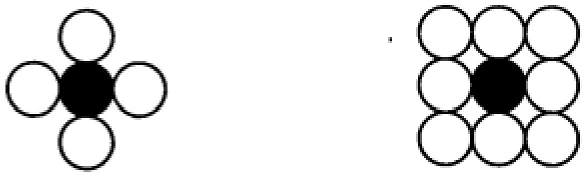
Experiment No. 5
Implement Area Filling Algorithm: Boundary Fill, Flood Fill.
Name: Tarun Premnarayan Pathak
Roll Number: 38
Date of Performance:
Date of Submission:

Experiment No. 5

Aim: To implement Area Filling Algorithm: Boundary Fill, Flood Fill.

Objective:
Polygon is an ordered list of vertices as shown in the following figure. For filling polygons with particular colors, we need to determine the pixels falling on the border of the polygon and those which fall inside the polygon. Objective is to demonstrate the procedure for filling polygons using different techniques.

Theory:
1) 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 processed outward pixel by pixel until the boundary color is encountered. A boundary-fill procedure accepts as input the coordinate of the interior point (x, y), a fill color, and a boundary color.

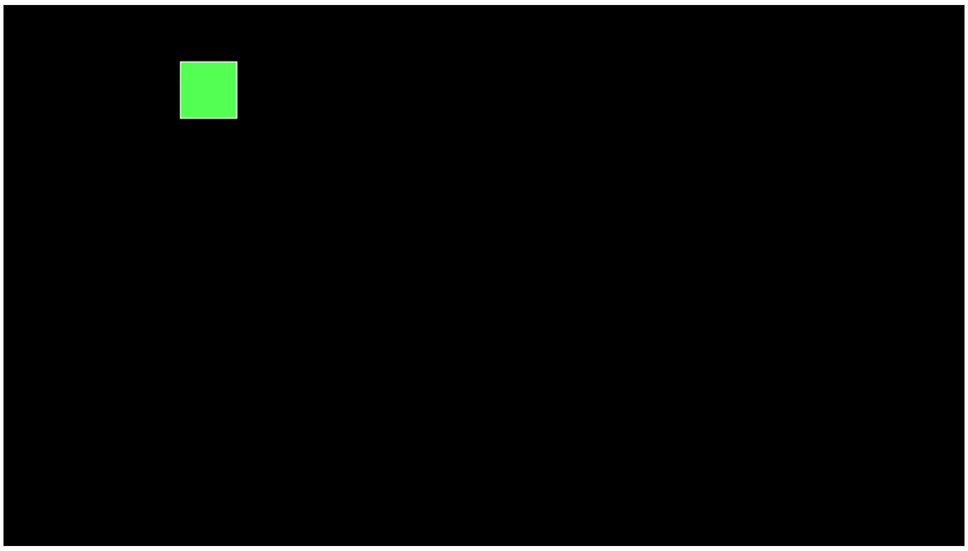

(a) Four connected region (b) Eight connected region

Procedure:
```
boundary_fill (x, y, f_color, b_color)
{
if (getpixel (x, y) != b_colour && getpixel (x, y) != f_colour)
        {
        putpixel (x, y, f_colour)
        boundary_fill (x + 1, y, f_colour, b_colour);
        boundary_fill (x, y + 1, f_colour, b_colour);
        boundary_fill (x - 1, y, f_colour, b_colour);
        boundary_fill (x, y - 1, f_colour, b_colour);
        }
}
```

Program:
```
#include <graphics.h>
#include <conio.h>
```

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

int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "c:\\turboc3\\bgi");

    rectangle(50, 50, 100, 100);

    boundary_fill(60, 61, 10, 15);

    getch();
    closegraph();
    return 0;
}
```

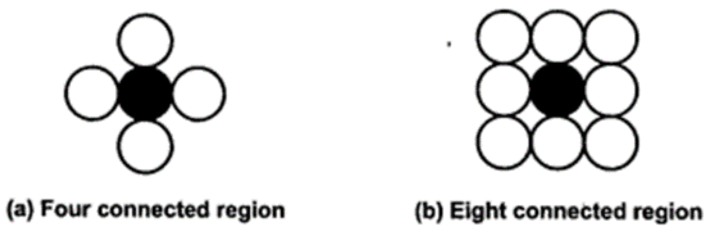
Output:



2) Flood Fill algorithm –

Sometimes we want to fill an area that is not defined within a single color boundary. We 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.

1. We start from a specified interior pixel (x, y) and reassign all pixel values that are currently set to a given interior color with the desired fill color.
2. If the area has more than one interior color, we can first reassign pixel values so that all interior pixels have the same color.
3. Using either 4-connected or 8-connected approach, we then step through pixel positions until all interior pixels have been repainted.



Procedure -

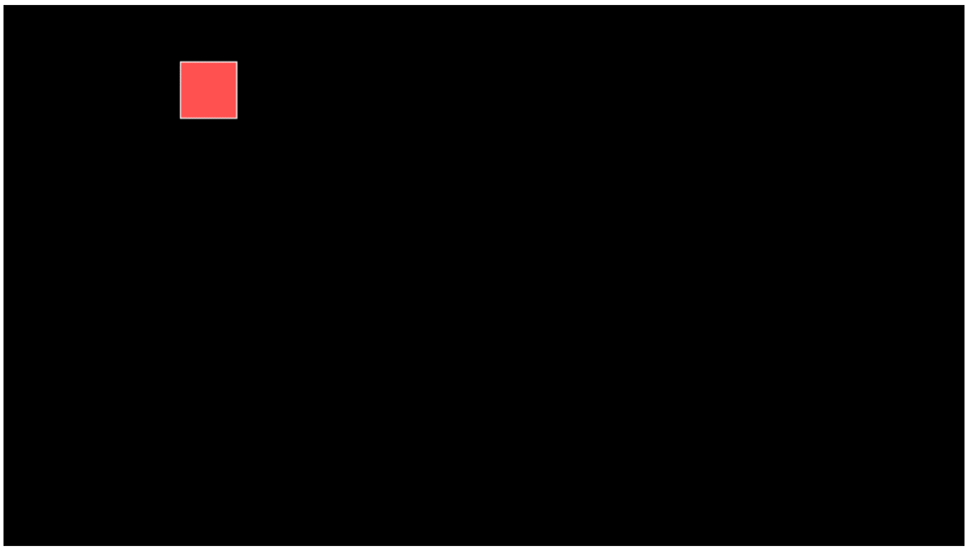
```
flood_fill (x, y, old_color, new_color)
{
if (getpixel (x, y) = old_colour)
        {
        putpixel (x, y, new_colour);
        flood_fill (x + 1, y, old_colour, new_colour);
        flood_fill (x - 1, y, old_colour, new_colour);
        flood_fill (x, y + 1, old_colour, new_colour);
        flood_fill (x, y - 1, old_colour, new_colour);
        flood_fill (x + 1, y + 1, old_colour, new_colour);
        flood_fill (x - 1, y - 1, old_colour, new_colour);
        flood_fill (x + 1, y - 1, old_colour, new_colour);
        flood_fill (x - 1, y + 1, old_colour, new_colour);
        }
}
```

Program:

```
#include<stdio.h>
#include<graphics.h>
#include<dos.h>
void flood(int,int,int,int);
int main()
```

```
{
int gd,gm=DETECT;
//detectgraph(&gd,&gm);
initgraph(&gd,&gm," ");
rectangle(50,50,100,100);
flood(55,55,12,0);
closegraph();
return 0;
}
void flood(int x,int y, int fill_col, int old_col)
{
if(getpixel(x,y)==old_col)
{
delay(10);
putpixel(x,y,fill_col);
flood(x+1,y,fill_col,old_col);
flood(x-1,y,fill_col,old_col);
flood(x,y+1,fill_col,old_col);
flood(x,y-1,fill_col,old_col);
flood(x + 1, y + 1, fill_col, old_col);
flood(x - 1, y - 1, fill_col, old_col);
flood(x + 1, y - 1, fill_col, old_col);
flood(x - 1, y + 1, fill_col, old_col);
}
}
```

Output:



Conclusion: Comment on

1. Importance of Flood fill- Flood fill is a vital graphics algorithm used for tasks such as coloring, image editing, and interactive interfaces. It's indispensable in graphic design, game development, and procedural content generation.
2. Limitation of methods- Flood fill struggles with areas containing gaps or complex boundaries. It may also lack fine control, leading to unwanted overspill, and its performance can degrade on large regions.
3. Usefulness of method- Flood fill's simplicity makes it an excellent choice for basic filling tasks in graphics, providing quick and interactive region filling. However, for intricate or more controlled operations, alternative techniques may be necessary.