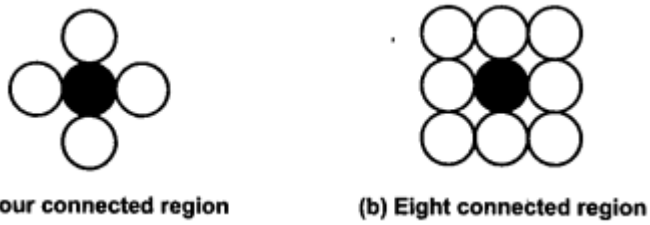
Experiment No. 5
Implement Area Filling Algorithm: Boundary Fill, Flood Fill.
Name: Aryan Gaikwad
Roll Number: 09
Date of Performance:
Date of Submission:

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:

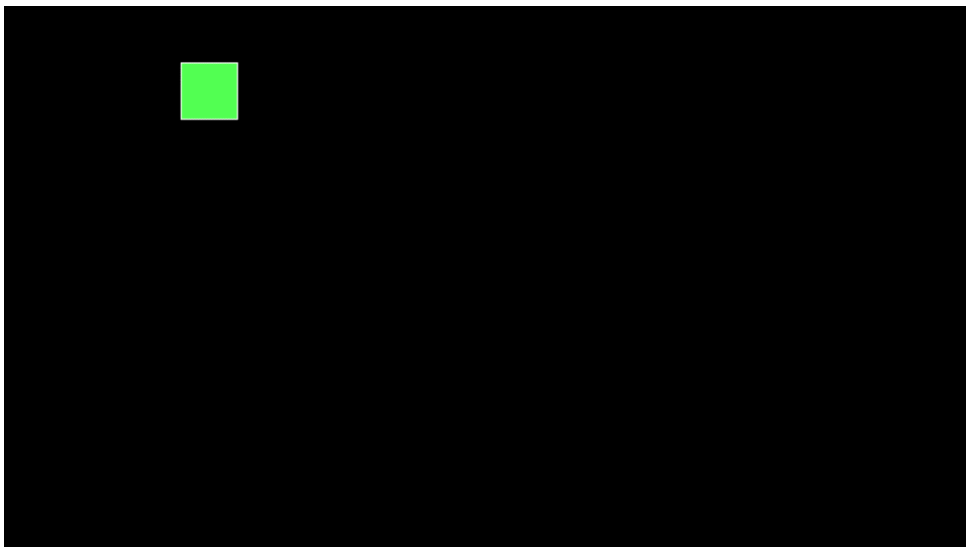
```
boundary_fill (x, y, f_color, b_color)
{
if (getpixel (x, y) != b_colour && getpixel (x, y) != f_colour)
        {
        putpixel (x, y, f_colour)
        boundary_fill (x + 1, y, f_colour, b_colour);
        boundary_fill (x, y + 1, f_colour, b_colour);
        boundary_fill (x - 1, y, f_colour,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

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

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
Importance of Flood Fill:

1. Vital for tasks like filling shapes and image segmentation, simplifying complex operations.
2. Essential in graphic design, image editing, and gaming for efficient area coloring.
3. Simplifies interactive drawing tools, eliminating the need for manual region filling.

Limitations:

1. Connectivity-dependent; best suited for simple 4 or 8-connectivity regions.
2. Prone to edge bleeding, causing colors to spill beyond region boundaries.
3. Performance may suffer in large images or complex regions.

Usefulness of Flood Fill:

1. Widely used in graphic design for efficient area coloring and erasing.
2. Vital in image segmentation to separate objects or regions of interest.
3. Applied in maze solving, gaming, and terrain modification for pathfinding and effects like filling and coloring.