Experiment No. 5
Topic : To implement Area Filling Algorithm:
Name: Pradeep Rathod
Roll Number: 49
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<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<doc.h>

 void boundary_fill(int x, int y, int fcolor, int
bcolor)

{

 if ((getpixel(x, y) != bcolor) && (getpixel(x, y) !=
fcolor))

 {

delay(10);

putpixel(x, y, fcolor);

boundary_fill(x + 1, y, fcolor, bcolor);
boundary_fill(x , y+1, fcolor, bcolor);
boundary_fill(x+1, y + 1, fcolor, bcolor);
boundary_fill(x-1, y - 1, fcolor, bcolor);
boundary_fill(x-1, y, fcolor, bcolor);
boundary_fill(x , y-1, fcolor, bcolor);
boundary_fill(x-1, y + 1, fcolor, bcolor);
boundary_fill(x+1, y - 1, fcolor, bcolor);

}

}

void main()

{

int x, y, fcolor, bcolor;

int gd=DETECT,gm;
```

```
initgraph(&gd, &gm, "C:\\TurboC3\\BGI");
printf("Enter the seed point (x,y) : ");
scanf("%d%d", &x, &y);

printf("Enter boundary color : "); scanf("%d",
&bcolor); printf("Enter new color : ");

scanf("%d", &fcolor); rectangle(50,50,100,100);
boundary_fill(x,y,fcolor,bcolor); getch();

}
```

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 -

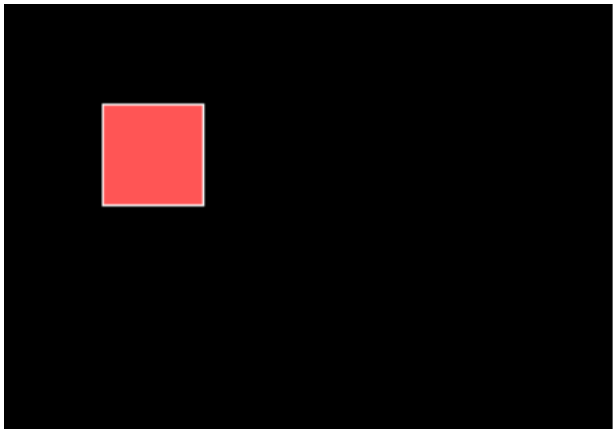
```
flood_fill (x, y, old_color, new_color)
{
if (getpixel (x, y) = old_colour)
        {
        putpixel (x, y, new_colour);
        flood_fill (x + 1, y, old_colour, new_colour);
        flood_fill (x - 1, y, old_colour, new_colour);
        flood_fill (x, y + 1, old_colour, new_colour);
        flood_fill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

1. Importance of Flood fill
2. Limitation of methods
3. Usefulness of method

Flood fill is crucial in computer graphics for various tasks, such as coloring regions, filling areas with patterns or textures, and identifying bounded regions. It is instrumental in creating

visually appealing and intuitive graphics in applications like drawing software, image editing tools, and video games.

Limitation of methods:

Despite its usefulness, flood fill algorithms can encounter limitations. One primary limitation is the possibility of stack overflow when dealing with large areas or when the recursion depth becomes too significant. Additionally, the traditional implementation of flood fill might not be suitable for handling complex shapes or irregular boundaries, as it may lead to leakage or bleeding of the fill outside the desired region

Usefulness of method:

The flood fill algorithm is incredibly useful in various applications, particularly in graphics and image processing. It enables the efficient filling of closed areas with desired colors, textures, or patterns. Moreover, it simplifies the implementation of numerous functionalities such as the paint bucket tool, region labeling, and boundary detection.