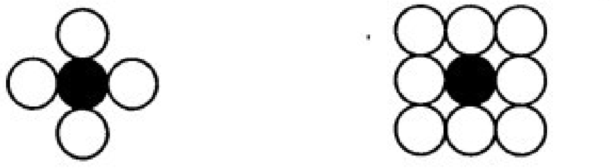
Experiment No. 5

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

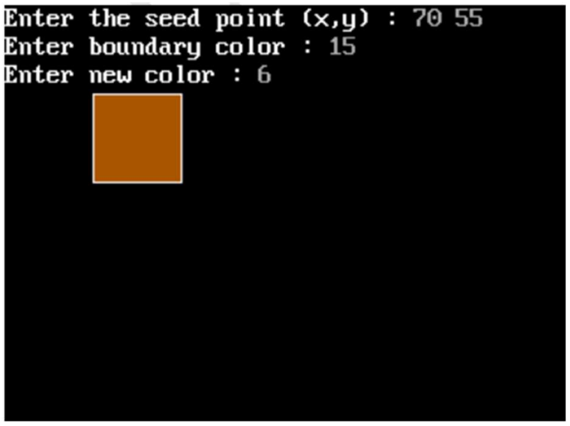
Procedure:
```
boundary_fill (x, y, f_color, b_color)
{
if (getpixel (x, y) != b_colour && getpixel (x, y) != f_colour)
        {
        putpixel (x, y, f_colour) boundary_fill (x +
        1, y, f_colour, b_colour); boundary_fill (x,
        y + 1, f_colour, b_colour); boundary_fill (x
        - 1,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

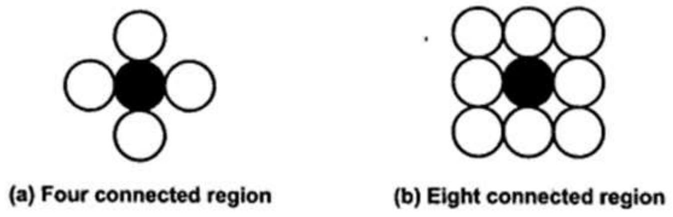


```
Enter the seed point (x,y) : 70 55
Enter boundary color : 15
Enter new color : 6
```

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.



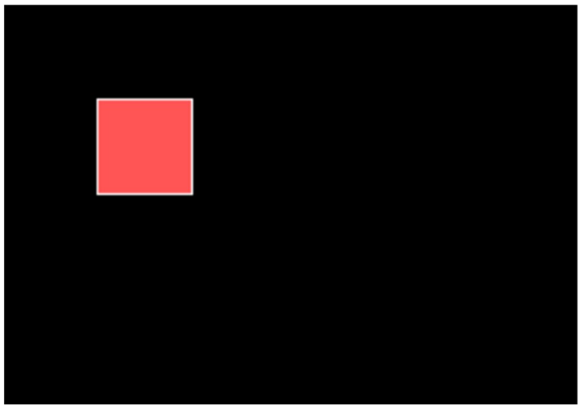
```
Procedure flood_fill (x, y, old_color,
new_color)
{
if (getpixel (x, y) = old_colour)
        {
        putpixel (x, y, new_colour); flood_fill (x + 1, y,
        old_colour, new_colour); flood_fill (x - 1, y,
        old_colour, new_colour); flood_fill (x, y + 1,
        old_colour, new_colour); 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,&g
m);
initgraph(&gd,&gm,"
");
rectangle(50,50,100,10
0);    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_c
ol)
{ 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

1. The flood fill method holds significant importance in computer graphics and image processing. Its primary function is to efficiently fill closed areas with a specific color or pattern, making it a fundamental tool for coloring, masking, and various graphic design tasks. It enhances user interaction in software applications through features like the "paint bucket" tool. Moreover, flood fill can be used for error correction, image segmentation, boundary tracing, and connected component analysis. Its applications extend to maze solving, data compression, terrain generation, resource allocation in GIS, and more, making it a versatile technique with a wide range of practical uses in the field of computer graphics and beyond.

2. Limitations of Flood Fill Method:

The flood fill method has several limitations. Firstly, it can only be applied to closed regions, making it ineffective for areas with gaps or openings in the boundaries. Furthermore, flood fill is primarily designed for uniform color or pattern filling, rendering it unsuitable for tasks requiring gradient fills or textured patterns. In some cases, the algorithm's computational complexity can be a drawback, particularly when filling very large regions, leading to performance issues. Additionally, the recursive version of flood fill can result in stack overflow errors when dealing with extensive areas or deep recursion, posing a challenge in memory-constrained environments.

Limitations of Boundary Fill Method:

The boundary fill method is primarily suited for contiguous regions with well-defined boundaries, and it struggles with irregular or disconnected shapes. It can sometimes produce ambiguous results or leak into unintended areas when confronted with complex or irregular boundaries. Moreover, the algorithm's recursive nature can be inefficient for filling large areas, limiting its use in real-time applications with performance constraints. Additionally, boundary fill requires the selection of a single seed point, and choosing the appropriate seed point for irregular shapes can be a challenging task, affecting the accuracy of the fill.

3. Flood fill and boundary fill methods are valuable tools in computer graphics and image processing. Flood fill is particularly useful for efficiently filling closed regions with a uniform color or pattern, making it an essential tool for coloring, masking, and user-friendly interactive features like the "paint bucket" tool. It aids in error correction, image segmentation, and boundary tracing, and it's integral in tasks such as maze solving, connected component analysis, and resource allocation in GIS.

On the other hand, the boundary fill method is beneficial for precisely filling regions with well-defined boundaries, which is essential in tasks like text rendering and creating precise shapes. It ensures that only the intended area gets filled without leakage. While it has limitations with irregular or disconnected shapes, when applied correctly, it can produce high-quality, clean results in boundary-based graphic applications. Both flood fill and boundary fill methods offer unique advantages and find their utility in various aspects of computer graphics and design, catering to different needs within the field.