Name:	MAHEK SHAH
Roll No:	
Class/Sem:	SE/III
Experiment No.:	5
Title:	Boundary Fill and Flood Fill Polygon filling Algorithm
Date of Performance:	
Date of Submission:	
Marks:	
Sign of Faculty:	

Experiment No. 5

Aim :Write a program to implement Boundary Fill and Flood Fill Polygon filling Algorithm in C.

Obective : **Boundary Fill Algorithm:**

This algorithm uses the recursive method. First of all, a starting pixel called the seed is considered. The algorithm checks whether boundary pixels or adjacent pixels are colored or not. If the adjacent pixel is already filled or colored then leave it, otherwise fill it. The filling is done using four connected or eight connected approaches.

1. Four connected approaches: In this approach, left, right, above, below pixels are tested.

2. Eight connected approaches: In this approach, left, right, above, below and four diagonals are selected.

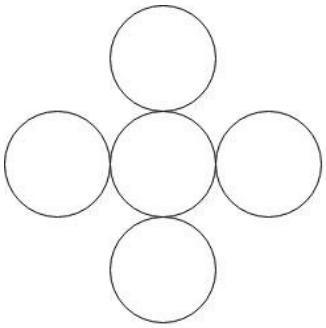
Boundary can be checked by seeing pixels from left and right first. Then pixels are checked by seeing pixels from top to bottom. The algorithm takes time and memory because some recursive calls are needed.

Flood Fill Algorithm:

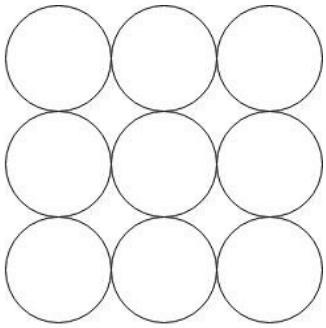
In this method, a point or seed which is inside region is selected. This point is called a seed point. Then four connected approaches or eight connected approaches is used to fill with specified color.

The flood fill algorithm has many characters similar to boundary fill. But this method is more suitable for filling multiple colors boundary. When boundary is of many colors and interior is to be filled with one color we use this algorithm.

In fill algorithm, we start from a specified interior point (x, y) and reassign all pixel values are currently set to a given interior color with the desired color. Using either a 4-connected or 8-connected approaches, we then step through pixel positions until all interior points have been repainted.



Four Connected



Eight Connected

Algorithm:

Boundary Fill

boundary_fill_8(x, y, fill_color, boundary_color) int current;
current=getpixel (x, y);
if ((current != fill_color) && (current != boundary_color)) {
setpixel (x, y, fill_color)
boundary_fill_8(x+1, y, fill_color , boundary_color); boundary_fill_8(x-1, y, fill_color , boundary_color); boundary_fill_8(x, y+1, fill_color , boundary_color); boundary_fill_8(x, y-1, fill_color , boundary_color); boundary_fill_8(x+1, y+1, fill_color , boundary_color); boundary_fill_8(x+1, y-1, fill_color , boundary_color); boundary_fill_8(x-1, y+1, fill_color , boundary_color); boundary_fill_8(x-1, y-1, fill_color , boundary_color); }

Flood Fill

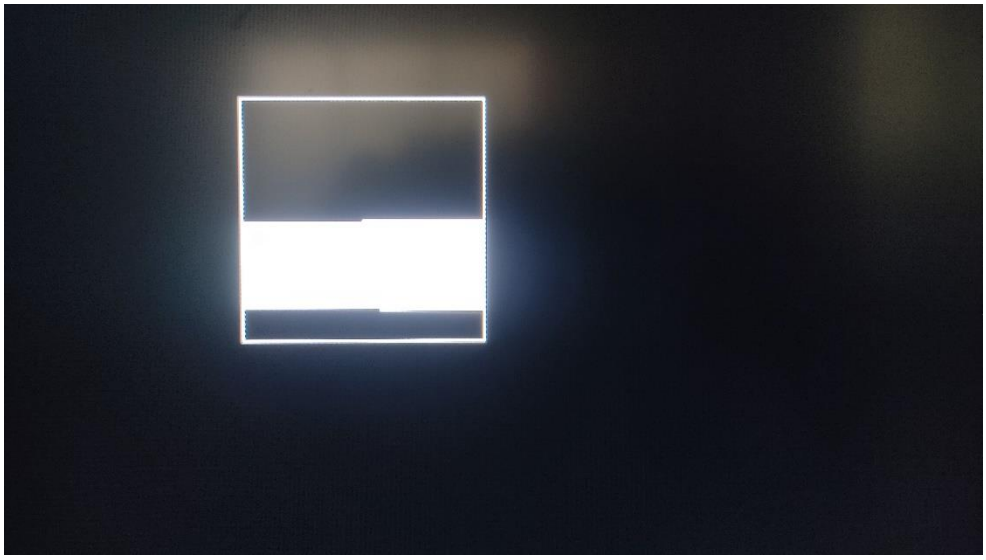
flood_fill_8(x, y, fill_color, default_color)
{
int current; current=getpixel (x, y);
if (current == default_color) {
setpixel (x, y, fill_color)
flood_fill_8(x+1, y, fill_color , default_color); flood_fill_8(x-1, y, fill_color , default_color); flood_fill_8(x, y+1, fill_color , default_color); flood_fill_8(x, y-1, fill_color , default_color); flood_fill_8(x+1, y+1, fill_color , default_color); flood_fill_8(x-1, y+1, fill_color , default_color); flood_fill_8(x+1, y-1, fill_color , default_color); flood_fill_8(x-1, y-1, fill_color , default_color); }
}

Code :

```c
#include <stdio.h>
#include <graphics.h>
void boundaryFill(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);
        boundaryFill(x + 1, y, fill_color, boundary_color);
        boundaryFill(x, y + 1, fill_color, boundary_color);
        boundaryFill(x - 1, y, fill_color, boundary_color);
        boundaryFill(x, y - 1, fill_color, boundary_color);
    }
}
void floodFill(int x, int y, int old_color, int new_color) {
    if (getpixel(x, y) == old_color) {
        putpixel(x, y, new_color);
        floodFill(x + 1, y, old_color, new_color);
        floodFill(x, y + 1, old_color, new_color);
        floodFill(x - 1, y, old_color, new_color);
        floodFill(x, y - 1, old_color, new_color);
```

```
    }
}
int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, NULL);
    int poly[] = {100, 100, 300, 100, 200, 300, 100, 100};
    drawpoly(4, poly);
    boundaryFill(200, 200, 6, 15);
    delay(2000);
    cleardevice();
    drawpoly(4, poly);
    floodFill(200, 200, 0, 6);
    delay(2000);
    closegraph();
    return 0;
}
```

Output:



Conclusion:

The Boundary Fill and Flood Fill algorithms are fundamental techniques for filling closed areas in computer graphics.

Boundary Fill:

- **Pros:** It is easy to implement and understand.
- **Cons:** It may face issues with performance and efficiency, especially for complex shapes. The algorithm might fill areas that are not intended to be filled due to the similarity of colors.

Flood Fill:

- **Pros:** It is generally more efficient than Boundary Fill for filling large areas.
- **Cons:** The recursive nature of the algorithm may lead to a stack overflow for large areas. It can also fill areas that are not enclosed by the boundary.
-

Both algorithms have their strengths and weaknesses, and the choice between them depends on the specific requirements of the application. In practice, more advanced algorithms like scanline fill algorithms or seed fill algorithms are often used for efficient and accurate polygon filling in computer graphics.