VIT Vidyalankar Institute of Technology ACCREDITED A+ BY NAAC	Department of Computer Engineering

Semester	S.E. Semester IV – CMPN
Division & Batch	Div C Batch 1
Subject	Computer Graphics
Laboratory Teacher:	Divya Nimbalkar
Laboratory	M310A

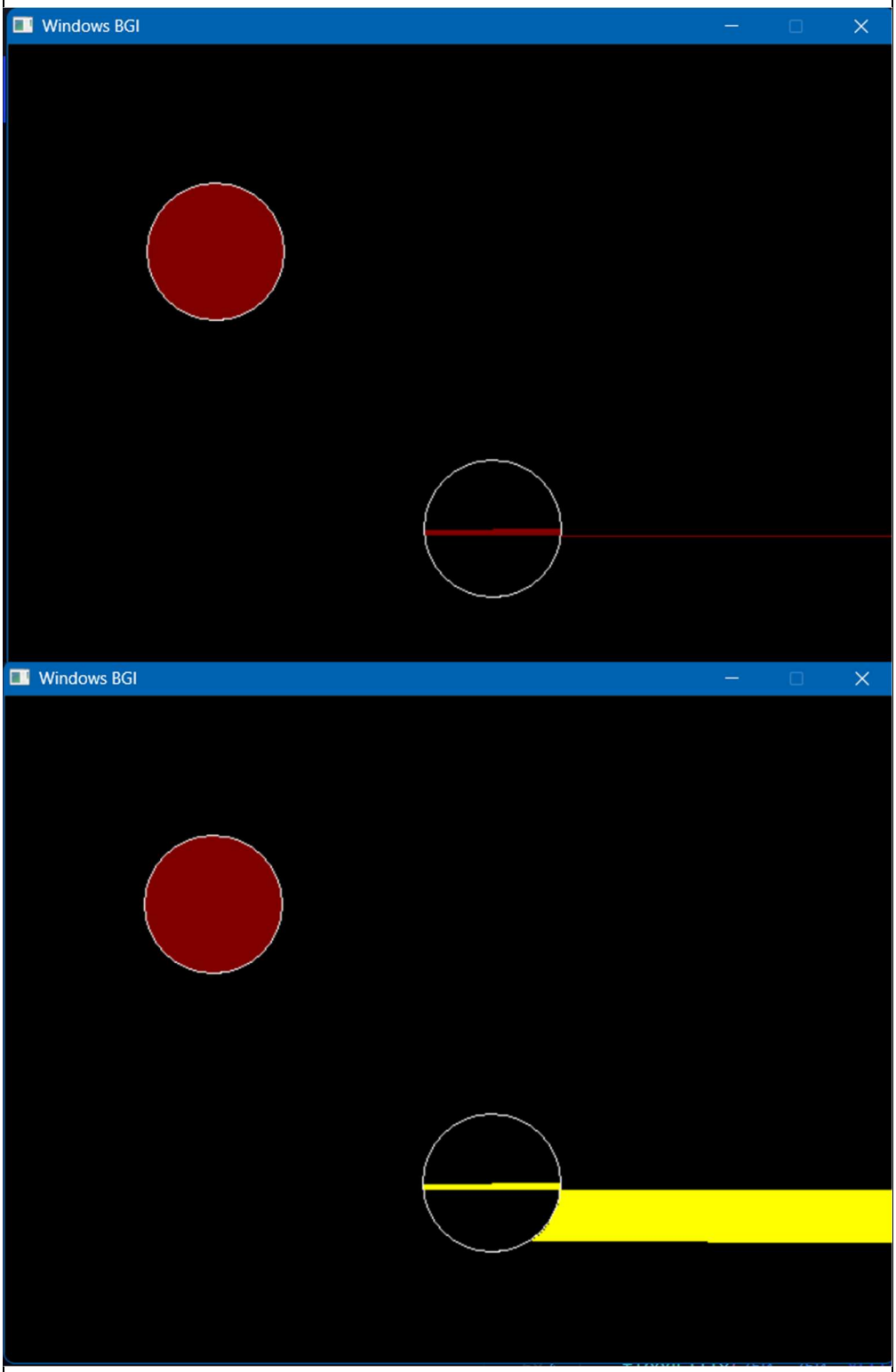
Student Name	Atharva Sheramkar	
Roll Number	24102B0069	
Grade and Subject Teacher's Signature		

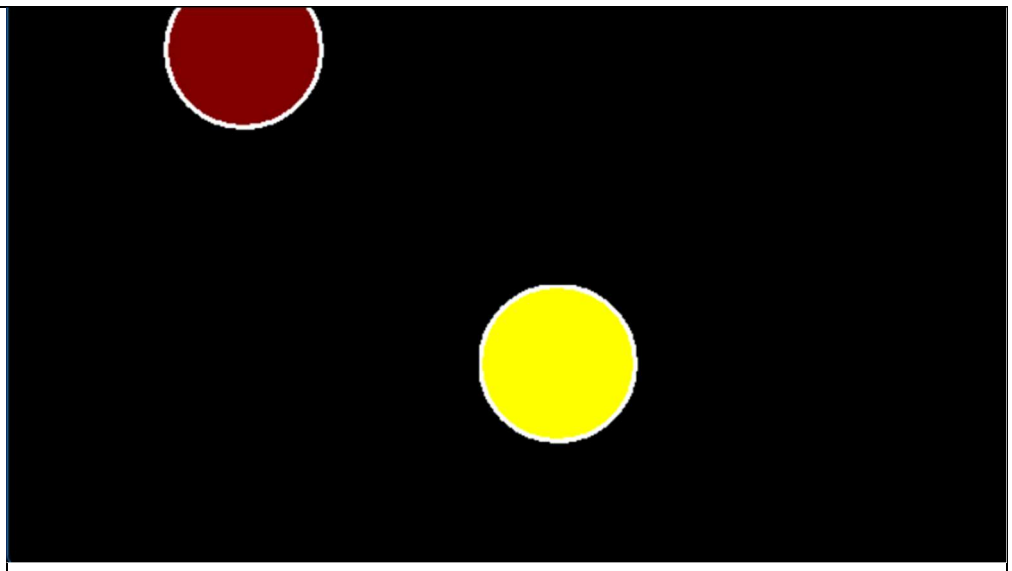
Experiment Number	4	
Experiment Title	**Implementation of Boundary Fill and Flood Fill Algorithm**	
Problem Statement	Implement these algorithms using 4 and 8 connected approaches Explicitly showcases cases where 4 connected may fail where 8 connected approaches may be used or vice versa.	
Resources / Apparatus Required	Hardware: Desktop	Software: Dev C++

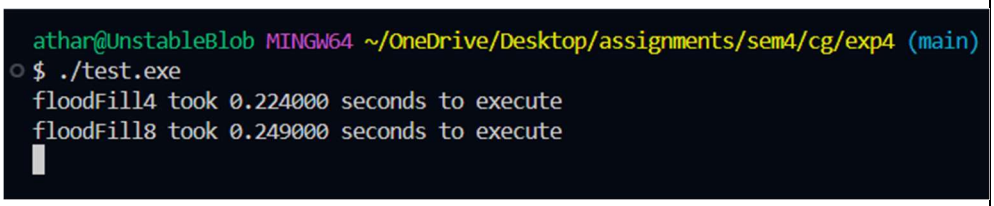
Code	BOUNDARY FILL #include <graphics.h> #include <stdio.h> #include <conio.h> #include<time.h> void boundaryFill4(int *x*, int *y*, int *fill_color*, int *boundary_color*) { int current_color = getpixel(*x*, *y*); if (current_color != *boundary_color* && current_color != *fill_color*) { putpixel(*x*, *y*, *fill_color*); boundaryFill4(*x* + 1, *y*, *fill_color*, *boundary_color*); boundaryFill4(*x* - 1, *y*, *fill_color*, *boundary_color*); boundaryFill4(*x*, *y* + 1, *fill_color*, *boundary_color*); boundaryFill4(*x*, *y* - 1, *fill_color*, *boundary_color*); } } void boundaryFill8(int *x*, int *y*, int *fill_color*, int *boundary_color*) { int current_color = getpixel(*x*, *y*); if (current_color != *boundary_color* && current_color != *fill_color*) { putpixel(*x*, *y*, *fill_color*); boundaryFill8(*x* + 1, *y*, *fill_color*, *boundary_color*); boundaryFill8(*x* - 1, *y*, *fill_color*, *boundary_color*); boundaryFill8(*x*, *y* + 1, *fill_color*, *boundary_color*); boundaryFill8(*x*, *y* - 1, *fill_color*, *boundary_color*); boundaryFill8(*x* + 1, *y* + 1, *fill_color*, *boundary_color*); boundaryFill8(*x* - 1, *y* - 1, *fill_color*, *boundary_color*); boundaryFill8(*x* + 1, *y* - 1, *fill_color*, *boundary_color*); boundaryFill8(*x* - 1, *y* + 1, *fill_color*, *boundary_color*); } } int main() { int x1, y1, x2, y2; x1 = 100; x2 = 200; y1=100; y2 = 200; int gd = DETECT, gm; initgraph(&gd, &gm, (char*)""); clock_t start, end; double cpu_time_used; setcolor(WHITE); *// setlinestyle(SOLID_LINE, 0, 3);*

```
    circle(150,150,50);

    start = clock();
    boundaryFill4(150, 150, RED, WHITE);
    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("boundaryFill4 took %f seconds to execute \n", cpu_time_used);

    x1 = 300;
    x2 = 400;
    y1  = 300;
    y2 = 400;

    setcolor(WHITE);
    // setlinestyle(SOLID_LINE, 0, 3);
    circle(350,350,50);

    start = clock();
    boundaryFill8(350, 350, RED, WHITE);
    end = clock();

    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("boundaryFill8 took %f seconds to execute \n", cpu_time_used);

    getch();
    closegraph();
    return 0;
}
 FLOOD FILL
#include <graphics.h>
 #include <stdio.h>
 #include <conio.h>
 #include <time.h>

 // 4-connected Flood Fill Algorithm
 void floodFill4(int x, int y, int new_color, int old_color) {
    int current_color = getpixel(x, y);

    if (current_color == old_color) {
        putpixel(x, y, new_color);

        floodFill4(x + 1, y, new_color, old_color);  // Right
        floodFill4(x - 1, y, new_color, old_color);  // Left
        floodFill4(x, y + 1, new_color, old_color);  // Down
        floodFill4(x, y - 1, new_color, old_color);  // Up
    }
 }

 // 8-connected Flood Fill Algorithm
 void floodFill8(int x, int y, int new_color, int old_color) {
    int current_color = getpixel(x, y);

    if (current_color == old_color) {
```

```
        putpixel(x, y, new_color);

        // 4-connected neighbors
        floodFill8(x + 1, y, new_color, old_color);      // Right
        floodFill8(x - 1, y, new_color, old_color);      // Left
        floodFill8(x, y + 1, new_color, old_color);      // Down
        floodFill8(x, y - 1, new_color, old_color);      // Up

        // Diagonal neighbors
        floodFill8(x + 1, y + 1, new_color, old_color);  // Bottom-right
        floodFill8(x - 1, y - 1, new_color, old_color);  // Top-left
        floodFill8(x + 1, y - 1, new_color, old_color);  // Top-right
        floodFill8(x - 1, y + 1, new_color, old_color);  // Bottom-left
    }
}

int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, (char*)"");

    clock_t start, end;
    double cpu_time_used;

    setlinestyle(SOLID_LINE, 0, 3);
    setcolor(WHITE);
    circle(150, 150, 50);

    int old_color = getpixel(150, 150);  // Get the color to be replaced

    start = clock();
    floodFill4(150, 150, RED, old_color);
    end = clock();

    cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("floodFill4 took %f seconds to execute\n", cpu_time_used);

    setlinestyle(SOLID_LINE, 0, 3);
    setcolor(WHITE);
    circle(350, 350, 50);

    old_color = getpixel(350, 350);  // Get the color to be replaced

    start = clock();
    floodFill8(350, 350, YELLOW, old_color);
    end = clock();

    cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("floodFill8 took %f seconds to execute\n", cpu_time_used);

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


Windows BGI
Windows BGI




athar@UnstableBlob MINGW64 ~/OneDrive/Desktop/assignments/sem4/cg/exp4 (main)
$ ./test.exe
floodFill4 took 0.224000 seconds to execute
floodFill8 took 0.249000 seconds to execute

Conclusion	In this experiment, the **Flood Fill** and **Boundary Fill** algorithms were implemented to fill enclosed areas on the screen using both **4-connected** and **8-connected** approaches. The Flood Fill algorithm successfully filled a region by replacing all connected pixels of the same interior color with a new color, while the Boundary Fill algorithm filled the area until a specified boundary color was encountered. Using the **4-connected method**, pixels were filled based on their four direct neighbors (up, down, left, right). In contrast, the **8-connected method** also considered diagonal neighbors, resulting in more complete filling for regions with diagonal gaps. The output confirms that the filling process follows the screen coordinate system with the origin at the top-left corner. Overall, both algorithms proved efficient and suitable for basic computer graphics applications involving region filling and shape coloring.