**EXPERIMENT NO 5**

**TRUSHANT RATHOD**

**COMPS 18**

**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.



**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);

}

}

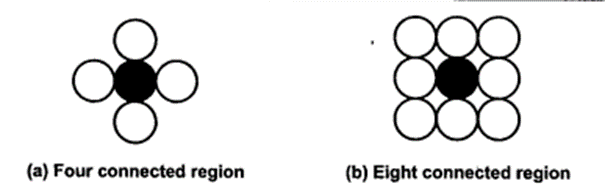
**2) Flood Fill algorithm –**

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<conio.h>

#include<graphics.h>

void boundaryfill(int x,int y,int f\_c,int b\_c)

{

int current;

current=getpixel(x,y);

if(current!=b\_c && current!=f\_c)

{

putpixel(x,y,f\_c);

boundaryfill(x,y+1,f\_c,b\_c);

boundaryfill(x,y-1,f\_c,b\_c);

boundaryfill(x+1,y,f\_c,b\_c);

boundaryfill(x-1,y,f\_c,b\_c);

}

}

void floodfill(int x1,int y1,int o\_c,int n\_c)

{

int current;

current=getpixel(x1,y1);

if(current==o\_c)

{

putpixel(x1,y1,n\_c);

floodfill(x1,y1+1,o\_c,n\_c);

floodfill(x1,y1-1,o\_c,n\_c);

floodfill(x1-1,y1,o\_c,n\_c);

floodfill(x1+1,y1,o\_c,n\_c);

}

}

void main()

{

int gd,gm,x,y,x1,y1,r;

gd=DETECT;

printf("Enter the origin points for circle 1 & 2:");

scanf("%d %d %d %d",&x,&y,&x1,&y1);

printf("\n Enter radius:");

scanf("%d",&r);

initgraph(&gd,&gm,"..//bgi");

circle(x,y,r);

circle(x1,y1,r);

boundaryfill(x,y,5,15);

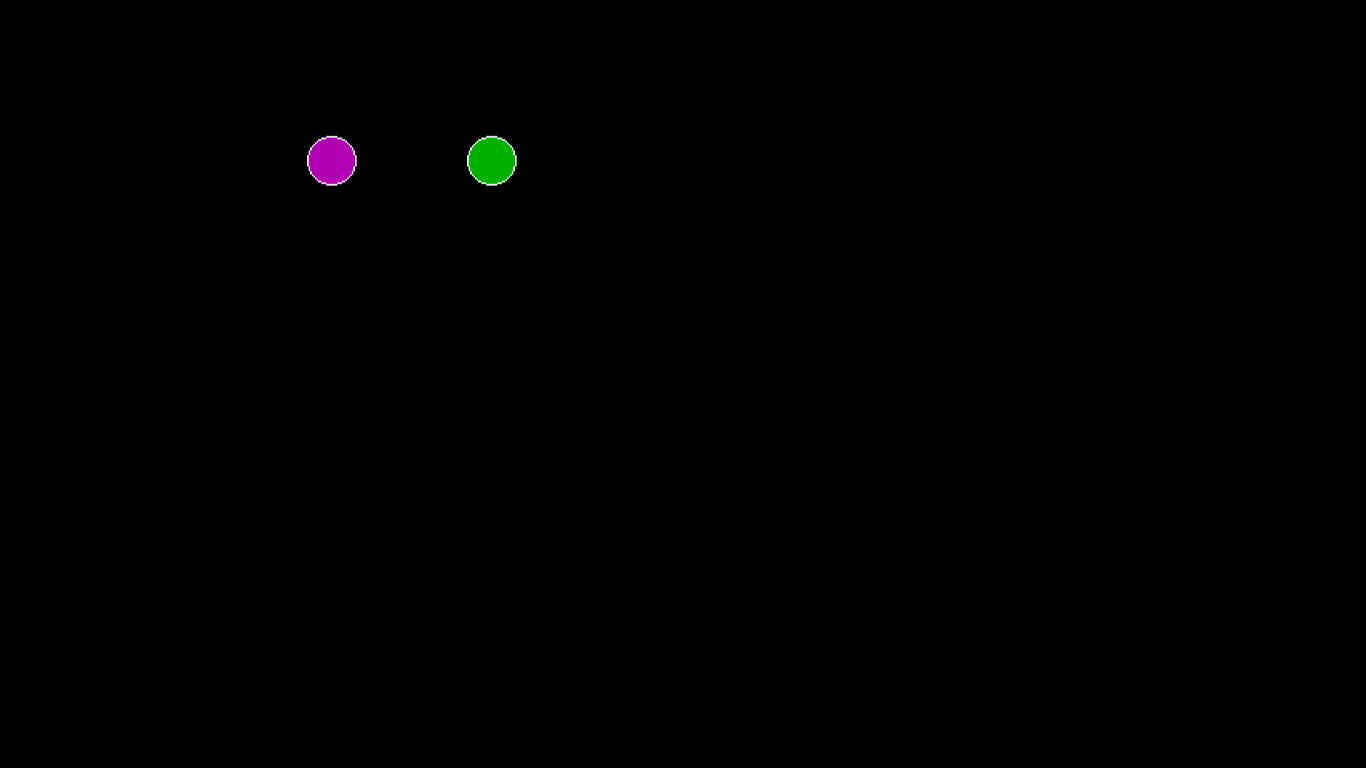
floodfill(x1,y1,0,2);

getch();

closegraph();

}

**Output:**

****

**Conclusion:**

**1.Importance of Flood fill**

1. **Image Editing**: Flood fill is widely used in image editing software to perform tasks like colour replacement, painting, and erasing within a defined region.
2. **Segmentation**: It plays a crucial role in image segmentation, helping separate different objects or regions of interest within an image.
3. **Data Visualization**: Flood fill can be used to highlight and visually represent specific data points or regions in data visualization and charting.
4. **Computer Vision**: In computer vision applications, flood fill helps identify and analyse connected components or regions in an image, which is vital for tasks like object detection and tracking.
5. **Interactive Graphics**: It's essential for interactive graphics, enabling users to paint or fill regions with colours in applications, games, and graphical user interfaces.
6. **Boundary Detection**: Flood fill can be used in combination with edge detection algorithms to find the boundaries of regions or objects.

**2. Limitation of methods:**

1. **Overfilling**: Both algorithms may overfill regions if not properly constrained, leading to unintended results.
2. **Complex Boundaries**: They struggle with complex or overlapping boundaries, potentially causing leakage.
3. **Recursive Depth**: Recursive implementations can lead to stack overflow errors for large regions.
4. **Slow for Large Areas**: Flood fill can be slow for large areas due to the time needed to process all pixels.

**3. Usefulness of method:**

1. **Flood Fill**: Useful for filling closed areas with a specific color, essential in image editing, segmentation, and interactive graphics.
2. **Boundary Fill**: Helpful for colouring regions with distinct boundaries, often used in graphics and paint software for colouring enclosed shapes.