Program 7

Aim: Write a program to fill the polygon using

a) Boundary Fill

b) Flood Fill

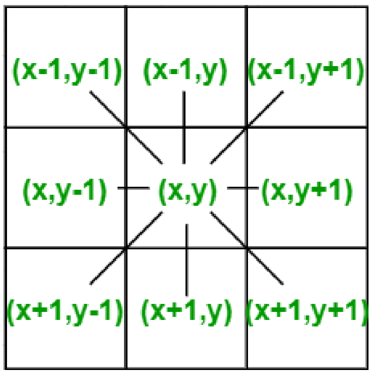
c) Scan Fill methods.

Theory:

Boundary Fill

The boundary fill algorithm works as its name. This algorithm picks a point inside an object and starts to fill until it hits the boundary of the object. The color of the boundary and the color that we fill should be different for this algorithm to work.

In this algorithm, we assume that color of the boundary is same for the entire object. The boundary fill algorithm can be implemented by 4-connected pixels or 8-connected pixels.

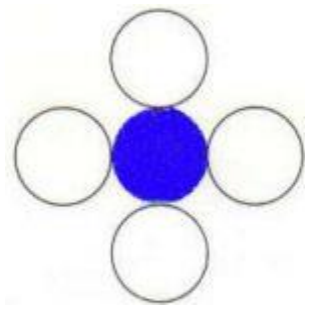


Flood Fill

Sometimes we come across an object where we want to fill the area and its boundary with different colors. We can paint such objects with a specified interior color instead of searching for particular boundary color as in boundary filling algorithm.

Instead of relying on the boundary of the object, it relies on the fill color. In other words, it replaces the interior color of the object with the fill color. When no more pixels of the original interior color exist, the algorithm is completed.

In Flood Fill algorithm we start with some seed and examine the neighboring pixels, however pixels are checked for a specified interior color instead of boundary color and is replaced by a new color. It can be done using 4 connected or 8 connected region method.

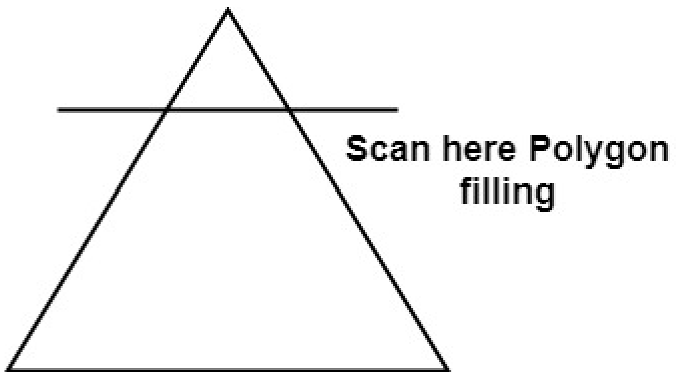


4 Connected Region

Scan Fill

This algorithm lines interior points of a polygon on the scan line and these points are done on or off according to requirement. The polygon is filled with various colors by coloring various pixels.

In above figure polygon and a line cutting polygon in shown. First of all, scanning is done. Scanning is done using raster scanning concept on display device. The beam starts scanning from the top left corner of the screen and goes toward the bottom right corner as the endpoint. The algorithms find points of intersection of the line with polygon while moving from left to right and top to bottom. The various points of intersection are stored in the frame buffer. The intensities of such points is keep high. Concept of coherence property is used. According to this property if a pixel is inside the polygon, then its next pixel will be inside the polygon.



Algorithm:

Boundary Fill

```
Procedure fill (x, y, color, color1: integer)
int c;
c=getpixel (x, y);
if (c!=color) (c!=color1) {
    setpixel (x, y, color);
    fill (x+1, y, color, color 1);
    fill (x-1, y, color, color 1);
    fill (x, y+1, color, color 1);
    fill (x, y-1, color, color 1);
}
```

Flood Fill

Step 1 − Initialize the value of seed point (seedx, seedy), fcolor and dcol.

Step 2 − Define the boundary values of the polygon.

Step 3 − Check if the current seed point is of default color, then repeat the steps 4 and 5 till the boundary pixels reached.

If getpixel(x, y) = dcol then repeat step 4 and 5

Step 4 − Change the default color with the fill color at the seed point.

setPixel(seedx, seedy, fcol)

Step 5 − Recursively follow the procedure with four neighborhood points.

FloodFill (seedx – 1, seedy, fcol, dcol)

FloodFill (seedx + 1, seedy, fcol, dcol)

FloodFill (seedx, seedy - 1, fcol, dcol)

FloodFill (seedx – 1, seedy + 1, fcol, dcol)

Step 6 – Exit

Scan Fill

Step 1 − Find out the Y_{min} and Y_{max} from the given polygon.

Step 2 – Scan Line intersects with each edge of the polygon from Y_{min} to Y_{max}. Name each intersection point of the polygon. As per the figure shown above, they are named as p0, p1, p2, p3.

Step 3 − Sort the intersection point in the increasing order of X coordinate i.e. (p0, p1), (p1, p2), and (p2, p3).

Step 4 − Fill all those pair of coordinates that are inside polygons and ignore the alternate pairs.

Step 5 − **Exit**

Code:

Boundary Fill

```
#include <conio.h>
#include <stdio.h>
#include <graphics.h>
#include <dos.h>

void fill_right(int x, int y);
void fill_left(int x, int y);

int main(){
    int n, i, x, y;
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

    line(50, 50, 200, 50);
    line(200, 50, 200, 300);
    line(200, 300, 50, 300);
    line(50, 300, 50, 50);

    x = 100;
    y = 100;
    fill_right(x, y);
    fill_left(x - 1, y);
    getch();
    return 0;
}

void fill_right(int x, int y){
    if((getpixel(x, y) != WHITE) && (getpixel(x, y) != RED)){
        putpixel(x, y, RED);
        fill_right(++x, y);
        x = x - 1;
            fill_right(x, y - 1);
            fill_right(x, y + 1);
    }
    delay(0);
}

void fill_left(int x, int y){
    if((getpixel(x, y) != WHITE) && (getpixel(x, y) != RED)){
        putpixel(x, y, RED);
        fill_left(--x, y);
        x = x + 1;
            fill_left(x, y - 1);
            fill_left(x, y + 1);
    }
    delay(0);
}
```

Flood Fill

```
#include <stdio.h>
#include <graphics.h>
#include <dos.h>
#include <conio.h>

void floodFill(int x, int y, int oldcolor, int newcolor){
    if(getpixel(x, y) == oldcolor){
        putpixel(x, y, newcolor);
        floodFill(x + 1, y, oldcolor, newcolor);
        floodFill(x, y + 1, oldcolor, newcolor);
        floodFill(x - 1, y, oldcolor, newcolor);
        floodFill(x, y - 1, oldcolor, newcolor);
    }
}

int main(){
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
    int x, y, radius;

    printf("Enter x and y positions for circle\n");
    scanf("%d%d", &x, &y);
    printf("Enter radius of circle\n");
    scanf("%d", &radius);
    circle(x, y, radius);
    floodFill(x, y, 0, 15);
    delay(3000);
    getch();
    closegraph();
    return 0;
}
```

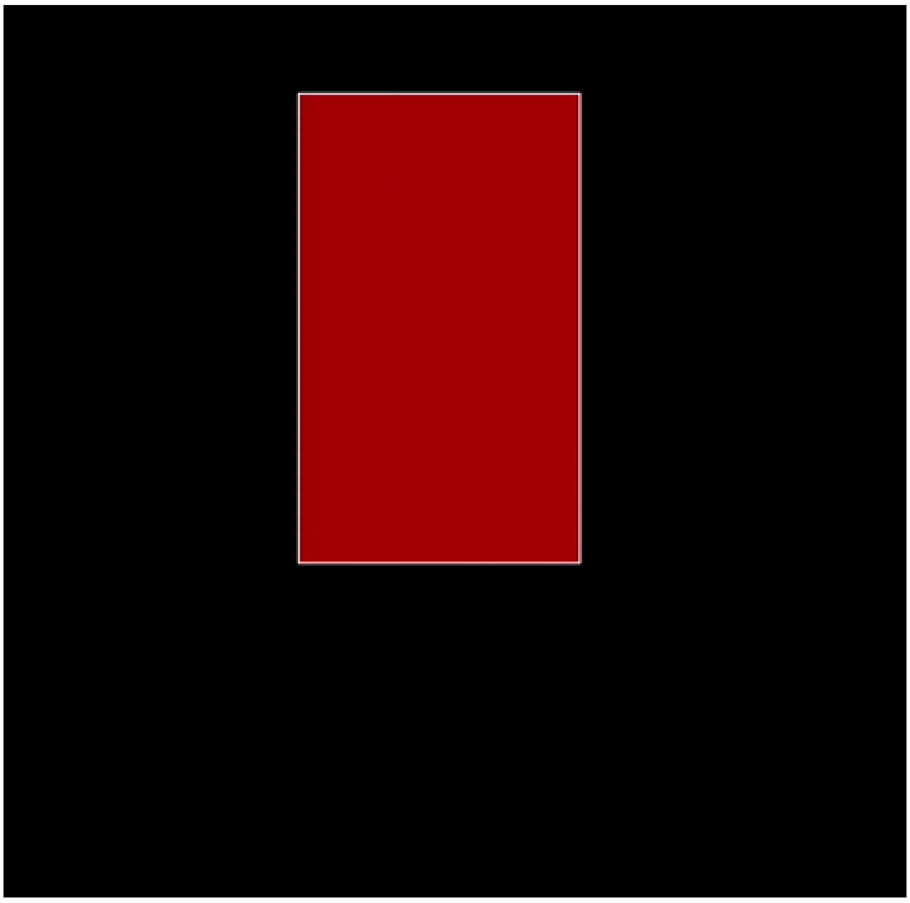
Scan Fill

```
#include <iostream.h>
#include <conio.h>
#include <graphics.h>
#include <dos.h>

struct edge{
    int x1, y1, x2, y2, flag;
};

int main(){
    int n, i, j, k;
    struct edge ed[10], temped;
    float dx, dy, m[10], x_int[10], inter_x[10];
    int x[10], y[10], ymax = 0, ymin = 480, yy, temp;
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

    cout <<"Enter the no.of vertices of the graph: ";
    cin >> n;
```

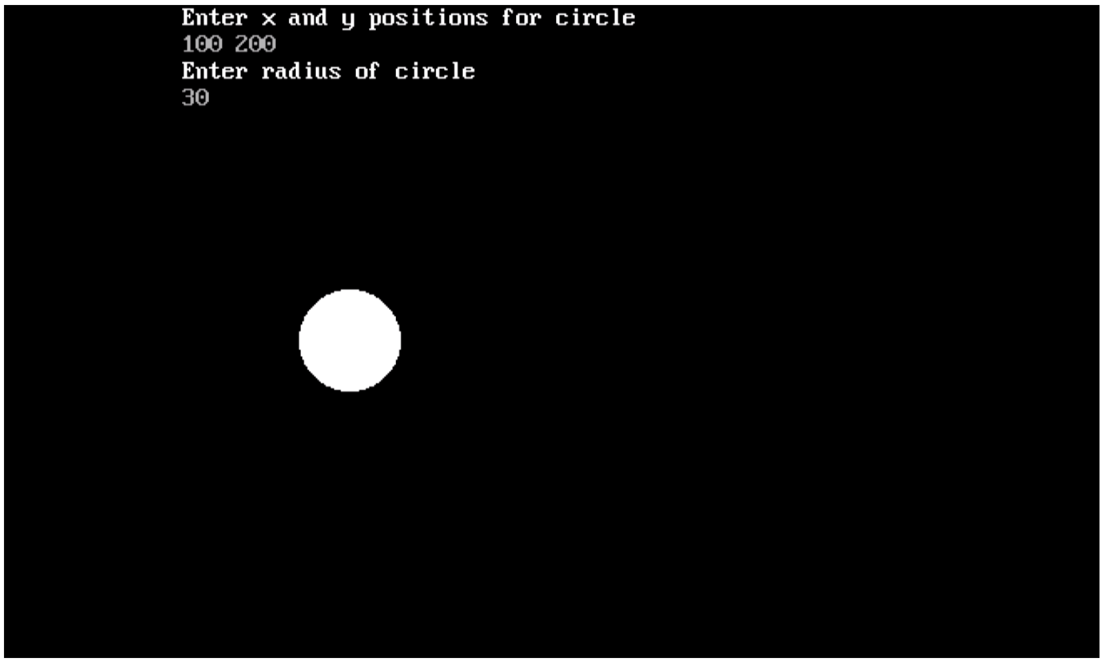
```cpp
    cout <<"Enter the vertices";
    for(i = 0; i < n; i++){
        cin >> x[i];
        cin >> y[i];
        if (y[i] > ymax)
            ymax = y[i];
        if (y[i] < ymin)
            ymin = y[i];
        ed[i].x1 = x[i];
        ed[i].y1 = y[i];
    }
    for(i = 0; i < n - 1; i++){
        ed[i].x2 = ed[i + 1].x1;
        ed[i].y2 = ed[i + 1].y1;
        ed[i].flag = 0;
    }
    ed[i].x2 = ed[0].x1;
    ed[i].y2 = ed[0].y1;
    ed[i].flag = 0;
    for(i = 0; i < n; i++){
        if(ed[i].y1 < ed[i].y2){
            temp = ed[i].x1;
            ed[i].x1 = ed[i].x2;
            ed[i].x2 = temp;
            temp = ed[i].y1;
            ed[i].y1 = ed[i].y2;
            ed[i].y2 = temp;
        }
    }
    for(i = 0; i < n; i++){
        line(ed[i].x1, ed[i].y1, ed[i].x2, ed[i].y2);
    }
    for(i = 0; i < n - 1; i++){
        for(j = 0; j < n - 1; j++){
            if (ed[j].y1 < ed[j + 1].y1){
                temped = ed[j];
                ed[j] = ed[j + 1];
                ed[j + 1] = temped;
            }
            if(ed[j].y1 == ed[j + 1].y1){
                if (ed[j].y2 < ed[j + 1].y2){
                    temped = ed[j];
                    ed[j] = ed[j + 1];
                    ed[j + 1] = temped;
                }
                if(ed[j].y2 == ed[j + 1].y2){
                    if (ed[j].x1 < ed[j + 1].x1){
                        temped = ed[j];
                        ed[j] = ed[j + 1];
                        ed[j + 1] = temped;
                    }
                }
            }
        }
    }
    for(i = 0; i < n; i++){
        dx = ed[i].x2 - ed[i].x1;
        dy = ed[i].y2 - ed[i].y1;
        if(dy == 0){
            m[i] = 0;
        }else{
            m[i] = dx / dy;
        }
        inter_x[i] = ed[i].x1;
    }
    yy = ymax;
    while(yy > ymin){
        for(i = 0; i < n; i++){
            if(yy > ed[i].y2 && yy <= ed[i].y1){
                ed[i].flag = 1;
            }else
                ed[i].flag = 0;
        }
        j = 0;
        for(i = 0; i < n; i++){
            if(ed[i].flag == 1){
                if(yy == ed[i].y1){
                    j++;
                    if(ed[i - 1].y1 == yy && ed[i - 1].y1 < yy)
{
                        x_int[j] = ed[i].x1;
                        j++;
                    }
                    if(ed[i + 1].y1 == yy && ed[i + 1].y1 < yy)
{
                        x_int[j] = ed[i].x1;
                        j++;
                    }
                }else{
                    x_int[j] = inter_x[i] + (-m[i]);
                    inter_x[i] = x_int[j];
                    j++;
                }
            }
        }
        for(i = 0; i < j; i++){
            for(k = 0; k < j - 1; k++){
                if (x_int[k] > x_int[k + 1])
                {
                    temp = (int)x_int[k];
                    x_int[k] = x_int[k + 1];
                    x_int[k + 1] = temp;
                }
            }
        }
        for(i = 0; i < j; i = i + 2){
            line((int)x_int[i], yy, (int)x_int[i + 1], yy);
        }
        yy--;
        delay(50);
    }
    getch();
    closegraph();
}
```

Output:

Boundary Fill

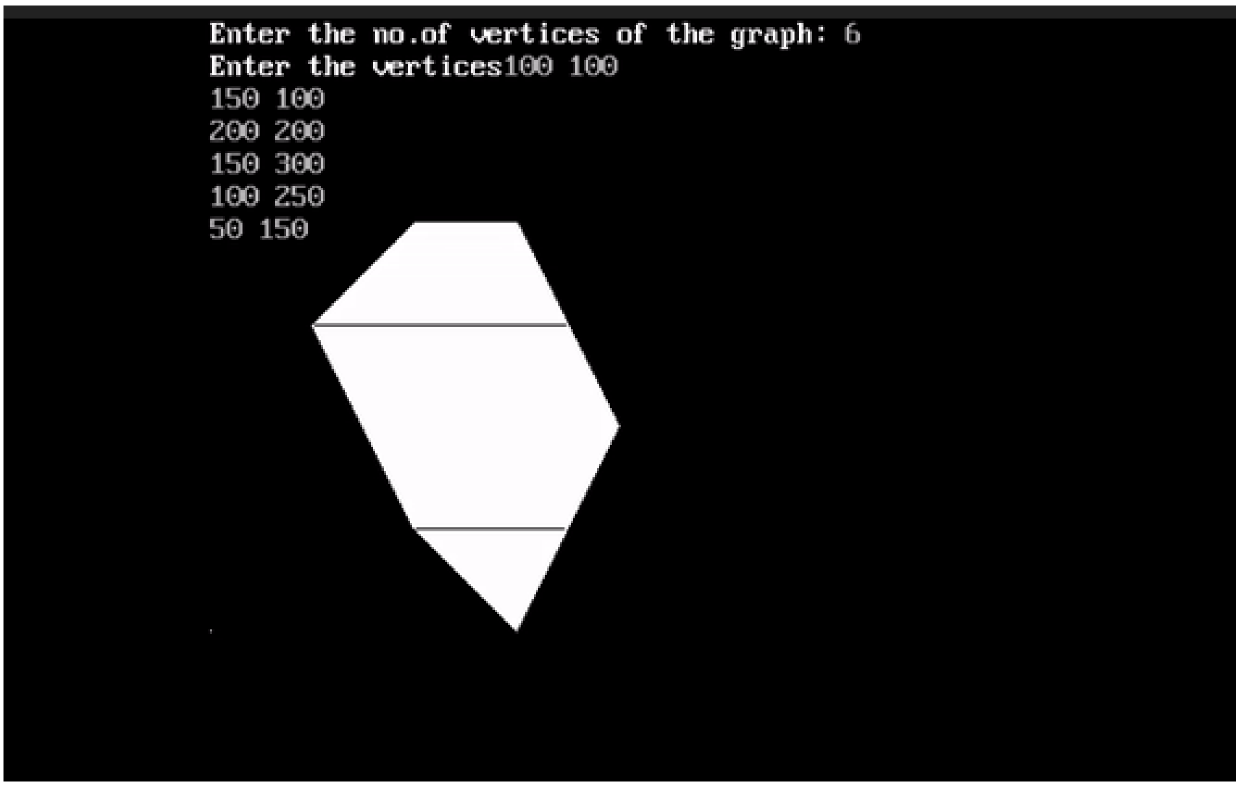


Flood Fill



```
Enter x and y positions for circle
100 200
Enter radius of circle
30
```

Scan Fill



```
Enter the no.of vertices of the graph: 6
Enter the vertices100 100
150 100
200 200
150 300
100 250
50 150
```