**Practical No 6.A**

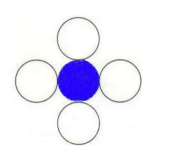
**Implementation of boundary fill algorithm.**

**Aim: Write a program to implement a boundary fill algorithm.**

**Theory:**

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.

**4-Connected Polygon**

In this technique 4-connected pixels are used as shown in the figure. We are putting the pixels above, below, to the right, and to the left side of the current pixels and this process will continue until we find a boundary with different color.

**Algorithm:**

**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)

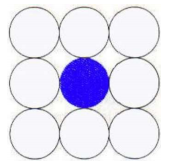
FloodFill (seedx + 1, seedy, fcol, dcol)

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

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

**Step 6 –** Exit

**8-Connected Polygon**

In this technique 8-connected pixels are used as shown in the figure. We are putting pixels above, below, right and left side of the current pixels as we were doing in 4-connected technique. In addition to this, we are also putting pixels in diagonals so that entire area of the current pixel is covered. This process will continue until we find a boundary with different color.

**Algorithm:**

**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 neighbourhood points

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

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

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

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

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

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

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

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

**Step 6 –** Exit

**Conclusion: We have implemented Boundary Fill Algorithm.**

**Code:**

**Boundary Fill Algorithm using 4-Connectors**

#include<iostream.h>

#include<graphics.h>

#include<dos.h>

#include<conio.h>

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

{

if(getpixel(x,y)!=b\_color && getpixel(x,y)!=f\_color)

{

putpixel(x,y,f\_color);

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

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

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

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

}

}

void main()

{

clrscr();

int gm,gd=DETECT,r;

int x,y;

cout<<"Enter 'X' and 'Y' positions of the Circle:\n";

cin>>x>>y;

cout<<"Enter Radius of the Circle:";

cin>>r;

initgraph(&gd,&gm,"C:/TURBOC3/BGI");

circle(x,y,r);

boundaryfill(x,y,4,15);

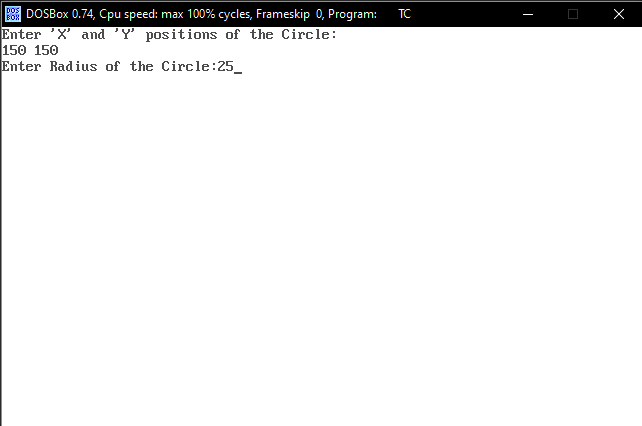
delay(2);

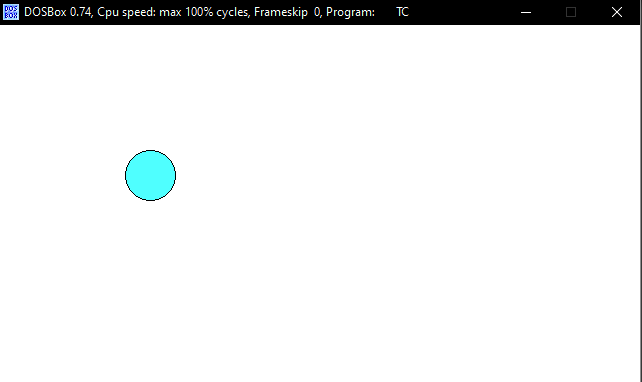
getch();

closegraph();

}

**Output:**





**Code:**

**Boundary Fill Algorithm using 8-Connectors**

#include<iostream.h>

#include<graphics.h>

#include<dos.h>

#include<conio.h>

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

{

if(getpixel(x,y)!=b\_color && getpixel(x,y)!=f\_color)

{

putpixel(x,y,f\_color);

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

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

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

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

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

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

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

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

}

}

void main()

{

clrscr();

int gd = DETECT, gm;

initgraph(&gd,&gm,"C:/TURBOC3/BGI");

int a,b,c,d;

cout<<"Enter the X and Y positions of rectangle: \n";

cin>>a>>b;

cout<<"Enter the breadth and length of Rectangle: \n";

cin>>c>>d;

rectangle(a,b,c,d);

int x = (a+c)/2;

int y = (b+d)/2;

boundaryfill(x,y,4,15);

delay(100);

getch();

closegraph();

}

**Output:**

