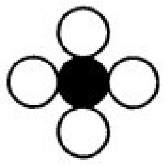
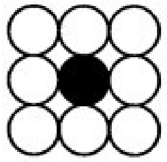
Aim: To implement Area Filling Algorithm: Boundary Fill, Flood Fill.

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 thecoordinate of the interior point (x, y), a fill color, and a boundary color.


(a) Four connected region


(b) Eight connected region

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);
        }
}
```

Program:
FOR 4- CONNECTED BOUNDARY FILL:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void bf(int,int,int,int);
void main()
{
int gd=DETECT,gm;
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
rectangle(50,50,100,100);
bf(70,70,10,15);
getch();
closegraph();
}
void bf (int x,int y,int fcolour, int bcolour)
{
int ccolour=getpixel(x,y);
if((ccolour!=fcolour)&&(ccolour!=bcolour))
{
putpixel(x,y,fcolour);
delay(5);
bf(x,y-1,fcolour,bcolour);
bf(x,y+1,fcolour,bcolour);
bf(x-1,y,fcolour,bcolour);
bf(x+1,y,fcolour,bcolour);
}
}
```

FOR 8- CONNECTED BOUNDARY FILL:

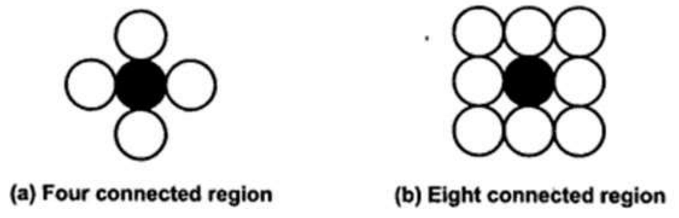
```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void bf(int,int,int,int);
void main()
{
int gd=DETECT,gm;
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
rectangle(50,50,100,100);
bf(70,70,10,15);
getch();
closegraph();
}
void bf (int x,int y,int fcolour, int bcolour)
{
int ccolour=getpixel(x,y);
if((ccolour!=fcolour)&&(ccolour!=bcolour))
{
putpixel(x,y,fcolour);
delay(5);
bf(x,y-1,fcolour,bcolour);
bf(x,y+1,fcolour,bcolour);
bf(x-1,y,fcolour,bcolour);
bf(x+1,y,fcolour,bcolour);
bf(x-1,y-1,fcolour,bcolour);
bf(x,y-1,fcolour,bcolour);
bf(x-1,y+1,fcolour,bcolour);
bf(x+1,y+1,fcolour,bcolour);


}
}
```

Output:

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.


(a) Four connected region (b) Eight connected region

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:

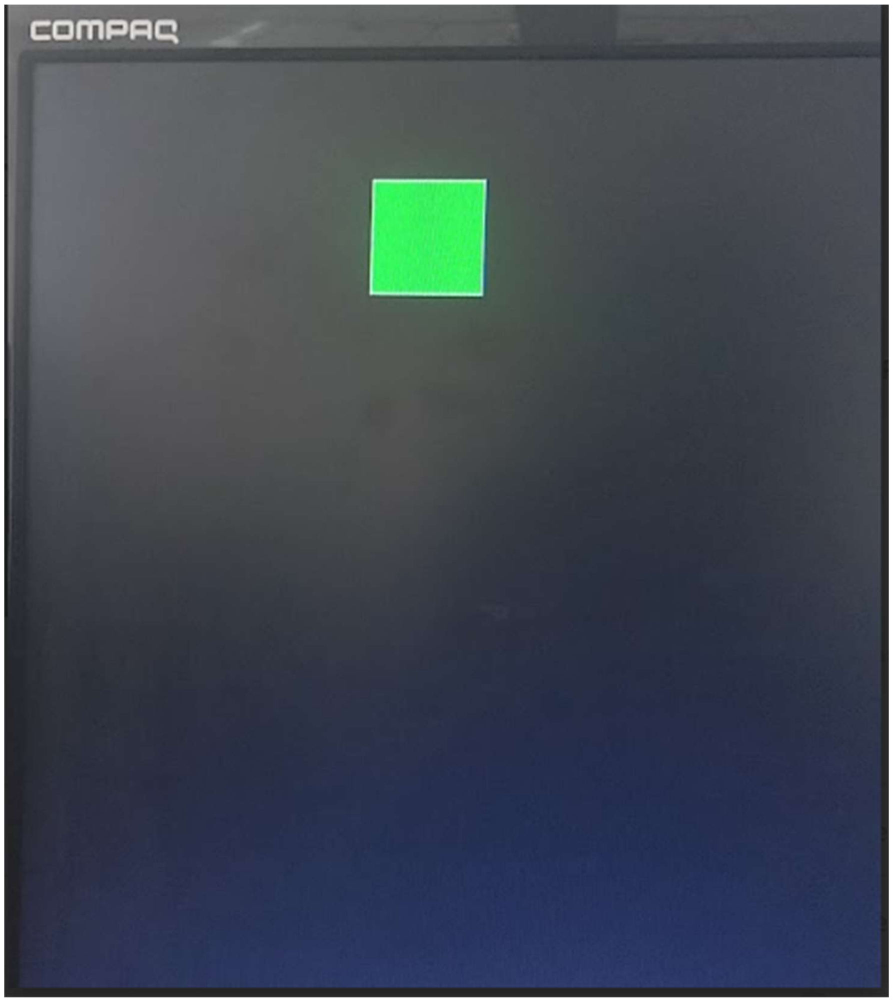
FOR 4- CONNECTED FLOOD FILL:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void ff (int ,int,int,int);
void main()
{
int gd=DETECT,gm;
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
rectangle(50,50,100,100);
ff(70,70,0,15);
getch();
closegraph();
}
void ff ( int x, int y,int ocolour,int ncolour)
{
if(getpixel(x,y)==ocolour)
{
putpixel(x,y,ncolour);
delay (5);
ff (x,y-1,ocolour,ncolour);
ff (x,y+1,ocolour,ncolour);
ff (x+1,y,ocolour,ncolour);
ff (x,y-1,ocolour,ncolour);
}
}
```

FOR 8- CONNECTED FLOOD FILL:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void ff (int ,int,int,int);
void main()
{
int gd=DETECT,gm;
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
rectangle(50,50,100,100);
ff(70,70,0,15);
getch();
closegraph();
}
void ff ( int x, int y,int ocolour,int ncolour)
{
if(getpixel(x,y)==ocolour)
{
putpixel(x,y,ncolour);vc k,m,hgkuyktgf,mytiktfrg
delay (5);
ff (x,y-1,ocolour,ncolour);
ff (x,y+1,ocolour,ncolour);
ff (x+1,y,ocolour,ncolour);
ff (x-1,y,ocolour,ncolour);
ff (x+1,y-1,ocolour,ncolour);
ff (x-1,y+1,ocolour,ncolour);
ff (x+1,y,ocolour,ncolour);
ff (x+1,y+1,ocolour,ncolour);
}
}
```

Output:



Conclusion: Comment on

1. Importance of Flood fill:

   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.
2. Limitation of methods:

   Boundary fill:

   The limitations of boundary fill algorithm is like if the polygon is having boundaries with different colors then boundary fill algorithm fails.

   Flood fill:

   1. Very slow Algorithm.
   2. Maybe fail for large polygons.
   3. Initial pixel require more knowledge about surrounding pixels.

3. Usefulness of method:

   Boundary fill:

   1. This method is good for a polygon with a multicolour area and single-coloured boundary.

   2. In this method, pixels may be visited more than once.

   Flood fill:

   It is an easy way to fill color in the graphics. One just takes the shape and starts flood fill. The algorithm works in a manner so as to give all the pixels inside the boundary the same color leaving the boundary and the pixels outside.