Aim: Implement Area filling Algorithms: Boundary fill, Flood fill.

Theory: * Boundary fill

- Boundary fill algorithm starts with some interior pixel of a polygon, called seed pixel and keeps filling neighbouring pixels in outward directions until the boundary colour is encountered.
- It starts with 3 parameters: interior point (x,y), fill & boundary colour.
- This approach retrieves colour of current pixel and compares it with fill colour & boundary colour.
- If colour of pixel ≠ fill colour && boundary colour then fill it with fill colour and make a recursive call, otherwise skip pixel.
- Neighbour pixel connected using 4-connectivity fails to paint entire region, so 8 is used which is time consuming & memory intensive.
- This method is used in interactive painting packages, where selection of interior pixel can be done easily using a mouse.
- Recursively this algorithm checks all pixels in given polygon & fills them if not already filled.

Algorithm:

// fillColour: Colour to be filled in Polygon.
// boundaryColor: Color of boundary.

Current ← getColor(x,y) // Retrieve color of current pixel.

if Current ≠ fillcolor && Current ≠ boundary Color
putpixel (x,y, fillcolor).

Boundary fill (x+1, y, fillcolor, boundarycolor)
Boundary fill (x-1, y, fillcolor, boundarycolor)
Boundary fill (x, y+1, fillcolor, boundarycolor)
Boundary fill (x, y-1, fillcolor, boundarycolor).

end.

- This algorithm may not work properly if some pixels are already filled, because algo returns when the current pixel is either boundary pixel or it is filled.
- To avoid this, we shall set the colour of all interior ~~point~~ pixels to background color.

Advantages: Simple, easy to implement, works for any type of polygon.

Disadvantages: Requires extensive stacking due to recursion, require more memory, not suitable for large polygons, etc.

* Flood fill.

- Many realistic objects have different colored boundaries. Boundary filling algorithm can't fill the polygon with multi-colored boundary.

- Flood fill can handle such issues.
The algorithm starts with interior ~~point~~ pixel, fill color and old color.

- If color pixel is same as old pixel, it fills it with fill color and examines its neighbours recursively.

- Like boundary fill, if interior pixel is already filled with some other color, flood fill algo fails.
To handle this, ~~wh~~ we shall ~~point~~ ~~f~~ first point all the exterior points with ~~bad~~ background (old) color.

~~Alg~~ Algorithm:

```
Floodfill (x, y, fillcolor, oldcolor)
current ← getcolor (x,y)
if current ≠ old color then
    putpixel (x,y, color)
floodfill ( x+1 , y , fillcolor, old color)
floodfill ( x-1 , y , fill color, old color)
floodfill ( x , y+1 , fillcolor, oldcolor)
floodfill ( x , y-1 , fillcolor, old color)
end
```

Advantages: Simple, easy to implement, Boundary fill algorithm cannot handle the object with multi-color boundary, whereas flood fill can.

Disadvantages:
- Requires extensive stacking
- Slow in nature
- Not suitable for large polygons.

BOUNDARY FILL

PROGRAM:

```
#include <stdio.h>
#include <graphics.h>
void boundaryFill8(int x, int y, int fill_color,int boundary_color)
{
   if(getpixel(x, y) != boundary_color &&
     getpixel(x, y) != fill_color)
   {
      putpixel(x, y, fill_color);
      boundaryFill8(x + 1, y, fill_color, boundary_color);
      boundaryFill8(x, y + 1, fill_color, boundary_color);
      boundaryFill8(x - 1, y, 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);
   }
}
void main()
{
   int gd = DETECT, gm;
   initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

   rectangle(50, 50, 100, 100);
   boundaryFill8(55, 55, 4, 15);
   delay(10000);
   getch();
   closegraph();
 restorecrtmode();
}
```

OUTPUT:

FLOOD FILL

PROGRAM:

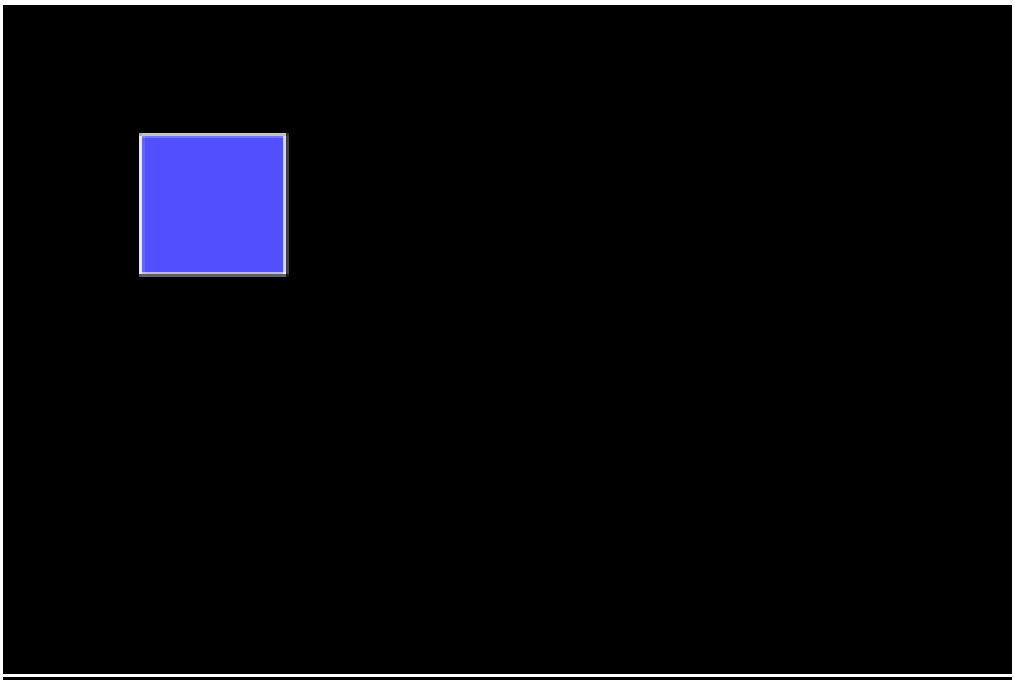
```
#include <graphics.h>
#include <stdio.h>

void flood (int x,int y, int new_col, int old_col)
{
if (getpixel(x,y)==old_col){
putpixel(x,y,new_col);
flood(x+1,y,new_col,old_col);
flood(x-1,y,new_col,old_col);
flood(x,y+1,new_col,old_col);
flood(x,y-1,new_col,old_col);
}}

int main(int top,int bottom,int left,int right, int x, int y, int newcolor, int oldcolor)
{
int gd=DETECT, gm;
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
top=left=50;
```

```
bottom=right=100;
x=51;
y=51;
newcolor=9;
oldcolor=0;
rectangle(top,left,bottom,right);
flood(x,y, newcolor, oldcolor);
getch();
closegraph();
return 0;
}
```

OUTPUT:



Conclusion : We have studied Boundary fill & flood fill algorithms. We learnt in detail about thier algorithms, pros & cons. Thereafter, we have also implemented programs for both of these methods.