PUNE INSTITUTE OF COMPUTER TECHNOLOGY

IT Department

CGL Assignment 4

Name: Suyash More **Class**:SE9

Roll No: 23148 **Batch**: G9

Title:

Implement the following polygon filling methods : i) Flood fill / Seed fill ii) Boundary fill ; using mouse click, keyboard interface and menu driven programming

Problem Statement:

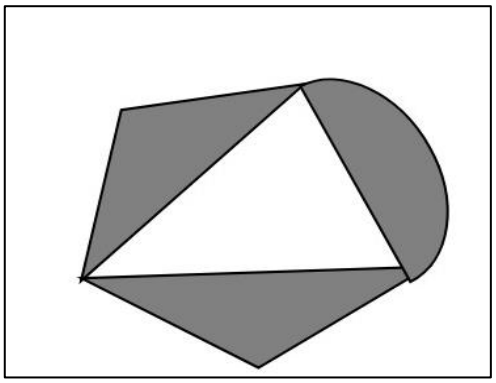
Implement the following polygon filling methods : i) Flood fill / Seed fill ii) Boundary fill ; using mouse click, keyboard interface and menu driven programming

Theory:

1) Flood Fill / Seed Fill Algorithm :-

In this method, a point or seed which is inside region is selected. This point is called a seed point. Then four connected approaches or eight connected approaches is used to fill with specified color.

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.



In fill algorithm, we start from a specified interior point (x, y) and reassign all pixel values are currently set to a given interior color with the desired color. Using either a 4-connected or 8-connected approaches, we then step through pixel positions until all interior points have been repainted.

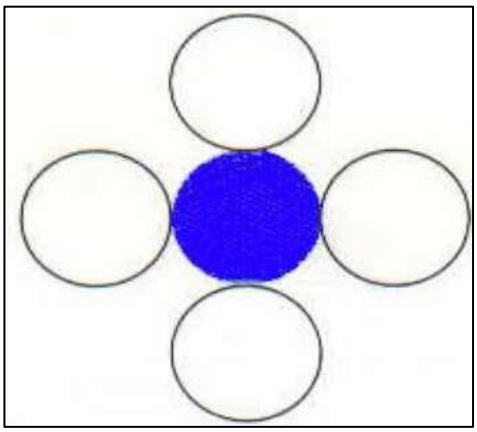
2) Boundary Fill Algorithm :-

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.

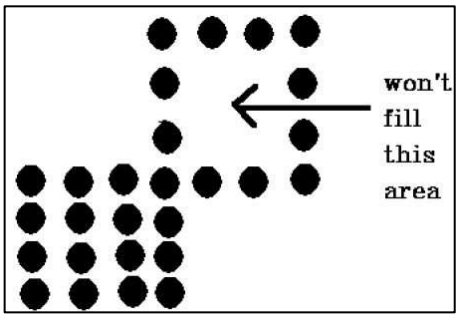
In this algorithm, we assume that color of the boundary is same for the entire object. The boundary fill algorithm can be implemented by 4-connected pixels or 8-connected pixels.

(i) 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.



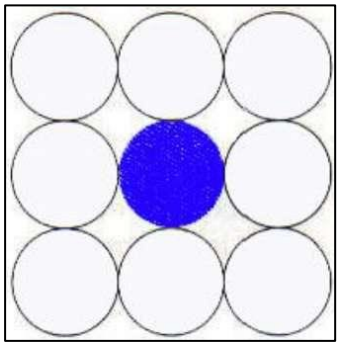
There is a problem with this technique. Consider the case as shown below where we tried to fill the entire region. Here, the image is filled only partially. In such cases, 4-connected pixels technique cannot be used.



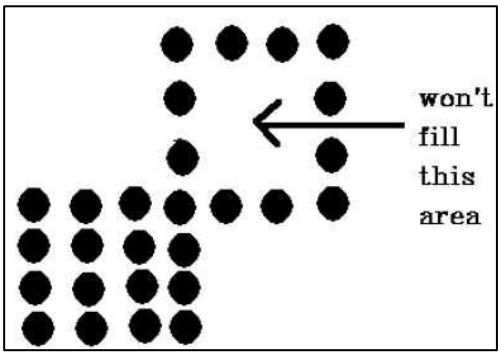
(ii) 8-Connected Polygon :-

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.



The 4-connected pixel technique failed to fill the area as marked in the following figure which won't happen with the 8-connected technique.



Algorithm:

1) Flood Fill / Seed Fill Algorithm :-

1. Procedure floodfill (x, y,fill_ color, old_color: integer)
2. If (getpixel (x, y)=old_color)
3. {
4. setpixel (x, y, fill_color);
5. fill (x+1, y, fill_color, old_color);
6. fill (x-1, y, fill_color, old_color);
7. fill (x, y+1, fill_color, old_color);
8. fill (x, y-1, fill_color, old_color);
9. }
10. }

2) Boundary Fill Algorithm :-

(i) 4-Connected Polygon :-

Step 1 − Initialize the value of seed point seedx, seedyseedx, 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

(ii) 8-Connected Polygon :-

Step 1 − Initialize the value of seed point seedx, seedyseedx, 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:

Advantages Flood Fill :-

- Flood fill colors an entire area in an enclosed figure through interconnected pixels using a single color.
- 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.
- Flood Fill is also sometimes referred to as Seed Fill as you plant a seed and more and more seeds are planted by the algorithm. Each seed takes the responsibility of giving the same color to the pixel at which it is positioned.

Disadvantages of Flood Fill :-

- Very slow algorithm
- May be fail for large polygons
- Initial pixel required more knowledge about surrounding pixels.

Disadvantages of Boundary-Fill over Flood-Fill :-

- In boundary-fill algorithms each pixel must be compared against both the new colour and the boundary colour. In flood-fill algorithms each pixel need only be compared against the new colour. Therefore flood-fill algorithms are slightly faster.
- Boundary-fill algorithms can leak. There can be no leakage in flood-fill algorithms.

Advantages of Boundary-Fill over Flood-Fill :-

- Flood-fill regions are defined by the whole of the region. All pixels in the region must be made the same colour when the region is being created. The region cannot be translated, scaled or rotated.
- 4-connected boundary-fill regions can be defined by lines and arcs. By translating the line and arc endpoints we can translate, scale and rotate the whole boundary-fill region. Therefore 4-connected boundary-fill regions are better suited to modelling.

Code :-

```
#include<stdio.h>
#include<GL/gl.h>
#include<GL/glu.h>
#include<GL/glut.h>
#include<math.h>

/*draw chess pattern rotate it and fill it with different colours*/

typedef struct pixel
{
	GLubyte r,g,b;
}pixel;

pixel f_color,b_color;

float mat1[20][3];
float ans1[20][3];
float trans1[3][3];
int ch=1;

void initial_co()
{
	int i,y,x;

	y=90;
	//horizontal lines
	for(i=0;i<10;i+=2)
	{
		//first point
		mat1[i][0]=90;
		mat1[i][1]=y;
		mat1[i][2]=1;

		//second point
		mat1[i+1][0]=210;
		mat1[i+1][1]=y;
		mat1[i+1][2]=1;

		y+=30;
	}

	x=90;
```

```
	//vertical lines
	for(i;i<20;i+=2)
	{
		//first point
		mat1[i][0]=x;
		mat1[i][1]=90;
		mat1[i][2]=1;

		//second point
		mat1[i+1][0]=x;
		mat1[i+1][1]=210;
		mat1[i+1][2]=1;

		x+=30;
	}
}

void rotate_fig()
{
	int i,j,k;
	float theta;
	theta=45*3.14/180;

	/*----------------translation to origin ---------------------------------*/
	for(i=0;i<3;i++)
	{
		for(j=0;j<3;j++)
		{
			if(i==j)
				trans1[i][j]=1;
			else
				trans1[i][j]=0;
		}
	}
	trans1[2][0]=trans1[2][1]=-150;
	/*
		trans1= 1  0  0
			0  1  0
			tx ty 1
	*/

	for(i=0;i<20;i++)
	{
		for(j=0;j<3;j++)
```

```
	{
		ans1[i][j]=0;
		for(k=0;k<3;k++)
			ans1[i][j]+=mat1[i][k]*trans1[k][j];
	}
}

/*----------------------rotation at origin-------------------------------*/
for(i=0;i<3;i++)
{
	for(j=0;j<3;j++)
	{
		if(i==j)
			trans1[i][j]=1;
		else
			trans1[i][j]=0;
	}
}

trans1[0][0]=trans1[1][1]=cos(theta);
trans1[0][1]=sin(theta);
trans1[1][0]=-sin(theta);
/*
	trans1=  cos sin 0
		-sin cos 0
		 0    0  1
*/

for(i=0;i<20;i++)
{
	for(j=0;j<3;j++)
	{
		mat1[i][j]=0;
		for(k=0;k<3;k++)
			mat1[i][j]+=ans1[i][k]*trans1[k][j];
	}
}

/*----------------------translation back----------------------------*/
for(i=0;i<3;i++)
{
	for(j=0;j<3;j++)
	{
		if(i==j)
```

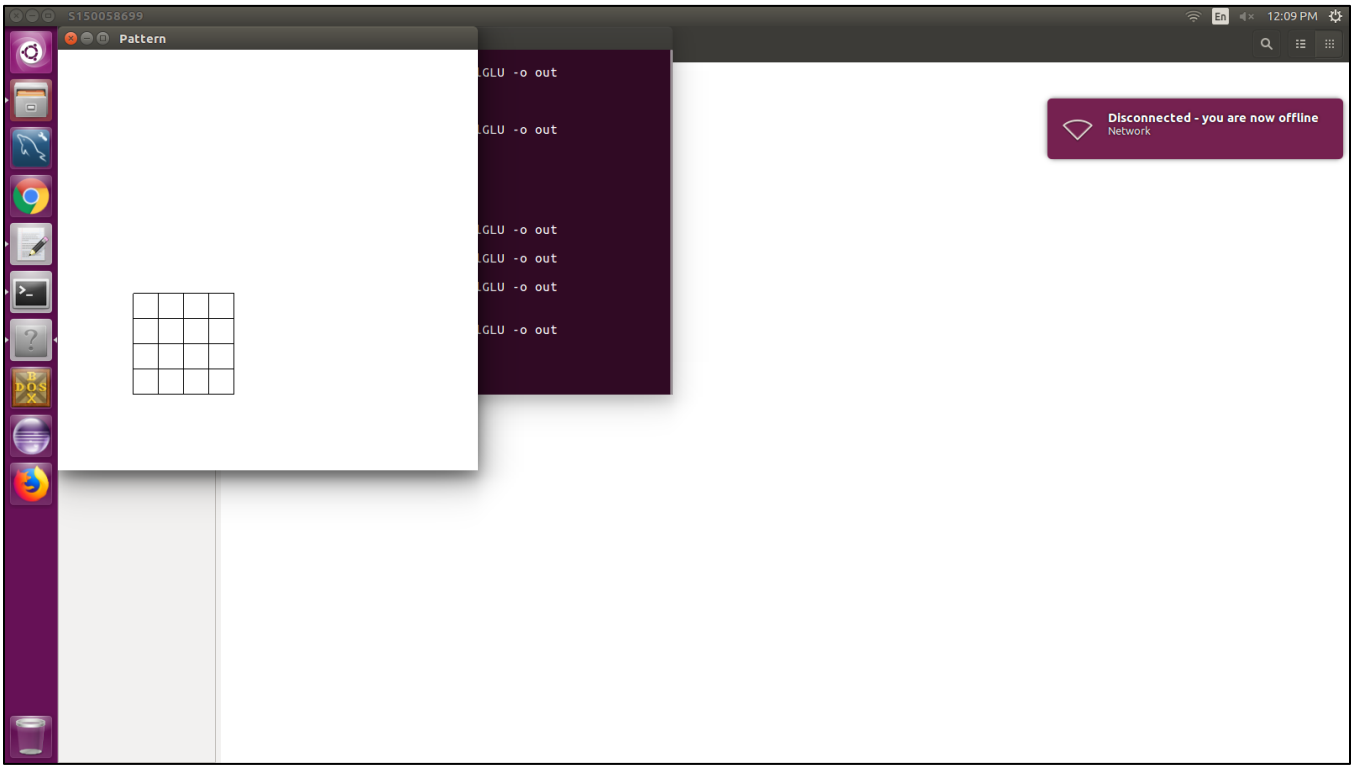
```
                        trans1[i][j]=1;
                else
                        trans1[i][j]=0;
            }
        }
        trans1[2][0]=trans1[2][1]=150;

        for(i=0;i<20;i++)
        {
            for(j=0;j<3;j++)
            {
                ans1[i][j]=0;
                for(k=0;k<3;k++)
                    ans1[i][j]+=mat1[i][k]*trans1[k][j];
            }
        }

}

void boundary_fill(int x,int y)
{
        pixel c;

        glReadPixels(x,y,1,1,GL_RGB,GL_UNSIGNED_BYTE,&c);//values are put into c

        //if color not equal to backgroung color and filling color put color
        if((c.r!=b_color.r || c.g!=b_color.g || c.b!=b_color.b )&&(c.r!=f_color.r || c.g!=f_color.g || c.b!=f_color.b
))
        {
            glColor3ub(f_color.r,f_color.g,f_color.b);//set fill color for pixel
            glBegin(GL_POINTS);
                glVertex2d(x,y);//put pixel
            glEnd();
            glFlush();
            boundary_fill(x+1,y);//right pixel
            boundary_fill(x-1,y);//left pixel
            boundary_fill(x,y+1);//upper pixel
            boundary_fill(x,y-1);//lower pixel
        }
}

void before()
{
        int i;
```

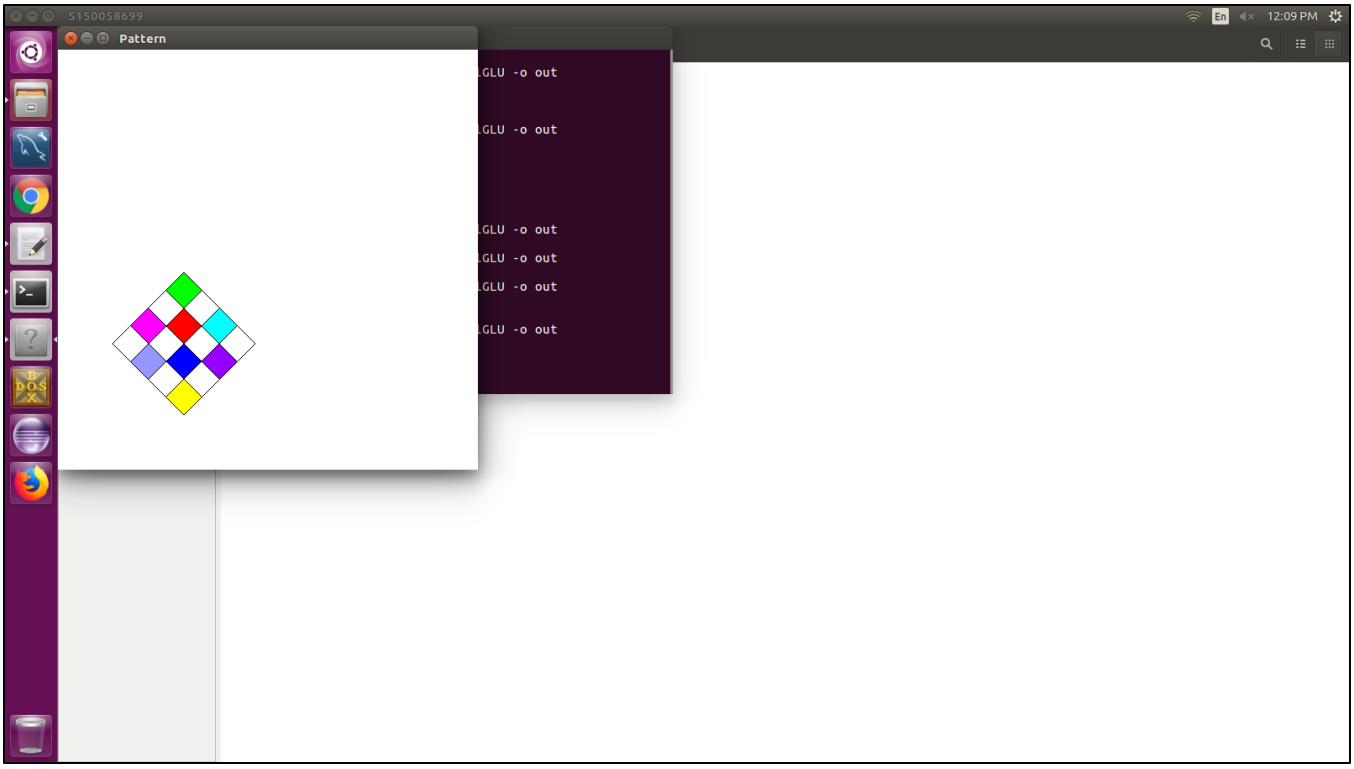
```
	initial_co();
	glBegin(GL_LINES);//draws the new figure
		for(i=0;i<20;i+=2)
		{
			glVertex2f(mat1[i][0],mat1[i][1]);
			glVertex2f(mat1[i+1][0],mat1[i+1][1]);
		}
	glEnd();
	glFlush();
}

void figure()
{
	glClear(GL_COLOR_BUFFER_BIT);
	int i;
	float factor=30*cos(45*3.14/180);

	rotate_fig();//rotates the figure about the middle point (150,150)

	glBegin(GL_LINES);//draws the new figure
		for(i=0;i<20;i+=2)
		{
			glVertex2f(ans1[i][0],ans1[i][1]);
			glVertex2f(ans1[i+1][0],ans1[i+1][1]);
		}
	glEnd();
	glFlush();

	//filling the boxes with colours
	//red
	boundary_fill(150,150+factor);

	//green
	f_color.r=0;
	f_color.g=255;
	f_color.b=0;
	boundary_fill(150,150+3*factor);

	//blue
	f_color.r=0;
	f_color.g=0;
	f_color.b=255;
	boundary_fill(150,150-factor);
```

```
	//yellow
	f_color.r=255;
	f_color.g=255;
	f_color.b=0;
	boundary_fill(150,150-3*factor);

	//light blue
	f_color.r=0;
	f_color.g=255;
	f_color.b=255;
	boundary_fill(150+2*factor,150+factor);

	//pink
	f_color.r=255;
	f_color.g=0;
	f_color.b=255;
	boundary_fill(150-2*factor,150+factor);

	//purple
	f_color.r=150;
	f_color.g=0;
	f_color.b=255;
	boundary_fill(150+2*factor,150-factor);

	//light violet
	f_color.r=150;
	f_color.g=150;
	f_color.b=255;
	boundary_fill(150-2*factor,150-factor);
}

void mouse_click(int btn,int state,int x,int y)
{
	//left click shows and changes the figure
	if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
	{
		switch(ch)
		{
		case 1:
			before();//initial figure
			ch=2;
			break;
		case 2:
			figure();//after transformation
```

```
                        ch=3;
                        break;
                case 3:
                        break;
                }
        }
}

void init_func()//empty function doesnt do anything
{
        glFlush();
}

void Init()
{
        glClearColor(1.0,1.0,1.0,0.0);//sets the background colour
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(0.0,0.0,0.0);//sets the drawing colour
        gluOrtho2D(0,500,0,500);//sets the co ordinates
}

int main(int argc,char **argv)
{

        //border color
        b_color.r=b_color.g=b_color.b=0;

        //fill color
        f_color.r=255;
        f_color.g=0;
        f_color.b=0;

        glutInit(&argc,argv);//initializing the library
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);//setting the display mode
        glutInitWindowPosition(0,0);//position of the window
        glutInitWindowSize(500,500);//size of the window
        glutCreateWindow("Pattern");//name of the window
        Init();//initializes the background colour and co ordinates
        glutDisplayFunc(init_func);//displays the function
        glutMouseFunc(mouse_click);//to display before and after figures
        glutMainLoop();//keeps the program open until closed
        return 0;
}
```

Output :-


S150058699
12:09 PM
Pattern
GLU -o out
GLU -o out
GLU -o out
GLU -o out
GLU -o out
GLU -o out
Disconnected - you are now offline
Network


S150058699
12:09 PM
Pattern
LGLU -o out
LGLU -o out
LGLU -o out
LGLU -o out
LGLU -o out
LGLU -o out