Experiment No 4

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

Polygon Filling Approaches:

1. Scan Line fill approaches

2. Seed fill approaches

There are two types of seed fill algorithms.

1. Boundary Fill Algorithm

2. Flood Fill Algorithm.

1 Boundary Fill

Boundary Fill is algorithm used for the purpose of coloring figures in computer graphics. Boundary fill fills

chosen area with a color until the given colored boundary is found.

Algorithm :

Step 1 : The boundary fill procedure accepts the input as coordinates of an interior point (x, y), a fill color, and

a boundary color.

Step 2 : Starting from (x, y)which is seed pixel, the procedure tests the neighboring positions to determine

whether they are boundary color.

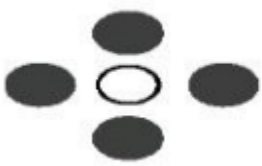
Step 3 : If not, they are painted with the fill color, and the neighbors are tested.

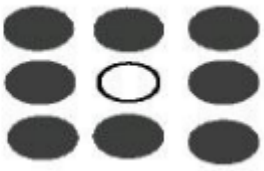
Step 4 : This process continues until all pixels up to the boundary color for the area have been tested.

There are two methods for filling the pixel and find the neighbor pixel :

(i) 4-connected.

(ii) 8-connected.





(i) 4-Connected Method :

Four_Fill (x, y, fill_col, bound_color)

if (curr_pixel_color != bound_color) and (curr_pixel_color != fill_col) then

set_pixel(x, y, fill_col)

Four_Fill (x+1, y, fill_col, bound_col);

Four_Fill (x-1, y, fill_col, bound_col);

Four_Fill (x, y+1, fill_col, bound_col);

Four_Fill(x, y-1, fill_col, bound_col);

end;

(ii) 8-Connected Method

The fill operation can proceed above, below, right and left side as well as through diagonal pixels of the

current

pixels. This process will continue until we find a boundary with different color.

2 Flood Fill

The purpose of Flood Fill is to color an entire area of connected pixels with the same color.

Similarly boundary fill algorithm, we start with seed pixel, seed pixel is examined for specified interior color

instead of boundary color

Pseudo Code for Flood fill Algorithm

Flood-fill (node, old-color, replacement-color):

If the color of node is not equal to old-color, return.

Set the color of node to replacement-color.

Perform Flood-fill (one step to the left of node, old-color, replacement-color).

Perform Flood-fill (one step to the right of node, old-color, replacement-color).

Perform Flood-fill (one step to the top of node, old-color, replacement-color).

Perform Flood-fill (one step to the bottom of node, old-color, replacement-color)

Code And Output

```
#include <iostream>

#include <math.h>

#include <GL/glut.h>

using namespace std;

float R=0,G=0,B=0;

int Algo;
```

```
void init(){
 glClearColor(1.0,1.0,1.0,0.0);
 glMatrixMode(GL_PROJECTION);
 gluOrtho2D(0,640,0,480);
}
void floodFill(int x, int y, float *newCol, float *oldcol){
 float pixel[3];
 glReadPixels(x,y,1,1,GL_RGB,GL_FLOAT,pixel);

 if(oldcol[0]==pixel[0] && oldcol[1]==pixel[1] && oldcol[2]==pixel[2]){

 glBegin(GL_POINTS);
 glColor3f(newCol[0],newCol[1],newCol[2]);
 glVertex2i(x,y);
 glEnd();
 glFlush();

 floodFill(x,y+1,newCol,oldcol);
 floodFill(x+1,y,newCol,oldcol);
 floodFill(x,y-1,newCol,oldcol);
 floodFill(x-1,y,newCol,oldcol);
 }
}
void boundaryFill(int x, int y, float* fillColor, float* bc){
 float color[3];
 glReadPixels(x,y,1.0,1.0,GL_RGB,GL_FLOAT,color);

 if((color[0]!=bc[0] || color[1]!=bc[1] || color[2]!=bc[2]) && (fillColor[0]!=color[0] ||
fillColor[1]!=color[1]
|| fillColor[2]!=color[2])){
```

```
 glColor3f(fillColor[0],fillColor[1],fillColor[2]);
 glBegin(GL_POINTS);
 glVertex2i(x,y);
 glEnd();
 glFlush();
 boundaryFill(x+1,y,fillColor,bc);
 boundaryFill(x-1,y,fillColor,bc);
 boundaryFill(x,y+1,fillColor,bc);
 boundaryFill(x,y-1,fillColor,bc);



 }


 return;
}
void mouse(int btn, int state, int x, int y){
 y = 480-y;
 if(btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN){

 float bcol[] = {1,0,0};
 float oldcol[] = {1,1,1};
 float newCol[] = {R,G,B};

 if(Algo==1){
 boundaryFill(x,y,newCol,bcol);
 }
 if(Algo==2){
 floodFill(x,y,newCol,oldcol);
 }
 }
}
```

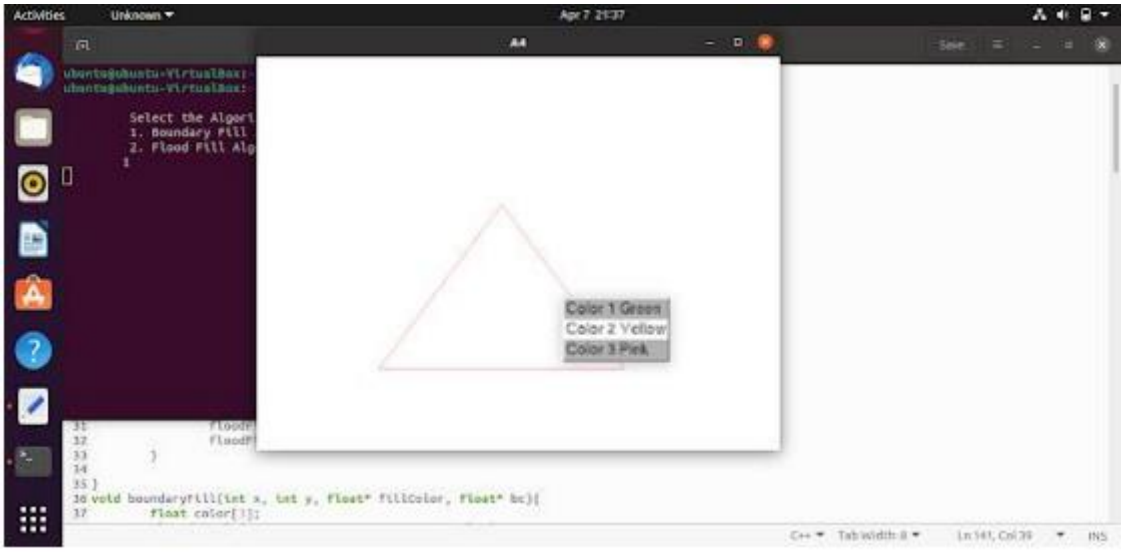
```
void B_Draw(){
 glClear(GL_COLOR_BUFFER_BIT);
 glColor3f(1,0,0);
 glBegin(GL_LINE_LOOP);
 glVertex2i(150,100);
 glVertex2i(300,300);
 glVertex2i(450,100);
 glEnd();
 glFlush();

}
void F_Draw(){
 glClear(GL_COLOR_BUFFER_BIT);

 glBegin(GL_LINES);
 glColor3f(1,0,0);glVertex2i(150,100);glVertex2i(300,300);
 glEnd();
 glBegin(GL_LINE_LOOP);
 glColor3f(0,0,1);glVertex2i(300,300);glVertex2i(450,100);
 glEnd();
 glBegin(GL_LINE_LOOP);
 glColor3f(0,0,0);glVertex2i(450,100);glVertex2i(150,100);
 glEnd();
 glFlush();

}
void goMenu(int value){
 switch(value){

 case 1:
 R = 0, G = 1, B=0;
```

```
 break;
 case 2:
 R = 1, G = 1, B=0;
 break;
 case 3:
 R = 1, G = 0, B=1;
 break;

 }
 glutPostRedisplay();
}
int main(int argc, char** argv){

 cout<<"\n \t Select the Algorithm ";
 cout<<"\n \t 1. Boundary Fill Algorithm ";
 cout<<"\n \t 2. Flood Fill Algorithm \n \t";
 cin>>Algo;

 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
 glutInitWindowSize(640,480);
 glutInitWindowPosition(200,200);
 glutCreateWindow("A4");
 init();
 glutCreateMenu(goMenu);
 glutAddMenuEntry("Color 1 Green",1);
 glutAddMenuEntry("Color 2 Yellow",2);
 glutAddMenuEntry("Color 3 Pink",3);
 glutAttachMenu(GLUT_RIGHT_BUTTON);

 if(Algo==1){
```

```
 glutDisplayFunc(B_Draw);
 }
 if(Algo==2){
 glutDisplayFunc(F_Draw);
 }
 glutMouseFunc(mouse);
 glutMainLoop();
 return 0;
}
```

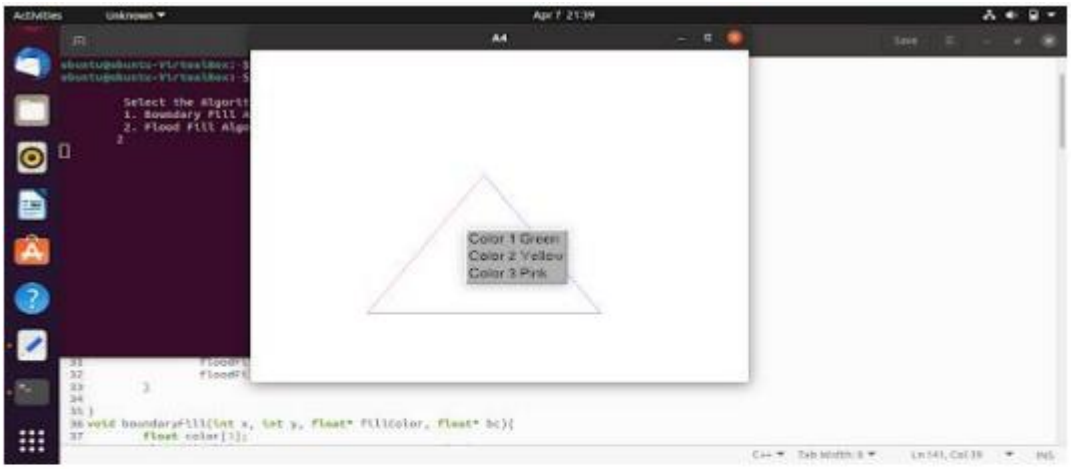
Output

```
g++ filename.cpp -lGL -lGLU -lglut
./a.out
```




Activities
Unknown
Apr 7 21:38
A4
Select the Algori
1. Boundary Fill
2. Flood Fill Alg
1
void BoundaryFill(int x, int y, float* fillColor, float* bc){
float color[3];


Activities
Unknown
Apr 7 21:39
A4
Select the Algori
1. Boundary Fill A
2. Flood Fill Alg
2
Color 1 Green
Color 2 Yellow
Color 3 Pink
void BoundaryFill(int x, int y, float* fillColor, float* bc){
float color[3];


Activities
Unknown
Apr 7 21:39
A4
Color 1 Green
Color 2 Yellow
Color 3 Pink


Activities
Unknown
Apr 7 21:39
A4
Color 1 Green
Color 2 Yellow
Color 3 Pink


Activities
Unknown
Apr 7 21:39
A4