CS352 Assignment-5

Krishanu Saini
190001029

Write a program using glut library for polygon filling by implementing the following algorithms:

1. Boundary fill 4, 8

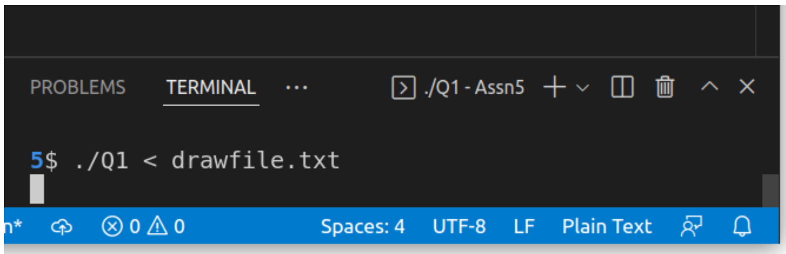
2. Scanline fill

Problem 1

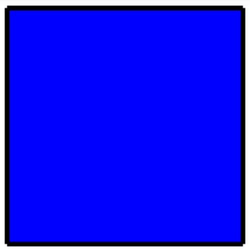
Code

```
#include <iostream>
#include <GL/glut.h>
using namespace std;
/*
 * Name: Krishanu Saini
 * Roll: 190001029
 * Ques: Problem 1a) Boundary Fill 4
 * Date: 10/02/22
 */

float colorFill[3] = {0, 0, 1}; // fill blue color
float boundFill[3] = {0, 0, 0}; // boundary is black

void DrawBoundary() {
    glLineWidth(4);
    glPointSize(4);
    glBegin(GL_LINE_LOOP);

    /*-------- Square --------*/
    glVertex2i(100, 100);
    glVertex2i(300, 100);
    glVertex2i(300, 300);
    glVertex2i(100, 300);

    /*-------- Triangle --------*/
    // glVertex2i(100, 100);
    // glVertex2i(300, 200);
    // glVertex2i(300, 600);

    glEnd();
}

void BoundaryFill(int x, int y, bool visited[1000][1000]) {
    if(x < 0 || x > 800 || y < 0 || y > 800) return; // out of bounds
    if(visited[x][y] == true) return;
    // else
    float pixel[3];
    glReadPixels(x,y,1.0,1.0,GL_RGB,GL_FLOAT,pixel);
    if(pixel[0] == boundFill[0] && pixel[1] == boundFill[1] &&  pixel[2] == boundFill[2]) {
        // boundary number
        return;
    }
    // else
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    visited[x][y] = true;
    glEnd();
    glFlush();
    BoundaryFill(x-3, y, visited);
    BoundaryFill(x+3, y, visited);
    BoundaryFill(x, y-3, visited);
    BoundaryFill(x, y+3, visited);
}

void Draw() {
    glClear(GL_COLOR_BUFFER_BIT);

    /*-------- boundary --------*/
    glPointSize(4);
    glColor3f(boundFill[0], boundFill[1], boundFill[2]);
    DrawBoundary();

    /*-------- b-fill --------*/
    bool visited[1000][1000];
    for(int i=0;i<1000;i++) {
        for(int j=0;j<1000;j++){
            visited[i][j] = false;
        }
    }
    glPointSize(4);
    glColor3f(colorFill[0], colorFill[1], colorFill[2]);
    BoundaryFill(200, 200, visited);


    glFlush();
}

int main(int argc, char *argv[]) {
    glutInit(&argc, argv);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(800, 800);
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutCreateWindow("");
    gluOrtho2D(0, 800, 0, 800);
    glClearColor(1,1,1,0.0);
    glutDisplayFunc(Draw);
    glutMainLoop();
    return 0;
}
```

Output



```
5$ ./Q1 < drawfile.txt
```

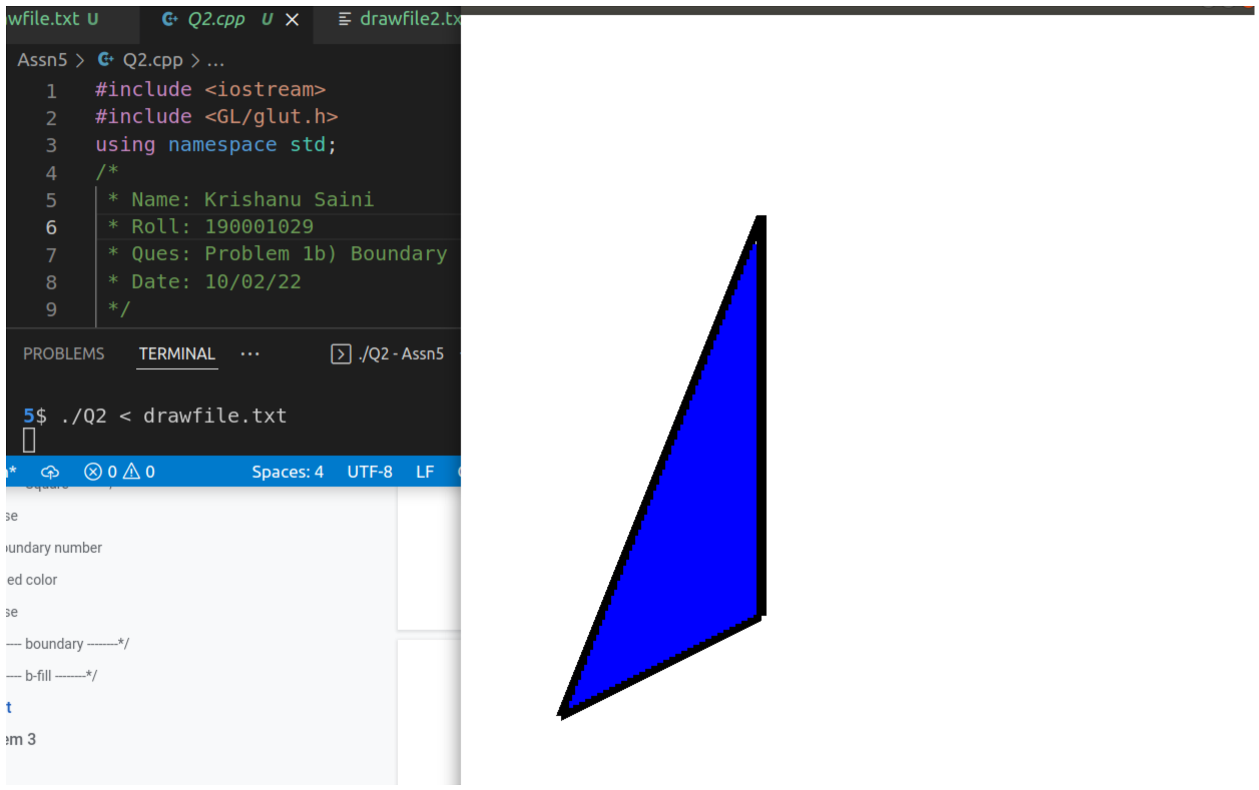


Problem 2

Code

```
#include <iostream>
#include <GL/glut.h>
using namespace std;
/*
* Name: Krishanu Saini
* Roll: 190001029
* Ques: Problem 1b) Boundary Fill 8
* Date: 10/02/22
*/

float colorFill[3] = {0, 0, 1}; // fill blue color
float boundFill[3] = {0, 0, 0}; // boundary is black

void DrawBoundary() {
   glLineWidth(10);
   glPointSize(4);
   glBegin(GL_LINE_LOOP);

   /*-------- Square --------*/
   glVertex2i(100, 100);
   glVertex2i(300, 100);
   glVertex2i(300, 300);
   glVertex2i(100, 300);

   /*-------- Triangle --------*/
   // glVertex2i(100, 100);
   // glVertex2i(300, 200);
   // glVertex2i(300, 600);

   glEnd();
}

void BoundaryFill(int x, int y, bool visited[1000][1000]) {
   if(x < 0 || x > 800 || y < 0 || y > 800) return; // out of bounds
   if(visited[x][y] == true) return;
   // else
   float pixel[3];
   glReadPixels(x,y,1.0,1.0,GL_RGB,GL_FLOAT,pixel);
   if(pixel[0] == boundFill[0] && pixel[1] == boundFill[1] &&  pixel[2] ==
boundFill[2]) {
       // boundary number
       return;
   }
   if(pixel[0] == colorFill[0] && pixel[1] == colorFill[1] &&  pixel[2] ==
colorFill[2]) {
       // filled color
       return;
   }
   // else
   glBegin(GL_POINTS);
   glVertex2i(x, y);
   visited[x][y] = true;
   glEnd();
   glFlush();
   BoundaryFill(x-3, y-3, visited);
   BoundaryFill(x, y-3, visited);
   BoundaryFill(x+3, y-3, visited);
   BoundaryFill(x-3, y, visited);
   BoundaryFill(x+3, y, visited);
   BoundaryFill(x-3, y+3, visited);
   BoundaryFill(x, y+3, visited);
   BoundaryFill(x+3, y+3, visited);
}

void Draw() {
   glClear(GL_COLOR_BUFFER_BIT);

   /*-------- boundary --------*/
   glPointSize(4);
   glColor3f(boundFill[0], boundFill[1], boundFill[2]);
   DrawBoundary();

   /*-------- b-fill --------*/
   bool visited[1000][1000];
   for(int i=0;i<1000;i++) {
       for(int j=0;j<1000;j++){
           visited[i][j] = false;
       }
   }
   glPointSize(4);
   glColor3f(colorFill[0], colorFill[1], colorFill[2]);
   BoundaryFill(200, 200, visited);


   glFlush();
}

int main(int argc, char *argv[]) {
   glutInit(&argc, argv);
   glutInitWindowPosition(100, 100);
   glutInitWindowSize(800, 800);
   glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);

   glutCreateWindow("");
   gluOrtho2D(0, 800, 0, 800);
   glClearColor(1,1,1,0.0);
   glutDisplayFunc(Draw);
   glutMainLoop();
   return 0;
}
```

Output



wfile.txt U   Q2.cpp U ×   drawfile2.tx

Assn5 > Q2.cpp > ...

```cpp
#include <iostream>
#include <GL/glut.h>
using namespace std;
/*
 * Name: Krishanu Saini
 * Roll: 190001029
 * Ques: Problem 1b) Boundary
 * Date: 10/02/22
 */
```

PROBLEMS   TERMINAL   ···   ./Q2 - Assn5

```
5$ ./Q2 < drawfile.txt
```

⊗ 0 ⚠ 0   Spaces: 4   UTF-8   LF

Problem 3

Code

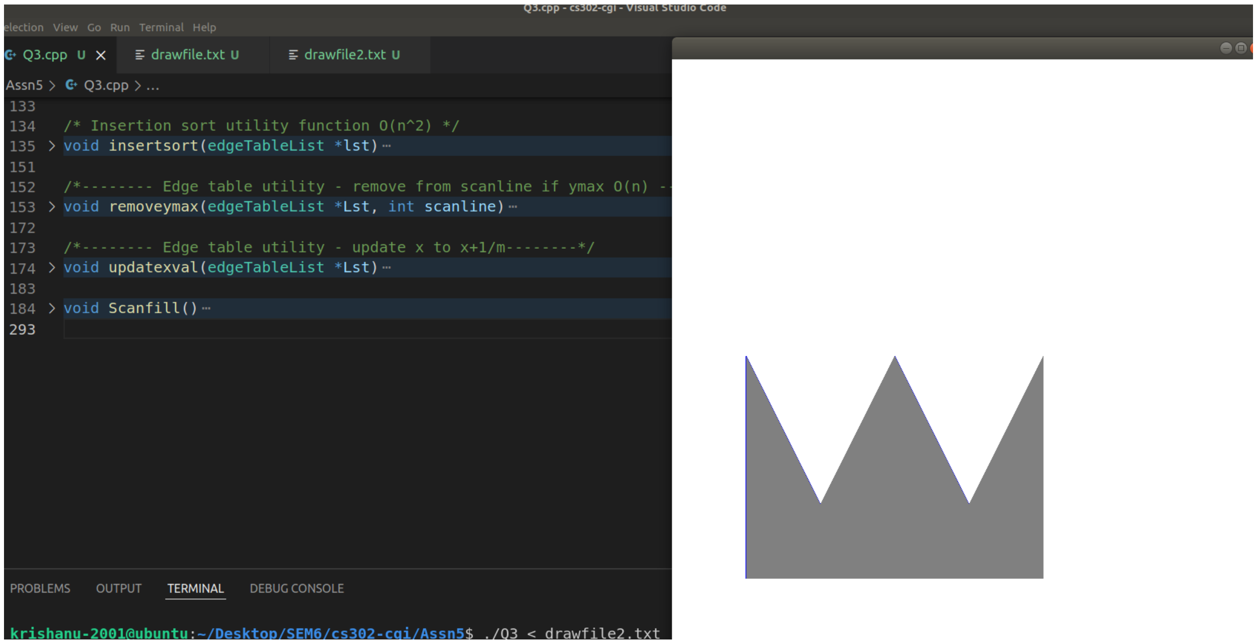
```cpp
#include <bits/stdc++.h>
#include <iostream>
#include <math.h>
#include <GL/glut.h>
using namespace std;
/*
* Name: Krishanu Saini
* Roll: 190001029
* Ques: Problem 2) Scanline Fill Algorithm
* Date: 10/02/22
*/
int inputSize;
int inputX[1000];
int inputY[1000];

// Start from lower left corner
struct edgeTuple
{
   // simplified version of [ymax, xatymin, delx, dely]
   int ymax;
   float xofymin;
   float slopeinverse;
};

struct edgeTableList
{
   int countEdge; // no. of items
   edgeTuple buckets[12345];
};

edgeTableList edgetable[800], activeEdgeList;

void insertsort(edgeTableList *);
void setupEdges(int, int, int, int);
void saveEdgeTuple(edgeTableList *, int, int, float);
void removeymax(edgeTableList *, int);
void updatexval(edgeTableList *);
void Scanfill();

void DrawBoundary();
void Draw();

void Draw()
{
   glClear(GL_COLOR_BUFFER_BIT);
   // initialize table
   activeEdgeList.countEdge = 0;
   for (int i = 0; i < 800; i++)
```

```
    {
        edgetable[i].countEdge = 0;
    }

    DrawBoundary();
    // polygon filling using scanline
    Scanfill();
    glFlush();
}

int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(800, 800);
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    cin >> inputSize;
    glutCreateWindow("");
    gluOrtho2D(0, 800, 0, 800);
    glClearColor(1, 1, 1, 0.0);
    glutDisplayFunc(Draw);
    glutMainLoop();
    return 0;
}

void DrawBoundary()
{
    glColor3f(0, 0, 1);
    glBegin(GL_LINE_LOOP);
    int n = inputSize;
    for (int i = 0; i < n; i++)
    {
        cin >> inputX[i] >> inputY[i];
        glVertex2i(inputX[i], inputY[i]);
    }
    for (int i = 1; i < n; i++)
    {
        int x1 = inputX[i - 1];
        int y1 = inputY[i - 1];
        int x2 = inputX[i];
        int y2 = inputY[i];
        setupEdges(x1, y1, x2, y2);
    }
    glFlush();
    glEnd();
}

/*-------- Edge table utility ------*/
void saveEdgeTuple(edgeTableList *eb, int ymax, int xatymin, float
slopeinverse)
{
    int n = eb->countEdge;
    eb->buckets[n] = {ymax, (float)xatymin, slopeinverse};
    /* sort it */
    insertsort(eb);
    eb->countEdge += 1;
}

/*-------- Create Edge Table --------*/
void setupEdges(int x1, int y1, int x2, int y2)
{
    int ymax, xatymin, y_scan;
    float slopeinverse;
    if (y2 == y1)
    {
        return;
    }
    slopeinverse = 0;
    if (x2 != x1)
    {
        slopeinverse = ((float)(x2 - x1)) / ((float)(y2 - y1));
    }

    y_scan = y1;
    ymax = y2;
    xatymin = x1;

    if (y1 > y2)
    {
        y_scan = y2;
```

```c
        ymax = y1;
        xatymin = x2;
    }
    saveEdgeTuple(&edgetable[y_scan], ymax, xatymin, slopeinverse);
}

/* Insertion sort utility function O(n^2) */
void insertsort(edgeTableList *lst)
{
    int n = lst->countEdge;
    edgeTuple temp;
    for (int i = 1; i < n; i++)
    {
        temp = lst->buckets[i];
        int j = i - 1;
        while (j >= 0 && lst->buckets[j].xofymin > temp.xofymin)
        {
            lst->buckets[j + 1] = lst->buckets[j];
            j--;
        }
        lst->buckets[j + 1] = temp;
    }
}

/*-------- Edge table utility - remove from scanline if ymax O(n)
--------*/
void removeymax(edgeTableList *Lst, int scanline)
{
    int n = Lst->countEdge;
    for (int i = 0; i < n; i++)
    {
        if (Lst->buckets[i].ymax == scanline)
        {
            int j = i;
            // shift left after deletion
            while (j < n - 1)
            {
                Lst->buckets[j] = Lst->buckets[j + 1];
                j++;
            }
            Lst->countEdge--;
            i--;
        }
    }
}

/*-------- Edge table utility - update x to x+1/m--------*/
void updatexval(edgeTableList *Lst)
{
    int n = Lst->countEdge;
    for (int i = 0; i < n; i++)
    {
        float minv = Lst->buckets[i].slopeinverse;
        Lst->buckets[i].xofymin = Lst->buckets[i].xofymin + minv;
    }
}

void Scanfill()
{
    /*
     * 1. Vectices at local extremum are counted twice
     * 2. otherwise once in pair
     */

    for (int i = 0; i < 800; i++)
    {
        // insert scanline into active edge list
        for (int j = 0; j < edgetable[i].countEdge; j++)
        {
            saveEdgeTuple(&activeEdgeList,
                          edgetable[i].buckets[j].ymax,
                          edgetable[i].buckets[j].xofymin,
                          edgetable[i].buckets[j].slopeinverse);
        }

        // Remove edges where ymax is reached
        removeymax(&activeEdgeList, i);

        // sort Active edge list
        insertsort(&activeEdgeList); // O(n^2)
        // pairing x coords
        int j = 0;
        int FillFlag = 0;
        int coordCount = 0;
        int x1 = 0, x2 = 0;
        int ymax1 = 0, ymax2 = 0;
        int n = activeEdgeList.countEdge;
        while (j < n)
        {
            if (coordCount % 2 == 0)
            {
                x1 = activeEdgeList.buckets[j].xofymin;
                ymax1 = activeEdgeList.buckets[j].ymax;
                if (x1 == x2)
                {
                    /* three cases can arrive-
                        1. lines are towards top of the intersection
                        2. lines are towards bottom
                        3. one line is towards top and other is towards
bottom
                    */
                    if (((x1 == ymax1) && (x2 != ymax2)) || ((x1 != ymax1)
&& (x2 == ymax2)))
                    {
                        x2 = x1;
                        ymax2 = ymax1;
                    }
                    else
                    {
                        coordCount++;
                    }
                }
                else
                {
                    coordCount++;
                }
            }
            else
            {
                x2 = activeEdgeList.buckets[j].xofymin;
                ymax2 = activeEdgeList.buckets[j].ymax;
                FillFlag = 0;
                if (x1 == x2)
                {
                    /*three cases can arrive-
                        1. lines are towards top of the intersection
                        2. lines are towards bottom
                        3. one line is towards top and other is towards
bottom
                    */
                    if (((x1 == ymax1) && (x2 != ymax2)) || ((x1 != ymax1)
&& (x2 == ymax2)))
                    {
                        x1 = x2;
                        ymax1 = ymax2;
                    }
                    else
                    {
                        coordCount++;
                        FillFlag = 1;
                    }
                }
                else
                {
                    coordCount++;
                    FillFlag = 1;
                }

                if (FillFlag)
                {
                    // drawing actual lines...
                    glColor3f(0.5, 0.5, 0.5);

                    glBegin(GL_LINES);
                    glVertex2i(x1, i);
                    glVertex2i(x2, i);
                    glEnd();
                    glFlush();
```

```
                        // printf("\nLine drawn from %d,%d to
%d,%d",x1,i,x2,i);
                   }
              }


              j++;
         }
         // get intersection points for next scanline
         updatexval(&activeEdgeList);
    }
}
```

Output



```

/* Insertion sort utility function O(n^2) */
void insertsort(edgeTableList *lst)…

/*-------- Edge table utility - remove from scanline if ymax O(n) -
void removeymax(edgeTableList *Lst, int scanline)…

/*-------- Edge table utility - update x to x+1/m--------*/
void updatexval(edgeTableList *Lst)…

void Scanfill()…

```

```
krishanu-2001@ubuntu:~/Desktop/SEM6/cs302-cgi/Assn5$ ./Q3 < drawfile2.txt
```