ГБОУ «Президентский ФМЛ №239»

Нахождение триангуляции Делоне множества точек

Годовой проект по информатике

Подготовили Коротченко Таисия, 10-1 и Лотников Алексей, 10-1.

Санкт-Петербург

2021 год

Задача

Пусть дано множество точек S. Тогда существует единственная триангуляция для заданного множества точек S на плоскости, при которой для любого треугольника все точки из S за исключением точек, являющихся его вершинами, лежат вне окружности, описанной вокруг треугольника. Такая триангуляция называется триангуляцией Делоне.

Напоминание. Триангуляция множества точек – это разбиение выпуклой оболочки на треугольники, вершинами которых являются точки исходного множества, причем все точки множества участвуют в триангуляции.

Далее нужно построить данную триангуляцию.

Планируемый внешний вид программы

Белое окно, на котором пользователь отмечает множество точек. Затем программа обрабатывает данные и рисует триангуляцию.

Исходные и выходные данные программы

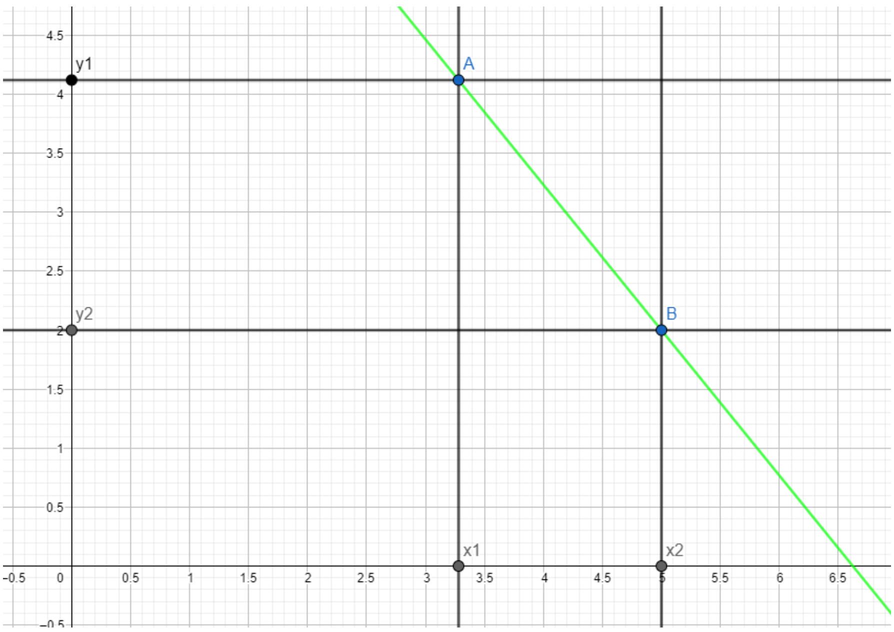
На вход поступает множество точек общего положения, отмеченных пользователем на экране. Каждая точка – это пара величина (int, int). Асимптотика работы нашей программы O(n^2), но строится массив отрезков, так что чтобы не было переполнения памяти, нельзя давать более 1000 точек на вход. Однако понятно, что так как наш проект визуализирован, то вводить более 100 точек не имеет смысла.

Выходными данными являются отрезки триангуляции, то есть четверка переменных (int, int, int, int). Также программа строит треугольники в процессе работы и по ним строит их описанные окружности. Координаты точек по оси Оу принадлежат отрезку (0, 1080), по оси Ох (0, 1920), так как пользователь нажимает на пискель.

Математическая модель

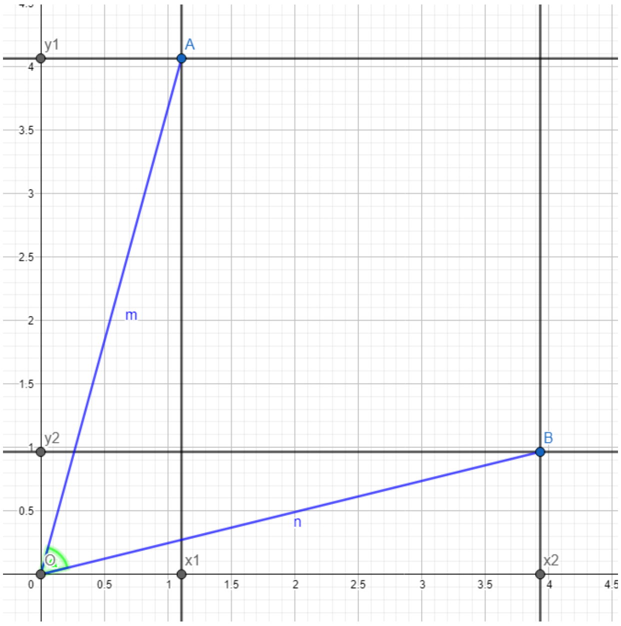
1. Уравнение прямой, проходящей через две данные точки A, B с координатами (x_1, y_1) и (x_2, y_2) соответственно.

$$(y_1 - y_2)x + (x_2 - x_1)y + (x_1 y_2 - x_2 y_1)$$



2. Вычисление косинуса угла A в треугольнике ABC, точка A – начало координат. Известны координаты точек B, C: (x_1, y_1) и (x_2, y_2) соответственно.

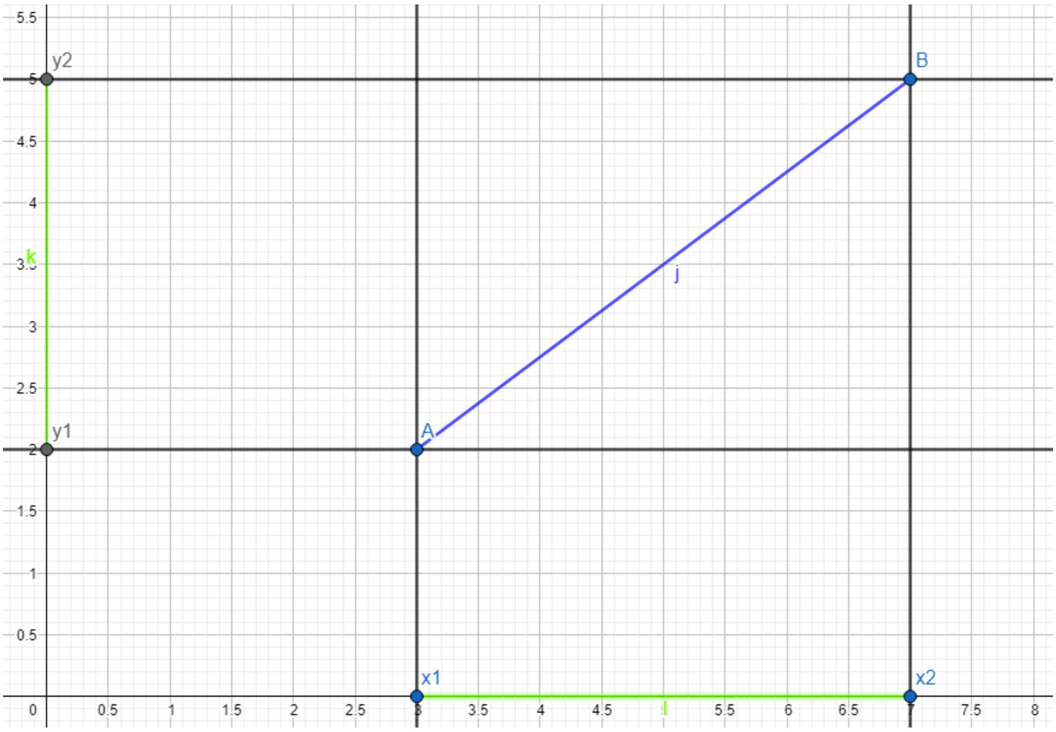
$cosA = \frac{x_1 \cdot x_2 + y_1 y_2}{\sqrt{x_1^2+y_1^2} \cdot \sqrt{x_2^2+y_2^2}}$. Данная формула следует из того, что скалярное произведение векторов – это произведение их модулей, умноженное на косинус угла между векторами.



3. В программе необходимо было разделить точки на две группы, в зависимости от того, в какой полуплоскости относительно данной прямой, проходящей через 2 точки из S, они лежат. Пусть уравнение прямой *ax+by+c=0*. Тогда за то, в какой полуплоскости лежит данная точка с координатами (p, q) отвечает знак выражения *px+qy+c*.

4. Расстояние между точками $A(x_1, y_1)$ $B(x_2, y_2)$

AB = $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$



5. Уравнение серединного перпендикуляра вида $ax+by+c=0$

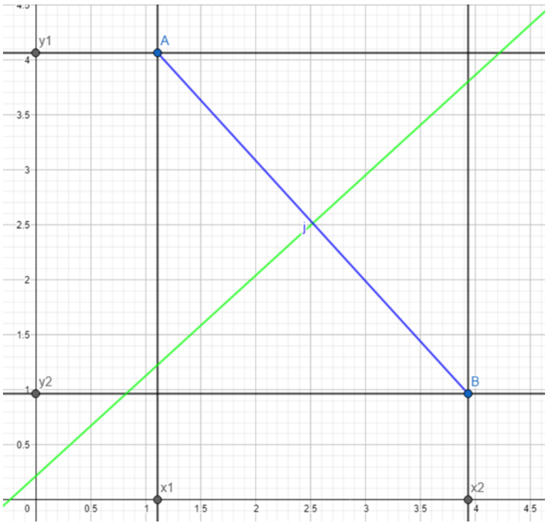
Координаты концов отрезка: $A\ (x_1, y_1)$, $B\ (x_2, y_2)$

Коэффициенты прямой:

$$a = x_2 - x_1$$

$$b = y_2 - y_1$$

$$c = \frac{(x_1^2 - x_2^2) + (y_1^2 - y_2^2)}{2}$$



Точка пересечения прямых вида $ax+by+c=0$

Прямые:

$$a_1x + b_1y + c_1 = 0$$

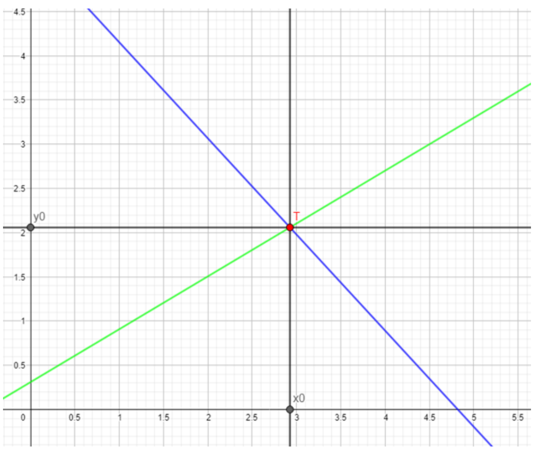
$$a_2x + b_2y + c_2 = 0$$

$\mathrm{T}(x_0, y_0)$ -точка пересечения:

$$x_0 = \frac{(c2 \cdot b1 - c1 \cdot b2)}{(a1 \cdot b2 - a2 \cdot b1)}$$

$$y_0 = \frac{(c2 \cdot a1 - c1 \cdot a2)}{(b1 \cdot a2 - b2 \cdot a1)}$$

В случае деления на ноль прямые параллельны.



Анализ используемой структуры данных

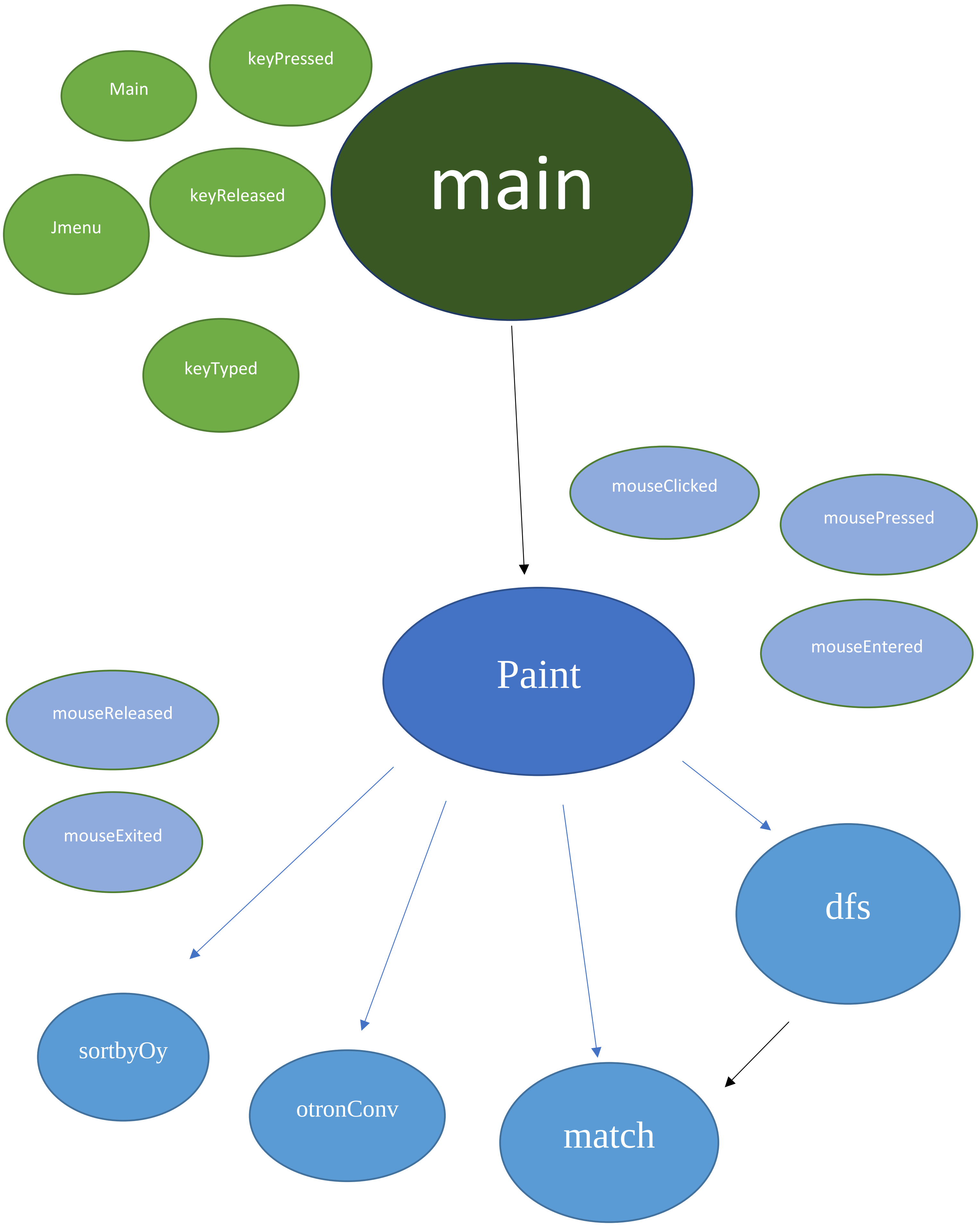
Точки множества S хранятся в массиве. Информация об отрезках хранятся в другом массиве, в котором про каждый отрезок понятно, взят ли он в триангуляцию. Переменная «num» отвечает за количество точек в множестве S. Всю информацию необходимо хранить в массивах, так как программа неоднократно использует эти данные. Так как по каждому номеру точки должны быть доступны обе координаты, а по каждой паре точек, несколько параметров отрезка удобно хранить информацию об отрезках и точках в виде массивов, а так как их удобно сделать глобальными их размеры лучше задать в соответствии с максимальным числом точек. Так как все координаты – координаты пикселей экрана, они целочисленные, значит массивы должны быть соответствующие. Результирующие величины записываются в уже созданные массивы и сразу выводятся.

Метод решения

В программе есть следующие блоки:

1. Подпрограмма: сортировка точек по координате y. (по убыванию)
2. Подпрограмма: Поиск отрезка выпуклой оболочки. Для этого ищем точку A с максимальный координатой по оси ординат. (Она уже известна, так как до этого массив точек был отсортирован). Затем ищем точку B, такую, что угол, образованный между горизонтальной прямой и прямой AB минимальный. Такой отрезок AB гарантировано лежит в выпуклой оболочке.
3. Подпрограмма: Поиск по отрезку AB двух точек C, D (с разных сторон от прямой, образованной данным отрезком), таких, что окружность ABC не содержит внутри себя точек S с той же стороны, что и C. Для D аналогично. Поиск такой точки состоит в поиске максимального угла ACB и проверки, с какой стороны лежит C относительно AB. Такие точки с каждой стороны единственны (если с каждой стороны точки существуют)
   (действия обоснованы свойствами вписанного четырехугольника)
4. Алгоритм работы программы. Работа нашей программы реализована с помощью подпрограммы dfs. По отрезку XY, обрабатываемому в данный момент ищется две точки Z, T (по подпрограмме, описанной в 3 пункте). Они точно лежат в триангуляции вместе с отрезком XY. Далее запускаем dfs для отрезков XT, XZ, YT, YZ. Dfs запускается с отрезка выпуклой оболочки, найденной в пункте 1.

Структура программы:



Комментированный листинг

```
package com.company;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.io.IOException;
import java.util.Scanner;

import static java.lang.Integer.valueOf;

public class Main extends JFrame implements  KeyListener{ // Приложение – наследник JFrame (окна)
    public Main(String title) {// Конструктор приложения
        super(title); // создать окно с указанным заголовком
        setSize(1920, 1080);// задаем размер окна приложения
        setDefaultCloseOperation( EXIT_ON_CLOSE ); // при закрытии окна заканчиваться
        JMenuBar menuBar = new JMenuBar(); // Создание строки главного меню
        menuBar.add(createFileMenu()); // Добавление в главное меню выпадающих пунктов
        setJMenuBar(menuBar); // Подключаем меню к интерфейсу приложения
        MyPanel panel = new MyPanel(); // создаем Панель, на которой можно будет рисовать
        add(panel);// подключаем Панель к текущему JFrame (окну)
        addKeyListener(this);
        setVisible(true); // делаем окно видимым
    }
    private JMenu createFileMenu() // создание меню приложения
    {
        JMenu file = new JMenu("Файл"); // Создание выпадающего меню
        JMenuItem open = new JMenuItem("Открыть"); // Пункт меню "Открыть"
        JMenuItem exit = new JMenuItem("Выход"); // Пункт меню "Выход"
        file.add(open); // Добавим в меню пункта "Открыть"
        file.addSeparator(); // Добавим в меню разделитель
        file.add(exit); // Добавим в меню пункт "Выход"
        JFileChooser fileChooser = new JFileChooser(); // Создадим объект,
        // умеющий показывать диалог выбора файла
        addKeyListener(this);
        open.addActionListener(new ActionListener() { // Действие при выборе меню "Открыть"
            @Override
            public void actionPerformed(ActionEvent arg0) {
                fileChooser.setDialogTitle("Открытие файла");// Заголовок окна диалога
                if (fileChooser.showOpenDialog(Main.this) ==
                        JFileChooser.APPROVE_OPTION) { // если файл был выбран
                    System.out.println("Выбран файл " + fileChooser.getSelectedFile());//Вывод
имени выбранного файла
                }
                try {

                } catch (Exception e) {//Внутри команды код, который показывает что нужно делать
в незапланированной ситуации
                    e.printStackTrace();
                }
            }
        });
        exit.addActionListener(new ActionListener() { // Действие при выборе меню "Выход"
            @Override
            public void actionPerformed(ActionEvent arg0) {
                System.exit(0);
            }
        });
        return file; // возвращаем построенное меню как результат метода
    }

    public static void main(String[] args) throws IOException {
        Main mw = new Main("Delone");//Задание названия файла
    }
    @Override
    public void keyTyped(KeyEvent e) {

    }

    @Override
    public void keyPressed(KeyEvent e) {//Нажатие клавиши
        if(e.getKeyCode() == KeyEvent.VK_ENTER){
            MyPanel.status = true;
            System.out.println(MyPanel.status);
            repaint();
        }
    }
```

```
    @Override
    public void keyReleased(KeyEvent e) {//Отпускание клавиши

    }
}
```

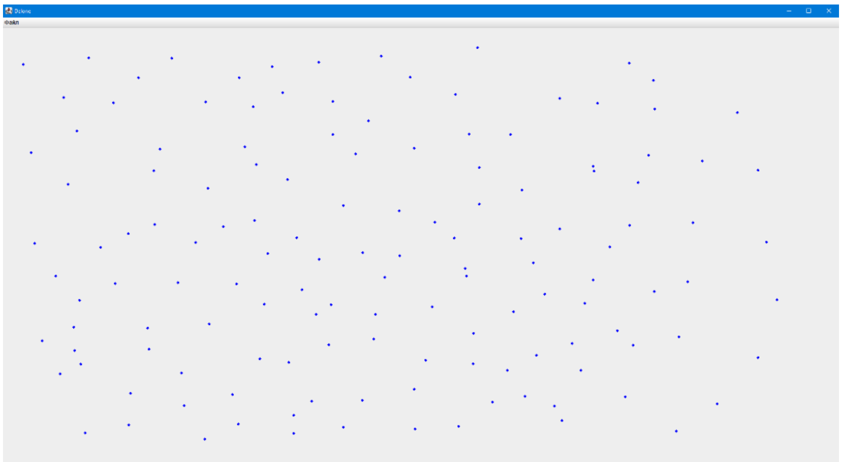
```
package com.company;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class MyPanel extends JPanel implements MouseListener {
    public MyPanel() {
        addMouseListener(this); // добавляем к текущей Панели обработчик мыши
    }
    static public int[][] dot = new int[1001][3]; // объявление массива точек (0 - координата по
X, 1 -координата по Y)
    static public int[][][] stretch = new int[1001][1001][4]; //объявление массива отрезков
    static public int num = 0, goodnum; // объявление переменной количества точек и номеры
вершины лежащей с верхней на выпуклой оболочке
    public static boolean status = false; // объявление статусной переменной
    public void paint(Graphics g) {//Рисование окна
        super.paint(g);// Очищение окна, нарисованного до этого
        for (int i = 0; i< num; i++) // вырисование точек в окне
      {
            g.setColor(Color.blue);
            g.drawOval(dot[i][0]-3, dot[i][1]-3, 5, 5);
            g.fillOval(dot[i][0]-3, dot[i][1]-3, 5, 5);

        }
        if (status == true) { // запуск программы при нажатии Enter
            sortbyOy(); // сортировка массива точек по коорденате у


            otronConv(); //поиск отрезка на выпуклой оболочке множества точек
            stretch[0][goodnum][3] = 1; // изменяем статуст отрезка на выпуклой облочке на "взят
в триангуляцию"
            stretch[goodnum][0][3] = 1;
            g.setColor(Color.BLUE); // прорисовка отрезка
            g.drawLine(dot[0][0], dot[0][1], dot[goodnum][0], dot[goodnum][1]);
            dfs(0, goodnum);// восстанавливаем триангуляию
            double a1, b1, c1, a2, b2, c2, x0, y0, r; // прорисовка полученных отрезков и
окружностей
            double  x1, y1, x2, y2, x3, y3;
            for (int x = 0; x < num; x++) {
                for (int y = 0; y < num; y++) {
                    if (stretch[x][y][3] == 1) {



                            g.setColor(Color.BLUE);
                            g.drawLine(dot[x][0], dot[x][1], dot[y][0], dot[y][1]); // прорисовка
отрезка

                            x1 = dot[x][0]; // запоминаем коорлинаты
                            y1 = dot[x][1];
                            x2 = dot[y][0];
                            y2 = dot[y][1];
                            x3 = dot[ stretch[x][y][1] ][0];
                            y3 = dot[ stretch[x][y][1] ][1];
                            a1 = x2-x1; // проводим серединные перпендикуляры к сторонам
треугольника
                            b1 = y2-y1;
                            c1 = (x1*x1-x2*x2)*0.5+(y1*y1-y2*y2)*0.5;
                            a2 = x2-x3; // проводим серединные перпендикуляры к сторонам
треугольника
                            b2 = y2-y3;
                            c2 = (x3*x3-x2*x2)*0.5+(y3*y3-y2*y2)*0.5;
                            x0 = (c2*b1-c1*b2)/(a1*b2-a2*b1); //пересекаем серединные
перпендикуляры
                            y0 = (c2*a1-c1*a2)/(b1*a2-b2*a1);
                            r = Math.sqrt((x1-x0)*(x1-x0)+(y1-y0)*(y1-y0)); // считаем ралиус
описанной окружности
                            g.setColor(Color.GREEN); // рисуем окружность
                            g.drawOval((int)Math.floor(x0+0.5)-(int)Math.floor(r+0.5),
(int)Math.floor(y0+0.5)-(int)Math.floor(r+0.5), 2*(int)Math.floor(r+0.5),
2*(int)Math.floor(r+0.5));
                     // проделываем аналогичные операции со второй окружностью
```

```java
                            x3 = dot[ stretch[x][y][2] ][0];
                            y3 = dot[ stretch[x][y][2] ][1];
                            a1 = x2-x1;
                            b1 = y2-y1;
                            c1 = (x1*x1-x2*x2)*0.5+(y1*y1-y2*y2)*0.5;
                            a2 = x2-x3;
                            b2 = y2-y3;
                            c2 = (x3*x3-x2*x2)*0.5+(y3*y3-y2*y2)*0.5;
                            x0 = (c2*b1-c1*b2)/(a1*b2-a2*b1);
                            y0 = (c2*a1-c1*a2)/(b1*a2-b2*a1);
                            r = Math.sqrt((x1-x0)*(x1-x0)+(y1-y0)*(y1-y0));
                            g.setColor(Color.GREEN);
                            g.drawOval((int)Math.floor(x0+0.5)-(int)Math.floor(r+0.5),
(int)Math.floor(y0+0.5)-(int)Math.floor(r+0.5), 2*(int)Math.floor(r+0.5),
2*(int)Math.floor(r+0.5));



                    }
                }
            }

        }
    }
    static public void match(int a, int b)
    {
            double minup,mindown, cs, A, B, C, alpha, beta;
            int mindotup, mindotdown, goodnum, x1, y1, x2, y2, x3, y3;
            x1 = dot[a][0]; // запоминанием координаты первого конца отрезка
            y1 = dot[a][1];
            x2 = dot[b][0]; // запоминание координаты второго конца отрезка
            y2 = dot[b][1];
            A = y1 - y2; //определяем коэффициенты прямой содержащей данный отрезок
            B = x2 - x1;
            C = x1 * y2 - x2 * y1;
            if (a != b) // если две точки не совпадают начинаем перебирать точки
                {
                    minup = 2; // минимальный косинус соответствующего для точек НАД прямой
                    mindotup = -1; // номер точки с минимальным косинусом угла среди точек НАД
прямой
                    mindown = 2; // минимальный косинус соответствующего для точек ПОД прямой
                    mindotdown = -1; // минимальный номер точки с минимальным косинусом угла
среди точек ПОД прямой
                    for (int c = 0; c < num; c++)
                    {
                        x3 = dot[c][0]; // запоминаем координаты точки
                        y3 = dot[c][1];
                        double x11 = x1 - x3; // сдвигаем вектора из с так, чтобы с попала в
(0,0)
                        double y11 = y1 - y3;
                        double x21 = x2 - x3;
                        double y21 = y2 - y3;
                        cs = (x11 * x21 + y11 * y21) / (Math.sqrt(x11 * x11 + y11 * y11) *
Math.sqrt(x21 * x21 + y21 * y21)); //считаем косинус угла между векторами по формуле
                        if (A * x3 + B * y3 + C >= 0) // определение расположение точки
относительно прямой
                        {
                            if (cs < minup) // обновление минимального косинуса
                            {
                                minup = cs;
                                mindotup = c;
                            }
                        } else { // определение расположение точки относительно точки
                            if (cs < mindown) // обновление минимального косинуса
                            {
                                mindown = cs;
                                mindotdown = c;
                            }
                        }

                    }

                    if (mindotup == -1) //дозаполнение парных точек в случае их отсутствия
                    {
                        mindotup = mindotdown;

                    }
                    if (mindotdown == -1) {
                        mindotdown = mindotup;
                    }
```

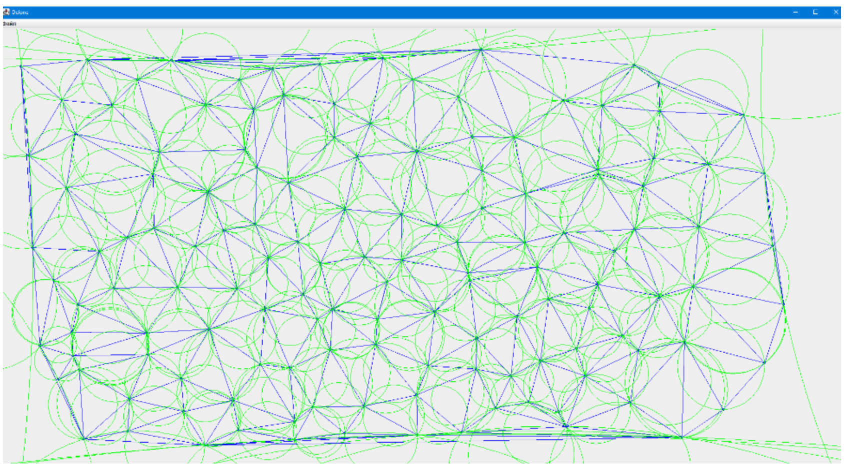
```
                    stretch[a][b][1] = mindotup; //запоминаем найденные парные точки
                    stretch[a][b][2] = mindotdown;
                    stretch[b][a][1] = mindotup; //запоминаем найденные парные точки
                    stretch[b][a][2] = mindotdown;
                    if (minup != 2 && mindown != 2) { // проверка возможности нахождения отрезка
в триангуляции
                        alpha = Math.acos(minup);
                        beta = Math.acos(mindown);
                        if (alpha + beta >= Math.PI) {
                            stretch[a][b][0] = 1;
                        }
                    }


                }




    }

    static public void dfs (int a, int b) // реализует метод dfs
    {
        match(a, b); // поиск соответствующих точек
        stretch[a][b][3]=1; // постановка маркера наличия в триангуляции
        stretch[b][a][3]=1;
        if(stretch[a][stretch[a][b][1]][3]==0) // проверка наличии в триангуляции, в противном
случае - запуск dfs
        {
            dfs(a, stretch[a][b][1] );
        }
        if(stretch[a][stretch[a][b][2]][3]==0) // проверка наличии в триангуляции, в противном
случае - запуск dfs
        {
            dfs(a, stretch[a][b][2] );
        }
        if(stretch[b][stretch[a][b][1]][3]==0) // проверка наличии в триангуляции, в противном
случае - запуск dfs
        {
            dfs(b, stretch[a][b][1] );
        }
        if(stretch[b][stretch[a][b][2]][3]==0) // проверка наличии в триангуляции, в противном
случае - запуск dfs
        {
            dfs(b, stretch[a][b][2] );
        }

    }

    static public void sortbyOy() //сортировка массива точек по координате y поплавковым методом
    {
        int a, b, c, d;
        for (int i = 0; i < num - 1; i++) {
            if (dot[i][1] > dot[i + 1][1]) {
                a = dot[i][0];
                b = dot[i][1];
                c = dot[i + 1][0];
                d = dot[i + 1][1];
                dot[i][0] = c;
                dot[i][1] = d;
                dot[i + 1][0] = a;
                dot[i + 1][1] = b;
                sortbyOy();
            }

        }
    }
    static public void otronConv( ) // поиск отрезка на выпуклой оболочке методом заметающей
прямой
    {
        double xx = dot[0][0]; // берем самую левую точку
        double yy = dot[0][1];
        double t = -1, t0;
        double x0, y0;
        goodnum = -1;
        for (int i = 0; i < num; i++) { // ищем точку отрезок от начальной до которой наименее
отклонен от оси Ox
            x0 = dot[i][0]; //запоминаем координаты
            y0 = dot[i][1];
            t0 = Math.abs(x0 - xx)*1.0 / Math.sqrt((xx - x0) * (xx - x0) + (yy - y0) * (yy -
y0)); // считаем косинус искомого угла
```

```
            if (t0 > t)
            {
                t = t0;
                goodnum = i;
            }
        }

    }

    @Override
    public void mouseClicked(MouseEvent e) {
    }
    @Override
    public void mousePressed(MouseEvent mouseEvent) {//Нажатие мыши
        if(mouseEvent.getButton() == MouseEvent.BUTTON1){
            dot[num][0] = mouseEvent.getX();
            dot[num][1] = mouseEvent.getY();
            System.out.println(dot[num][0] + " " + dot[num][1] + "/");
            num++;
            repaint();
        }
    }
    @Override
    public void mouseReleased(MouseEvent e) {//Отпускание мыши

    }
    @Override
    public void mouseEntered(MouseEvent e) {

    }
    @Override
    public void mouseExited(MouseEvent e) {

    }
}
```

Пример работы программы



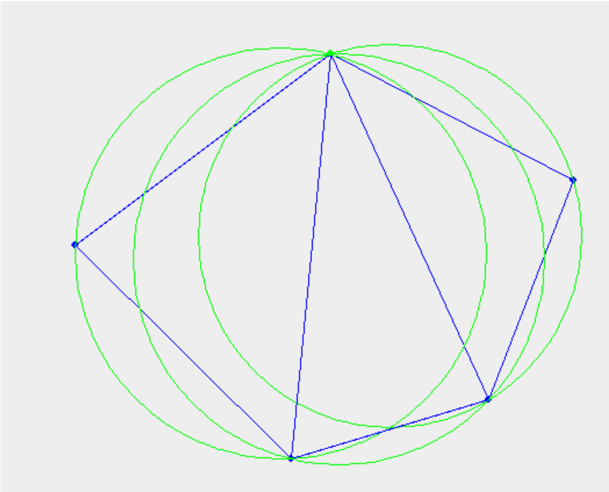
Введенные данные

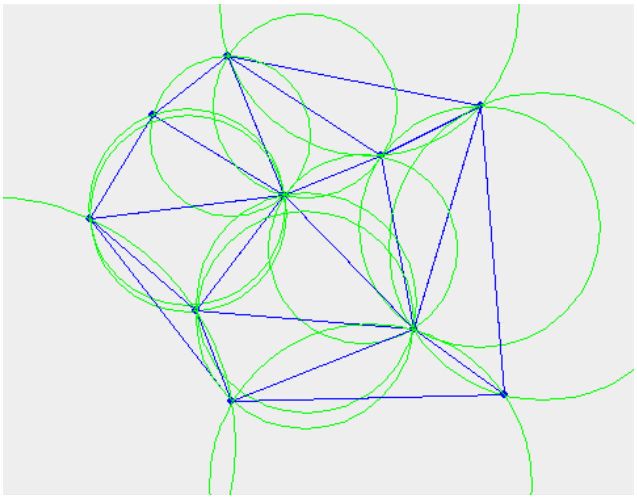


Выходные данные

Анализ работы программы

Существование такой триангуляции хорошо известно. Сейчас покажем, что так как триангуляция существует, то построенная нами триангуляция верна. Любой отрезок выпуклой оболочки гарантировано принадлежит триангуляции. Далее, если какой-то отрезок AB лежит, то и отрезки AC, BC тоже, где точка C, во-первых, лежит в фиксированной полуплоскости, относительно прямой AB, а во-вторых, среди всех точек C в данной полуплоскости, угол ACB наибольший. Продолжая алгоритм рекурсивно, строится искомая триангуляция.





Использованная литература

https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B8%D0%B0%D0%BD%D0%B3%D1%83%D0%BB%D1%8F%D1%86%D0%B8%D1%8F_%D0%94%D0%B5%D0%BB%D0%BE%D0%BD%D0%B5

https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BB%D0%BE%D0%BD%D0%B5,_%D0%91%D0%BE%D1%80%D0%B8%D1%81_%D0%9D%D0%B8%D0%BA%D0%BE%D0%BB%D0%B0%D0%B5%D0%B2%D0%B8%D1%87

http://grafika.me/node/285

https://studfile.net/preview/3731478/page:26/

https://support.microsoft.com/ru-ru/office/%D0%B2%D1%81%D1%82%D0%B0%D0%B2%D0%BA%D0%B0-%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D1%85-%D0%B7%D0%BD%D0%B0%D0%BA%D0%BE%D0%B2-91a4b04c-84a8-4de9-bd13-8609e14bed58