**Министерство образования Республики Беларусь**

**Учреждение образования**

**«Белорусский государственный университет информатики   
и радиоэлектроники»**

Кафедра интеллектуальных

информационных технологий

**Отчёт**

по лабораторной работе № 5

по курсу «Графический интерфейс интеллектуальных систем»

Выполнили:

студенты группы 821701

Булова М.И.

Титенков П.В

Минск 2011

**Цель:**

Изучить алгоритмы построения замкнутых областей. Разработать редактор, позволяющий разбиение плоскости на сегменты согласно алгоритму Вороного (диаграммы Вороного).

**Алгоритм:**

В лабораторной работе был применен алгоритм Форчуна. Он основан на применении заметающей прямой. Заметающая прямая — это вспомогательный объект, представляющий собой вертикальную прямую линию. На каждом шаге алгоритма диаграмма Вороного построена для множества, состоящего из заметающей прямой и точек слева от неё. При этом граница между областью Вороного прямой и областями точек состоит из отрезков парабол (так как геометрическое место точек, равноудалённых от заданной точки и прямой — это [парабола](http://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B1%D0%BE%D0%BB%D0%B0)). Прямая движется слева направо. Каждый раз, когда она проходит через очередную точку, эта точка добавляется к уже построенному участку диаграммы. Добавление точки к диаграмме при использовании [двоичного дерева поиска](http://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%D0%B0) имеет сложность *O*(log *n*), всего точек *n*, а сортировка точек по *x*-координате может быть выполнена за *O*(*n*log*n*), поэтому [вычислительная сложность](http://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C) алгоритма Форчуна равна *O*(*n*log*n*).

**class VoronoiClass extends Vector**

**{**

**public VoronoiClass (int width, int height, int points)**

**{**

**if(points > 0)**

**{**

**boolean flag = false;**

**for(int x = 30; x < width; x += width/8)**

**{**

**int start\_y = (flag ^= true) ? 0 : width/16;**

**// for(int y = 30 + start\_y; y < height; y += width/8)**

**for(int y = 30 + start\_y; y < height; y += height/2)**

**addElement(new MyPoint(x, y));**

**}**

**// addElement(new MyPoint(10D, height / 2));**

**}**

**checkDegenerate();**

**}**

**public void checkDegenerate ()**

**{**

**if(size() > 1)**

**{**

**MyPoint min = (MyPoint)elementAt(0), next = min;**

**for(int i = 1; i < size(); i++)**

**{**

**Object element = elementAt(i);**

**if(element instanceof MyPoint)**

**{**

**if(((MyPoint)element).x <= min.x)**

**{**

**next = min;**

**min = (MyPoint)element;**

**}**

**else if(((MyPoint)element).x <= min.x)**

**{**

**next = (MyPoint)element;**

**}**

**}**

**}**

**if(min.x == next.x && min != next)**

**{**

**min.x--;**

**System.out.println("Moved point: " + next.x + " -> " + min.x);**

**}**

**}**

**}**

**public void paint (Graphics g, boolean flag)**

**{**

**for(int i = 0; i < size(); i++)**

**{**

**if(flag || !(elementAt(i) instanceof MyLine))**

**((Paintable)elementAt(i)).paint(g);**

**}**

**}**

**public void clear()**

**{**

**for(int i = 0; i < size(); i++)**

**{**

**if(elementAt(i) instanceof MyLine)**

**removeElementAt(i--);**

**}**

**}**

**}**

**public CirclePoint calculateCenter(MyPoint mypoint, ArcNode arcnode, MyPoint mypoint1)**

**{**

**CirclePoint circlepoint = null;**

**MyPoint mypoint2 = new MyPoint(arcnode.x - mypoint.x, arcnode.y - mypoint.y);**

**MyPoint mypoint3 = new MyPoint(mypoint1.x - arcnode.x, mypoint1.y - arcnode.y);**

**if(mypoint3.y \* mypoint2.x > mypoint3.x \* mypoint2.y)**

**{**

**double d = -mypoint2.x / mypoint2.y;**

**double d1 = (mypoint.y + mypoint2.y / 2D) - d \* (mypoint.x + mypoint2.x / 2D);**

**double d2 = -mypoint3.x / mypoint3.y;**

**double d3 = (arcnode.y + mypoint3.y / 2D) - d2 \* (arcnode.x + mypoint3.x / 2D);**

**double d4;**

**double d5;**

**if(mypoint2.y == 0.0D)**

**{**

**d4 = mypoint.x + mypoint2.x / 2D;**

**d5 = d2 \* d4 + d3;**

**} else**

**if(mypoint3.y == 0.0D)**

**{**

**d4 = arcnode.x + mypoint3.x / 2D;**

**d5 = d \* d4 + d1;**

**} else**

**{**

**d4 = (d3 - d1) / (d - d2);**

**d5 = d \* d4 + d1;**

**}**

**circlepoint = new CirclePoint(d4, d5, arcnode);**

**}**

**return circlepoint;**

**}**

**Выводы:**

В ходе лабораторной работы были изучен алгоритм Вороного, а точнее его вариант реализации - алгоритм Форчуна. Был разработан графический редактор, который позволяет в разных режимах увидеть разбиение области на сектора. Предусмотрен режим пошаговой отладки алгоритма, когда мы можем перемещаться к следующему пикселю заметающей прямой или к следующей точке на плоскости. Также разработан интерфейс для расстановки точек на плоскости и их удаления.