Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 8

Дисциплина Компьютерная графика

Тема Реализация алгоритма отсечения отрезка произвольным выпуклым отсекателем.
(Алгоритм Кируса-Бека)

Студент Хетагуров П.К.

Группа ИУ7-45

Оценка (баллы) _____

Преподаватель Куров А. В

Москва.
2020 г.

Цель работы:

Изучение и программная реализация алгоритма отсечения отрезка

Задание:

Необходимо обеспечить ввод отсекателя – произвольного многоугольника. Высветить его первым цветом. Также необходимо обеспечить ввод нескольких (до десяти) различных отрезков (высветить их вторым цветом). Отрезки могут иметь произвольное расположение: горизонтальные, вертикальные, имеющие произвольный наклон. Предусмотреть ввод отрезков, параллельных границе отсекателя.

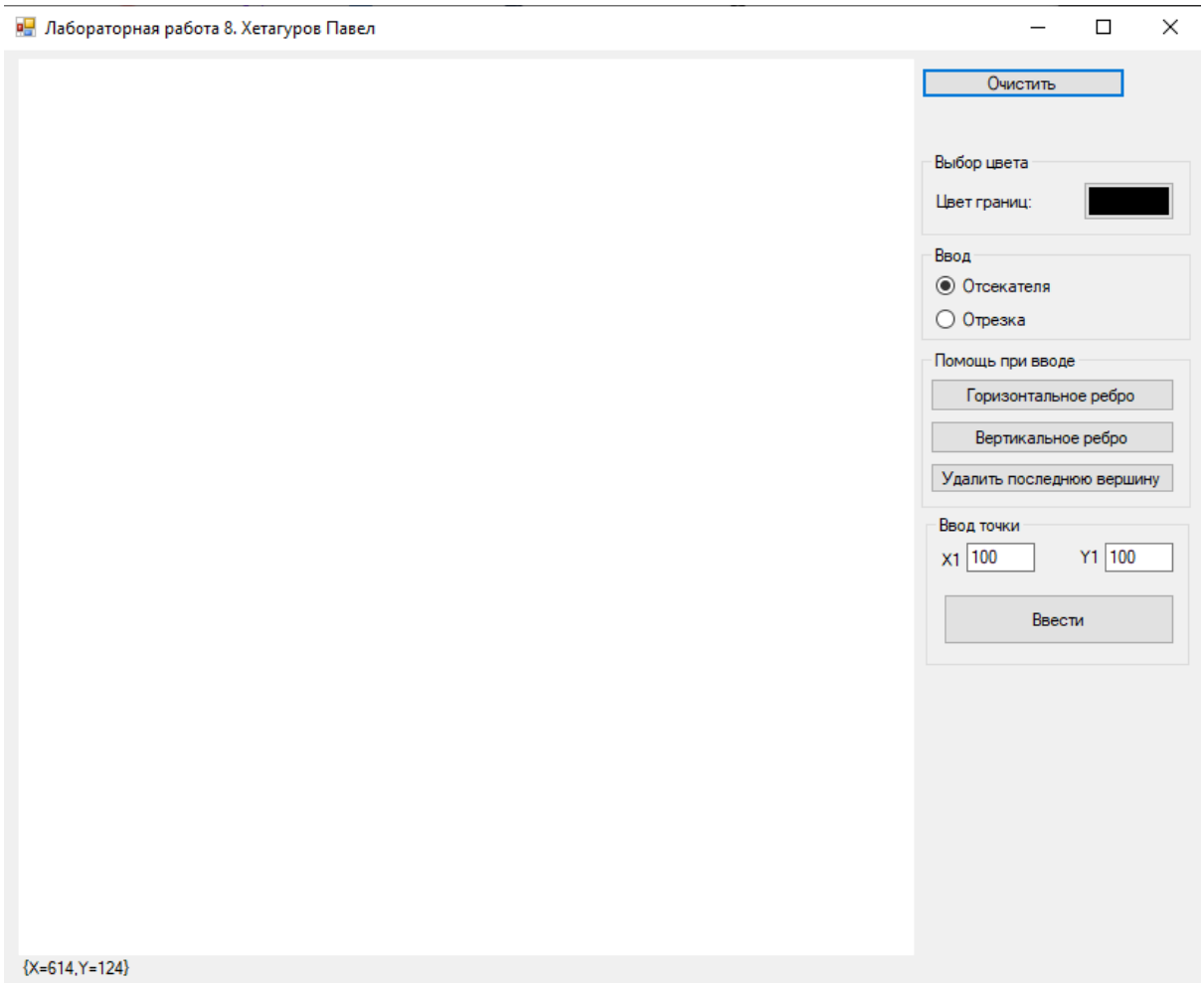
Ввод осуществлять с помощью мыши и нажатия других клавиш.

Выполнить отсечение отрезков, показав результат третьим цветом. Исходные отрезки не удалять.

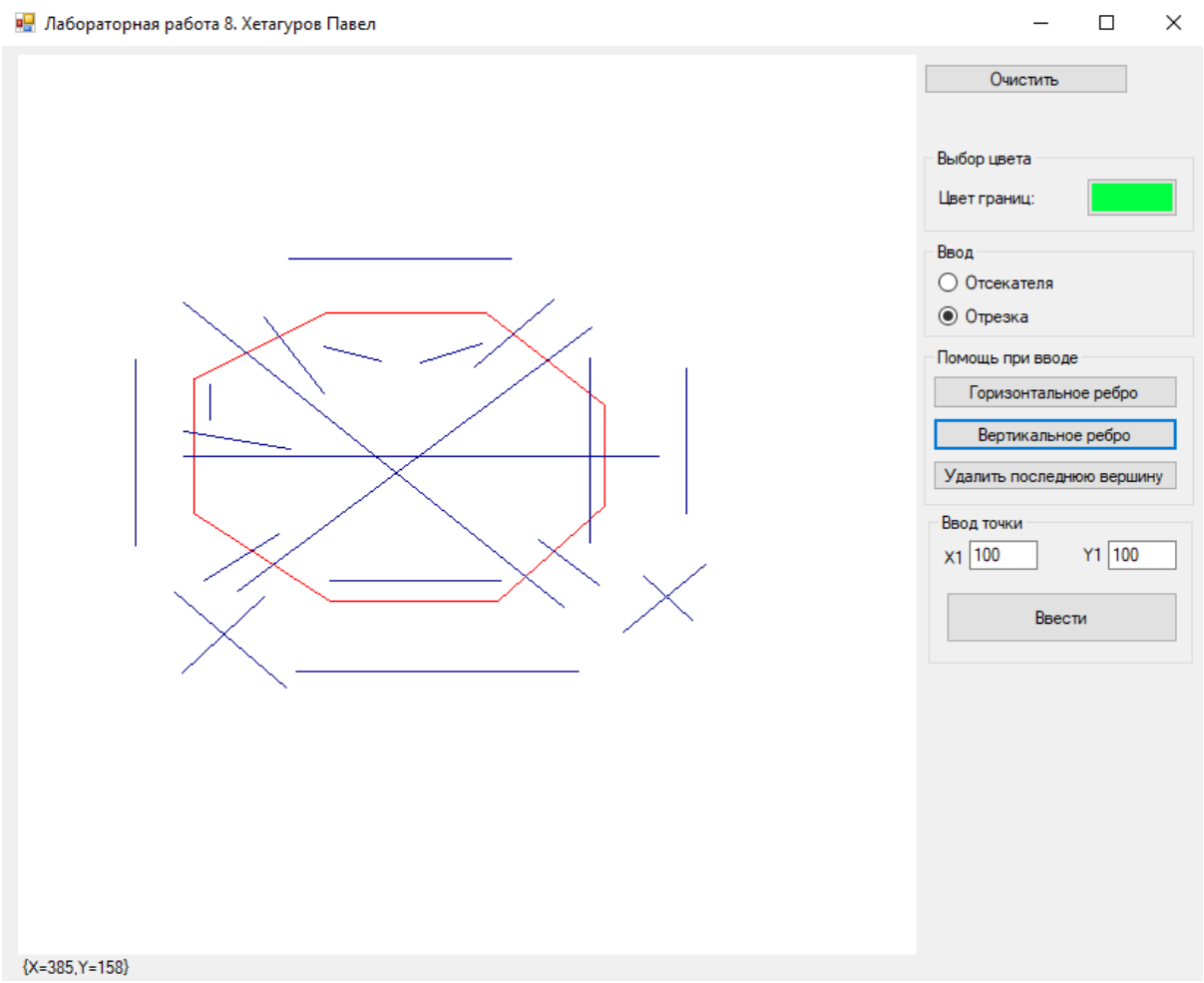
В трех уже рассмотренных алгоритмах отсечения отрезков предполагалось, что отсекатель является прямоугольником со сторонами, параллельными осям координат. Однако, очень часто оно повернуто относительно координатной сетки или не является прямоугольным. Поэтому Кирус и Бек предложили алгоритм отсечения окном произвольной выпуклой формы.
В этом алгоритме для определения местоположения точки относительно окна отсечения используется вектор нормали и параметрическая форма задания отрезка

Практическая часть

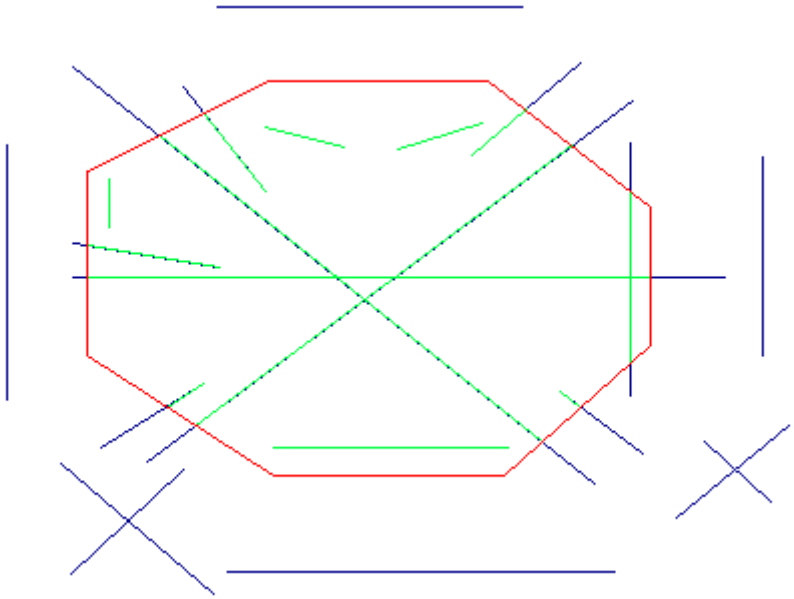
Интерфейс:



Введенное:



После отсечения:



Различия при наложении обуславливается работой алгоритма построения отрезка

Код:

```
        public static List<(Point, Point)> Cutting(List<Point> cutter, List<(Point,
Point)> lines)
        {

            List<(Point, Point)> resultLines = new List<(Point, Point)>();

            if (!IsConvex(cutter))
            {
                throw new Exception("Отсекатель не выпуклый");
            }
            else
            {
                List<Vector> normalVectors = FormNormalVectors(cutter);
                double t_down;
                double t_up;
                double t;

                Vector D;
                Vector W;
                double scalarD;
                double scalarW;

                foreach (var line in lines)
                {
                    t_down = 0;
                    t_up = 1;

                    D = new Vector(line.Item1, line.Item2);

                    for (int i = 0; i < cutter.Count; i++)
                    {
                        W = new Vector(cutter[i], line.Item1);
                        scalarD = D.ScalarMultiplication(normalVectors[i]);
                        scalarW = W.ScalarMultiplication(normalVectors[i]);

                        if (scalarD == 0)
                        {
                            if (scalarW < 0)
                            {
                                t_up = 0;
                                t_down = 1;
                                break;
                            }
                        }
                        else
                        {
                            t = -(scalarW / scalarD);
                            if (scalarD > 0)
                            {
                                t_down = Math.Max(t_down, t);
                                if (t > 1)
                                {
                                    break;
                                }
                            }
                            else
                            {
                                t_up = Math.Min(t_up, t);
                                if (t < 0)
                                {
                                    break;
                                }
                            }
                        }
```

```
                }

                if (t_down > t_up)
                {
                    continue;
                }

                resultLines.Add(CutByParam(t_down, t_up, line));

            }
        }

        return resultLines;
    }

    private static bool IsConvex(List<Point> cutter)
    {
        int sign = 0;
        bool isConvex = true;

        if (cutter.Count >= 3)
        {
            Vector first = new Vector(cutter[0], cutter[1]);
            Vector second;
            Vector result;

            for (int i = 0; i < cutter.Count && isConvex; i++)
            {
                second = new Vector(GetVertex(cutter, i), GetVertex(cutter, i +
1));
                result = first.VectorMultiplication(second);

                if (sign == 0)
                {
                    sign = Math.Sign(result.Z);
                }
                else if ((sign != Math.Sign(result.Z)) && (result.Z != 0))
                {
                    sign = 0;
                    isConvex = false;
                }

                first = second;
            }
        }

        if (sign == 0)
        {
            isConvex = false;
        }

        return isConvex;
    }

    private static List<Vector> FormNormalVectors(List<Point> cutter)
    {
        List<Vector> normalVectors = new List<Vector>();
        Vector vector;
        Vector result;

        for (int i = 0; i < cutter.Count; i++)
        {
            vector = new Vector(GetVertex(cutter, i), GetVertex(cutter, i + 1));
            if (vector.X != 0)
            {
                result = new Vector(-vector.Y / vector.X, 1);
            }
```

```
                else
                {
                    result = new Vector(1, -vector.X / vector.Y);
                }

                if (result.ScalarMultiplication(new Vector(GetVertex(cutter, i - 1),
GetVertex(cutter, i + 1))) > 0)
                {
                    result.X = -result.X;
                    result.Y = -result.Y;
                }

                normalVectors.Add(result);
            }

            return normalVectors;
        }
```