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 № 7

Дисциплина Компьютерная графика

Тема Реализация алгоритма отсечения отрезка регулярным отсекателем

Студент Хетагуров П.К.

Группа ИУ7-45

Оценка (баллы) _____

Преподаватель Куров А. В

Москва.
2020 г.

Вариант 27. Алгоритм разбиения отрезка средней точкой

Цель работы:

Изучение и программная реализация алгоритма отсечения отрезка

Задание:

Необходимо обеспечить ввод регулярного отсекателя - прямоугольника. 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.

Ввод осуществлять с помощью мыши и нажатия других клавиш.

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

Теоретическая часть:

В данном алгоритме непосредственное вычисление координат отсутствует, точка пересечения определяется с использованием двоичного поиска. Рассматриваемый алгоритм является частным случаем алгоритма отсечения Сазерленда-Коэна.

Программная реализация алгоритма медленнее, чем алгоритм Сазерленда-Коэна. Но аппаратная гораздо эффективнее, так как операции сложения и деления на два очень быстры

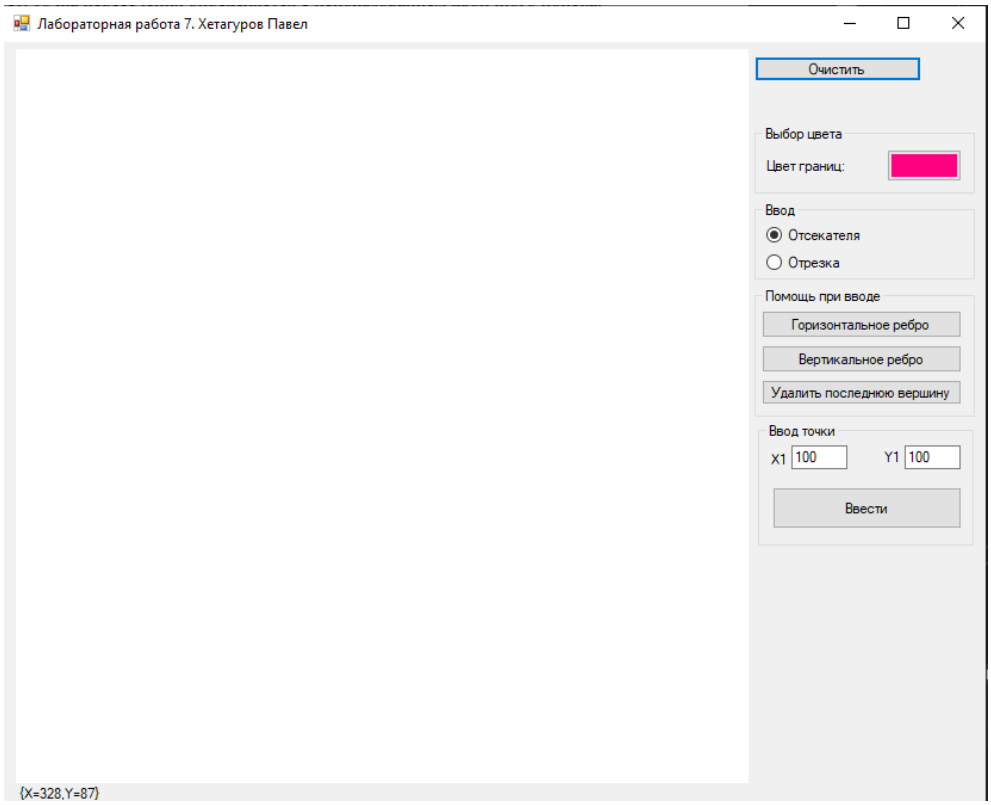
Суть алгоритма:

Сначала выявляется тривиальная видимость или невидимость отрезка, путем сравнения кодов его конечных точек.

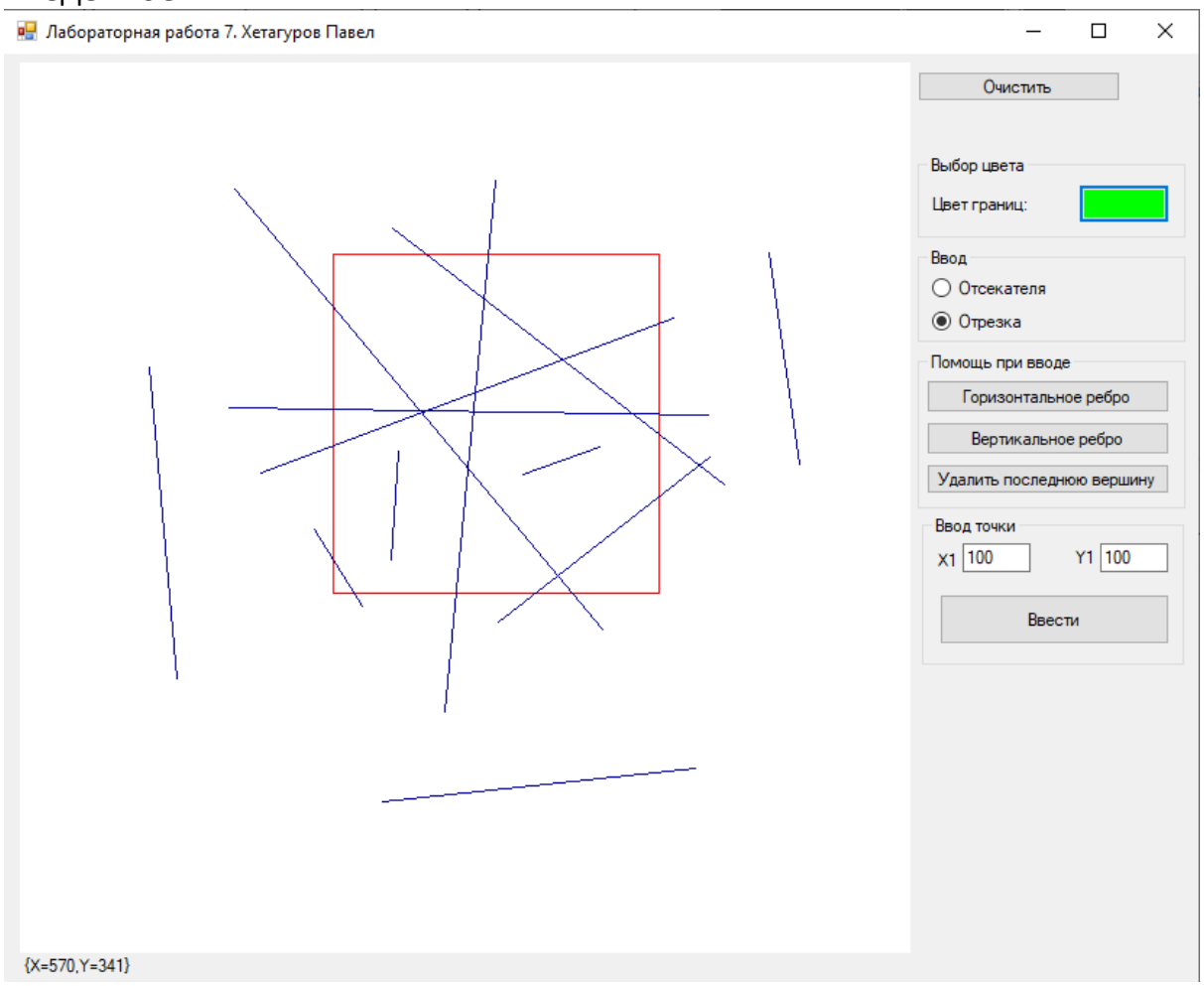
Если отрезок оказался частично видимым, то он разбивается на две равные части, к которым применяют те же проверки до тех пор, пока не будет обнаружено пересечение со стороной окна отсечения, либо пока он не выродится в точку. Затем определяется видимость вычисленной точки. Таким образом, процесс определения пересечения сводится к двоичному поиску.

Практическая часть

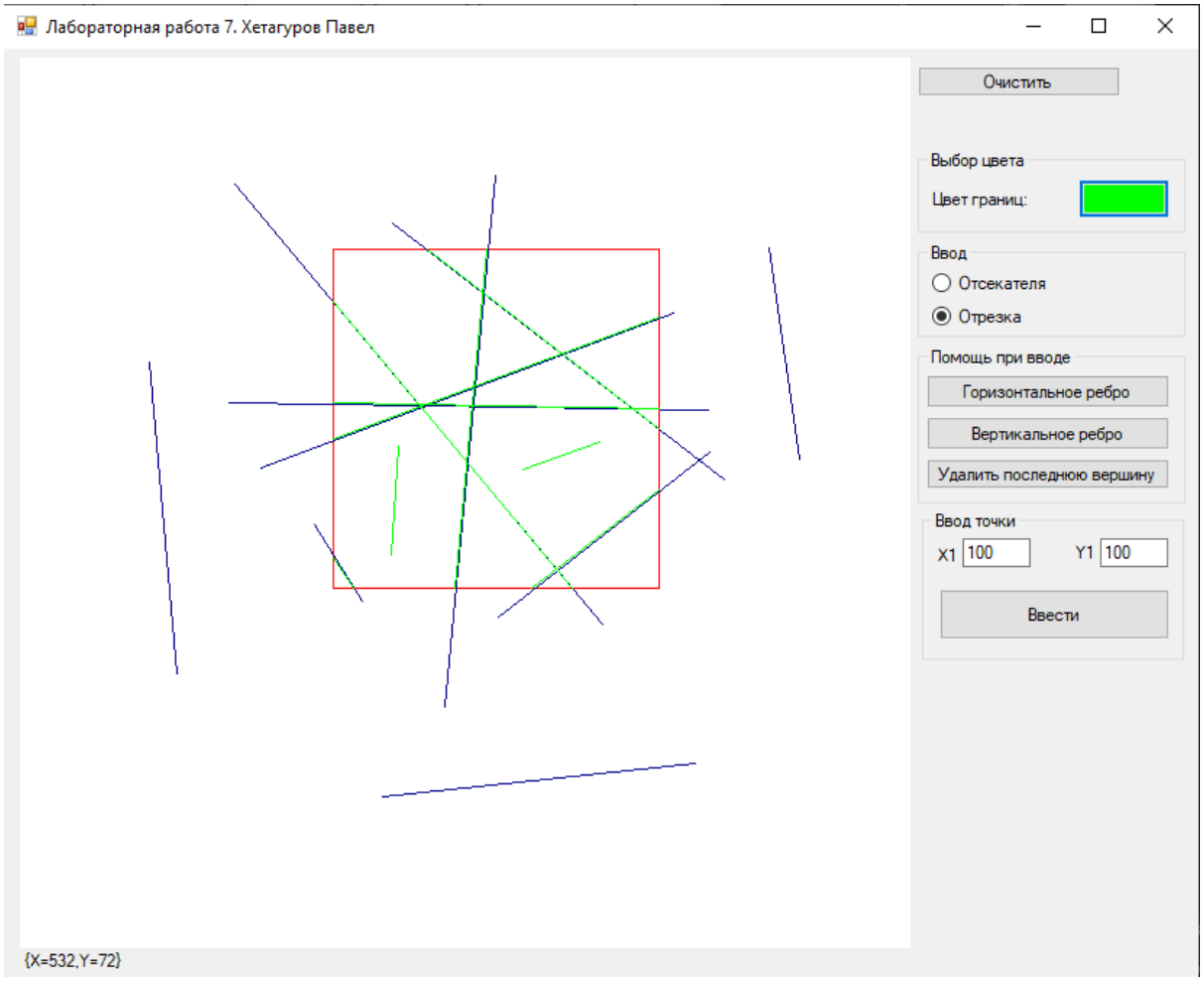
Интерфейс:



Введенное:



После отсечения:



Код:

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

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

        Point first, second, middle, buff, temp;
        middle = Point.Empty;
        byte codeFirst, codeSecond;
        int i;

        foreach (var line in lines)
        {

           i = 1;

           first = line.Item1;
           second = line.Item2;

           while (true)
```

```
            {
                codeFirst = ComputeBinCode(cutter, first);
                codeSecond = ComputeBinCode(cutter, second);

                if (codeFirst == 0 && codeSecond == 0)
                {
                    // отрезок тривиально видим
                    curLines.Add(line);
                    break;
                }

                if ((codeFirst & codeSecond) != 0)
                {
                    // отрезок тривиально не видим
                    break;
                }
                else // частично видим
                {
                    buff = first;

                    if (i > 2)
                    {
                        if ((codeFirst & codeSecond) == 0) // отрезок видим
                        {
                            curLines.Add((first, second));
                        }
                        break;
                    }

                    if (codeSecond == 0)
                    {
                        first = second;
                        second = buff;
                        i++;
                    }
                    else
                    {
                        // пока не вырождается в точку
                        while (Math.Abs(first.X - second.X) > 1 || Math.Abs(first.Y -
second.Y) > 1)
                        {
                            middle.X = (first.X + second.X) / 2;
                            middle.Y = (first.Y + second.Y) / 2;
```

```
                temp = first;
                first = middle;
                codeFirst = ComputeBinCode(cutter, first);
                if ((codeFirst & codeSecond) != 0)
                {
                    // эта часть не видима, обрабатываем вторую половину
                    first = temp;
                    second = middle;
                }
            }

            first = second;
            second = buff;
            i++;
        }
    }
  }

  return curLines;

}
```