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 № 9

Дисциплина Компьютерная графика

Тема ОТСЕЧЕНИЕ ПРОИЗВОЛЬНОГО МНОГОУГОЛЬНИКА ВЫПУКЛЫМ ОТСЕКАТЕЛЕМ
(АЛГОРИТМ САЗЕРЛЕНДА-ХОДЖМЕНА)

Студент Хетагуров П.К.

Группа ИУ7-45

Оценка (баллы) _____

Преподаватель Куров А. В

Москва.
2020 г.

Цель работы:

Изучение и программная реализация алгоритма Сазерленда-Ходжмена отсечения многоугольников.

Задание:

Необходимо обеспечить ввод отсекателя – произвольного многоугольника. Высветить его первым цветом. Также необходимо обеспечить ввод отсекаемого многоугольника (высветить вторым цветом). Должна присутствовать проверка отсекателя на выпуклость. Должен быть предусмотрен ввод вершин многоугольника в произвольных точках ребер отсекателя (включая его вершины)

Ввод осуществлять с помощью мыши и нажатия других клавиш.

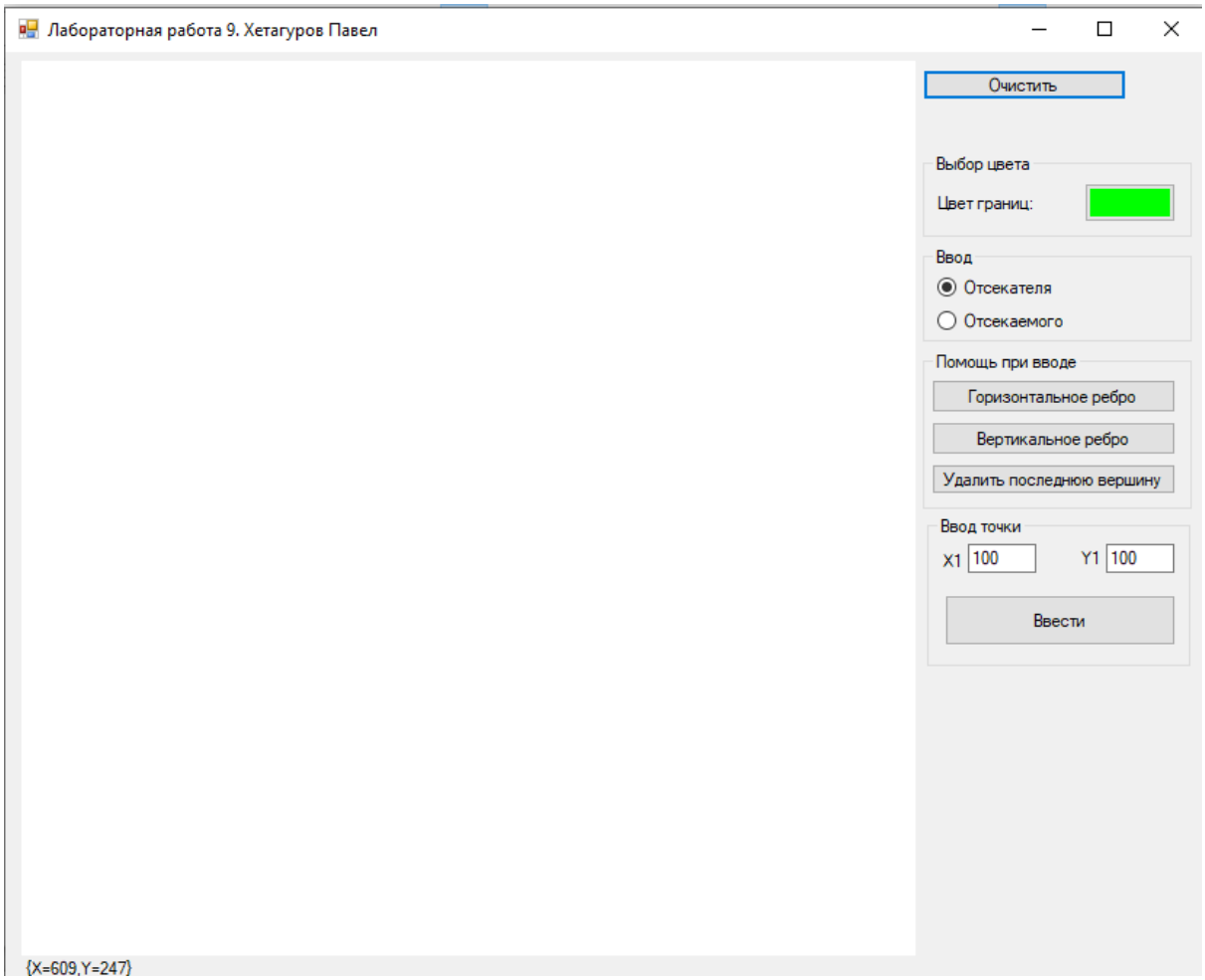
Выполнить отсечение многоугольника, показав результат третьим цветом. Исходный многоугольник не удалять.

Идея алгоритма:

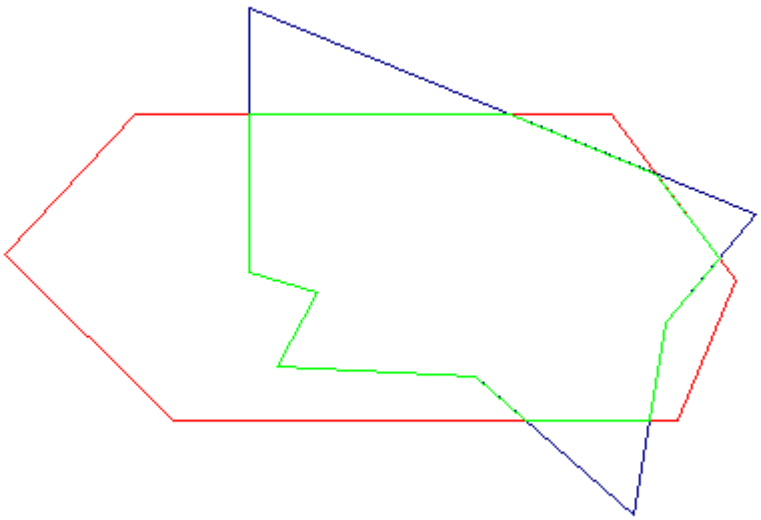
Алгоритм Сазерленда-Ходжмена позволяет провести отсечение произвольного (выпуклого или невыпуклого) многоугольника по границам выпуклого отсекателя. Идея алгоритма достаточно проста. На каждом шаге отсечения исходный и промежуточные многоугольники отсекаются последовательно очередной границей отсекателя. Отсечение многоугольника относительно одной прямой не представляет больших затруднений.

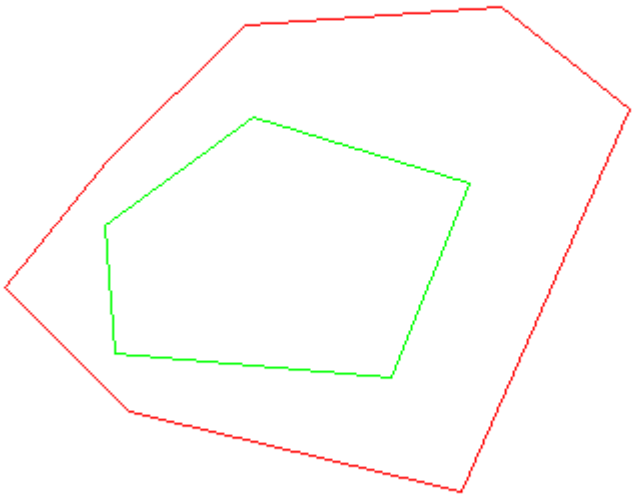
Практическая часть

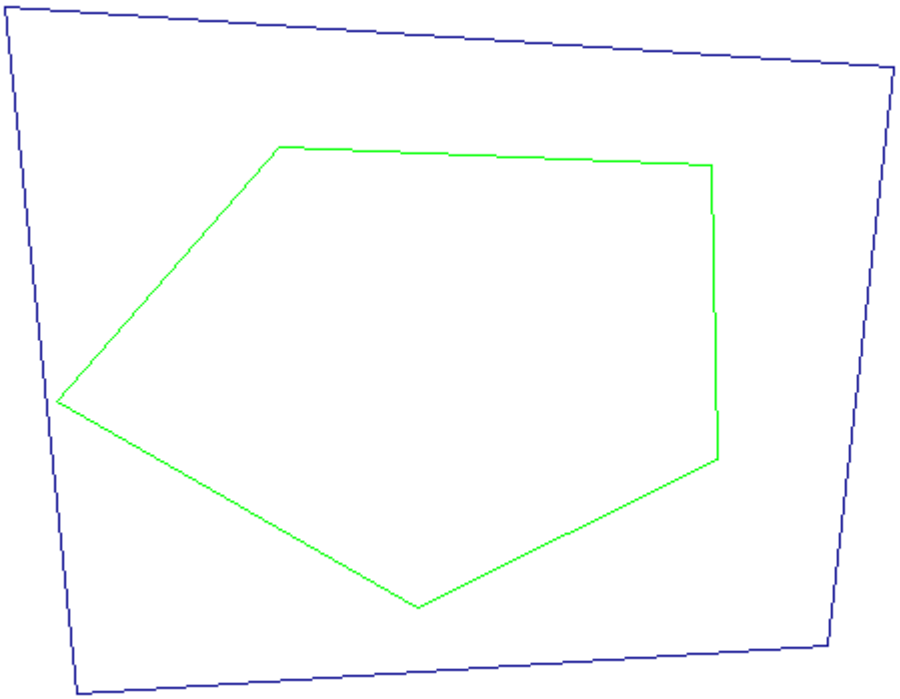
Интерфейс:



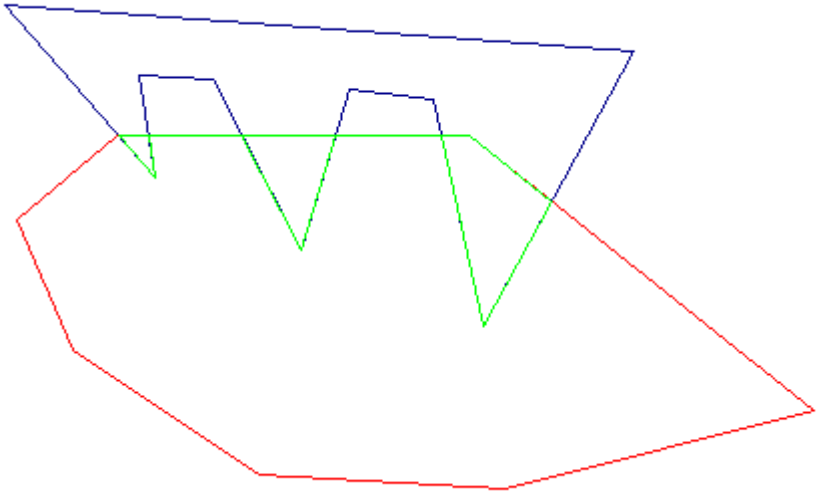
Примеры работы (красным — отсекатель, синим — отсекаемоый многоугольник, зеленым - результат):







Ложные ребра:



Код:

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

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

                Vector D;
                Vector W;

                double scalarD;
                double scalarW;
                double t;

                Point tempPoint;
                for (int i = 0; i < cutter.Count; i++)
                {
                    newTempPolygon = new List<Point>();
                    if (new Vector(cutter[i], GetVertex(polygon,
0)).ScalarMultiplication(normalVectors[i]) > 0)
                    {
                        newTempPolygon.Add(GetVertex(polygon, 0));
                    }
```

```
                    for (int j = 0; j < polygon.Count; j++)
                    {
                        D = new Vector(GetVertex(polygon, j), GetVertex(polygon, j +
1));
                        scalarD = D.ScalarMultiplication(normalVectors[i]);

                        if (scalarD != 0)
                        {

                            W = new Vector(cutter[i], polygon[j]);
                            scalarW = W.ScalarMultiplication(normalVectors[i]);
                            t = -scalarW / scalarD;

                            if (0 <= t && t <= 1)
                            {
                                newTempPolygon.Add(CutByParam(t, (GetVertex(polygon,
j), GetVertex(polygon, j + 1))));
                            }
                        }

                        if (new Vector(cutter[i], GetVertex(polygon, j +
1)).ScalarMultiplication(normalVectors[i]) > 0)
                        {
                            newTempPolygon.Add(GetVertex(polygon, j + 1));
                        }
                    }

                    polygon = newTempPolygon;
                }
            }

            return polygon;
        }

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;
        }
```