|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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**

|  |  |
| --- | --- |
| **Тема**  Отсечение произвольного многоугольника выпуклым отсекателем.  (алгоритм Сазерленда-Ходжмена)  **Студент** Козлов М. А.  **Группа** ИУ7-45Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель \_**Куров А. В.**\_\_\_\_\_** |  |

Москва.

2020 г.

**Цель работы:** изучение и программная реализация алгоритма Сазерленда-Ходжмена отсечения многоугольников.

**Техническое 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.

**Теоретический материал.**

Если определять многоугольник, как часть плоскости, ограниченной замкнутой ломанной линией, то результат полученный при отсечении рёбер многоугольника, алгоритмом отсечения отрезков, необходимо дополнить участками рёбер отсекателя. Для этих целей был разработан алгоритм Сазерленда-Ходжмена. Он основан на последовательном отсечении многоугольника границами отсекателя.

Для определения видимости точки можно воспользоваться

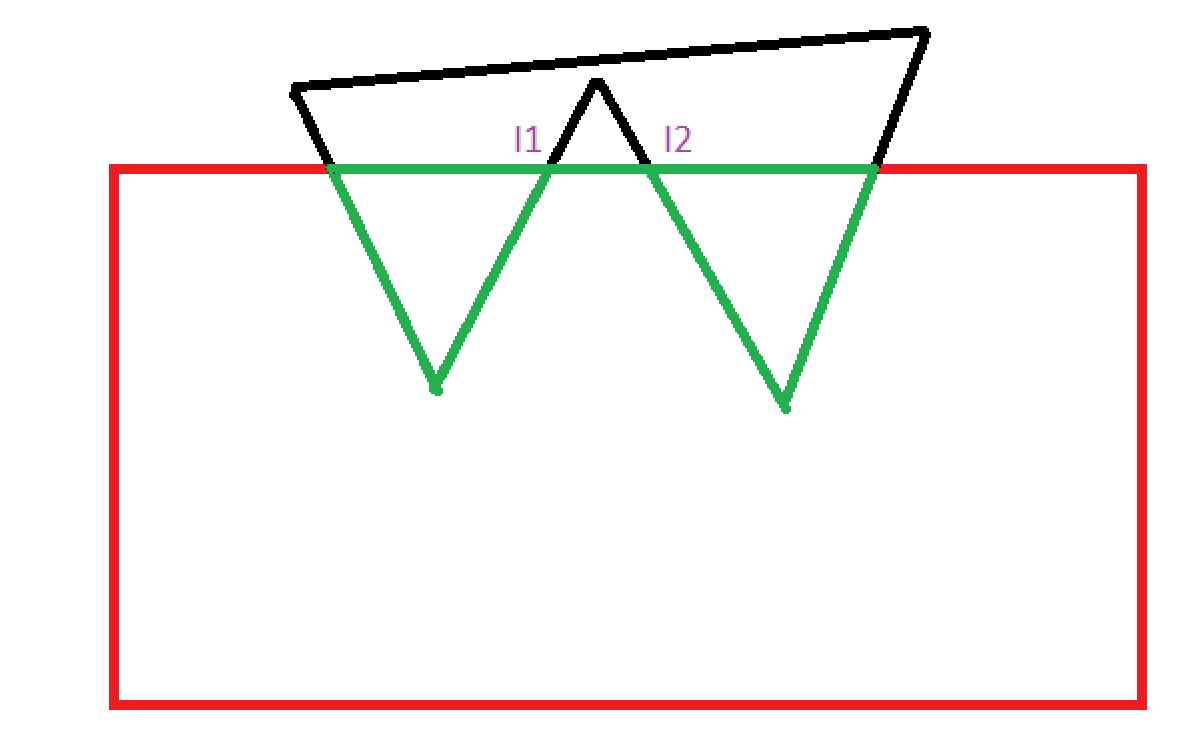
1. Скалярным произведением вектора внутренней нормали на вектор, соединяющий ребро отсекателя с исследуемой точкой.
2. Пробной функцией – уравнение прямой, проходящей через ребро отсекателя Ax+By + C, знак пробной функции сравнивается со знаком пробной функции, положение которой известно.

Для нахождения точки пересечения, необходимом убедиться в её существовании – видимость концов отрезка различная.

Возможны следующие комбинации расположения ребра многоугольника и внутренней области:

1. Обе вершины невидимы, в результат ничего не заносится.
2. Начальная вершина невидима, конечная видима, необходимо найти точку пересечения и занести в результат точку пересечения и конечную точку.
3. Обе вершины видимы, занести обе вершины в результат, но с учётом того, что начальная вершина текущего ребра является конечной вершиной предыдущего ребра, она будет занесена на предыдущем шаге.
4. Начальная вершина видима, конечная невидима. Начальная вершина текущего ребра является конечной вершиной предыдущего ребра, она будет занесена на предыдущем шаге. Найти точку пересечения и занести её в результат.

Данный алгоритм имеет недостаток – возможны ложные рёбра.

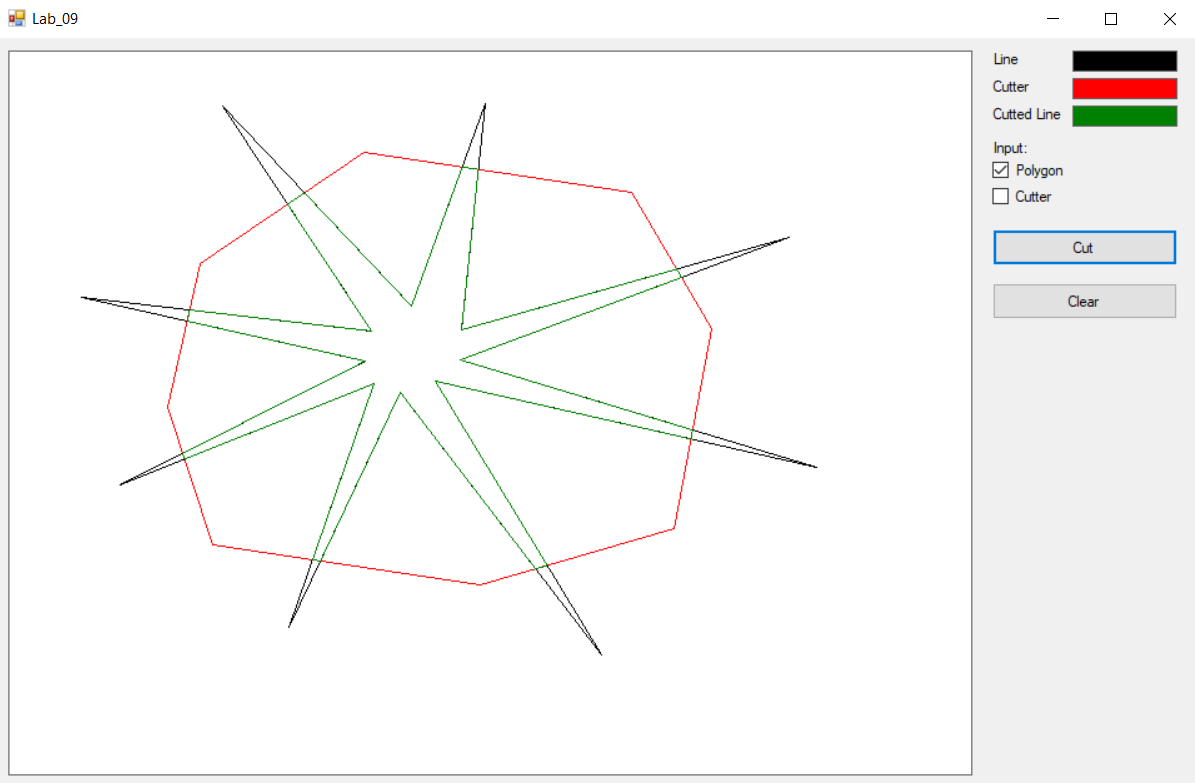


I1I2 – ложное ребро.

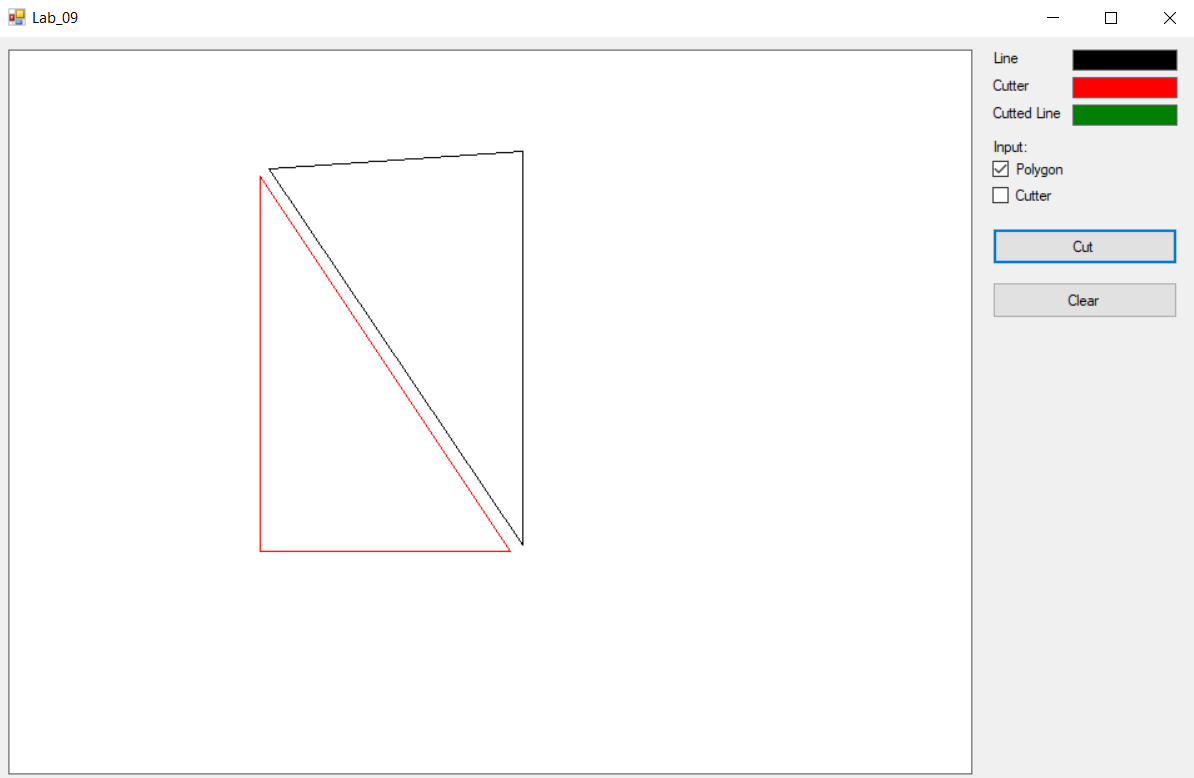
При работе с массивом вершин, и последовательным обходом, ложным будет ребро, которое обходится два раза.

**Примеры работы программы.**

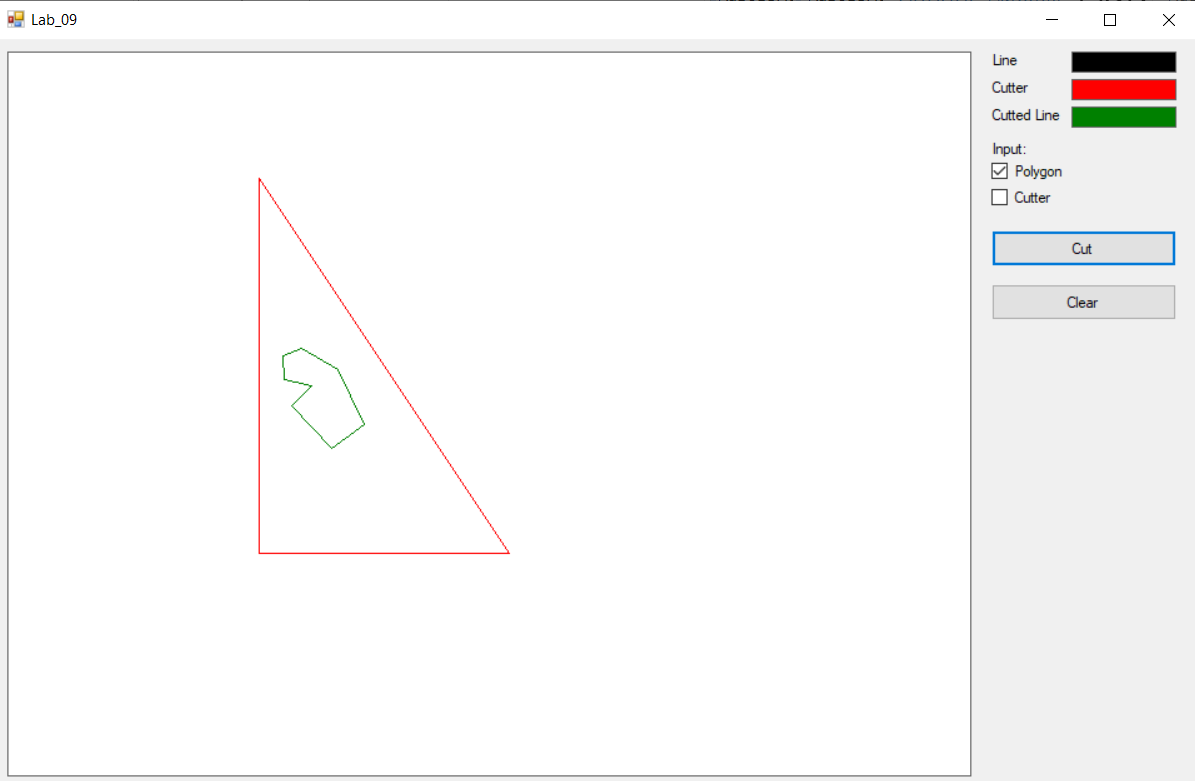
Ввод горизонтальных и вертикальных рёбер осуществляется при зажатии клавиши SHIFT.



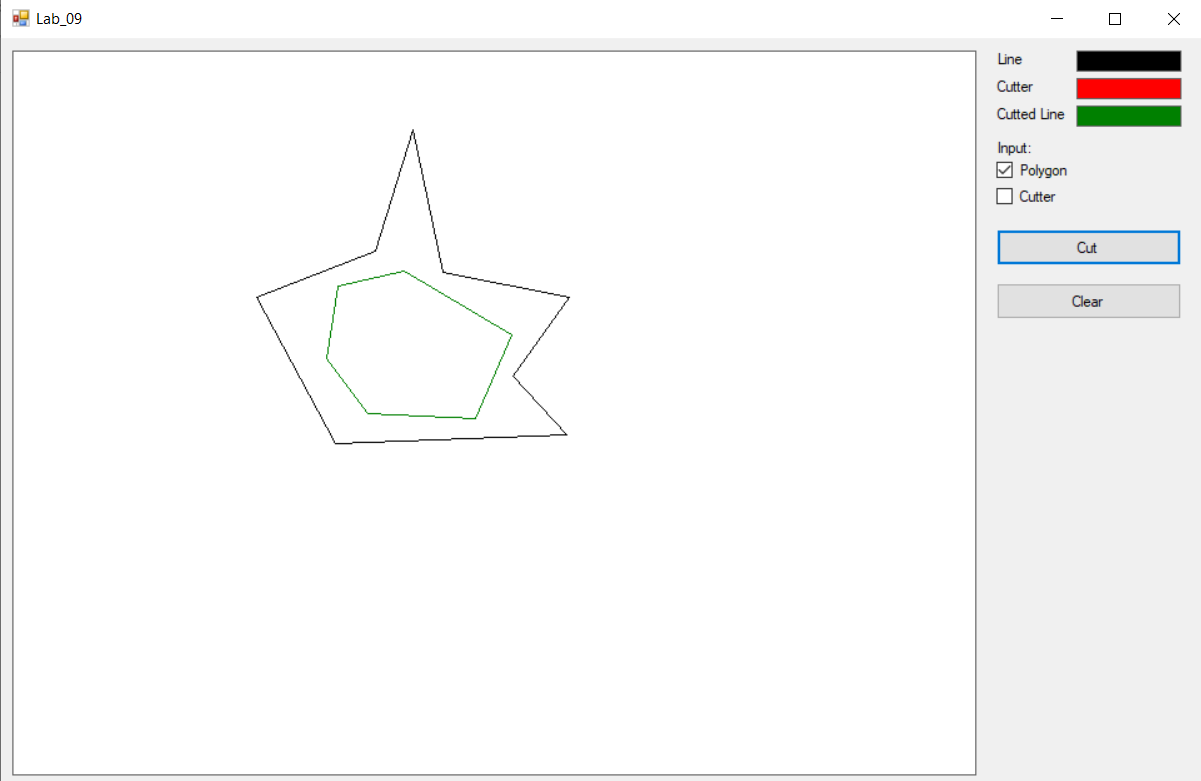
Многоугольник вне отсекателя



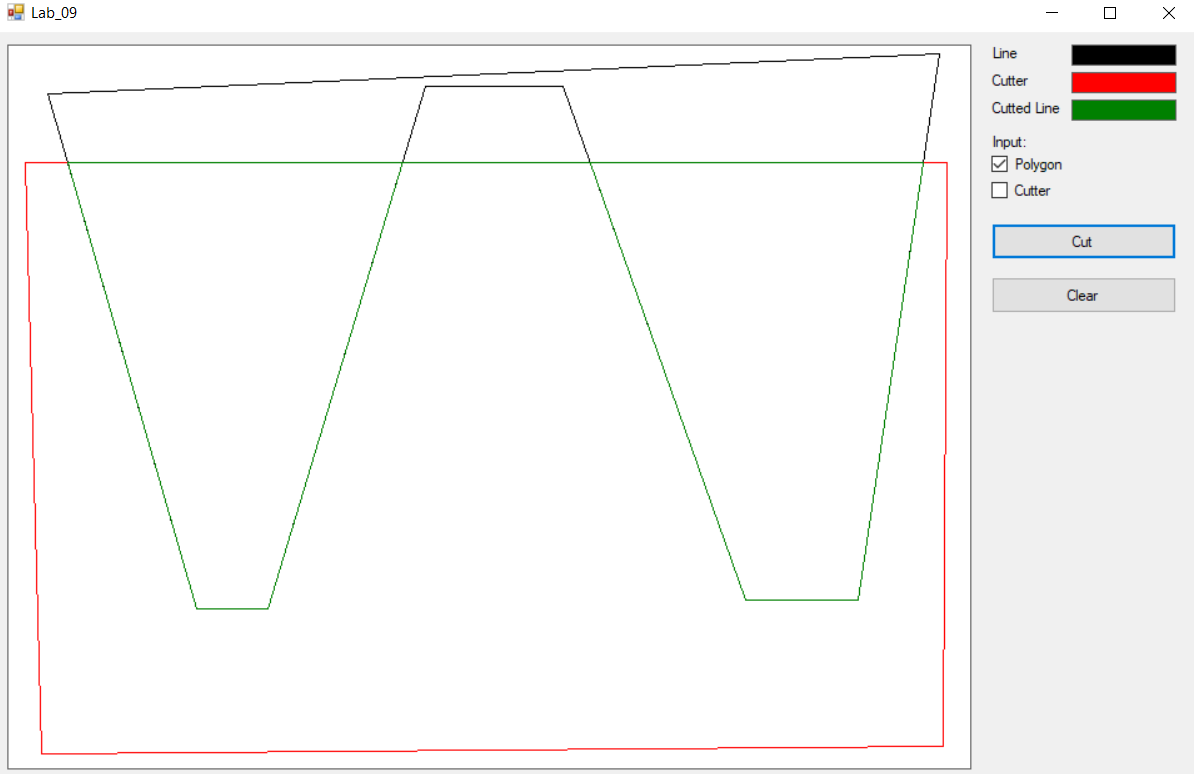
Многоугольник целиком в отсекателе



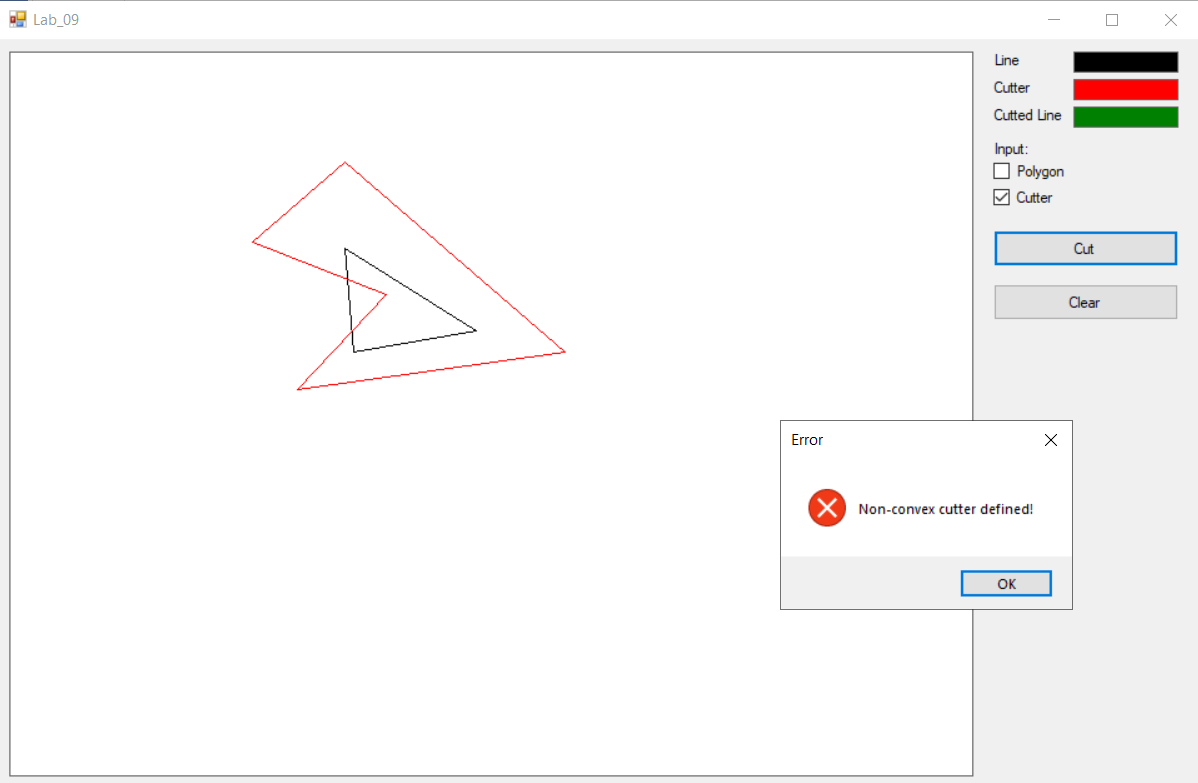
Отсекатель целиком в многоугольнике



«Ложные» рёбра



Введён невыпуклый многоугольник.



**Исходный код.**

private bool IsVisible(PointF point, int i)

{

var W = new Vector(Points[i], point);

var WScalar = Vector.Scalar(W, Normals[i]);

return WScalar > 0;

}

private PointF? GetPointIntersection(PointF V0, PointF V1, int i)

{

// директриса текущего отрезка

Vector D = new Vector(V0, V1);

var DScalar = Vector.Scalar(D, Normals[i]);

if (DScalar != 0) // Провека, что не точка и не парралельно

{

var W = new Vector(Points[i], V0);

var WScalar = Vector.Scalar(W, Normals[i]);

float t = -WScalar / DScalar;

if (t >= 0 && t <= 1)

{

PointF I = new PointF();

I.X = V0.X + D.X \* t;

I.Y = V0.Y + D.Y \* t;

return I;

}

}

return null;

}

public List<PointF> Cut(List<Point> polygon)

{

List<PointF> cuttedPolygon = new List<PointF>();

foreach (var points in polygon)

cuttedPolygon.Add(points);

cuttedPolygon.Add(cuttedPolygon.First());

List<PointF> tempPolygon = new List<PointF>();

PointF? I;

// по всем сторонам отсекателя

for (int i = 0; i < Points.Count(); i++)

{

// по всем сторонам многоугольника

for (int j = 1; j < cuttedPolygon.Count(); j++)

{

I = GetPointIntersection(cuttedPolygon[j - 1], cuttedPolygon[j], i);

if (I.HasValue)

tempPolygon.Add(I.Value);

if (IsVisible(cuttedPolygon[j], i))

tempPolygon.Add(cuttedPolygon[j]);

}

if (tempPolygon.Count() == 0)

return null;

I = GetPointIntersection(cuttedPolygon.First(), cuttedPolygon.Last(), i);

if (I.HasValue)

tempPolygon.Add(I.Value);

tempPolygon.Add(tempPolygon.First());

cuttedPolygon.Clear();

cuttedPolygon.AddRange(tempPolygon);

tempPolygon.Clear();

}

return cuttedPolygon;

}

// Получить направление обхода

// 1 = против часовой

// 0 = мн-к невыпуклый

// -1 = по часовой стрелке

public static int GetRoundDirection(List<Point> points)

{

if (points.Count < 3)

return 0;

Vector A = new Vector(points.Last(), points.First());

int sign = 0;

for (int i = 0; i < points.Count; i++)

{

var B = new Vector(points[i], points[(i + 1) % points.Count]);

int res = Math.Sign(Vector.Multiplication(A, B));

if (res != 0)

{

if (sign == 0)

sign = res;

else if (sign != res)

return 0;

}

A = B;

}

return sign;

}

public static Vector[] GetNormals(List<Point> points, int direction)

{

var normals = new Vector[points.Count];

Vector N;

float temp;

for (int i = 0; i < points.Count - 1; i++)

{

N = new Vector(points[i], points[i + 1]);

// Nx = -Vy; Ny = Vx

// (N, V) = Nx\*Vx + Ny\*Vy = -Vy\*Vx + Vx\*Vy = 0

temp = N.X;

N.X = -N.Y;

N.Y = temp;

if (direction == -1)

N.Negative();

normals[i] = N;

}

N = new Vector(points.Last(), points.First());

temp = N.X;

N.X = -N.Y;

N.Y = temp;

if (direction == -1)

N.Negative();

normals[points.Count - 1] = N;

return normals;

}