|  |  |
| --- | --- |
| 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 № 8**

|  |  |
| --- | --- |
| **Тема**  Реализация алгоритма отсечения отрезка  произвольным выпуклым отсекателем.  (Алгоритм Кируса-Бека)  **Студент** Козлов М. А.  **Группа** ИУ7-45Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель \_**Куров А. В.**\_\_\_\_\_** |  |

Москва.

2020 г.

**Цель работы:** изучение и программная реализация алгоритма Кируса-Бека отсечения отрезка.

**Техническое 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.

**Теоретический материал.**

Во многих случаях отсекатель не является координатно-ориентированным прямоугольником. При таком отсечении наиболее удобно использовать параметрическое представление линии, так как такое представление не зависит от системы координат.

Для выполнения отсечения в параметрическом представлении необходимо иметь метод определения местоположения относительно окна точки, принадлежащей отрезку. Для этих целей в алгоритме Кируса-Бека, реализующем отсечение произвольным выпуклым многоугольником, используется вектор внутренней нормали к ребру окна.

Внутренняя нормаль N в произвольной точке A принадлежащей стороне окна, удовлетворяет условию

Где B – любая точка на границе отсекателя.

Алгоритм Кируса-Бека.

Отсекатель является выпуклым многоугольником, поэтому может быть не более двух точек пересечения отрезка с отсекателем. Необходимо найти два параметра t: t\_down – соответствующей начальной и t\_up – соответствующей конечной точке видимой части отрезка.

Параметрическое представление отрезка P1P2

Из этого следует уравнение для нахождения точки пересечения отрезка с границей отсекателя.

Обозначим

если отрезок выродился в точку или параллелен текущей границе отсекателя, то необходимо проанализировать его расположение.

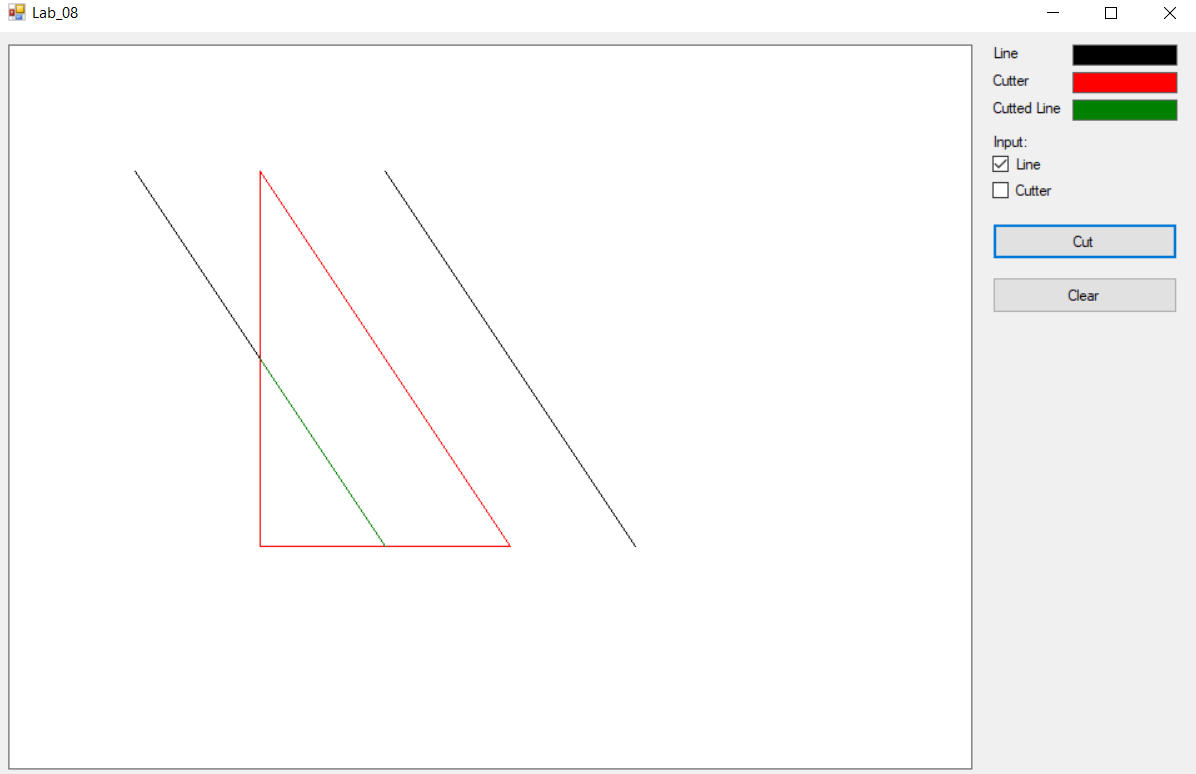
Но если отрезок не видим относительно одной стороны окна, то он является невидим.

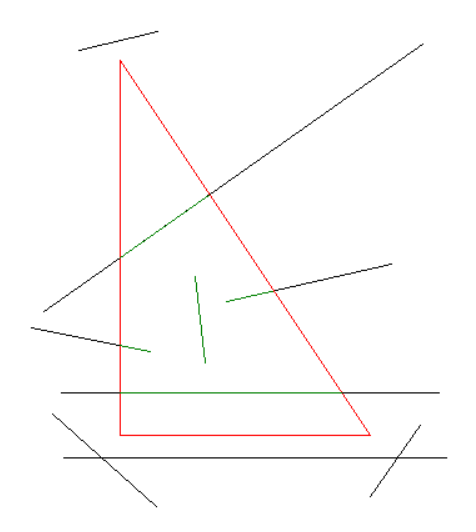
Если

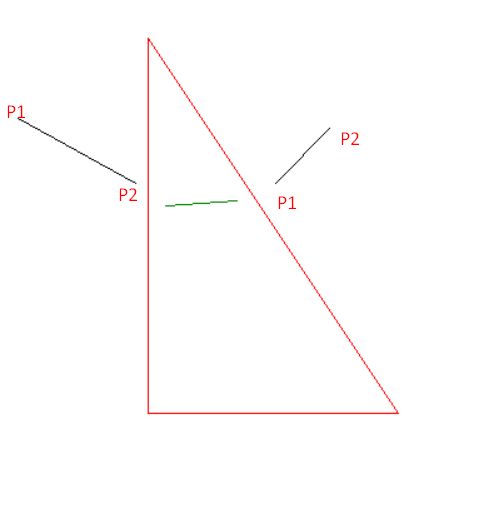
Если

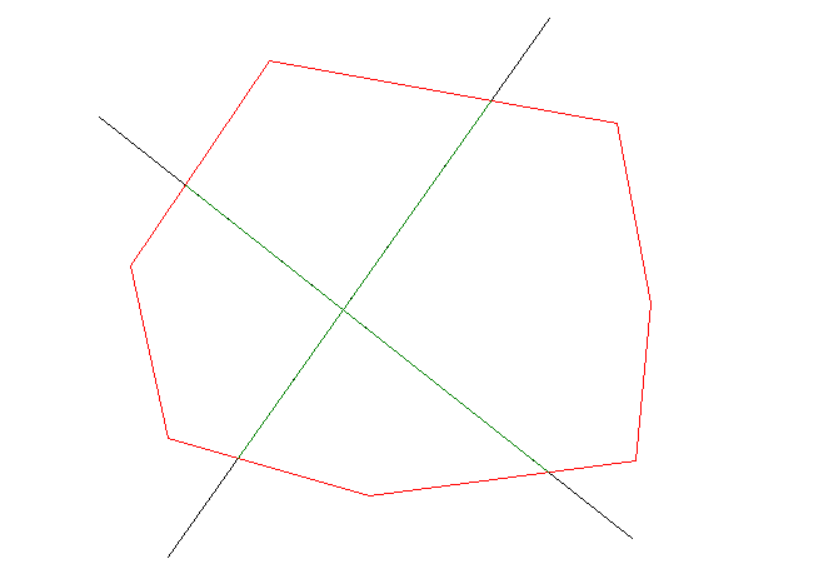
**Примеры работы программы.**

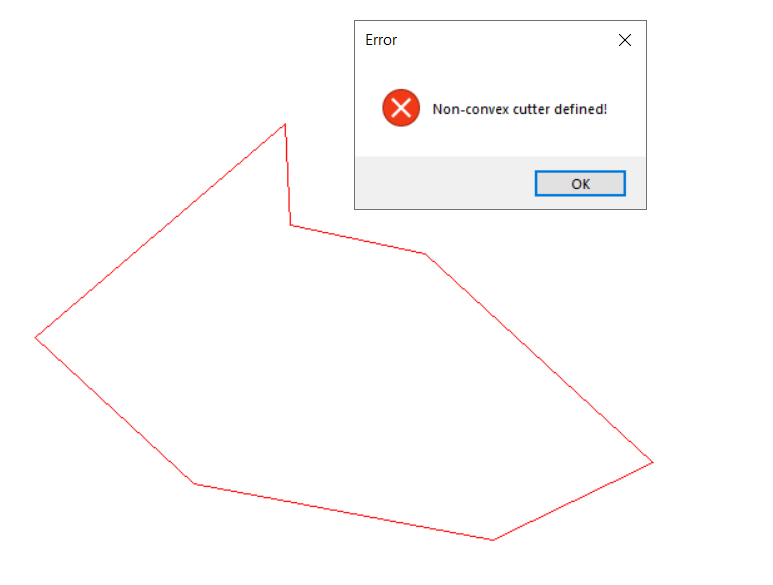
Ввод горизонтальных и вертикальных рёбер осуществляется при зажатии клавиши SHIFT.











**Исходный код.**

// Получить направление обхода

// 1 = против часовой

// 0 = мн-к невыпуклый

// -1 = по часовой стрелке

private int GetRoundDirection()

{

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;

}

private Vector[] GetNormals(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;

}

// если мн-к не выпуклый будет брошено исключение NoConvexCutterException

private void SetPoints(List<Point> points)

{

this.points = points;

int direction = GetRoundDirection();

if (direction == 0)

throw new NoConvexCutterException();

Normals = GetNormals(direction);

}

public PointF[] Cut(PointF A, PointF B)

{

float tDown = 0, tUp = 1;

float tTemp;

Vector D = new Vector(A, B);

Vector W;

float DScalar, WScalar;

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

{

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

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

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

if (DScalar == 0) // отрезок выродился в точку или отрезок параллелен

{

if (WScalar < 0)

return null;

// иначе точка видима относительно текущей границы

}

else

{

tTemp = -WScalar / DScalar;

if (DScalar > 0) // поиск нижнего предела

{

if (tTemp > 1)

return null;

tDown = Math.Max(tDown, tTemp);

}

else // поиск верхнего предела

{

if (tTemp < 0)

return null;

tUp = Math.Min(tUp, tTemp);

}

}

if (tDown > tUp)

return null;

}

var newA = new PointF(A.X + D.X \* tDown, A.Y + D.Y \* tDown);

var newB = new PointF(A.X + D.X \* tUp, A.Y + D.Y \* tUp);

return new PointF[2] { newA, newB };

}