|  |  |
| --- | --- |
| **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**

**по дисциплине «Компьютерная графика»**

|  |  |
| --- | --- |
| **Тема: Реализация алгоритма отсечения отрезка произвольным выпуклым отсекателем (Алгоритм Кируса-Бека).**  **Студент:** Пересторонин Павел  **Группа:** ИУ7-43Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель:** Куров А. В. |  |

Москва.

2020 г.

Цель работы: изучение и программная реализация алгоритма отсечения отрезка произвольным выпуклым отсекателем.

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**

В первую очередь стоит отметить форму, в которой описываются отрезки в данном алгоритме (параметрическая форма задания отрезка):

P(t) = P1 + (P2- P1) \* t; где t - параметр (0 <= t <= 1).

Это векторное уравнение, которое в двумерной графике можно свести к двум одномерным параметрическим уравнениям след. вида:

Px(t) = P1.x + (P2.x- P1.x) \* t

Py(t) = P1.y+ (P2.y- P1.y) \* t

параметр t так же принадлежит [0, 1]

При нахождении пересечений с гранями многоугольника находится значение параметра t, при котором происходит пересечение, и если значение параметра лежит вне допустимых границ, то пересечение происходит с продолжением отрезка (получается, что его нет). Такие пересечения отвергаются.

В данном алгоритме используется понятие "вектора внутренней нормали" nв. Вектор внутренней нормали - вектор, перпендикулярный грани многоугольника и направлен внутрь этого многоугольника. Этот факт проверяется аналитическим выражением след. вида: nв \* (B - A) >= 0, где А - точка грани, из которой исходит данная нормаль, а В любая другая точка нормали, однако в качестве точки В следует брать точку, не принадлежащую рассматриваемой грани, иначе что для внутренней, что для внешней нормали скалярное произведение будет равно 0.

Так же в алгоритме представляет интерес следующий вектор:

[P(t) - fi], где fi - произвольная точка рассматриваемой грани (не совпадающая с точкой пересечения рассматриваемых грани и отрезка)

Проанализируем скалярное произведение этого вектора и вектора нормали к рассматриваемой грани:

nв[P(t) - fi] > 0 - вектор направлен внутрь области многоугольника (так как из скалярного произведения следует, что модуль угла между этим вектором и вектором внутренней нормали < 90).

nв[P(t) - fi] = 0 - вектор перпендикулярен нормали (то есть параллелен грани)

nв[P(t) - fi] < 0 - направлен вне области многоугольника (противоположность первой ситуации)

Очевидно, что при различных параметрах t, рассматриваемая точка P(t) может находиться как внутри, так и вне области многоугольника относительно рассматриваемой грани, однако в данном случае нас больше интересует (и понадобится) тот факт, что мы можем определить "входит" или "выходит" отрезок из многоугольника при пересечении определенной грани. Это выясняется так: если отрезок пересекает грань и его начало было внутри многоугольника относительно этой грани (анализ скалярного произведение рассмотренного выше при P(t), t = 0), то получается что при пересекании он выйдет за грань (получается, он выходит за пределы многоугольника).

Теперь рассмотрим еще лучше ситуацию: nв[P(t) - fi] = 0. Мы уже сказали, что она соответствует ситуации, когда вектор, состоящий из точки отрезка (P(t)) и точки fi грани параллелен этой грани. Очевидно, что если стоит задача построить параллельную прямую к некоторой прямой L через точку А, находящейся на этой прямой L, то решение этой задачи - прямая, совпадающая с прямой L. Из этого следует, что nв[P(t) - fi] = 0 выполняется тогда, когда вектор лежит на одной прямой с гранью [P(t) - fi], а P(t) для некоторого t - точка пересечения грани и отрезка.

Подставим параметрическую форму уравнения в данное выражение:

nв[P1 + (P2- P1)t - fi] = 0

Преобразуем: nв[P1 - fi] + nв[P2- P1]t = 0 (\*)

В данном уравнении вектор [P2- P1] - вектор, определяющий направление (ориентацию) отрезка, а вектор [P1 - fi] рассматривался выше в общем виде, но в данном случае - вектор, соединящий некоторую точку грани с началом отрезка (по его скалярному произведению с внутренней нормалью можно судить о положении отрезка относительно внутренней нормали)

Введем обозначения (почти такие же обозначения указаны в коде):

Wi = nв[P1 - fi]

Dск = nв[P2- P1]

Выразим t из уравнения (\*), используя обозначения выше:

t = -Wi / Dск

Данное выражение нельзя рассматривать при Dск = 0. Рассмотрим случаи, когда Dск = 0:

1. вектор ориентации отрезка вырожден (нулевой). Такой случай нас не очень интересует.

2. Dcк перпендикулярен nв. Данный случай инетресен: получается, что отрезок парллелен грани. Здесь может быть 2 случая: отрезок лежит вне многоугольника относительно грани - тогда можно однозначно сделать вывод о том, что он не видим и закончить операцию отсечения данного отрезка. Отрезок лежит внутри многоугольника относительно грани - тогда следует перейти на следующую итерацию и продолжить операцию отсечения. Определить "вне" или "внутри" довольно-таки легко с помощью вектора Wi. Этот вектор начинается в некоторой точке рассматриваемой грани многоугольника и заканчивается в некоторой точке отрезка, таким образом можно сказать, что он направлен "от грани к отрезку". Внутренняя нормаль начинается в некоторой точке грани и может быть направлена к отрезку и в противоположную сторону. Проверяется это, например, вот таким скалярным произведением: Wi = nв[P1 - fi] (это тот самый коэффициент, который мы уже выразили и посчитали выше). Если скалярное произведение < 0, то угол между вектором нормали и вектором, направленным к отрезку > 90, и вектор лежит вне фигуры, иначе - внутри.

Теперь когда мы научились находить значения t всех пересечений с гранями, а так же научились определять "входит" отрезок в многоугольник или нет, осталось решить только одну проблему: какие конкретно значения t выбрать в качестве начального и конечного. Очевидно, что если отрезок виден, то он виден относительно всех граней. Из этого можно сделать вывод, что если отрезок входит в многоугольник относительно какой-то грани, то относительно других граней он должен был уже войти (то есть надо выбирать в качестве начала последний вход), и при этом не должен был выйти (то есть получается последний вход должен быть раньше всех выходов). С выходами ситуация та же: выйдя за первую грань, отрезок перестанет быть виден относительно нее и, следовательно, будет на оставшемся промежутке (из этого вывод - в качестве конца следует брать первых выход). Далее, убедившись, что параметр tвх , соответствующий последнему входу, меньше, чем параметр tвых, соответствующий первому выходу, чертим видимую часть отрезка (если условие не выполняется - не чертим).

**Исходный код программы.**

# Функция получения свободного вектора по 2 точкам (p1 - начало вектора,

# p2 - конец вектора)

**def** **get\_vect**(p1, p2):

**return** [p2[**0**] - p1[**0**], p2[**1**] - p1[**1**]]

# Функция расчета векторного произведения 2 векторов

**def** **vect\_mul**(v1, v2):

**return** v1[**0**] \* v2[**1**] - v1[**1**] \* v2[**0**]

# Функция расчета скалярного произведения 2 векторов

**def** **scalar\_mul**(v1, v2):

**return** v1[**0**] \* v2[**0**] + v1[**1**] \* v2[**1**]

# Функция проверки многоугольника (на выпуклость)

**def** **check\_polygon**():

# Не существует многоугольника, у которого меньше 3 вершин

**if** len(verteces\_list) < **3**:

**return** False

# Знаки всех векторых произведений должны быть одинаковыми:

# запомним знак первого векторного произведения

sign = **1** **if** vect\_mul(get\_vect(verteces\_list[**1**], verteces\_list[**2**]),

get\_vect(verteces\_list[**0**], verteces\_list[**1**])) > **0** **else** -**1**

# В цикле проверяем совпадения знаков векторных произведений

# всех пар соседних ребер со знаком первого

# векторного произведения

**for** i **in** range(**3**, len(verteces\_list)):

**if** sign \* vect\_mul(get\_vect(verteces\_list[i - **1**], verteces\_list[i]),

get\_vect(verteces\_list[i - **2**], verteces\_list[i - **1**])) < **0**:

# Возвращаем False при несовпадении знаков: прямоугольник невыпуклый

**return** False

**if** sign < **0**:

# если знак отрицательный, значит обход был по часовой стрелке.

# В дальнейших шагах мне нужно работать с обходом против часовой

# стрелке (при выяснении направления нормали, например), поэтому

# я переворчиваю список вершин (ну и соответственно при проходе

#в обратном порядке, будет обход против часовой стрелки)

verteces\_list.reverse()

**return** True

# Функция получения нормали для грани многоугольника между вершинами p1, p2

# cp = check point, следующая вершина в многоугольнике: нужна для проверки,

# направлена ли нормаль внутрь многоугольника или же из многоугольника

**def** **get\_normal**(p1, p2, cp):

vect = get\_vect(p1, p2)

# Если ищется нормаль к вертикальному вектору - то это нормаль [1, 0],

# иначе вектор нормали находится из условия равенства 0 скалярного произведения

# исходного вектора и искомого вектора нормали

norm = [**1**, **0**] **if** vect[**0**] == **0** **else** [-vect[**1**] / vect[**0**], **1**]

# Если скалярное произведение найденного вектора нормали и вектора, совпадающего

# со следующей стороной многоугольника меньше нуля - нормаль направлена из

# многоугольника, берем обратный вектор

**if** scalar\_mul(get\_vect(p2, cp), norm) < **0**:

**for** i **in** range(len(norm)):

norm[i] = -norm[i]

**return** norm

# Функция, составляющая список векторов нормалей ко всем сторонам многоугольника

**def** **get\_normals\_list**(verteces):

length = len(verteces\_list)

normal\_list = list()

**for** i **in** range(length):

normal\_list.append(get\_normal(verteces[i], verteces[(i + **1**) % length], verteces[(i + **2**) % length]))

**return** normal\_list

# Функция, отсекающая отрезок и рисующая полученный новый отрезок

**def** **cut**(section, verteces\_list, normals\_list):

# список параметров t рассматриваемого отрезка, при которых он пересекает ребро и

# входит в многоугольник (не может быть меньше 0)

t\_start\_list = [**0**]

# аналогично списку выше, но выходит (не может быть больше 1)

t\_end\_list = [**1**]

# Вектор направления отрезка ( вектор = [P2 - P1] )

d = get\_vect(section[**0**], section[**1**])

# Цикл по всем граням многоугольника и поиск параметров t точек пересечения

# (а также получение информации о "входе-выходе", см. ниже)

**for** i **in** range(len(verteces\_list)):

# В общем случае в качестве "точки многоугольника" берется начальная вершина

# этой грани, однако если она совпадает с точкой начала отрезка, берется

# конечная точка грани

**if** verteces\_list[i] != section[**0**]:

wi = get\_vect(verteces\_list[i], section[**0**])

**else**:

wi = get\_vect(verteces\_list[(i + **1**) % len(verteces\_list)], section[**0**])

# Скалярное произведение вектора нормали и вектора ориентации (если = 0, то вектор

# параллелен стороне многоугольника)

Dck = scalar\_mul(d, normals\_list[i])

# Скалярное произведение вектора нормали и вектора от "точки многоугольника" до

# начала отрезка (если оно = 0, то начало отрезка лежит на рассматриваемой

# грани многоугольника)

Wck = scalar\_mul(wi, normals\_list[i])

# Если отрезок параллелен грани, и лежит вне многоугольника - выход

**if** Dck == **0**:

**if** scalar\_mul(wi, normals\_list[i]) < **0**:

**return**

**else**:

**continue**

# Параметр t, соответствующий точке пересечения рассматриваемого отрезка

# с очередной гранью

t = -Wck / Dck

# Если Dck > 0 - точка входа в многоугольник

**if** Dck > **0**:

t\_start\_list.append(t)

**else**:

t\_end\_list.append(t)

# Видимый отрезок находится между "последним" входом и "первым" выходом

t\_start = max(t\_start\_list)

t\_end = min(t\_end\_list)

# Если "входной" t < "выходной" t, то отрезок видимый - чертим его

**if** t\_start < t\_end:

p1 = [round(section[**0**][**0**] + d[**0**] \* t\_start), round(section[**0**][**1**] + d[**1**] \* t\_start)]

p2 = [round(section[**0**][**0**] + d[**0**] \* t\_end), round(section[**0**][**1**] + d[**1**] \* t\_end)]

draw\_section(\*p1, \*p2, res\_color)

**def** **solve**():

# Проверка многоугольника на выпуклость

**if** **not** check\_polygon():

mb.showerror("Невыпуклый многоугольник", "Для осуществления отсечения отрезка алгоритмом Кируса-Бека прямоугольник должен быть выпуклым")

**return**

# Получение нормалей для всех граней многоугольника

normals\_list = get\_normals\_list(verteces\_list)

# Отсечение всех отрезков из списка отрезков

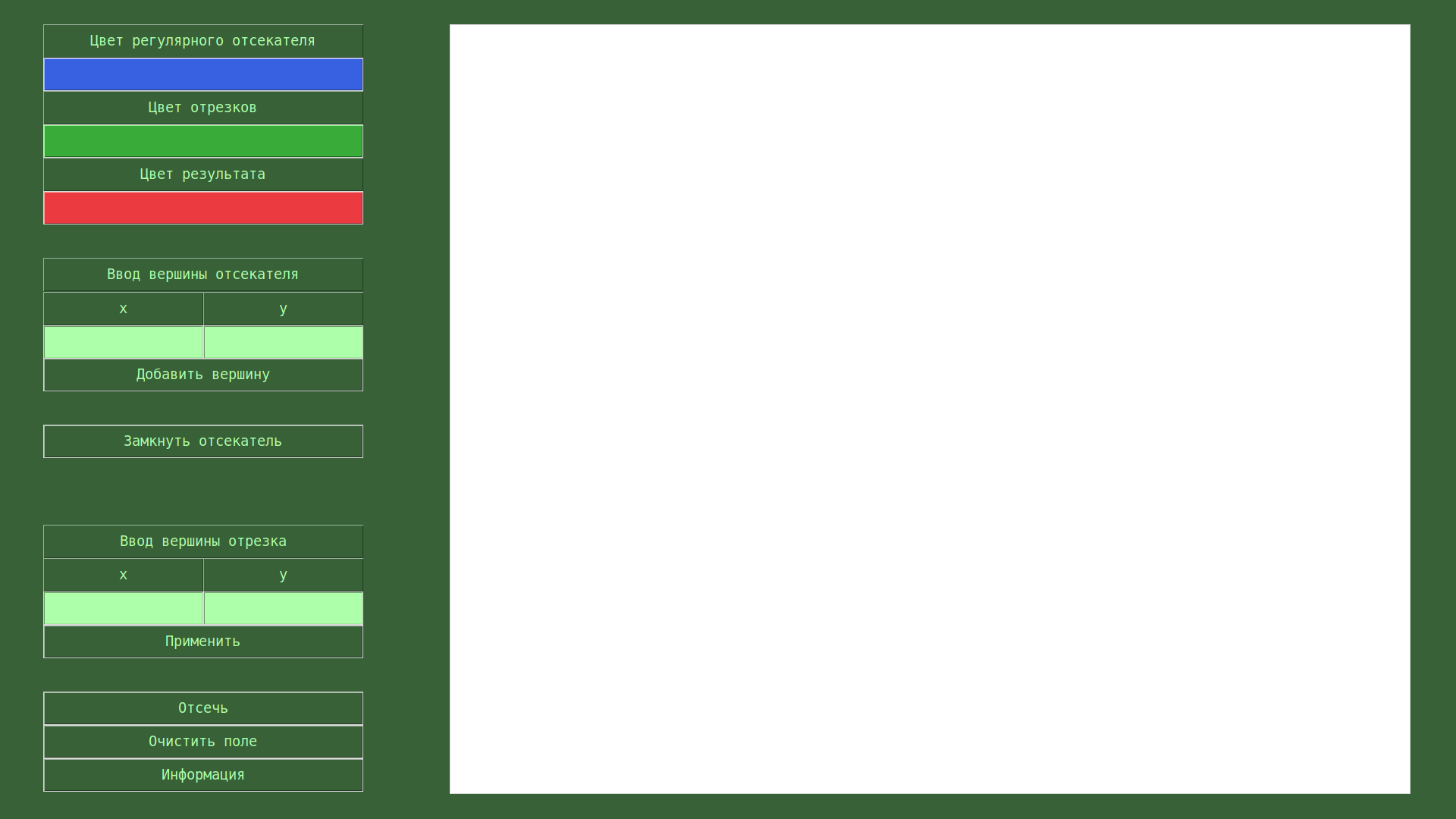
**for** section **in** sections:

cut(section, verteces\_list, normals\_list)

Конец кода.

**Интерфейс и примеры работы.**

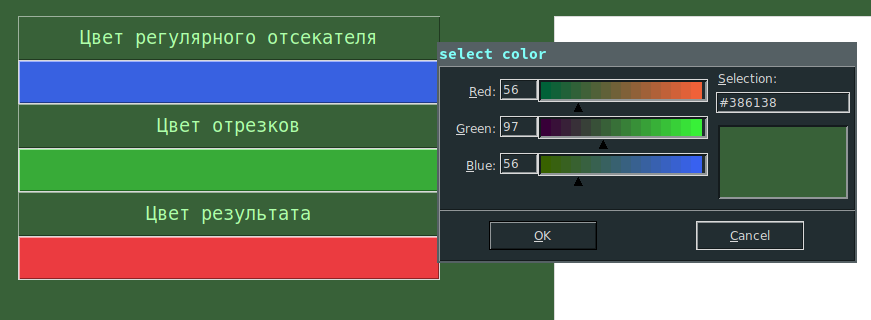
Интерфейс:



Интерфейс предусматривает 2 формата ввода: через поля (координаты) и через выбор мышью точки на плоскости.

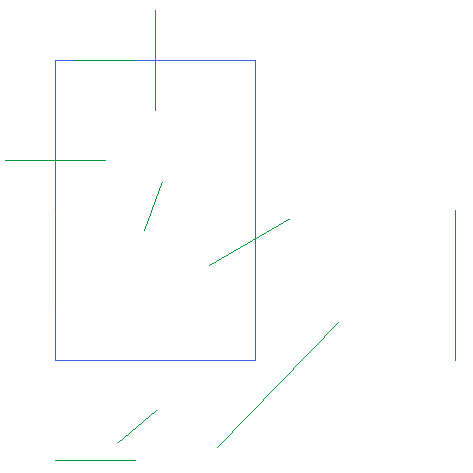
При вводе с помощью мышки границ отсекателя вершины вводятся нажатием правой кнопки мыши. Для замыкания следует нажать кнопку Enter.

Отрезки вводятся с помощью левой кнопки мыши. Первое нажатие – начало отрезка, второе – конец. Можно ввести начало координатами, а конец выбрать мышкой, и наоборот. Цвета могут быть любые, используется палитра:

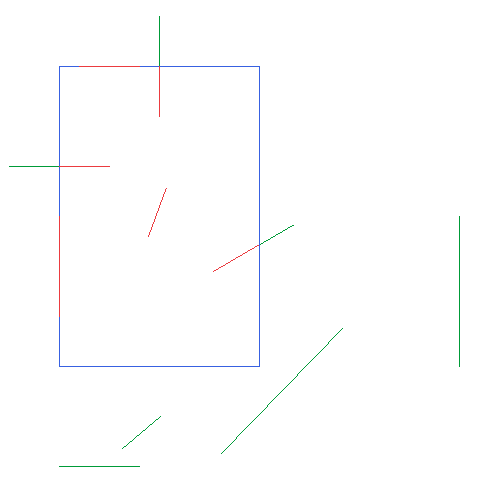


Примеры работы:

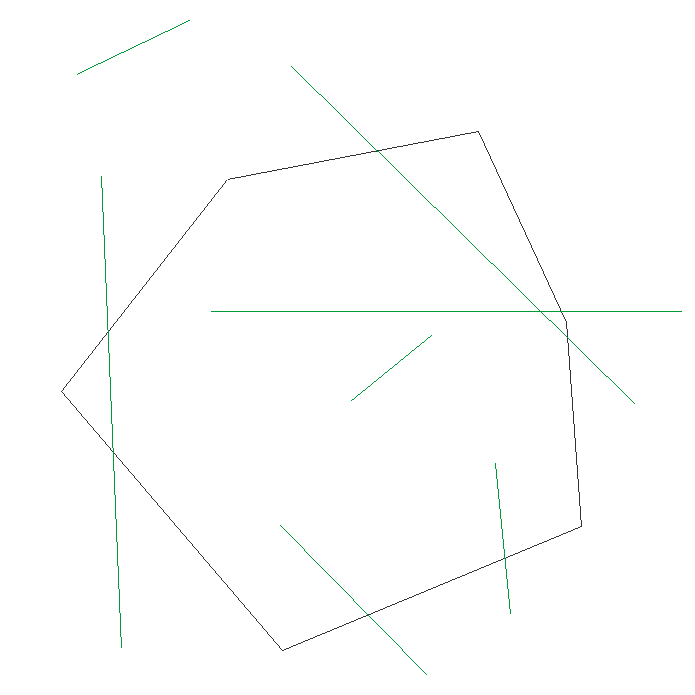
Прямоугольный отсекатель:

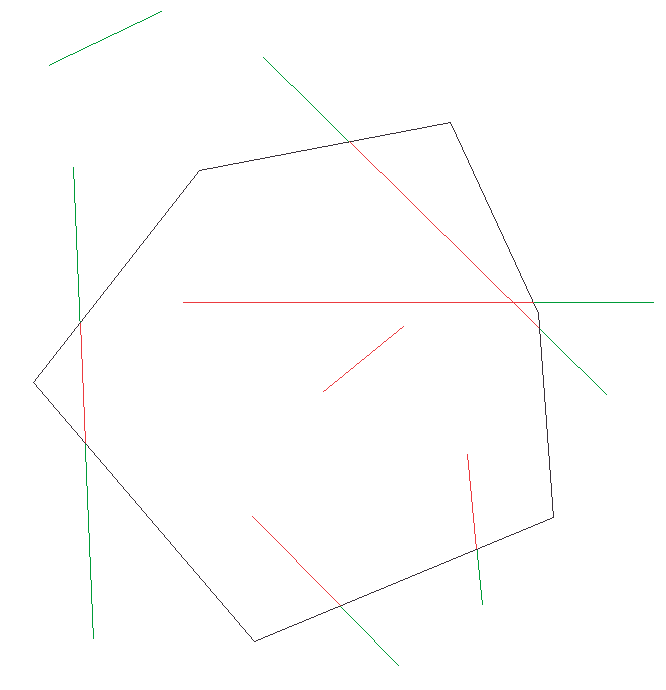
До:

После (обратите внимание на ребра, совпадающие с гранями окна):



Произвольный отсекатель:

До:



После: