|  |  |
| --- | --- |
| **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: Отсечение произвольного многоугольника выпуклым отсекателем (Алгоритм Сазерленда-Ходжмена)**  **Студент:** Пересторонин Павел  **Группа:** ИУ7-43Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель:** Куров А. В. |  |

Москва.

2020 г.

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

РЕЗУЛЬТАТ: должна быть разработана программа, позволяющая осуществлять ввод отсекателя, отсекаемого многоугольника и выполнять отсечение многоугольника по границам отсекателя.

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**

Алгоритм Сазерленда-Ходжмена позволяет выпуклым отсекателем отсечь многоугольник. Для начала стоит договориться о том, что мы будем называть многоугольником.

Есть 2 определения:

1. Многоугольник - замкнутая ломаная линия (для реализации данного задачи можно использовать алгоритм Кируса-Бека)

2. Плоский многоугольник - часть плоскости, ограниченная замкнутой плоской линией. Именно **этим определением** мы будем пользоваться в нашей программе (при отсечении результат может включать ребра отсекателя)

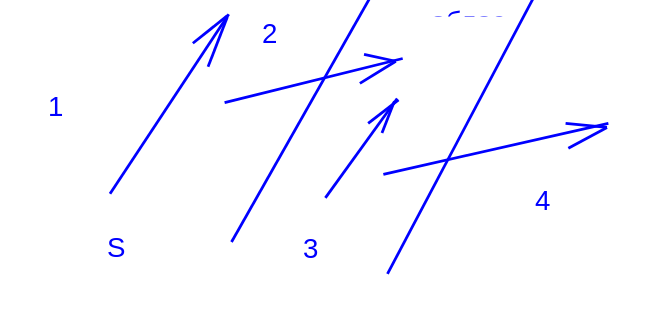
Идея алгоритма - решение задачи последовательно: многоугольник отсекается относительно каждой грани отсекателя по отдельности и при этом, результат, полученный на i-ом шаге служит исходным для i+1-ого отсечения.

Отсечение многоугольника относительно грани - не самая тривиальная задача. Во-первых требуется некоторая возможность определять принадлежность точки рассматриваемого многоугольника плоскости отсекателя относительно очередной грани. Эту задачу можно решить с помощью векторного произведения:

Из прошлой ЛР я взял алгоритм, который позволял рассматривать вершины многоугольника против часовой стрелки (то есть если они рассматриваются по часовой стрелке, он переворачивает список вершин). При таком рассмотрении выпуклого многоугольника внутренняя часть относительно грани находится с левой стороны. Используя этот факт, можно с помощью векторного произведения 2 векторов (вектор грани, при обходе против часовой стрелки, и вектора, проведенного из одной из вершин грани к проверяемой точке) установить положение точки: если векторное произведение в правовинтовой системе меньше нуля, то вершина лежит вне плоскости отсекателя, иначе - внутри (но только относительно рассматриваемой на данный момент грани). Так же можно было использовать скалярное произведение внутренней нормали и вектора, проведенного из произвольной точки рассматриваемой грани к проверяемой точке (если это произведение больше нуля, то точка лежит внутри отсекателя, иначе - снаружи).

Теперь, решив подзадачу определения принадлежности точки выпуклому отсекателю относительно некоторой грани, перейдем к самому рассмотрению алгоритма отсечения.

В процессе рассмотрения ребер отсекаемого многоугольника возможны следующие положения (пусть для определенности направление ребра от вершины P к вершине S; параллельные линии без стрелочек - область отсекателя):



Возможные ситации:

1. Ребро полностью вне рассматриваемой границы отсекателя. В таком случае не добавляем никакую вершину.

2. Ребро пересекает границу отсекателя, причем вершина S - внутри, а Р - нет. В таком случае нужно добавить в список вершин искомого отсеченного многоугольника вершину S, а так же точку пересечения текущего ребра с рассматириваемой гранью.

3. Ребро полностью находится внутри отсекателя. В таком случае следует добавить обе вершины в список вершин искомого многоугольника.

4. Ребро пересекает границу отсекателя, причем вершина S - вне отсекателя, а Р - внутри. В таком случае нужно добавить в список вершин искомого отсеченного многоугольника вершину Р, а так же точку пересечения текущего ребра с рассматириваемой гранью.

Пересечение в данном алгоритме я находил так же, как в алгоритме Кируса-Бека: у меня имеется вектор внутренней нормали, а так же есть отрезок (заданный двумя точками, очередное ребро отсекаемого многоугольника) и грань (тоже задана 2 точками, вершинами отсекателя), пересечение которых надо найти.

Находим d - вектор ориентации ребра многоугольника, wi - вектор, соединяющий точку грани (первую для определенности), а также скалярные произведенения Wck = (n, wi), Dck = (d, n), где n - вектор внутренней нормали.

Далее найдем параметр t = -Wck / Dck (он точно будет в [0, 1], потому что точки ребра многоугольника лежат по разные стороны грани; про параметр t см. отчет №8) и выразим Х и У точки пересечения:

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

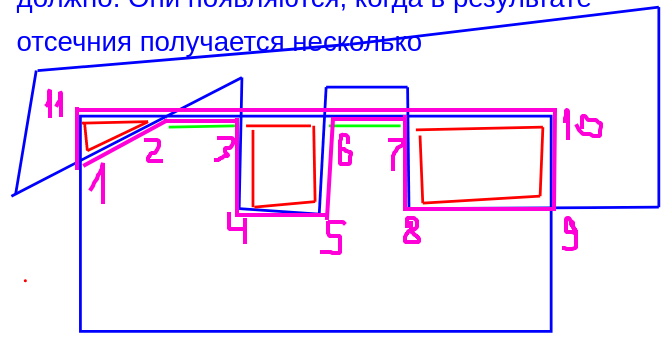
Y = P1.y + (P2.y - P1.y) \* t

При обходе и отсечении многоугольника требуется избежать ситуации, когда какая-либо точка дважды добавляется в список вершин получающегося многоугольника, поэтому я добавлял только конечную точку рассматриваемого вектора, если вектор целиком в области отсекателя, только точку пересечения, если вектор выходит из области отсекателя, и 2 точки, если вектор входит в область отсекателя.

Так же на лекции была продемонстрирована нетривиальная ситуация, в которой алгоритм дает нехороший результат (ложные ребра):



На рисунке ложные ребра обозначены зеленым цветом. Я решаю данную задачу следующим образом:



Из рисунка видно, что через ложные ребра полученный многоугольник проходит 2 раза. В своем алгоритме я пользуюсь этим фактом. В первую очередь я разбиваю отрезки, проходящие через вершины многоугольника (такие как 10-11, который помимо 10 и 11 вершины, через которые он строится, проходит через вершины 7, 6, 3, 2), на несколько (то есть например отрезок 10-11, я разбиваю на 10-7, 7-6, 6-3, 3-2, 2-11) и после этого проверяю, сколько раз у меня встречается каждый элементарный отрезок. Если четное количество раз - он соответствует ложному ребру, иначе - ребру результата.

**Исходный код программы.**

# Функция, которая удаляет ложные ребра

**def** **make\_uniq**(sections):

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

section.sort()

**return** list(filter(**lambda** x: (sections.count(x) % **2**) == **1**, sections))

# Функция проверки, принадлежит ли точка point отрезку section

**def** **point\_in\_section**(point, section):

**if** abs(vect\_mul(get\_vect(point, section[**0**]), get\_vect(\*section))) <= **1e-6**:

**if** (section[**0**] < point < section[**1**] **or** section[**1**] < point < section[**0**]):

**return** True

**return** False

# Функция получения "элементарных" отрезков многоугольника

# (см. мой алгоритм)

**def** **get\_sections**(section, rest\_points):

points\_list = [section[**0**], section[**1**]]

**for** p **in** rest\_points:

**if** point\_in\_section(p, section):

points\_list.append(p)

points\_list.sort()

sections\_list = list()

**for** i **in** range(len(points\_list) - **1**):

sections\_list.append([points\_list[i], points\_list[i + **1**]])

**return** sections\_list

# Функция выброса ложных ребер из результирующего многоугольника

**def** **get\_uniq\_sections**(figure):

all\_sections = list()

rest\_points = figure[**2**:]

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

cur\_section = [figure[i], figure[(i + **1**) % len(figure)]]

all\_sections.extend(get\_sections(cur\_section, rest\_points))

rest\_points.pop(**0**)

rest\_points.append(figure[i])

**return** make\_uniq(all\_sections)

# Функция рисования результата (многоугольника)

**def** **draw\_figure**(figure):

**for** section **in** get\_uniq\_sections(figure):

draw\_section(round(section[**0**][**0**]), round(section[**0**][**1**]),

round(section[**1**][**0**]), round(section[**1**][**1**]), res\_color)

# Функция вычисления векторного произведения

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

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

# Функция вычисления скалярного произведения

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

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

# Функция проверки многоугольника на выпуклость

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

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

**return** False

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

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

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

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

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

**return** False

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

verteces.reverse()

**return** True

# Функция получения внутренней нормали к грани

# p1, p2 - вершины грани, cp - одна из точек многоугольника для проверки

# внутренняя нормаль или внешняя нашлась

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

vect = get\_vect(p1, p2)

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

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

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

norm[i] = -norm[i]

**return** norm

# Функция нахождения нормалей ко всем граням многоугольника (отсекателя)

**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 проверки принадлежности точки point отсекателю относительно

# грани [p1, p2]

**def** **check\_point**(point, p1, p2):

**return** True **if** vect\_mul(get\_vect(p1, p2), get\_vect(p1, point)) <= **0** **else** False

**def** **find\_intersection**(section, edge, normal):

wi = get\_vect(edge[**0**], section[**0**])

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

Wck = scalar\_mul(wi, normal)

Dck = scalar\_mul(d, normal)

diff = [section[**1**][**0**] - section[**0**][**0**], section[**1**][**1**] - section[**0**][**1**]]

t = -Wck / Dck

**return** [section[**0**][**0**] + diff[**0**] \* t, section[**0**][**1**] + diff[**1**] \* t]

# Функция отсечения многоугольника относительно одной грани отсекателя

**def** **edgecut\_figure**(figure, edge, normal):

res\_figure = list()

**if** len(figure) < **3**:

**return** []

prev\_check = check\_point(figure[**0**], \*edge)

**for** i **in** range(**1**, len(figure) + **1**):

cur\_check = check\_point(figure[i % len(figure)], \*edge)

**if** prev\_check:

**if** cur\_check:

res\_figure.append(figure[i % len(figure)])

**else**:

res\_figure.append(find\_intersection([figure[i - **1**],

figure[i % len(figure)]], edge, normal))

**else**:

**if** cur\_check:

res\_figure.append(find\_intersection([figure[i - **1**],

figure[i % len(figure)]], edge, normal))

res\_figure.append(figure[i % len(figure)])

prev\_check = cur\_check

**return** res\_figure

# Функция отсечения фигуры

**def** **cut\_figure**(figure, cutter\_verteces, normals\_list):

res\_figure = figure

**for** i **in** range(len(cutter\_verteces)):

cur\_edge = [cutter\_verteces[i],

cutter\_verteces[(i + **1**) % len(cutter\_verteces)]]

res\_figure = edgecut\_figure(res\_figure, cur\_edge,

normals\_list[i])

**if** len(res\_figure) < **3**:

**return** []

**return** res\_figure

# Функция-решение задачи (проверка выпуклости и затем отсечение)

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

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

mb.showerror("Невыпуклый многоугольник")

**return**

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

cutted\_figure = cut\_figure(figure\_list, verteces\_list, normals\_list)

draw\_figure(cutted\_figure)

Конец кода.

**Интерфейс и примеры работы.**

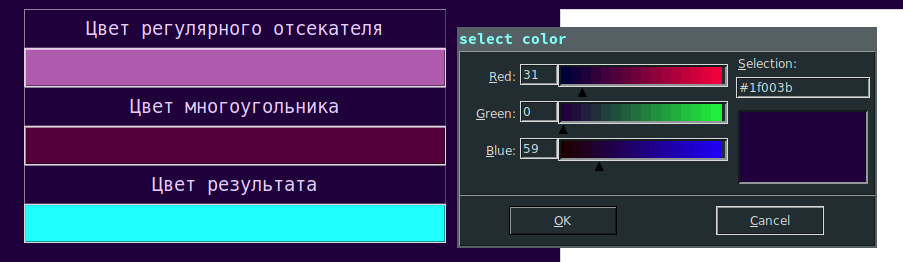
Интерфейс:



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

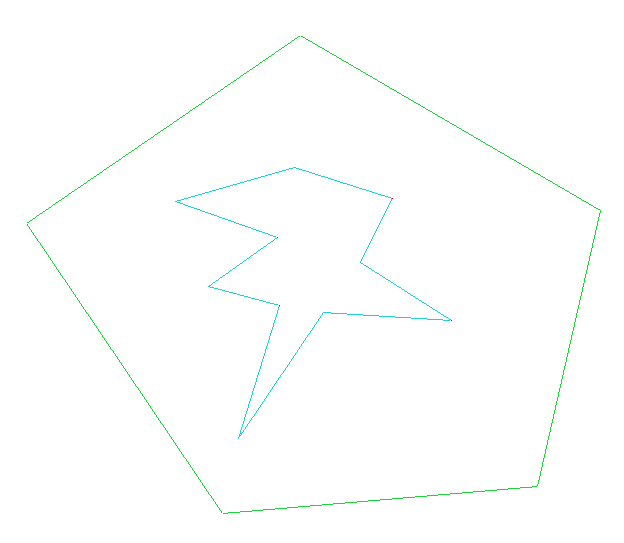
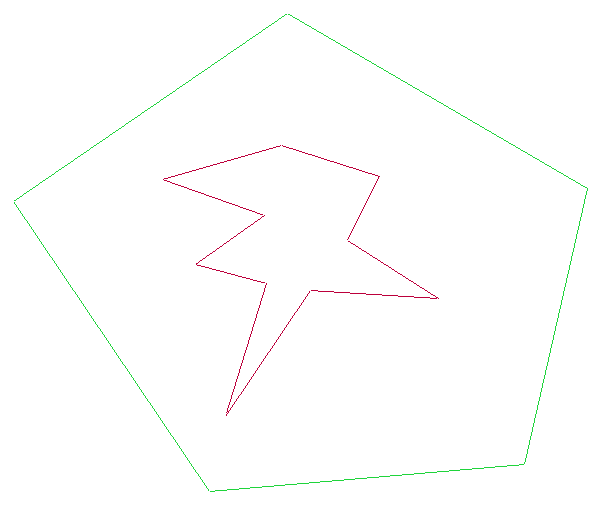
При вводе с помощью мышки границ отсекателя вершины вводятся нажатием правой кнопки мыши. Для замыкания следует нажать кнопку Enter.

При вводе с помощью мышки границ многоугольника вершины вводятся нажатием левой кнопки мыши. Для замыкания следует нажать кнопку **с**. Цвета могут быть любые, используется палитра:

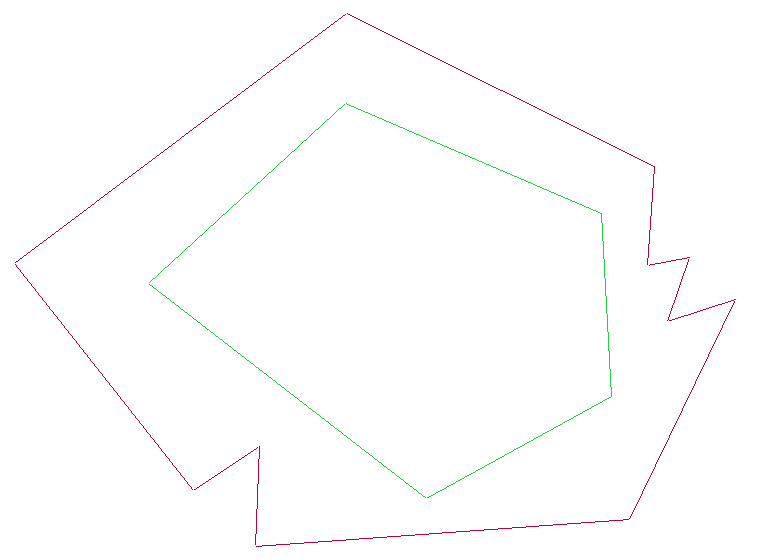


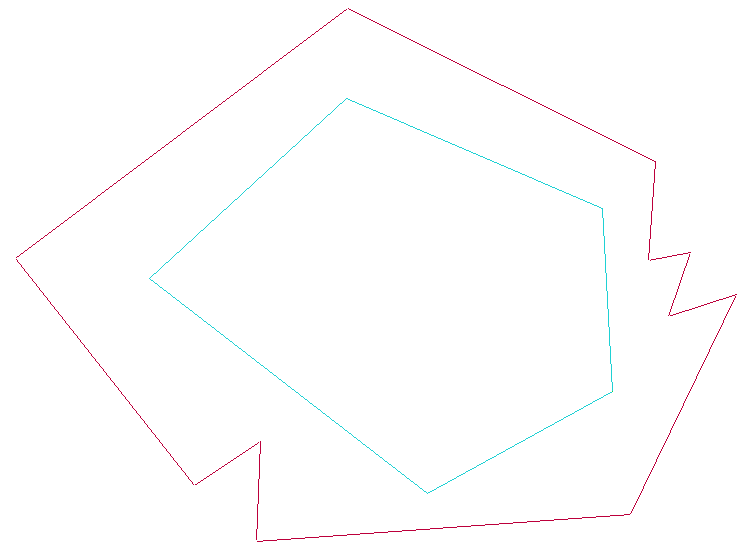
**Примеры работы**

Многоугольник полностью внутри отсекателя:

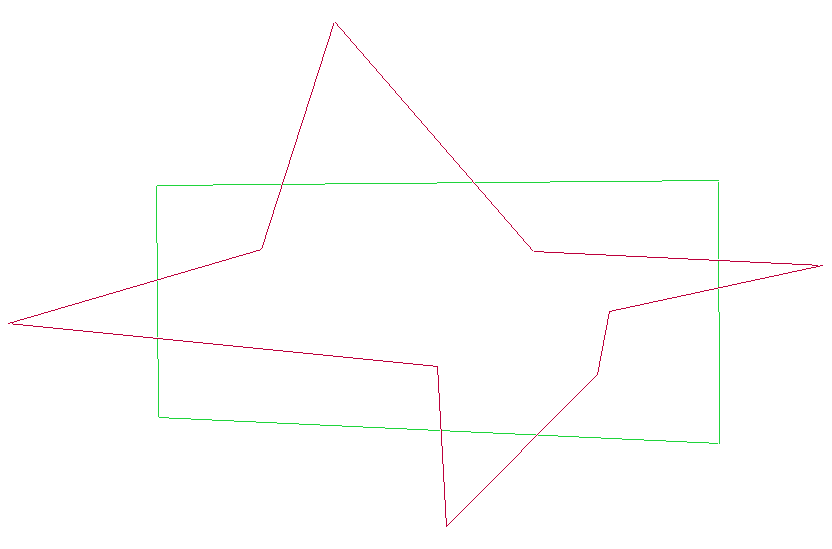


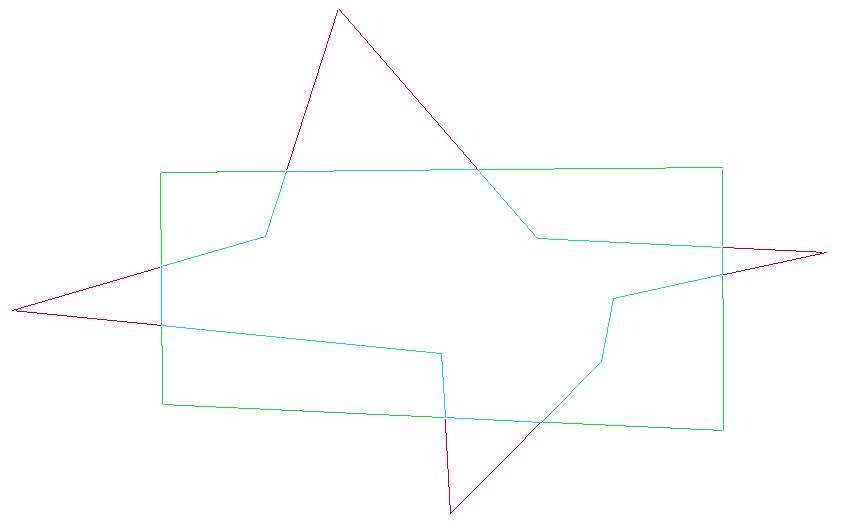
Многоугольник вне отсекателя:



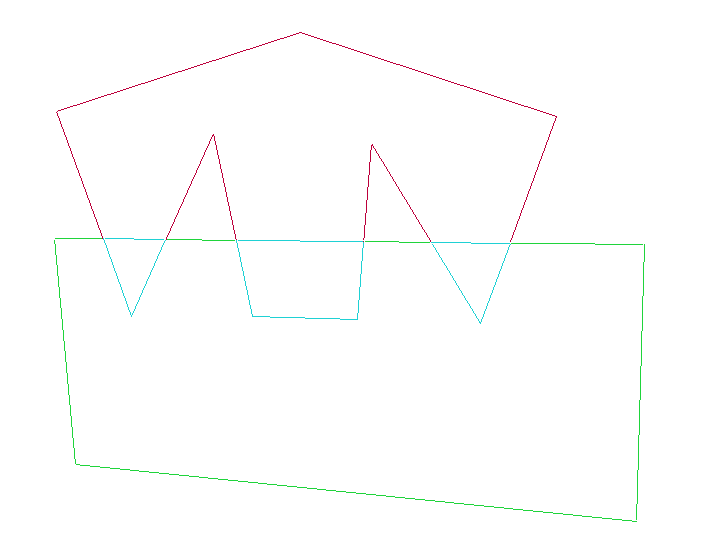
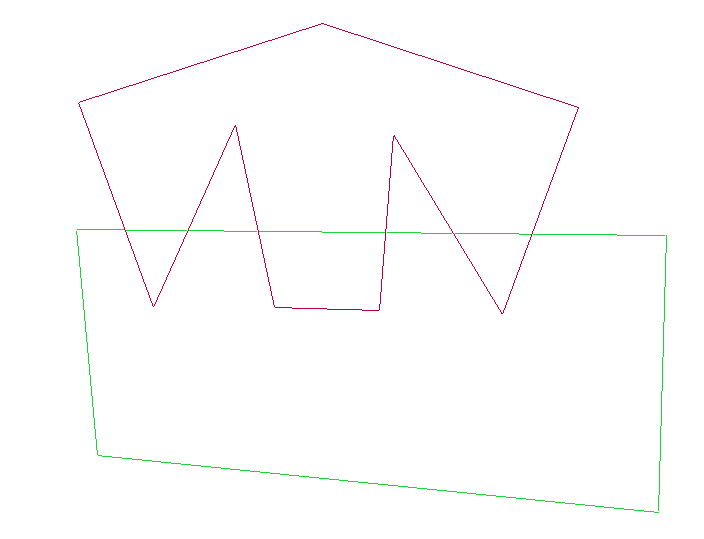


Многоугольник внутри отсекателя (пример похожий на пример с лекции):





Исключительная ситуация с лекции:



Пример, похожий на исключительную ситуацию:

