

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Дисциплина « Компьютерная графика»**

**Лабораторная работа №9**

**по теме:**

**«ОТСЕЧЕНИЕ ПРОИЗВОЛЬНОГО МНОГОУГОЛЬНИКА ВЫПУКЛЫМ ОТСЕКАТЕЛЕМ.»**

**Работу выполнил:**

студент группы ИУ7-43Б

Сукочева А.

**Работу проверил:**

Куров А. В.

2020 г.

**Цель работы:**

Изучение и программная реализация алгоритма Сазерленда-Ходжмена отсечения многоугольников.

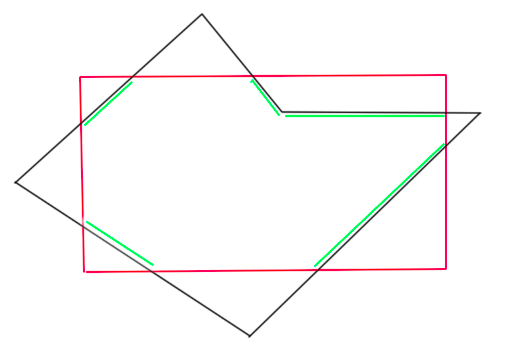
**Задание:**

Реализация алгоритма Сазерленда-Ходжмена.

**Требования:**

1. Необходимо обеспечить ввод отсекателя – произвольного многоугольника. Высветить его первым цветом.
2. Необходимо обеспечить ввод отсекаемого многоугольника. Высветить вторым цветом.
3. Необходимо проверить отсекатель на выпуклость.
4. Ввод осуществлять с помощью мыши и нажатия других клавиш.
5. Должен быть предусмотрен ввод вершин многоугольника в произвольных точках ребер отсекателя
6. Выполнить отсечение многоугольника, показав результат третьим цветом. Исходный многоугольник не удалять.

**Теоретический материал:**

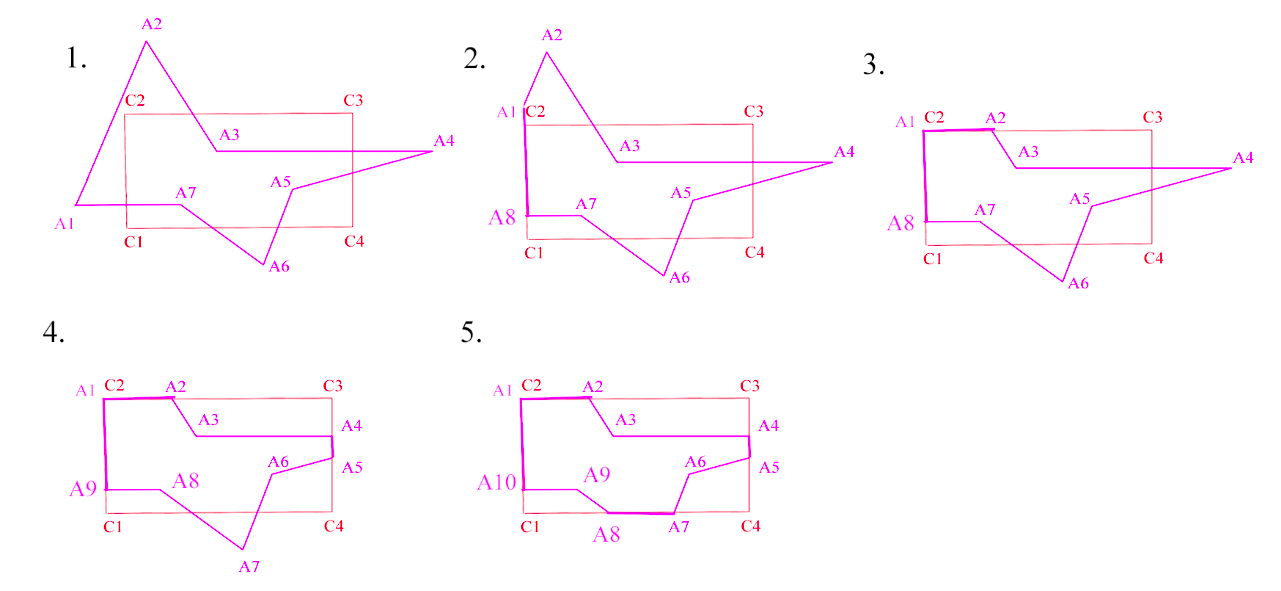
Если рассматривать отсечение многоугольника, как отсечение каждого его ребра по отдельности и воспользоваться предыдущим алгоритмом, то в результате мы получим неполную картину, а именно:

Зеленым цветом показан результат, если мы рассматриваем многоугольник как набор отрезков и каждую отсекаем многоугольником. Как мы видим - результат нас не удовлетворяет, поэтому были придуманы алгоритмы отсечения многоугольников которые в результате дают часть плоскости, ограниченной замкнутой ломаной линией.

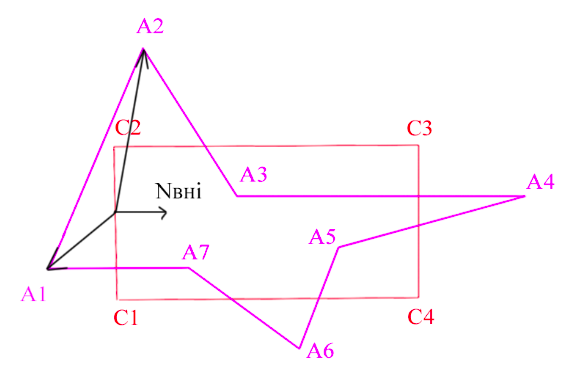
Сазерленд и Ходжмен предложили алгоритм, который решает данную задачу последовательно. Т.е. выполнять отсечение многоугольника последовательно каждой границей отсекателя. Результат, полученный на i-ом шаге отсечения является исходным данным для выполнения отсечения i+1 границы отсекателя.

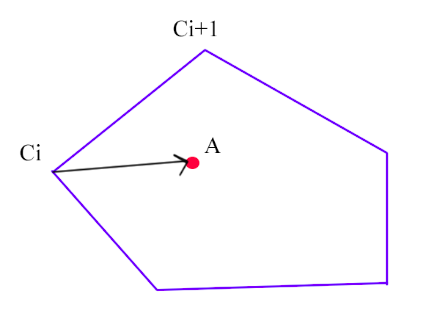
Рассмотрим последовательно работу алгоритма:

Мы должны последовательно определять видимость вершин многоугольника и определять точки пересечения его ребер с прямыми, проходящими через границы отсекателя. Если разная видимость концов ребра, то находим точку пересечения и заносим в результат.



Способы определения видимости точки:

1. Скалярное произведение вектора внутренней нормали на вектор, соединяющий точку ребра отсекателя с исследуемой точкой.
2. Использование пробной функции, получаемой из уравнения прямой, проходящей через ребро отсекателя. (Уравнение прямой: Ax+By+C=0). Чтобы получить пробную функцию мы отбрасываем равенство нулю(Пробная функция: f = Ax+By+C). Подставляя координаты исследуемой точки в пробную функцию мы определяем знак пробной функции. Знак пробной функции сравниваем со знаком пробной функции точки, положение которой нам известно.
3. Соединяем начало ребра отсекателя с исследуемой точкой и вычисляем векторное произведение CiA\*CiCi+1(Рис. ниже) и определить его знак. Если знак положительный, то точка видима, относительно текущей границы отсекателя. Если равно нулю, то точка расположена на границе отсекателя (Считаем, что видима). Если отрицательно, то точка невидима.



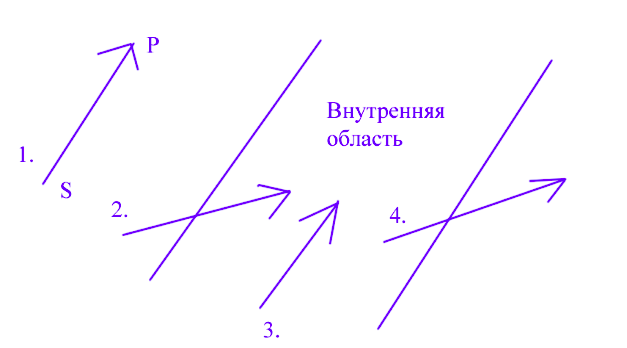
Определим пересечение, используя параметр уравнение:

, где 0<=t<=1 - для ребра отсекаемого многоугольника.

(s - без ограничения) - для ребра отсекателя.

= - решаем и получаем значение параметра. После этого можем вычислить x-ую и y-ую координату. Если нет решений, то отрезки параллельны.

Расположение ребер отсекаемого многоугольника относительно отсекателя:



Начальная вершина обозначается через S, а заключительная - P.

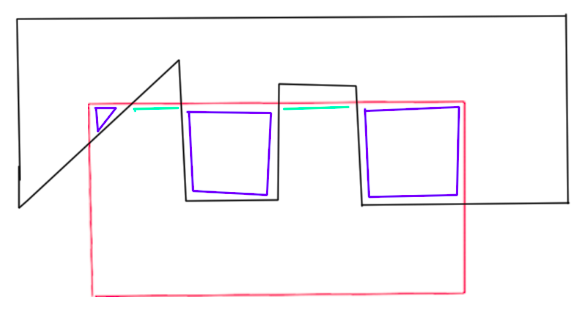
Выполняемые действия:

1. Обе вершины невидимы. В результат ничего не заносим.
2. S - невидима, P - видима. Найти точку пересечения и занести в результат точку пересечения и P.
3. S - видима, P - видима.В результат заносим обе вершины, но начальная вершина текущего ребра является конечной вершиной предыдущего ребра. Она будет занесена на предыдущем шаге.
4. S - видима, P - невидима. Найти точку пересечения и занести в результат. S - будет занесена на предыдущем шаге.

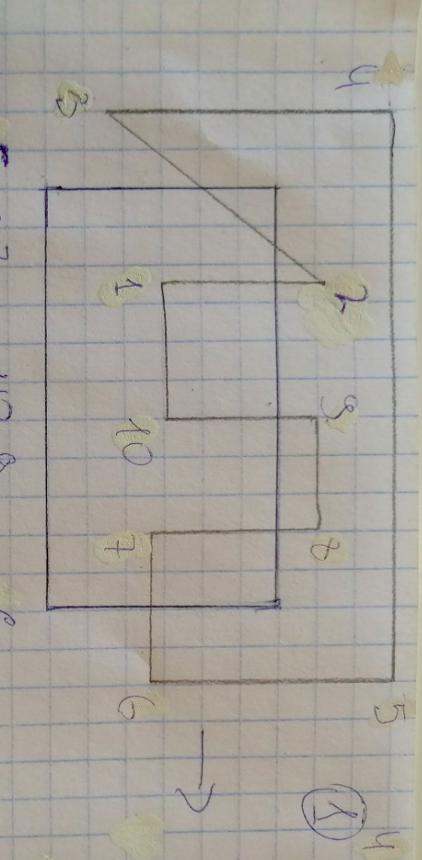
Особым образом нужно обрабатывать первую точку многоугольника: для нее требуется определить только видимость. Если точка видима, то она заносится в результирующий список и становится начальной точкой первого ребра. Если же она невидима, то она просто становится начальной точкой ребра и в результирующий список не заносится.   
 Можно убедиться в полной невидимости многоугольника, относительно текущей границы отсекателя. При анализе последнего замыкающего ребра отсутствие результата означает невидимость многоугольника относительно текущей границы отсекателя, а значит многоугольник невидимый.

Недостаток: Появление ложных ребер.

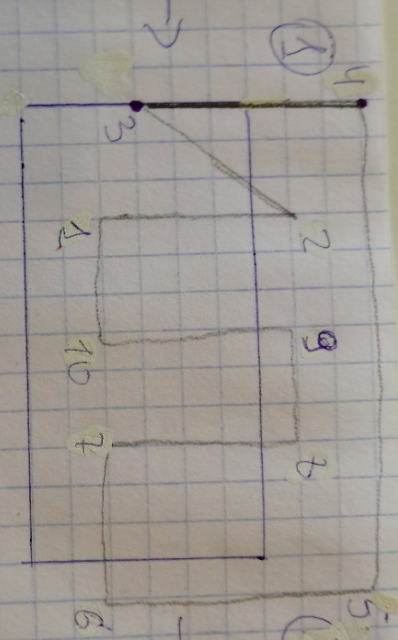
Ложное ребро - ребро, которого не должно быть. Они появляются, когда в результате отсечения получается несколько многоугольников. Ложное ребро как раз и соединяет эти многоугольники. (Рис. ниже. Зеленым показаны ложные ребра.)



Рассмотрим работу алгоритма. Допустим отсечение проходит сначала по левой границе, потом по верхней, правой и нижней.

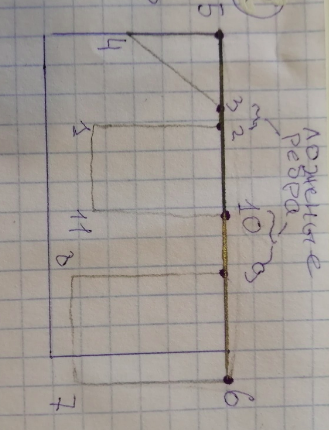


Вершина 1 видима, значит заносим ее в новый массив. Далее вершины 1 и 2 видимы, значит заносим в массив вершину 2. Далее вершина 2 видима, а вершина 3 нет (относительно рассматриваемой границы отсекателя) значит есть пересечение. Находим его и заносим в массив. Далее вершины 3 и 4 невидимы (Не заносим их). Далее вершина 4 невидима, а вершина 5 видима, значит находим пересечение и заносим в массив пересечение и 5 вершину. Остальные вершины будут видимы и занесены в массив. (Результат отсечения левой границей представлен ниже)



Отсечение по верхней границе:

Вершина 1 видима, заносим ее в массив. Далее вершина 1 видима, а 2 нет, ищем пересечение и заносим в его массив. Далее вершина 2 невидима, а 3 видима, находим пересечение и заносим в массив пересечение и вершину 3. Далее вершина 3 видима, а 4 нет, значит заносим в массив пересечение. Далее 4 и 5 невидимы, ничего не заносим. Далее 5 невидима, 6 видима, значит заносим пересечение и 6 вершину. Далее 6 и 7 видимы, заносим 7. Далее 7 видима, а 8 нет, значит заносим пересечение. 8 и 9 невидимы, ничего не заносим. 9 невидима, а 10 видима, значит заносим пересечение и 10. И в конце мы проверим, есть ли пересечение (10 и 1) и т.к. его нет, мы его не занесем (первая вершина вначале занесена была, последняя на предыдущем шаге).



Как мы видим в результате ложные ребра 3-2 и 10-9. По этим ребрам многоугольник проходит дважды (Т.е. сверху еще проходит ребро 5-6).

**Алгоритм:**

**def FindDirection(line):**

**return [line[1][0] - line[0][0], line[1][1] - line[0][1]]**

**def Find\_W(p1, p2):**

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

**def ConvertParametric(line, t):**

**return [round(line[0][0] + (line[1][0] - line[0][0]) \* t), round(line[0][1] + (line[1][1] - line[0][1]) \* t)]**

**def IsIntersection(ed1, ed2, peak):**

**# ed1 - ребро отсекаемого многоугольника.**

**# ed2 - ребро отсекателя.**

**# peak - след. вершина отсекателя, нужна для**

**# корректного определения нормали**

**# Определяем видимость вершин относительно рассматриваемого ребра.**

**visiable1 = IsVisiable(ed1[0], ed2[0], ed2[1], peak)**

**visiable2 = IsVisiable(ed1[1], ed2[0], ed2[1], peak)**

**# Если одна вершина видна, а вторая нет (Есть пересечение).**

**# Иначе пересечения нет.**

**if not (visiable1 ^ visiable2):**

**return False**

**# Ищем пересечение**

**N = FindNormal(ed2[0], ed2[1], peak)**

**D = FindDirection(ed1)**

**W = Find\_W(ed1[0], ed2[0])**

**# Скалярное произведение D на N.**

**DScalar = Scalar(D, N)**

**# Скалярное произведение W на N.**

**WScalar = Scalar(W, N)**

**# DScalar может быть равен нулю в двух случаях:**

**# 1. Если ребро отсекателя вырождается в точку**

**# Т.е. p1 == p2. В интерфейсе обработан данный случай**

**# (Пользователь не может ввести ребро у которого начало и конец совпадают)**

**# 2. Если текущее ребро отсекаемого многоугольника параллельно**

**# Ребру отсекателя. Такие ребра не дойдут до этого момента -**

**# Они будут обработаны выше. Т.к. в этом случае нет пересечения**

**# Обе вершины отсекаемого многоугольника будут либо по видимую сторону**

**# Отсекателя, либо по невидимую.**

**t = -WScalar/DScalar**

**return ConvertParametric(ed1, t)**

**def Scalar(v1, v2):**

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

**def FindNormal(peak1, peak2, peak3):**

**n = [peak2[1] - peak1[1], peak1[0] - peak2[0]]**

**# Если скалярное произведение вектора нормали**

**# На вектор, который является следующим ребром**

**# Многоугольника, дает нам отрицательное значение,**

**# То вектор нормали нужно домножить на -1**

**# Чтобы он был направлен внутрь многоугольника.**

**if Scalar([peak3[0] - peak2[0], peak3[1] - peak2[1]], n) < 0:**

**n = [-n[0], -n[1]]**

**return n**

**def IsVisiable(point, peak1, peak2, peak3):**

**# Находим нормаль к ребру (peak1, peak2)**

**# peak3 нужно, чтобы проверить нормаль (Внутренняя ли она).**

**n = FindNormal(peak1, peak2, peak3)**

**if Scalar(n, GetVector([peak2, point])) < 0:**

**return False**

**return True**

**def GetVector(line):**

**return [line[1][0] - line[0][0], line[1][1] - line[0][1]]**

**def SutherlandHodgman(cutter, polygon):**

**# Для удобства работы алгоритма первая вершина**

**# отсекателя заносится в массив дважды (В начало и конец).**

**# Т.к. последнее ребро отсекателя образуется**

**# последней и первой вершинами многоугольника.**

**cutter.append(cutter[0])**

**# Также, т.к. для поиска нормали для ребра i и i+1**

**# Мне нужна вершина i+2. Поэтому я дублирую еще вторую вершину.**

**# (Чтобы могла в цикле отправлять i+2).**

**cutter.append(cutter[1])**

**# Цикл по вершинам отсекателя.**

**for i in range(len(cutter) - 2):**

**new = [] # новый массив вершин**

**# Особым образом нужно обрабатывать первую**

**# точку многоугольника: для нее требуется определить**

**# только видимость. Если точка видима, то она заносится**

**# В результирующий список и становится начальной точкой первого ребра.**

**# Если же она невидима, то она просто становится начальной точкой ребра**

**# И в результирующий список не заносится.**

**f = polygon[0] # Запоминаем первую вершину.**

**if IsVisiable(f, cutter[i], cutter[i + 1], cutter[i + 2]):**

**new.append(f)**

**s = polygon[0]**

**# Цикл по вершинам многоугольника**

**for j in range(1, len(polygon)):**

**# Определяем пересечение текущего ребра отсекателя (cutter[i], cutter[i + 1])**

**# И рассматриваемого ребра отсекаемого многоугольника (s, polygon[j]),**

**# Где s = polygon[j - 1]. cutter[i + 2] нам нужно, чтобы корректно найти нормаль.**

**t = IsIntersection([s, polygon[j]], [cutter[i],**

**cutter[i + 1]], cutter[i + 2])**

**# Если есть пересечение, то заносим его в новый массив вершин.**

**if t:**

**new.append(t)**

**# Запоминаем в s текущую вершину. (Чтобы на следующем шаге**

**# Искать пересечение polygon[j - 1] и polygon[j])**

**s = polygon[j]**

**# Проверяем, видна ли текущая вершина**

**if IsVisiable(s, cutter[i], cutter[i + 1], cutter[i + 2]):**

**# Если видна, то заносим ее в новый массив вершин.**

**new.append(s)**

**# Можно убедиться в полной невидимости многоугольника,**

**# относительно текущей границы отсекателя. При анализе**

**# последнего замыкающего ребра отсутствие результата означает невидимость**

**# многоугольника относительно текущей границы отсекателя, а значит многоугольник невидимый.**

**if not len(new):**

**return False**

**t = IsIntersection([s, f], [cutter[i], cutter[i + 1]], cutter[i + 2])**

**if t:**

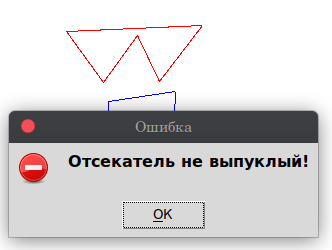
**new.append(t)**

**polygon = deepcopy(new)**

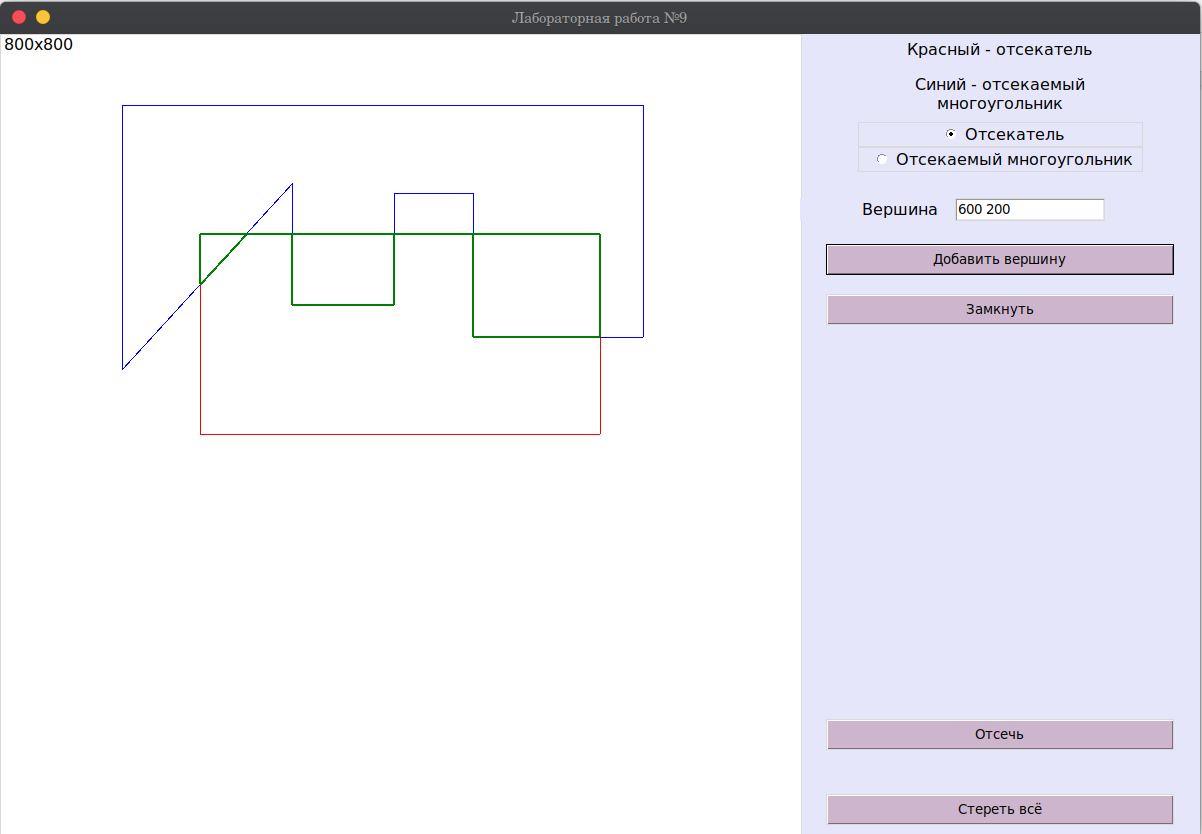
**return polygon**

**Результат:**

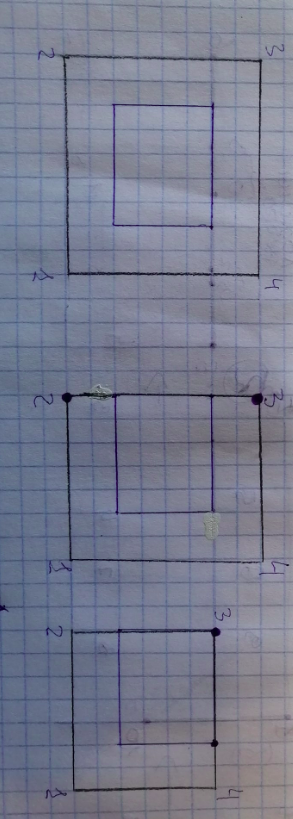
Как и в предыдущей лабораторной работе присутствует проверка на выпуклость отсекателя:

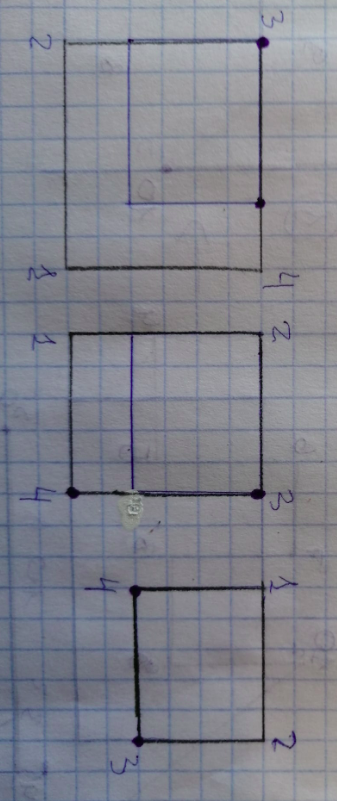


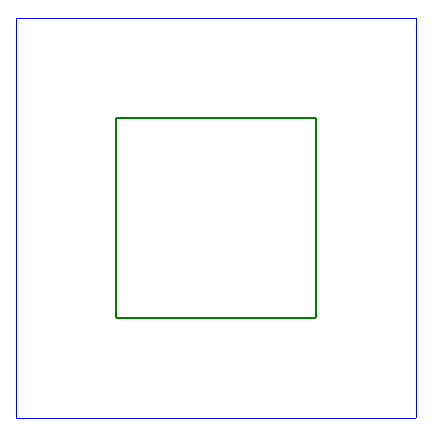
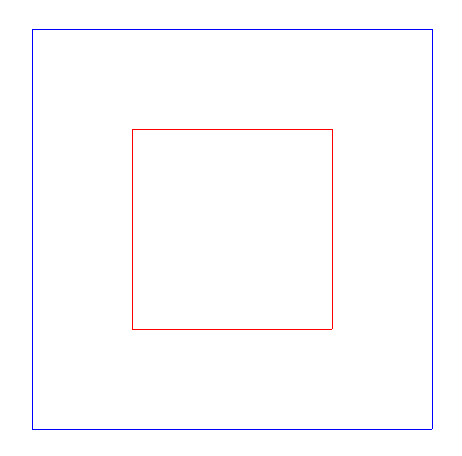
Появление ложных ребер: (Зеленым цветом показан результат)



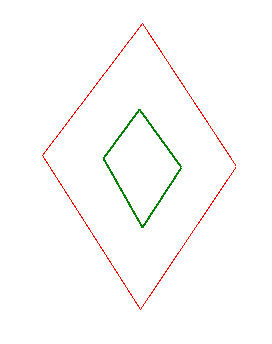
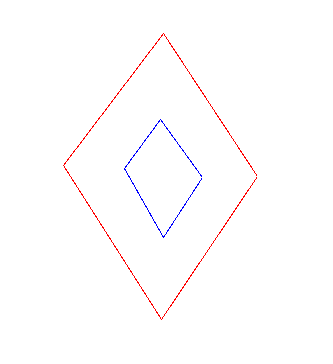
Теперь рассмотрим, если отсекатель находится внутри отсекаемого многоугольника (Ручкой нарисован отсекатель, карандашом отсекаемый многоугольник). Отсечение проводится сначала по левой стороне, потом верхней, правой и нижней.

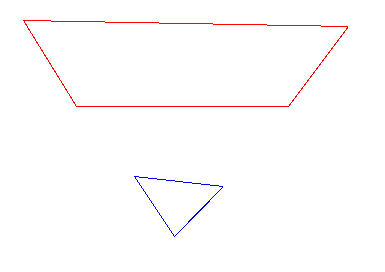






Рассмотрим ситуацию, когда отсекаемый многоугольник расположен внутри отсекателя. В этом случае относительно всех граней отсекателя все вершины отсекаемого многоугольника будут видимы и поэтому в результате будет исходный отсекаемый многоугольник.



Рассмотрим ситуацию, когда многоугольник полностью невидим. 

Относительно нижней грани отсекаемый многоугольник будет полностью невидимый и новый новый массив вершин будет пуст. После чего будет сделана проверка на пустоту нового массива и т.к. он пуст, то алгоритм завершит работу.

Еще некоторые примеры:

