|  |  |
| --- | --- |
| 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 остекателем (Алгоритм Сазерленда-Ходжмена)  **Студент** Якуба Д. В.  **Группа** ИУ7-43  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель** Куров А. В. |  |

Москва

2020 г.

Оглавление

[Цель работы 2](#_Toc41760924)

[Техническое задание 2](#_Toc41760925)

[Теоретическая часть 3](#_Toc41760926)

[Практическая часть 3](#_Toc41760927)

[Программная реализация алгоритма на ЯП Python 3](#_Toc41760928)

[Пользовательский интерфейс 3](#_Toc41760929)

[Демонстрация работы алгоритма 3](#_Toc41760930)

# Цель работы

Изучение и программная реализация алгоритма Сазерленда-Ходжмена отсечения многоугольников.

# Техническое задание

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 точка рёбер отсекателя (включая его вершины).

Ввод осуществлять с помощью мыши и нажатия других клавиш.

Выполнить отсечение многоугольника, показав результат третьим цветом. Исходный многоугольника не удалять.

# Теоретическая часть

## Отсечение многоугольников выпуклым отсекателем (алгоритм Сазерленда-Ходжмена)

Для решения поставленной задачи отсечения многоугольника выпуклым отсекателем нам несколько недостаточно знать изученный в предыдущей работе алгоритм отсечения отрезков Кируса-Бека, так как в решении задачи нам требуется «отсечь» часть плоскости, ограниченной заданной ломаной линией. Таким образом, при решении задачи от нас потребуется найти замкнутую ломанную линию, которая будет состоять не только из участков рёбер заданного исходного многоугольника, но и будет содержать в себе в качестве результата участки рёбер отсекателя.

Господами Сазерлендом и Ходжменом было предложено решать данную задачу последовательно, то есть выполнять отсечение многоугольника последовательно каждой границей отсекателя и результат, полученный на очередном шаге, является исходным многоугольником для выполнения отсечения следующей границей отсекателя.

При решении задачи мы должны последовательно определять видимость рассматриваемых вершин многоугольника и определять точки пересечения его рёбер с границами отсекателя.

Опишем алгоритм на конкретном примере:

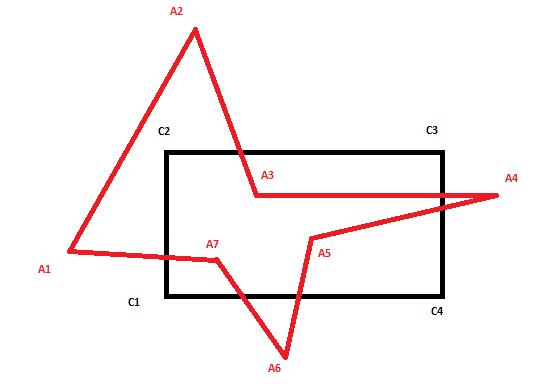


Рисунок 1, пример

В качестве первой рассматриваемой границы отсекателя возьмём границу C1-C2.

Вершина A1 невидима относительно левой границы отсекателя, поэтому в результат она занесена не будет. Вершина A2 видима относительно левой границы отсекателя. Видно, что видимость вершины A1 и A2 разная, а значит, ребро многоугольника пересекает рассматриваемую границу отсекателя. В таком случае нужно определить точку пересечения текущего ребра с границей и занести найденную точку в результат. Анализируя видимость следующих вершин (A3, A4, A5, A6, A7), убеждаемся, что все они видимы относительно текущей границы отсекателя, и все они попадают в результат. По завершению текущего шага потребуется сформировать замыкающее ребро многоугольника – A7 A1. Анализируя видимость этих вершин, мы определим, что ребро многоугольника пересекает текущую границу отсекателя. Поэтому будет найдена точка пересечения, которую мы и занесём в результат. Таким образом, по окончанию выполнения описанного выше шага будем иметь следующий результат:

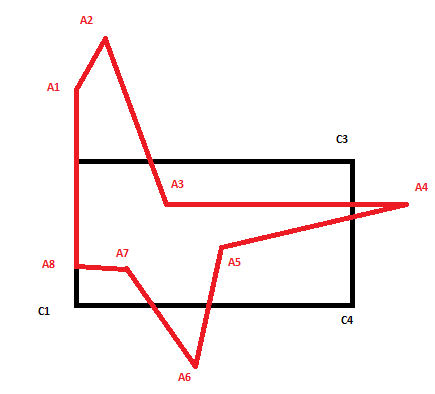


Рисунок 2, результат выполнения шага 1

Рассмотрим следующий шаг (граница отсекателя C2 C3). Вершина A1 невидима относительно текущей границы отсекателя, следовательно, в результат она не заносится. Вершина A2 также невидима. Вершина A3 видима. Видимость вершин ребра A2 A3 разная, следовательно, требуется найти точку пересечения. В результат будут занесены найденная точка пересечения рассматриваемого ребра с границей отсекателя и вершина A3. Следующие рассматриваемые вершины (A4, A5, A6, A7, A8) видимы и требуется определить замыкающее ребро: вершина A8 видима, A1 невидима, следовательно требуется найти точку пересечения данного ребра с верхней границей отсекателя и внести в результат вершину A8 и найденную точку пересечения.

По окончанию выполнения описанного выше шага будем иметь следующий результат:

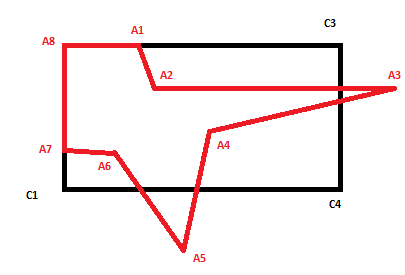


Рисунок 3, результат выполнения шага 2

Рассмотрим следующий шаг (граница отсекателя C3 C4). Вершина A1 видима. Вершина A2 видима. Вершина A3 невидима. Имеем разную видимость концов ребра, находим точку пересечения и заносим её в результат. Для ребра A3 A4 ситуация аналогичная, с тем лишь изменением, что видна вторая, а не первая, вершина рассматриваемого ребра. Находим точку пересечения этого ребра с границей отсекателя и заносим её в результат. Вершины A4, A5, A6, A7, A8 видимы, заносим их в результат.

По окончанию выполнения описанного выше шага будем иметь следующий результат:

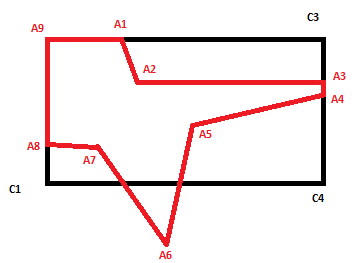


Рисунок 4, результат выполнения шага 3

Рассмотрим следующий шаг (граница отсекателя C1 C4). Вершина A1 видима. Вершина A2 видима. Вершина A3, A4, A5 видимы. Вершина A6 невидима, следовательно, находим точку пересечения ребёр A5 A6 и A6 A7 с рассматриваемой границей отсекателя и точки пересечения занесём в результат. Оставшиеся рёбра будут видимыми относительно рассматриваемой границы отсекателя.

По окончанию выполнения описанного выше шага будем иметь следующий результат:

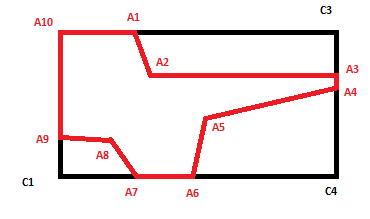


Рисунок 5, результат выполнения шага 4

Таким образом, для реализации алгоритма нужно уметь следующие две задачи: определение видимость вершин отсекаемого многоугольника относительно отсекателя и нахождение точек пересечения рёбер многоугольника со сторонами отсекателя.

На лекции нам было предложено ответить на следующий вопрос: «Пересечение каких двух геометрических объектов мы здесь должны находить?»

Ответ: мы ищем пересечение отрезка, которым является очередное ребро многоугольника, и прямой, проходящей через точки границы отсекателя.

## Определение видимости точки относительно стороны отсекателя

Видимость точки многоугольника будем определять так же, как и в предыдущей лабораторной работе, используя скалярное произведение скалярного произведения вектора внутренней нормали на вектор, соединяющий точку ребра отсекателя с исследуемой точкой.

Также возможен ещё один способ определения видимости точки посредством использования пробной функции, получаемой из уравнения прямой, проходящей через ребро отсекателя. При этом уравнение прямой: . Подставляя координаты исследуемой точки в пробную функцию, мы определяем её знак. Полученный знак сравнивается со знаком пробной функции точки, положение которой нам известно.

Ещё есть возможность использовать способ, основанный на векторном произведении. Допустим, что у нас есть некоторый отсекатель, изображённый на следующем рисунке:

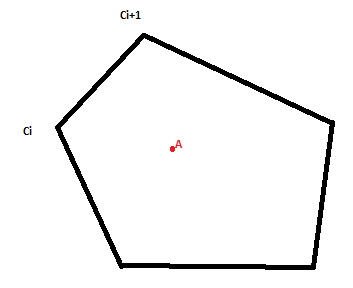


Рисунок , пример

Для решения поставленной задачи требуется вычислить векторное произведение вектором и вектором .

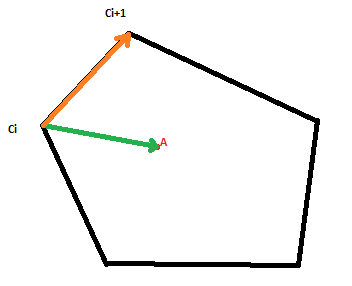


Рисунок , расположение описанных векторов

Далее достаточно просто проанализировать знак полученного произведения:

Если знак положителен, то точка видима относительно текущей границы отсекателя.

Если полученное значение нулевой, то точка находится на границе отсекателя, а, следовательно, видима.

Если знак отрицателен, то точка невидима относительно текущей границы отсекателя.

## Определение точки пересечения ребра отсекаемого многоугольника с границей отсекателя

Пересечение двух прямых найти достаточно несложно, но в нашем случае это будет выполнить наиболее удобно выполнить, используя параметрическую форму задания:

*– ребро отсекаемого многоугольника*

*– граница отсекателя*

Заметим, что мы ищем точку пересечения ребра отсекаемого многоугольника с прямой, проходящей через границу отсекателя. При этом легко заметить, что в таком случае мы не накладываем ограничения на параметр

Таким образом, чтобы найти требующуюся точку пересечения будем иметь:

То есть получаем систему двух уравнений с двумя неизвестными, решаем его и получаем значение параметра соответствующей точки пересечения и после этого вычисляются и координаты точки пересечения.

## Расположение рёбер многоугольника относительно внутренней области отсекателя



Рисунок , случай 1 - целиком невидимое ребро

В данном случае обе вершины невидимы, следовательно, в результат ничего не заносится.

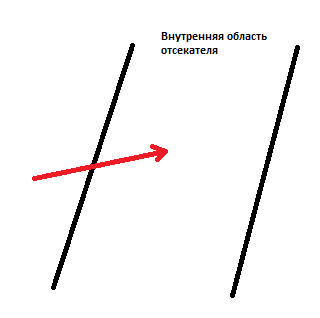


Рисунок , случай 2 - частично видимое входящее ребро

В данном случае начальная вершина невидима, а конечная вершина видима. Потребуется найти точку пересечения и занести в результат найденную и конечную точки.

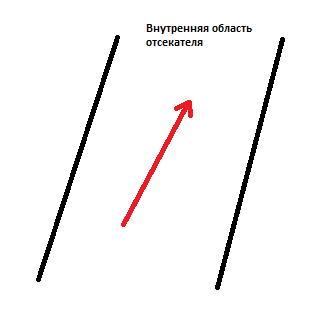


Рисунок , случай 3 - полностью видимое ребро

В данном случае обе вершины видимы, следовательно, обе вершины заносятся в результат, но начальная вершина текущего ребра является конечной вершиной предыдущего ребра – она будет занесена в результат не предыдущем шаге.

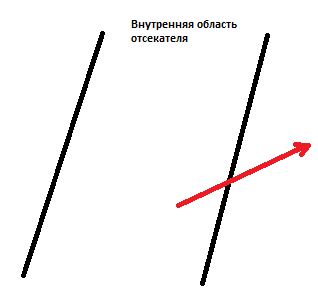


Рисунок , случай 4 - частично видимое выходящее ребро

В данном случае начальная вершина видима, а конечная – невидима. Начальная вершина будет занесена в результат на предыдущем шаге, а на текущем шаге потребуется определить точку пересечения и занести её в результат.

## Недостаток алгоритма

Главным недостатком описываемого алгоритма является появление ложных рёбер, то есть тех рёбер, которых быть не должны. Их появление обусловлено ситуацией, когда в результате отсечения получается несколько многоугольников.

Пример такой ситуации предоставлен ниже:

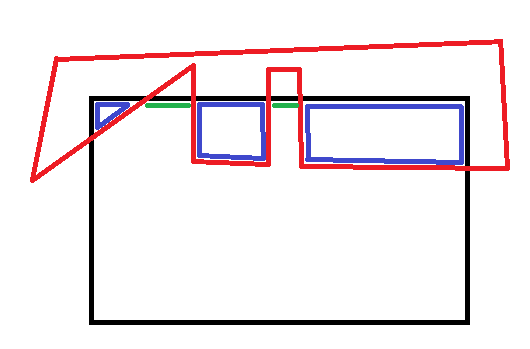


Рисунок , появление ложных рёбер в результате отсечения, зелёным цветом обозначены ложные рёбра

# Практическая часть

## Алгоритм Сазерленда-Ходжмена

1. Ввод исходных данных: количетсов вершин отсекааемого многоугольника, массива координат вершин отсекаемого многоугольника, количество вершин отсекателя, массива координат вершин отсекателя. Элементами массивов являются записи, каждая из которых содержит два поля – координаты и вершины. Для удобства работы алгоритма первая вершина отсекателя заносится в массив C дважды: на первое место и ещё раз в конец массива (это сделано потому, что последнее ребро отсекателя образуется последней и первой вершинами многоугольника).

2.Цикл по всем рёбрам отсекателя (переменная цикла изменяется от 1 до ).

2.1. Обнуление количества вершин результирующего многоугольника

2.2. Цикл по всем рёбрам отсекаемого многоугольника (переменная цикла изменяется от 1 до ).

2.2.1. Анализ номера обрабатываемой вершины многоугольника: если (первая вершина), то её координаты запоминаются в переменной .

Переход к пункту 2.2.7.

2.2.2. Определение факта пересечения ребра многоугольника и ребра отсекателя .

2.2.3. Если пересечение рёбер многоугольников установлено, то определение координат точки T пересечения этих рёбер, иначе переход к пункту 2.2.7.

2.2.4. Увеличение на единицу количества вершин результирующего многоугольника . Занесение в массив координат результирующего многоугольника координат найденной точки .

2.2.5. Изменение начальной точки ребра многоугольника: присвоение переменной значения переменной

2.2.6. Проверка видимости вершины относительно ребра . Если вершина видима, то занесение её координат в массив .

2.2.7. Конец цикла по переменной (цикл отсечения рёбер многоугольника по текущей границе отсекателя).

2.3. Проверка ненулевого количества вершин в результирующем массиве: если , то переход к пункту 2.10 (многоугольник невидим относительно текущей границы отсекателя, следовательно, он невидим относительно всего отсекателя).

2.4. Проверка факта пересечения ребра многоугольника с ребром отсекателя .

2.5. Если отсечение рёбер многоугольников установлено, то определение координат точки пересечения этих рёбер, иначе переход к пункту 2.

2.6. Увеличение на единицу количества вершин результирующего многоугольника . Занесение в массив координат результирующего многоугольника координат найденной точки .

2.7. Присвоение полученных значений количества вершин и их координат исходного многоугольника: (полученный многоугольника отсекается далее следующей стороной отсекателя).

2.8. Конец цикла по переменной (цикл отсечения по все границам отсекателя).

2.9. Визуализация полученного многоугольника P.

2.10. Конец алгоритма.

## Программная реализация алгоритма на ЯП Python

def scalProd(fVector, sVector):

return fVector[0] \* sVector[0] + fVector[1] \* sVector[1]

def vectProd(fVector, sVector):

return fVector[0] \* sVector[1] - fVector[1] \* sVector[0]

def isConvex(pointArray):

if len(pointArray) < 3:

return False

prev = sign(vectProd([pointArray[0][0] - pointArray[-1][0], pointArray[0][1] - pointArray[-1][1]],

[pointArray[-1][0] - pointArray[-2][0], pointArray[-1][1] - pointArray[-2][1]]))

for i in range(1, len(pointArray) - 2):

cur = sign(vectProd([pointArray[i][0] - pointArray[i - 1][0], pointArray[i][1] - pointArray[i - 1][1]],

[pointArray[i - 1][0] - pointArray[i - 2][0], pointArray[i - 1][1] - pointArray[i - 2][1]]))

if prev != cur:

return False

prev = cur

return True

def normal(fPoint, sPoint, posToPoint):

foundVector = [sPoint[0] - fPoint[0], sPoint[1] - fPoint[1]]

positiveForVector = [posToPoint[0] - sPoint[0], posToPoint[1] - sPoint[1]]

if foundVector[1]:

foundPoint = - foundVector[0] / foundVector[1]

normVec = [1, foundPoint]

else:

normVec = [0, 1]

if scalProd(positiveForVector, normVec) < 0:

normVec[0] = -normVec[0]

normVec[1] = -normVec[1]

return normVec

def isVisibleFor(point, fPointOfSide, sPointOfSide):

if vectProd([sPointOfSide[0] - fPointOfSide[0], sPointOfSide[1] - fPointOfSide[1]], [point[0] - fPointOfSide[0], point[1] - fPointOfSide[1]]) < 0:

return False

else:

return True

def crossingOfTwoSegments(segment, side, norm):

directrix = [segment[1][0] - segment[0][0], segment[1][1] - segment[0][1]]

wVec = [segment[0][0] - side[0][0], segment[0][1] - side[0][1]]

dScal = scalProd(directrix, norm)

wScal = scalProd(wVec, norm)

parameter = - wScal / dScal

return [segment[0][0] + directrix[0] \* parameter, segment[0][1] + directrix[1] \* parameter]

def cutSide(result, side, posToDot):

ret = []

norm = normal(side[0], side[1], posToDot)

previousVisibility = isVisibleFor(result[-2], side[0], side[1])

for curPointNum in range(-1, len(result)):

currentVisibility = isVisibleFor(result[curPointNum], side[0], side[1])

if previousVisibility:

if currentVisibility:

ret.append(result[curPointNum])

else:

ret.append(crossingOfTwoSegments([result[curPointNum - 1], result[curPointNum]], side, norm))

else:

if currentVisibility:

ret.append(crossingOfTwoSegments([result[curPointNum - 1], result[curPointNum]], side, norm))

ret.append(result[curPointNum])

previousVisibility = currentVisibility

return ret

def SutherlandHodgmanAlg(figureArray, cutterArray):

if not isConvex(cutterArray):

makeConvexError()

return

result = figureArray

for curDot in range(-1, len(cutterArray) - 1):

result = cutSide(result, [cutterArray[curDot], cutterArray[curDot + 1]], cutterArray[curDot + 1])

if len(result) <= 2:

return []

drawFigure(result)

## Пользовательский интерфейс

## Демонстрация работы алгоритма