|  |  |
| --- | --- |
| 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 № 7**

|  |  |
| --- | --- |
| **Тема** Изучение и программная реализация простого алгоритма отсечения отрезка  **Студент** Якуба Д. В.  **Группа** ИУ7-43  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель** Куров А. В. |  |

Москва

2020 г.

Оглавление

[Цель работы 3](#_Toc40529092)

[Техническое задание 3](#_Toc40529093)

[Теоретическая часть 3](#_Toc40529094)

[Алгоритм построчного затравочного заполнения 3](#_Toc40529095)

[Практическая часть 3](#_Toc40529096)

[Программная реализация алгоритма на ЯП Python 3](#_Toc40529097)

[Пользовательский интерфейс 3](#_Toc40529098)

[Демонстрация работы алгоритма 3](#_Toc40529099)

[Исследование временных характеристик 3](#_Toc40529100)

# Цель работы

Изучение и программная реализация алгоритма отсечения отрезка.

# Техническое задание

Алгоритм отсечения отрезка, согласно варианту 28 – Простой алгоритм отсечения.

Необходимо обеспечить ввод регулярного отсекателя – прямоугольника. 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.

Ввод осуществляется с помощью мыши и нажатия других клавиш.

Выполнить отсечение отрезков, показав результат третьим цветом. Исходные отрезки не удалять.

# Теоретическая часть

## Отсечение

Отсечение – операция удаления части изображения, находящейся за пределами выделенной области, называемой отсекателем.

Отсечение используется в алгоритмах удаления невидимых линий поверхностей и в некоторых алгоритмах учёта теней, поэтому от нас не только требуется скопировать или сохранить изображение, расположенное в пределах отсекателя, но и решить геометрическую задачу нахождения геометрических характеристик изображения, чтобы использовать их при решении других задач.

Также существует понятие стирания изображения – это операция удаления части изображения, находящейся в пределах заданной области (то есть отсекателя).

Само по себе отсечение может проводиться в двумерном или трёхмерном пространствах. При этом, трёхмерный случай является обобщением двумерного случая. То есть, умея решать задачу в двумерном пространстве, не составит труда реализовать и трёхмерный случай.

Существует следующая классификация двумерных отсекателей:

Регулярный (стандартный) отсекатель – это отсекатель прямоугольной формы со сторонами, параллельными координатным осям.

Нерегулярный отсекатель – отсекатель формы произвольного выпуклого многоугольника.

Более сложные отсекатели – отсекатели формы произвольного невыпуклого многоугольника.

Также следует определить классификацию трёхмерных отсекателей:

Отсекатели формы прямоугольного параллелепипеда

Отсекатели формы четырёхгранной усечённой пирамиды.

Также следует отметить, что границу отсекателя принято относить к внутренней области отсечения.

## Отсечение отрезков регулярным отсекателем

Перед тем, как обратиться к простому алгоритму отсечения отрезка регулярным отсекателем, нам потребуется решить задачу определения отношения заданного отрезка к области отсечения.

На следующей картинке определены три отрезка:

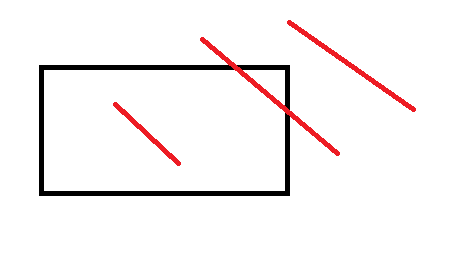


Рисунок , виды отрезков

Каждый из отрезков относительно отсекаемой области будет являться: полностью видимым (крайний левый отрезок), частично видимым (средний отрезок) или полностью невидимым (крайний правый отрезок).

Как видно из рисунка, для того чтобы определить принадлежность каждой из вершин отрезка отсекаемой области, достаточно проверить условие:

Каждый раз при обработке очередного отрезка проверять выполнения данных неравенств не очень удобно, поэтому было предложено формализовать данную процедуру и характеризовать положение точки четырёхразрядным кодом.

Каждый разряд в таком коде определяет положение точки относительно каждой из границ отсекателя. Обозначим такой код буквой .

В таком коде для каждой точки будем иметь:

*Стоит отметить, что в координатах экрана в условиях для знаки строгого неравенства поменяются между собой местами.*

Таким образом, точка будет видима в том случае, если поразрядная сумма её кода будет равна нулю.

Получается, что и отрезок будет полностью видим в том случае, если поразрядные суммы кодов концов этого отрезка равны нулю: .

Если отрезок не является полностью видимым, то он может являться либо частично видимым, либо полностью невидимым.

Возникает необходимость вновь «упростить» задачу и найти способ определить полную невидимость отрезка относительно заданного регулярного отсекателя.

Ниже приведены два вида полностью невидимых отрезков:

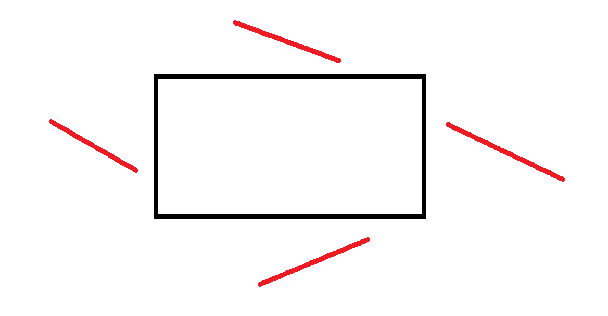


Рисунок , первый вид полностью невидимых отрезков

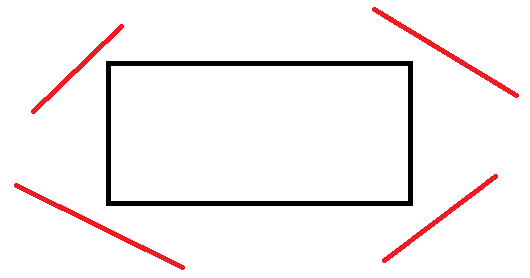


Рисунок , второй вид полностью невидимых отрезков

Сразу стоит обговорить тот факт, что для вида полностью невидимых отрезков, предоставленного на Рисунке 3, с помощью простого анализа кодов концов отрезков определить полную невидимость не получается.

Для определения полной невидимости отрезков, которые располагаются относительно отсекателя вполне определённым способом (Рисунок 2: полностью ниже нижней границы отсекателя, полностью левее левой границы, полностью правее правой границы, полностью выше верхней границы), достаточно провести следующий простой тест:

*, где & - поразрядное логические «и», а - код первого конца отрезка и - код второго конца отрезка.*

При этом мы также имеем возможность определить частичную видимость следующим простым тестом:

*Для таких отрезков от нас потребуется найти вторую вершину видимой части. При этом потребуется искать только одну координату точки пересечения, так как вторая координата будет совпадать с параметром границы отсекателя (для верхней и нижней границы это будет значение ординаты, а для левой и правой границы – значение ординаты).*

Для поиска координаты точки пересечение воспользуемся следующими формулами:

*С помощью данного уравнения мы можем определить точку пересечения отрезка с левой или правой границей регулярного отсекателя.*

*То есть для левой границы будем иметь:*

*А для правой:*

*В этих формулах m – это тангенс угла наклона отрезка.*

*То есть для нижней границы будем иметь:*

*А для верхней границы:*

*В этих формулах m – это тангенс угла наклона отрезка.*

Таким образом, перейдём к изложению Простого алгоритма отсечения отрезка.

## Простой алгоритм отсечения отрезка регулярным отсекателем

1. Ввод координат отсекателя

2. Ввод координат концов отрезка

3. Вычисление кодов концов отрезка:

Вычисление сумм кодов концов отрезка

4. FLAG = 1 (1 – отрезок видимый, -1 – отрезок невидимый)

5. Задание начального значения тангенса угла наклона отрезка:

То есть предположение вертикальности данного отрезка.

6. Проверка полной видимости отрезка:

Если , то выполнить:

Перейти к пункту 31.

7. Вычисление логического произведения кодов концов отрезка .

8. Проверка тривиальной невидимости отрезка:

Если , то установить FLAG = -1 и перейти к пункту 31.

9. Проверка видимости первого конца отрезка:

Если , то и перейти к пункту 15.

10. Проверка видимости второго конца отрезка:

Если и перейти к пункту 15.

11. Установка начального значения шага отсечения

12. Вычисление текущего номера шага отсечения

13. Проверка завершения процедуры отсечения:

Если перейти к пункту 31.

14. Занесение в рабочую переменную Q координат i-ой вершины отрезка:

15. Определение расположения отрезка:

Если отрезок вертикальный, перейти к пункту 23 (отсутствует пересечение с левой или правой границами отсекателя).

16. Вычисление тангенса угла наклона отрезка

17. Проверка возможности пересечения с левой границей отсекателя:

Если , то переход к пункту 20.

18. Вычисление ординаты точки пересечения отрезка с левой границей отсекателя:

19. Проверка корректности найденного пересечения:

Если , переход к пункту 12.

20. Проверка возможности пересечения отрезка с правой границей отсекателя:

Если , то переход к пункту 23.

21. Вычисление ординаты точки пересечения с правой границей:

22. Проверка корректности найденного пересечения:

Если , переход к пункту 12.

23. Проверка горизонтальности отрезка:

Если переход к пункту 12.

24. Проверка возможности пересечения с верхней границей отсекателя:

Если переход к пункту 27.

25. Вычисление абсциссы точки пересечения с верхней границей:

26. Проверка корректности найденного пересечения:

Если .

27. Проверка возможности пересечения с нижней границей отсекателя:

Если переход к пункту 30 (отрезок невидим)

28. Вычисление абсциссы точки пересечения с нижней границей:

29. Проверка корректности найденного пересечения

Если .

30. Установка FLAG = -1 (отрезок невидим полностью, так как ни одно пересечение не оказалось корректным).

31. Проверка признака видимости:

Если

32. Конец

# Практическая часть

Заранее хотелось бы отметить, что с возможностями языка Python алгоритм претерпел некоторые структурные (но ни в коем случае не алгоритмические) изменения. Одно из таких изменений – оператор continue, который позволяет непосредственно перейти в начало цикла при его вызове.

## Программная реализация алгоритма на ЯП Python

def getBinCodes(curLine, leftSide, rightSide, botSide, topSide):

firstPoint = 0b0000

secondPoint = 0b0000

if curLine[0][0] < leftSide:

firstPoint += 0b1000

if curLine[0][0] > rightSide:

firstPoint += 0b0100

if curLine[0][1] > botSide:

firstPoint += 0b0010

if curLine[0][1] < topSide:

firstPoint += 0b0001

if curLine[1][0] < leftSide:

secondPoint += 0b1000

if curLine[1][0] > rightSide:

secondPoint += 0b0100

if curLine[1][1] > botSide:

secondPoint += 0b0010

if curLine[1][1] < topSide:

secondPoint += 0b0001

return firstPoint, secondPoint

def simpleAlgCut(linesArray, cutterArray):

finalArray = []

if cutterArray[0][0] < cutterArray[2][0]:

leftSide = cutterArray[0][0]

rightSide = cutterArray[2][0]

else:

rightSide = cutterArray[0][0]

leftSide = cutterArray[2][0]

if cutterArray[0][1] < cutterArray[2][1]:

topSide = cutterArray[0][1]

botSide = cutterArray[2][1]

else:

botSide = cutterArray[0][1]

topSide = cutterArray[2][1]

for line in range(len(linesArray)):

binCodes = getBinCodes(linesArray[line], leftSide, rightSide, botSide, topSide)

firstPoint = binCodes[0]

secondPoint = binCodes[1]

fCoordinates = linesArray[line][0]

sCoordinates = linesArray[line][1]

if firstPoint == 0 and secondPoint == 0:

finalArray.append(linesArray[line])

continue

if firstPoint & secondPoint:

continue

flag = 1

i = -1

if not firstPoint:

result = [fCoordinates]

workVar = sCoordinates

i = 1

flag = 0

elif not secondPoint:

result = [sCoordinates]

workVar = fCoordinates

i = 1

flag = 0

else:

result = []

while i <= 1:

if flag:

workVar = linesArray[line][i]

i += 1

if fCoordinates[0] != sCoordinates[0]:

tan = (sCoordinates[1] - fCoordinates[1]) / (sCoordinates[0] - fCoordinates[0])

if workVar[0] <= leftSide:

crosser = tan \* (leftSide - workVar[0]) + workVar[1]

if (crosser <= botSide) and (crosser >= topSide):

result.append([leftSide, int(np.round(crosser))])

continue

elif workVar[0] >= rightSide:

crosser = tan \* (rightSide - workVar[0]) + workVar[1]

if (crosser <= botSide) and (crosser >= topSide):

result.append([rightSide, int(np.round(crosser))])

continue

if fCoordinates[1] != sCoordinates[1]:

if workVar[1] <= topSide:

crosser = (topSide - workVar[1])/tan + workVar[0]

if (crosser >= leftSide) and (crosser <= rightSide):

result.append([int(np.round(crosser)), topSide])

continue

elif workVar[1] >= botSide:

crosser = (botSide - workVar[1])/tan + workVar[0]

if (crosser >= leftSide) and (crosser <= rightSide):

result.append([int(np.round(crosser)), botSide])

continue

if result:

finalArray.append(result)

drawLines(finalArray)

## Пользовательский интерфейс

## Демонстрация работы алгоритма

# Исследование временных характеристик