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

Москва

2020 г.

Оглавление

[Цель работы 3](#_Toc41154214)

[Техническое задание 3](#_Toc41154215)

[Теоретическая часть 3](#_Toc41154216)

[Отсечение отрезка произвольным выпуклым отсекателем 3](#_Toc41154217)

[Видимость и невидимость отдельно взятой точки 5](#_Toc41154218)

[Нахождение точек пересечения произвольного отрезка с границами отсекателя. 7](#_Toc41154219)

[Выбор точек пересечения 9](#_Toc41154220)

[Способ определения внутренней нормали к заданной стороне многоугольника 12](#_Toc41154221)

[Способ определения выпуклости многоугольника 13](#_Toc41154222)

[Алгоритм Кируса-Бека 14](#_Toc41154223)

[Практическая часть 15](#_Toc41154224)

[Программная реализация алгоритма на ЯП Python 15](#_Toc41154225)

[Пользовательский интерфейс 18](#_Toc41154226)

[Демонстрация работы алгоритма 18](#_Toc41154227)

# Цель работы

Изучение и программная реализация алгоритма отсечения отрезка.

# Техническое задание

Алгоритм отсечения отрезка выпуклым отсекателем – алгоритм Кируса-Бека.

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.

# Теоретическая часть

## Отсечение отрезка произвольным выпуклым отсекателем

Поставленная задача отсечения в данном алгоритме, по сути, решается с использованием двух скалярных произведений, в которых задействован вектор нормали.

Легко заметить, что, в случае отсечения отрезка нерегулярным отсекателем, использование кодов концов отрезков (которые мы задействовали в рассмотрении предыдущего алгоритма отсечения отрезка регулярным отсекателем) невозможно. Данный факт вынуждает нас искать точки пересечения со сторонами отсекателя (или с их продолжениями). Количество точек пересечения будет равняться количеству сторон заданного многоугольного отсекателя, причём сам отрезок может пересекать многоугольник лишь в двух точках, которые нам и предстоит выбрать.

Таким образом, задача сводится к нахождению точек пересечения заданного отрезка с границами отсекателя и правильному определению из всех найденных точек пересечения вершины начала видимого отрезка и вершины конца видимого отрезка. В дополнении также потребуется определить полную невидимость отрезка. При этом стоит отметить, что определить полную видимость отрезка также не является некоторой тривиальной задачей, поэтому для тоже понадобится выполнить полный цикл нахождения точек пересечения. К моменту окончания работы алгоритма неизменным, что и будет означать полную видимость рассматриваемого отрезка.

Для идентификации полностью видимых или полностью невидимых отрезков удобно использовать параметрическую форму задания отрезка:

Рассмотрим следующий случай:



Рисунок , случай 1, полная видимость отрезка

В данном случае будем иметь:

– для нижней и левой границ.

– для верхней и правой границ.

Таким образом, для полностью видимого отрезка каждой точке пересечения соответствует недопустимое значение параметра.

Рассмотрим иной случай:



Рисунок , случай 2, полная невидимость отрезков

В данном случае точкам пересечения отрезков также соответствуют недопустимые значения параметра.

Получается, что простых способов определения полностью видимых или полностью невидимых отрезков предложено быть не может. То есть полностью видимые и полностью невидимые отрезки будут распознаваться уже по ходу работы алгоритма.

### Видимость и невидимость отдельно взятой точки

В процессе отсечения, как правило, требуется определять видимость или невидимость отдельно взятой точки. Рассмотрим следующий случай:



Рисунок , рассмотрение видимости точки

Задача состоит в определении видимости точки *A* относительно каждой из сторон отсекателя.

Рассмотрим ребро :

Построим внутреннюю нормаль к рассматриваемой стороне из произвольной её точки:



Рисунок , высота к рассматриваемой стороне

В качестве второго вектора построим вектор, начинающийся в произвольной точке рассматриваемого ребра и заканчивающийся в рассматриваемой точке:



Рисунок , вектор к рассматриваемой точки

Выбранную произвольную точку на рассматриваемой стороне назовём .

Рассмотрим скалярное произведение двух заданных векторов:

Если , то точка видима относительно рассматриваемой стороны, так как косинус угла между данными векторами в этом случае положителен, то есть лежит в диапазоне от до , не включая эти конечные значения.

Если , то точка лежит на границе отсекателя.

Если , то точка расположена по невидимую сторону отсекателя.

Таким образом имеем способ определение видимости отдельно взятой точки. Теперь же перейдём к рассмотрению решения задачи нахождения точек пересечения произвольного отрезка с границами отсекателя.

### Нахождение точек пересечения произвольного отрезка с границами отсекателя.

Рассмотрим следующий случай:



Рисунок , рассмотрение определения видимости отрезка

Определим вектор внутренней нормали к стороне отсекателя и вектор с началом в произвольной точке стороны к точке отрезка в области отсечения:



Рисунок , построение внутренней нормали и вектора от стороны к точке отрезка

Запишем скалярное произведение заданных векторов:

В соответствии с тем, что мы отметили выше можно записать:

*,* то точка отрезка видима относительно рассматриваемой границы отсекателя.

*,* то точка отрезка расположена на границе отсекателя.

Таким образом, было сформулировано условие для нахождения точки пересечения отрезка с границей отсекателя. Итак, из условия равенства нулю скалярного произведения вектора внутренней нормали и вектора, соединяющего произвольную точку границы отсекателя с точкой на рассматриваемом отрезке, и будет находиться значение параметра , которое соответствует искомой точке пересечения.

*,* то точка расположена по невидимую сторону отсекателя.

Приравняем скалярное произведение нулю:

Обозначим вектор следующим образом:

*Данный вектор является вектором направления отрезка. Его также называют директрисой отрезка.*

Обозначим вектор следующим образом:

Тогда заданное условие запишем следующим образом:

И теперь можем записать выражение для нахождения параметра ­­:

*В данном выражении знаменатель может равняться нулю. Это происходит в следующих частных случаях:*

*1. – отрезок вырожденный.*

*2. Векторы перпендикулярны. Это означает, что отрезок расположен параллельно рассматриваемой границе отсекателя. В данном случае нас интересует вопрос, по какую сторону от текущей границы отсекателя он расположен: по видимую или по невидимую сторону. При этом стоит отметить тот факт, что, если отрезок полностью невидим для одной стороны отсекателя, то он является полностью невидимым, так как чтобы быть полностью видимым отрезком, отрезок должен быть видим относительно каждой границы отсекателя. В ином случае отрезок точек пересечения с текущей границей не имеет и целесообразно перейти к поиску пересечения со следующей границей отсекателя.*

***Для определения того, по какую сторону параллельный границе отрезок находится, достаточно проверить на видимость произвольную точку отрезка.***

Знак скалярного произведения, стоящего в числителе, позволяет определить положение первой вершины отрезка относительно границы отсекателя и, следовательно, положение всего отрезка.

Обозначим скалярные произведения в числителе и знаменателе следующим образом:

Если , то отрезок расположен по видимую сторону. Если же , отрезок является полностью невидимым.

### Выбор точек пересечения

Как говорилось выше, в случае частичной видимости отрезка, находится столько точек пересечения, сколько сторон у отсекателя. Таким образом, для примера, для пятиугольного отсекателя будут определены параметры . От нас требуется определить, какая из найденных точек пересечения будет являться началом видимой части отрезка, а какая концом видимой части отрезка.

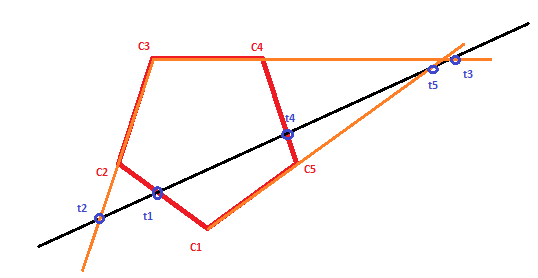


Рисунок , пример случая частичной видимости отрезка

Для удобства воспользуемся следующей картинкой, обозначающей интервал значений параметра :

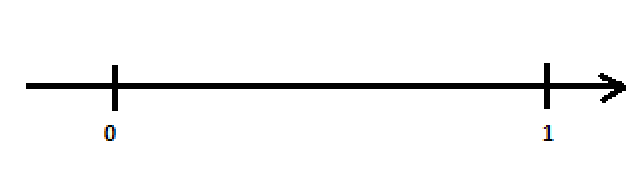
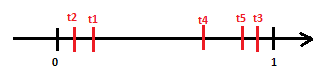


Рисунок , интервал значений t

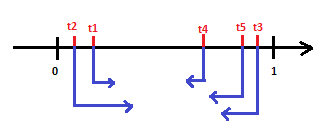
Занесём на заданный интервал значения найденных параметров:



Рисунок

*(в рассмотренном мною случае отсутствуют параметры >=1 и <=0, но в том случае, если такие параметры были бы, то располагались бы они соответственно за пределами интервала (0; 1), либо ан его концах)*

Покажем, в каком направлении при заданном параметре будет располагаться видимая часть отрезка.



Рисунок

Видно, что все найденные точки пересечения можно разделить на две группы: 1. Точки пересечения, расположенные ближе к началу отрезка. Они определяют начало видимой части отрезка.

2. Точки, расположенные ближе к концу отрезка. Они определяют конец видимой части отрезка.

То, в какую группу можно отнести точку, определяется по знаку скалярного произведения :

Если , то точка расположена ближе к началу отрезка.

Если , то точка расположена ближе к концу отрезка.

Из иллюстрации выше видно, что начало видимой части отрезка будет определяться наибольшим значением параметра из тех значений, которые определяют начало видимой части. В случае, указанном выше, таким параметром будет . Конец же видимой части отрезка будет определяться наименьшим значением параметра из тех значений, которые определяют конец видимой части. В случае, указанном выше, таким параметром будет .

Также стоит отметить следующий частный случай расположения отрезка относительно отсекаемой области:

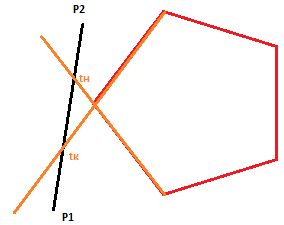


Рисунок , частный случай

Выполняя описанную процедуру нахождения точек пересечения заданного отрезка с границами отсекателя, как итог получим две точки, соответствующие параметрам и , определяющие начало и конец видимой части отрезка. То есть формально мы получаем два значения параметра, расположенных в интервале от нуля до единицы. Если не выполнить проверку того, что начало видимой части отрезка располагается до его конца, то будет получен неверный результат. То есть будет внесена следующая проверка:

*Следует отметить, следующие факты для дальнейшего описания алгоритма:*

*1. Изначально весь отрезок считается видимым, то есть .*

*2. Алгоритм работает правильно только в случае выпуклого отсекателя, то есть целесообразно добавить проверку выпуклости заданного отсекателя.*

### Способ определения внутренней нормали к заданной стороне многоугольника

Из курса аналитической геометрии нам известно, что равенство нулю скалярного произведения некоторых двух векторов означает, что эти векторы перпендикулярны. Данный факт мы и будем использовать для того, чтобы определить нормаль к i-ой стороне отсекателя.

Пусть некоторый вектор, нормаль к которому мы ищем, а вектор искомая нормаль.

Запишем формулу для вычисления скалярного произведения двух этих векторов:

Так как нам требуется найти направление нормали, следовательно можем задать одну из проекций нормали = 1.

Следовательно получим:

При этом не стоит пренебрегать перед вычислением выражения сверху проверкой на равенство нулю, так как в таком случае требуется задать вектор (0, 1) как вектор нормали заданной стороны.

При этом, после выполнения вычислений, стоит проверить также, была определена внутренняя или внешняя нормаль, и во втором случае вернуть вектор, обратный найденному.

### Способ определения выпуклости многоугольника

Для того, чтобы определить, является ли многоугольник выпуклым, его стороны стоит рассматривать как векторы:

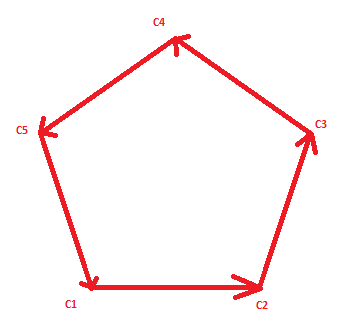


Рисунок , пример рассмотрения многоугольника

Для определения выпуклости многоугольника мы вычисляем векторное произведение двух сторон, связанных с некоторой вершиной и инцидентных этой вершине.

Для того, чтобы многоугольник был выпуклым, для сторон этого многоугольника должно выполняться следующее условие:

При этом векторы внутренних нормалей в этом случае ориентированы влево от направления обхода.

Если все , то многоугольник вырожденный.

Если среди таких произведений есть как положительные, так и отрицательные величины, то многоугольник невыпуклый.

При этом стоит отметить, что в своей реализации я несколько «упростил» условия выпуклости данного способа (чтобы дополнительно не определять «правильное» направления векторов, определённых сторонами многоугольника), так как мне не требовалось дополнительно опознавать направление внутренних нормалей (определение нормалей было вынесено в другой модуль), поэтому анализируется факт того, что все произведения не изменяют своего знака, либо являются нулевыми значениями.

## Алгоритм Кируса-Бека

1. Ввод исходных данных: точки отрезка

2. Ввод числа сторон выпуклого многоугольника и координат его вершин (массив )

3. Вычисление директрисы заданного отрезка:

4. Инициализация пределов значений параметра при условии, что отрезок полностью видим:

5. Начало цикла по всем сторонам отсекателя.

Для каждой i-ой стороны отсекателя выполнить следующие действия:

5.1. Вычисление вектора внутренней нормали к очередной i-ой стороне отсекателя -

5.2. Определение граничной точки каждой стороны отсекателя

5.3. Вычисление вектора

5.4. Вычисление скалярного произведения векторов:

5.5. Вычисление скалярного произведения векторов:

5.6. Проверка на равенство нулю скалярного произведения (вырождение отрезка в точку или его параллельность стороне отсекателя).

Если , тогда переход к пункту 5.9.

5.7. Вычисление параметра :

5.8. Определение верхнего и нижнего пределов параметра :

5.8.1. Поиск нижней границы параметра , если :

Если , то переход к пункту 7 (отрезок невидим, так как нижний предел параметра , превышает единицу и пересечение с отсекателем имеет место для самого отрезка, а для его продолжения за вершину ).

Если , то (выбор максимального значения из текущего значения параметра и ранее вычисленного значения нижней границы параметра).

1.8.2. Поиск верхней границы параметра , если :

Если , то переход к пункту 7 (отрезок невидим, так как верхний предел параметра отрицателен и пересечение с отсекателем имеет место не для самого отрезка, а для его продолжения за вершину ).

Если , то (выбор минимального значения из текущего значения параметра и раннее вычисленного значения верхней границы параметра).

5.9. Проверка видимости точки, в которую выродился отрезок, или проверка видимости произвольной точки отрезка в случае его параллельности стороне отсекателя: если , то отрезок (точка) невидим и переход к пункту 7. Если , то отрезок (точка) видим(-а) относительно текущей стороны отсекателя и переход к пункту 5.10.

5.10. Конец цикла по сторонам отсекателя.

6. Проверка фактической видимости отсечённого отрезка. Если , то переход к пункту 7, иначе визуализация отрезка в интервале от до .

7. Конец алгоритма.

# Практическая часть

## 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] - fPoint[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 cutOne(line, numOfSides):

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

topLimit = 0

bottomLimit = 1

for i in range(-2, numOfSides - 2):

norm = normal(cutterArray[i], cutterArray[i + 1], cutterArray[i + 2])

wVec = [line[0][0] - cutterArray[i][0], line[0][1] - cutterArray[i][1]]

dirScal = scalProd(directrix, norm)

wScal = scalProd(wVec, norm)

if dirScal == 0:

if wScal < 0:

break

else:

continue

parameter = - wScal / dirScal

if dirScal > 0:

topLimit = max(topLimit, parameter)

elif dirScal < 0:

bottomLimit = min(bottomLimit, parameter)

if topLimit > bottomLimit:

break

if topLimit <= bottomLimit:

return [[round(line[0][0] + directrix[0] \* topLimit), round(line[0][1] + directrix[1] \* topLimit)],

[round(line[0][0] + directrix[0] \* bottomLimit), round(line[0][1] + directrix[1] \* bottomLimit)]]

return []

def CyrusBeckAlg(linesArray, cutterArray):

if not isConvex(cutterArray):

makeConvexError()

return

numOfSides = len(cutterArray)

drawArr = []

for line in linesArray:

cuted = cutOne(line, numOfSides)

if cuted:

drawArr.append(cuted)

drawLines(drawArr)

return

def drawLines(array):

global img, curColorLines, curColorCuted, crutch

temp = curColorLines

crutch = curColorLines

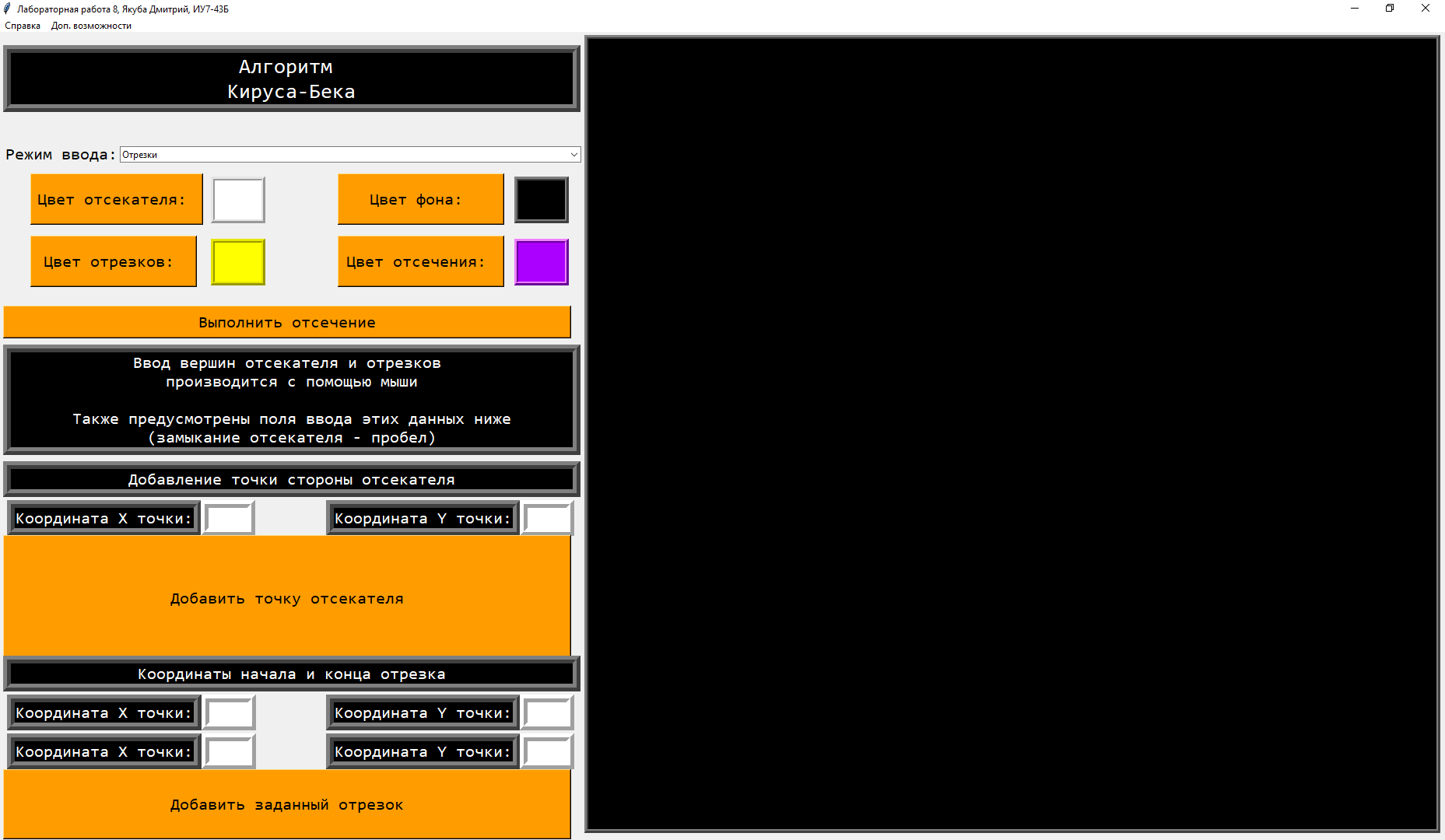
curColorLines = curColorCuted

for line in array:

digitBresenhamForCuted(img, line[1][0], line[1][1], line[0][0], line[0][1])

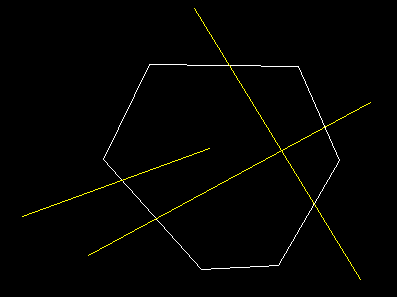
curColorLines = temp

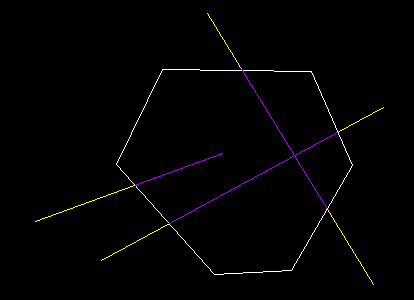
## Пользовательский интерфейс



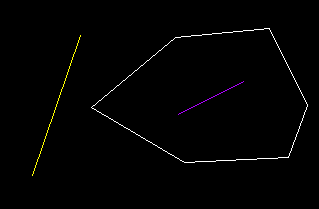
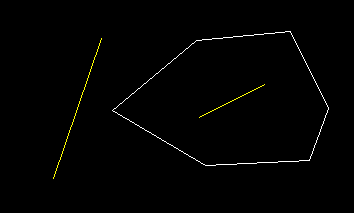
## Демонстрация работы алгоритма

Общий случай:

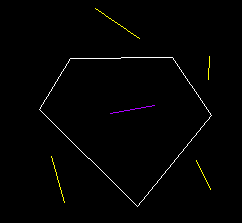
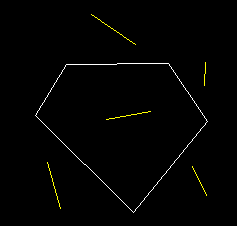




Частный случай невидимости отрезка с проверкой (здесь и далее внутренний отрезок приведён для того, чтобы не быть голословным и показать, что закраска произошла):



Случай полной невидимости отрезка:



И случай невидимости параллельного стороне отрезка: