|  |  |
| --- | --- |
| 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 № 6**

|  |  |
| --- | --- |
| **Тема** Реализация и исследование алгоритма построчного затравочного заполнения сплошных областей  **Студент** Якуба Д. В.  **Группа** ИУ7-43  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель** Куров А. В. |  |

Москва

2020 г.

Оглавление

[Цель работы 3](#_Toc39329704)

[Техническое задание 3](#_Toc39329705)

[Теоретическая часть 3](#_Toc39329706)

[Алгоритм построчного затравочного заполнения 3](#_Toc39329707)

[Практическая часть 6](#_Toc39329708)

[Программная реализация алгоритма на ЯП Python 6](#_Toc39329709)

[Пользовательский интерфейс 8](#_Toc39329710)

[Демонстрация работы алгоритма 8](#_Toc39329711)

[Исследование временных характеристик 17](#_Toc39329712)

# Цель работы

Реализация и исследование алгоритма построчного затравочного заполнения.

# Техническое задание

Необходимо обеспечить ввод произвольной многоугольной области, содержащей произвольное количество отверстий. Ввод (вершин многоугольника) производить с помощью мыши, при этом для удобства пользователя должны отображаться ребра, соединяющие вводимые вершины. Предусмотреть ввод горизонтальных и вертикальных ребер. Должен быть предусмотрен ввод затравочной точки.

Пользователь должен иметь возможность задания цвета заполнения.

Работа программы должна предусматривать два режима – с задержкой и без задержки.

Режим с задержкой должен позволить проследить выполняемую последовательность действий.

(Задержку целесообразно выполнять после обработки очередной строки).

Обеспечить замер времени выполнения алгоритма (без задержки, с выводом на экран только окончательного результата).

# Теоретическая часть

В отличие от уже рассмотренных алгоритмов заполнения, в алгоритмах затравочного заполнения мы действуем несколько иначе. Одним из главных отличий является наличие в исходных данных первого затравочного пиксела, который лежит внутри закрашиваемой области, с него и будет начинаться работа алгоритма.

Также следует отметить, что алгоритмы затравочного заполнения справляются с заполнением не только многоугольных областей, но и произвольных областей, которые могут быть ограничены произвольной кривой (что будет продемонстрировано).

## Алгоритм построчного затравочного заполнения

Алгоритм построчного затравочного заполнения является гранично-заполняющим, то есть с его помощью мы можем заполнить гранично-определённую область. Вследствие этого (по отношению к первому пункту следующего списка), в исходных данных для данного алгоритма мы будем иметь следующее:

1. Информация о границах заполняемой области

2. Затравочный пиксел

3. Цвет границы

4. Цвет заполнения

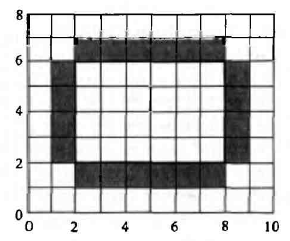
Одним из преимуществ данного алгоритма над рассмотренным ранее на лекции простым алгоритмом затравочного заполнения является уменьшение размера стека - мы рассматриваем «необходимый минимум», так как теперь хранится всего лишь один затравочный пиксел для некоторого непрерывного интервала (непрерывный интервал – это группа примыкающих друг к другу пикселей, расположенных в одной сканирующей строке, которые ещё не закрашены, но ограниченны уже закрашенными или граничными пикселами).

Рисунок . Пример гранично-определённой области

В общем виде алгоритм построчного затравочного заполнения будет выглядеть следующим образом:

1. Ввод исходных данных (перечень см. выше)

2. Инициализация стека и занесение переданного затравочного пиксела в стек.

3. Пока стек не пуст выполнять следующее:

3.1. Извлечь пиксел из стека

3.2. Закраска пикселей текущей строки влево и вправо от извлечённого затравочного пиксела (включая сам пиксел) до встречи с граничным

3.3. Поиск и занесение в стек новых затравочных пикселей в интервале от левого крайнего закрашенного пиксела до крайнего правого закрашенного пиксела. Важно отметить, что в качестве затравочного пиксела в стек следует поместить самый правый найденный затравочный пиксел на рассматриваемом интервале.

Алгоритм закраски пикселей текущей строки влево и вправо от извлечённого затравочного пиксела до встречи с граничным:

Текущий пиксел = извлечённый пиксел из стека [*x, y*]

Пока и

Закрасить текущий пиксел цветом закраски

Выполнить возврат к изначально переданному затравочному пикселу

(требуется выполнить, для того чтобы не обрабатывать уже закрашенный пиксел)

Пока и

Закрасить текущий пиксел цветом закраски

Алгоритм нахождения новых затравочных пикселей на строке ниже закрашенной:

ВАЖНО: в следующем описание под «текущим пикселом» и «пикселом» имеется в виду пиксел с координатами

Пока выполнять следующие действия:

Пока и

Если , то

Если , то:

Если и , и

, то добавить в стек новый затравочный пиксел с координатами

иначе добавить в стек новый затравочный пиксел с координатами

Пока и

выполнять:

Если , то

Жёлтым цветом обозначено немаловажное условие, так как во многих книгах используется строгое неравенство в этом месте, что не совсем правильно, так как в частном случае алгоритм не справится с закраской области шириной в один пиксел.

Алгоритм для анализа следующей строки аналогичен, и в своей реализации требует только того, чтобы перед циклом мы выполнили операцию , если до этого мы шли по строке, расположенной ниже начальной, и в ином случае - , то есть переход к строке ниже начальной.

# Практическая часть

Важно заметить, что в моей реализации границы фигуры очерчиваются с использованием целочисленного алгоритма Брезенхема.

## Программная реализация алгоритма на ЯП Python

def seedFill(img, xSeed, ySeed):

stack = list()

stack.append([xSeed, ySeed])

while len(stack):

gotDot = stack.pop()

curX = gotDot[0]

curY = gotDot[1]

gotColor = img.get(curX, curY)

while gotColor != linesRGB and gotColor != seedRGB:

img.put(seedColor, (curX, curY))

curX -= 1

gotColor = img.get(curX, curY)

xLeft = curX + 1

curX = gotDot[0] + 1

gotColor = img.get(curX, curY)

while gotColor != linesRGB and gotColor != seedRGB:

img.put(seedColor, (curX, curY))

curX += 1

gotColor = img.get(curX, curY)

img.put(seedColor, (gotDot[0], curY, xRight + 1, curY + 1))

curX = xLeft

curY += 1

flag = False

while curX <= xRight:

gotColor = img.get(curX, curY)

while gotColor != linesRGB and gotColor != seedRGB and curX <= xRight:

flag = True

curX += 1

gotColor = img.get(curX, curY)

if flag:

if curX == xRight and gotColor != linesRGB and gotColor != seedRGB:

stack.append([curX, curY])

else:

stack.append([curX - 1, curY])

flag = False

xStart = curX

while (gotColor == linesRGB or gotColor == seedRGB) and curX < xRight:

curX += 1

gotColor = img.get(curX, curY)

if curX == xStart:

curX += 1

curX = xLeft

curY -= 2

flag = False

while curX <= xRight:

gotColor = img.get(curX, curY)

while gotColor != linesRGB and gotColor != seedRGB and curX <= xRight:

flag = True

curX += 1

gotColor = img.get(curX, curY)

if flag:

if curX == xRight and gotColor != linesRGB and gotColor != seedRGB:

stack.append([curX, curY])

else:

stack.append([curX - 1, curY])

flag = False

xStart = curX

while (gotColor == linesRGB or gotColor == seedRGB) and curX < xRight:

curX += 1

gotColor = img.get(curX, curY)

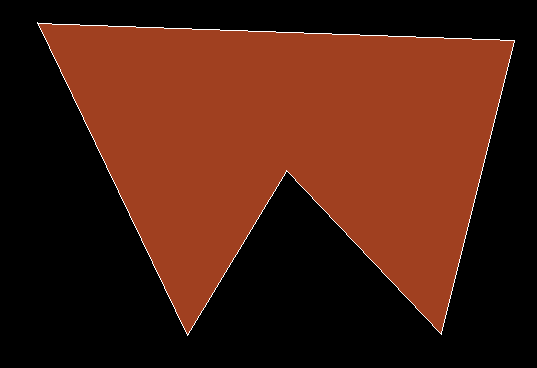
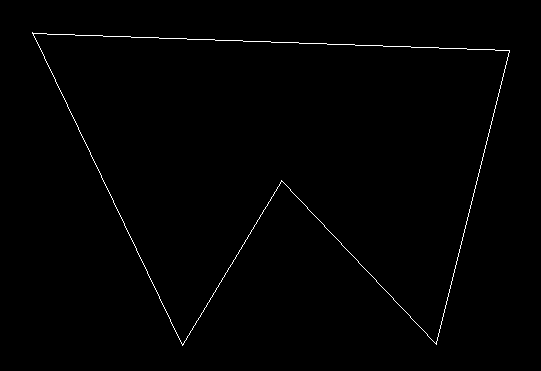
if curX == xStart:

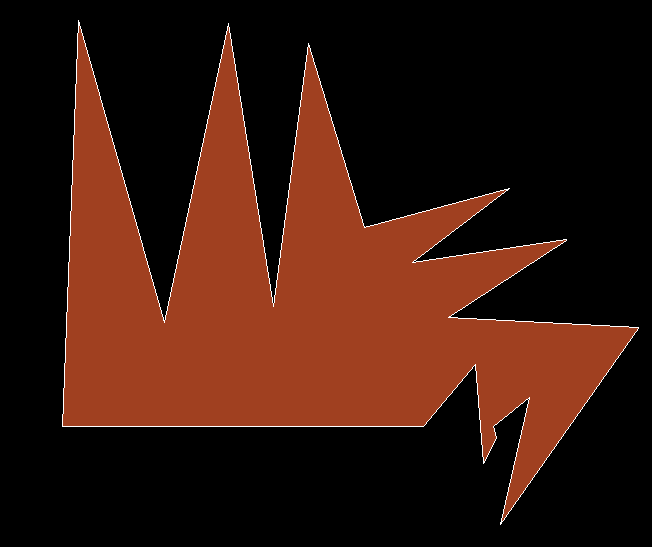
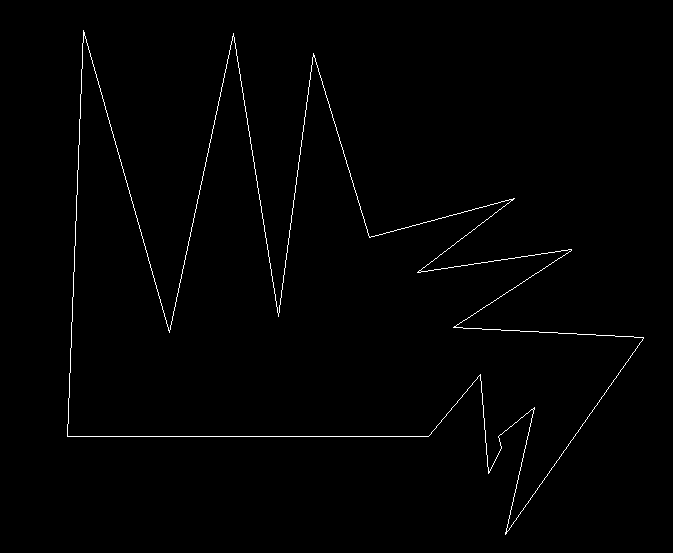
curX += 1

## Пользовательский интерфейс

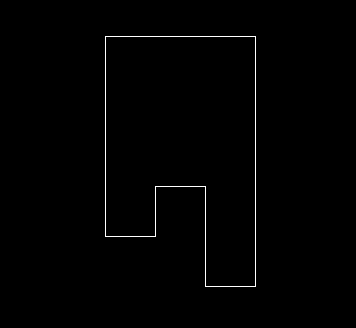
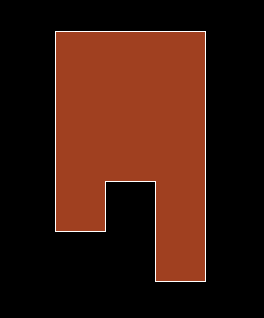


## Демонстрация работы алгоритма

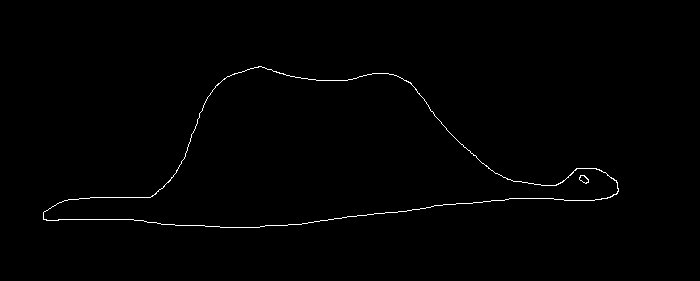


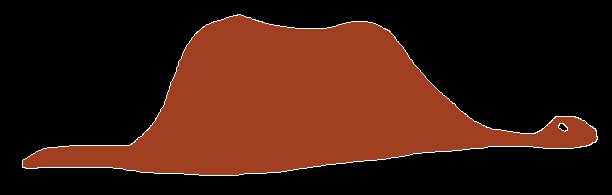


Работа с горизонтальными и вертикальными границами:

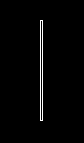
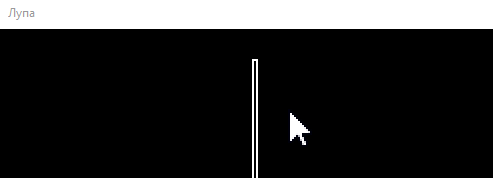
 

Работа с кривыми:

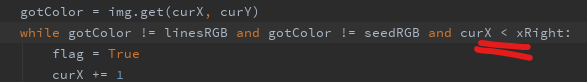




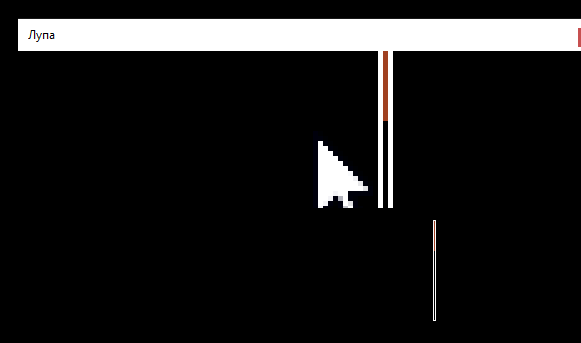
Область шириной в один пиксел:



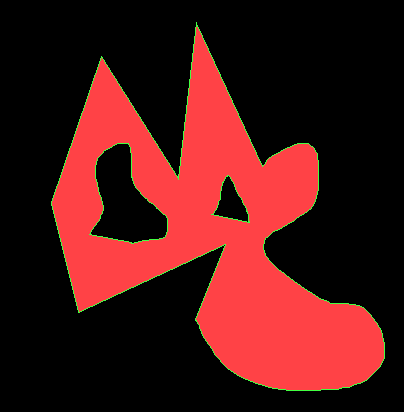
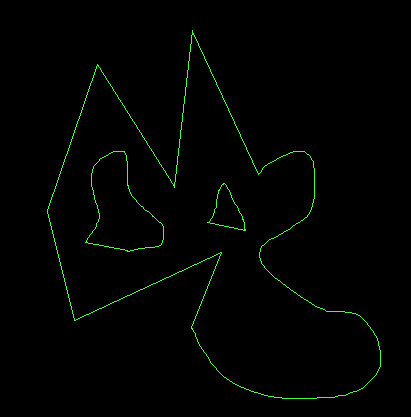
Если специально задать на одном из направлений обработки (в моём случае, в направлении вниз) нестрогое неравенство:



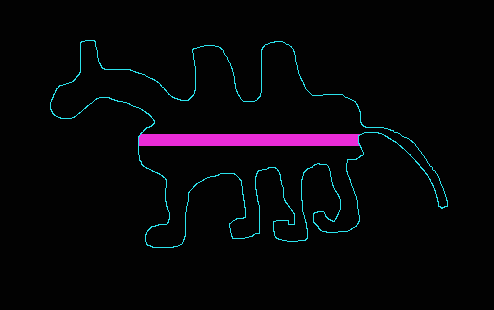
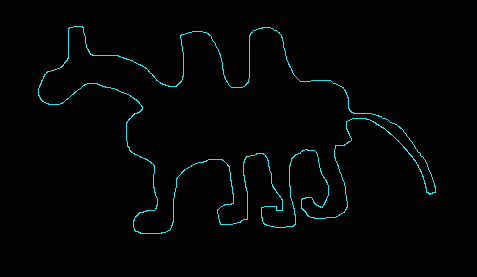
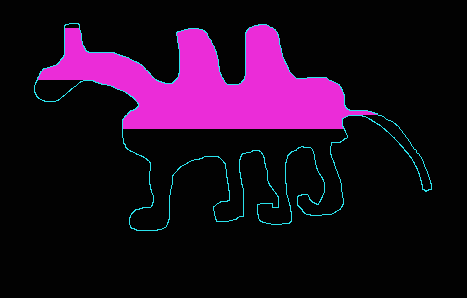
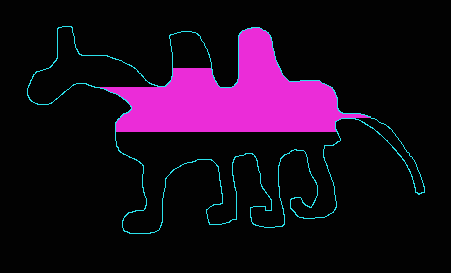
То получим:

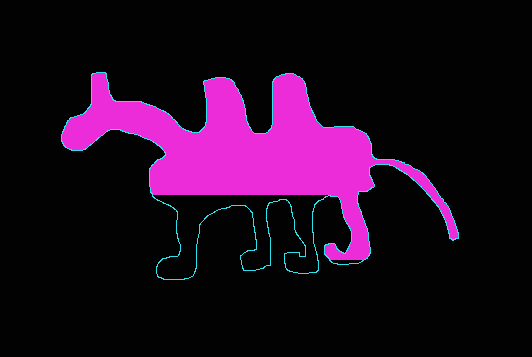
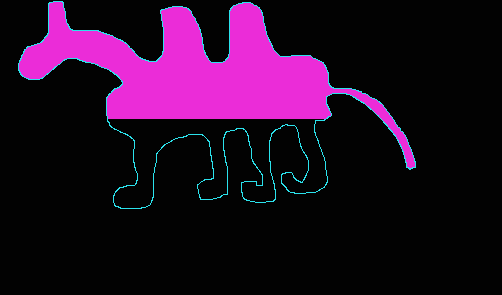
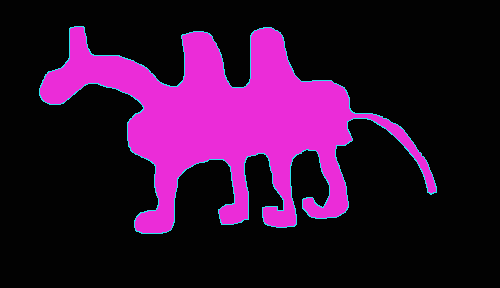
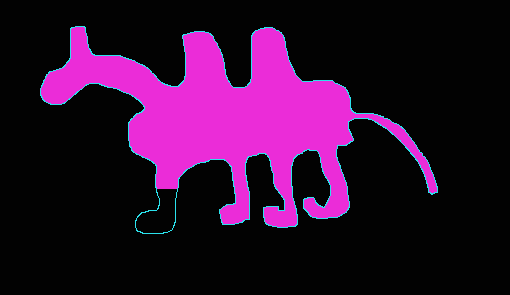


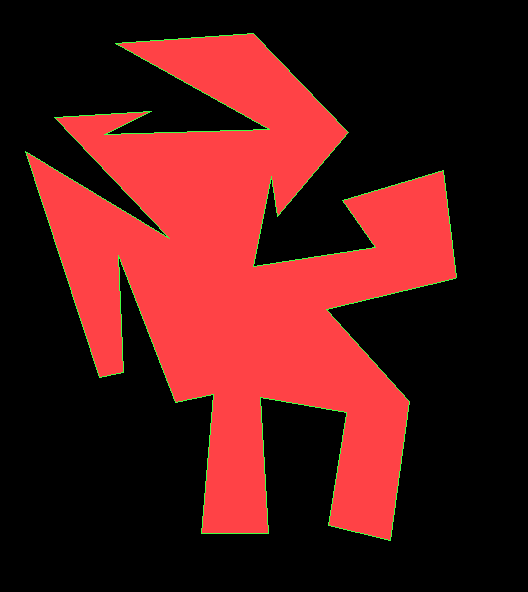
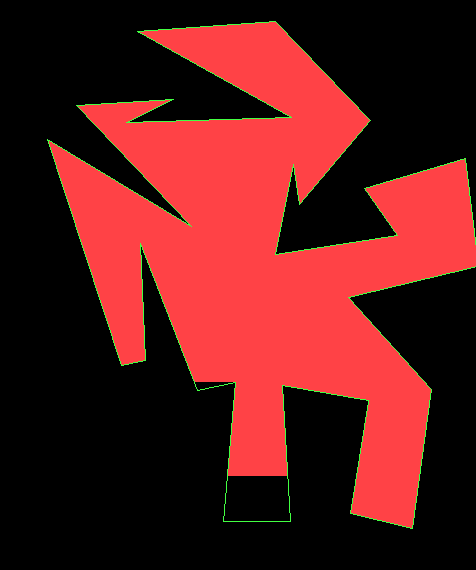
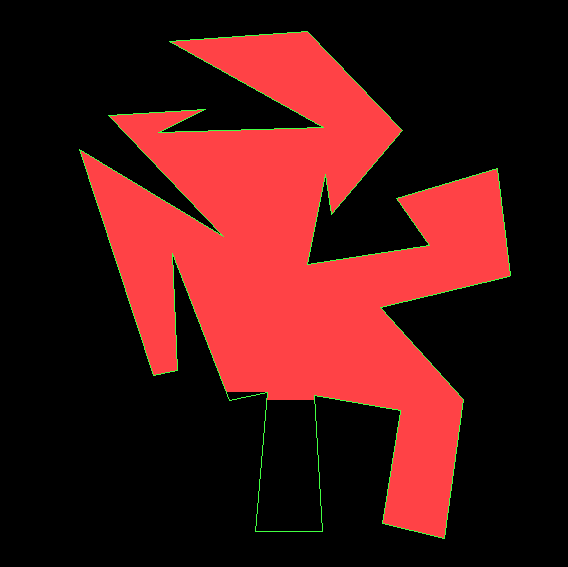
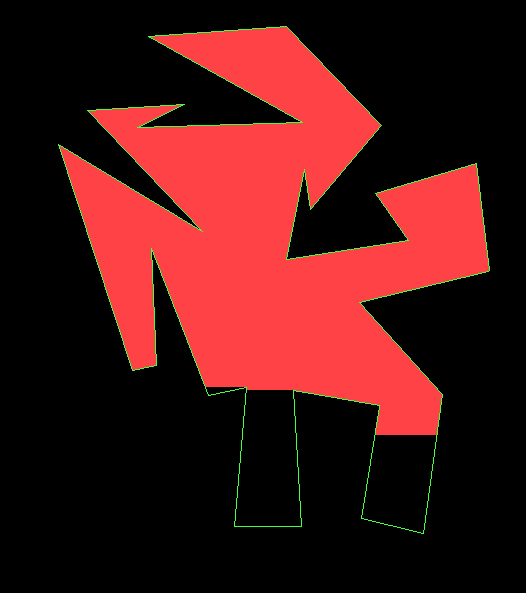
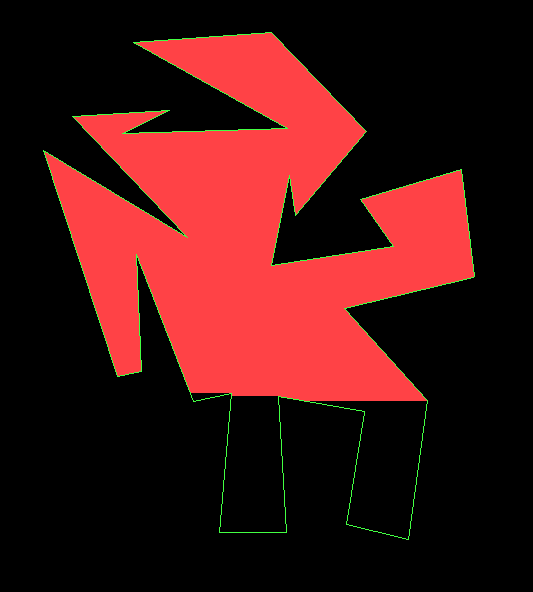
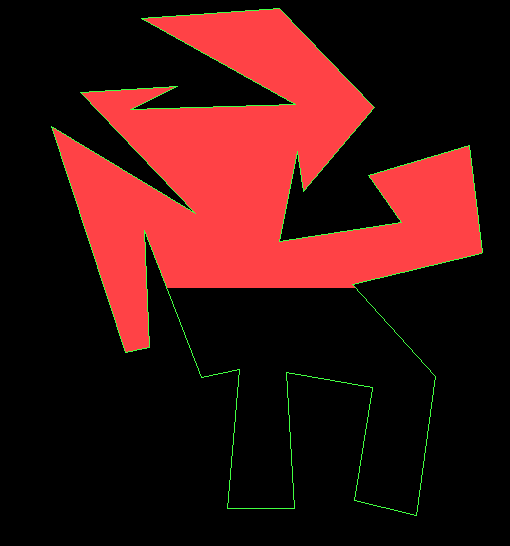
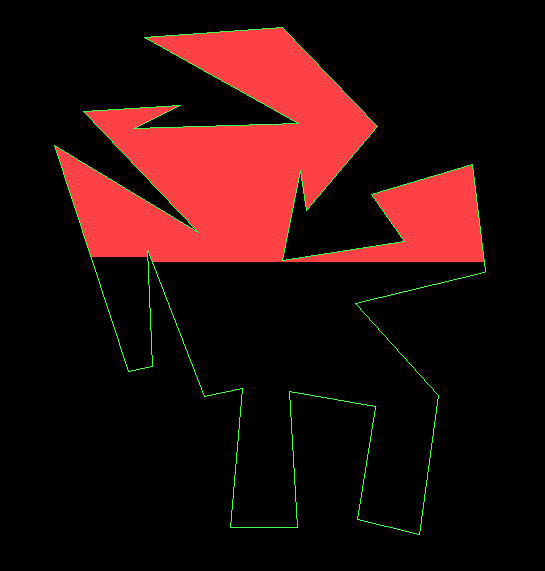
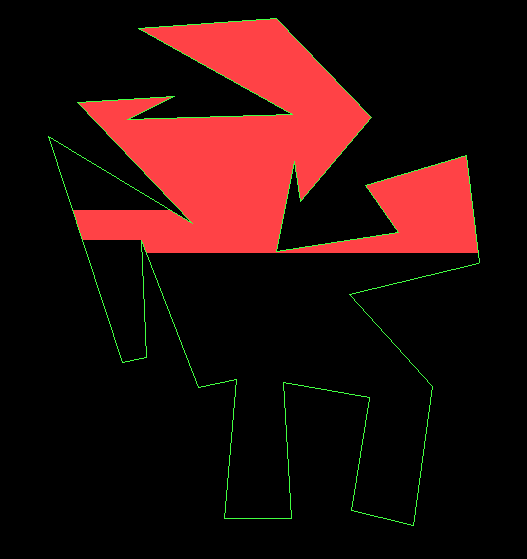
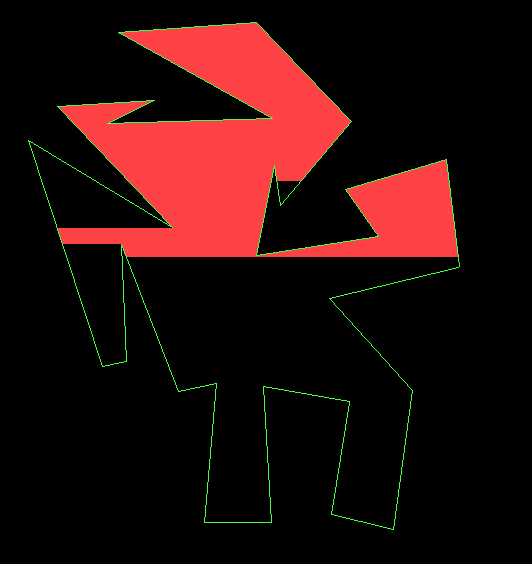
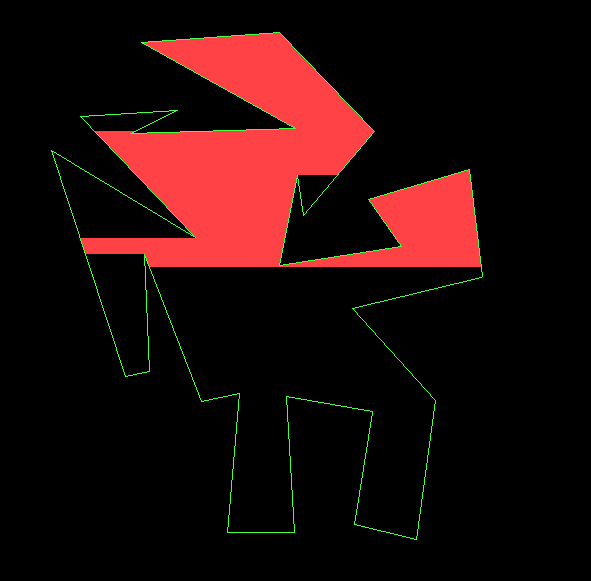
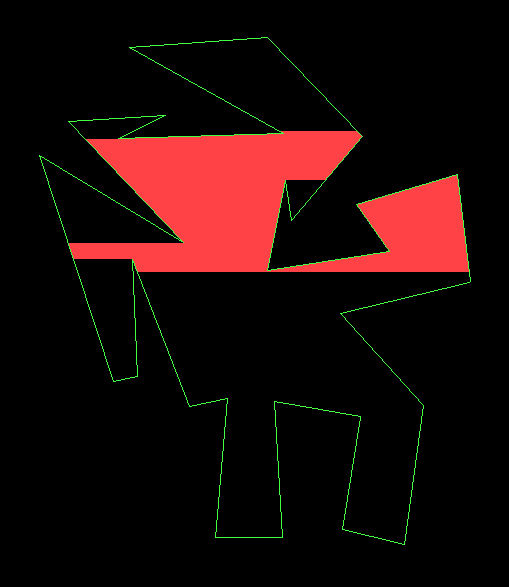
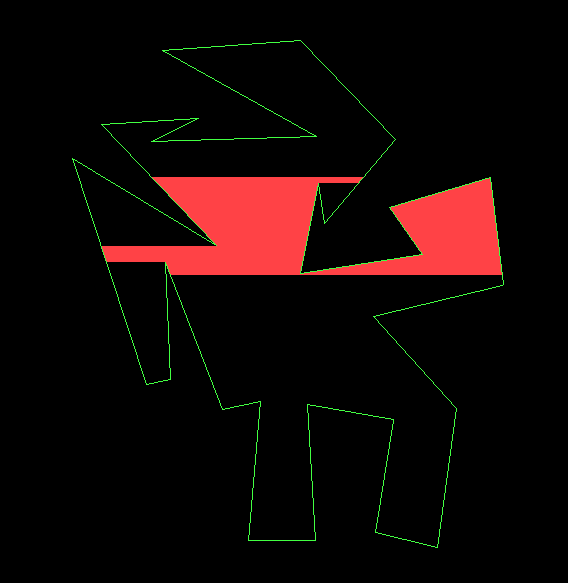
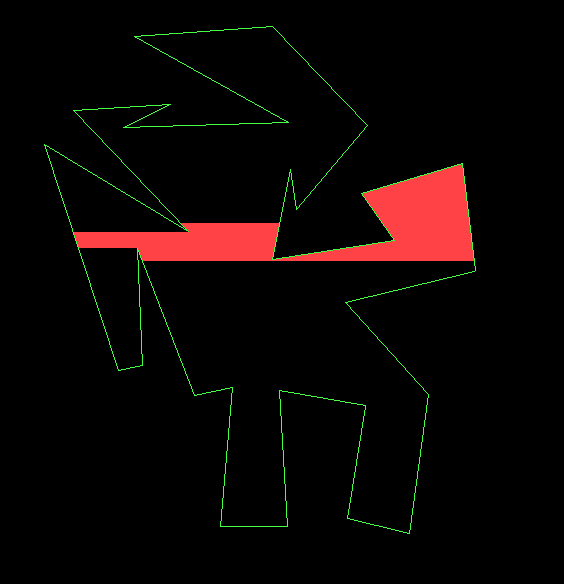
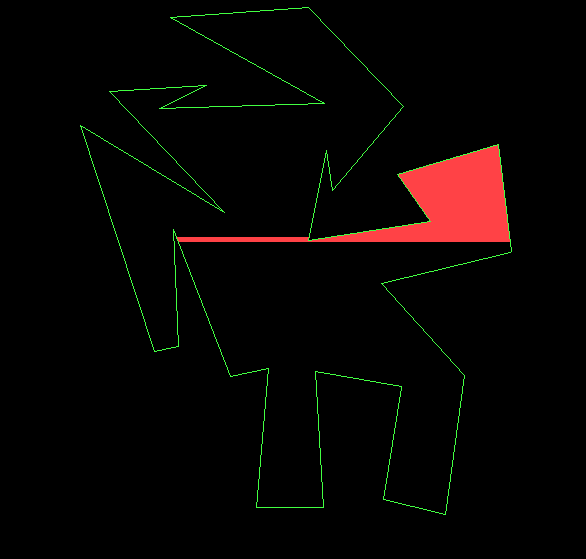
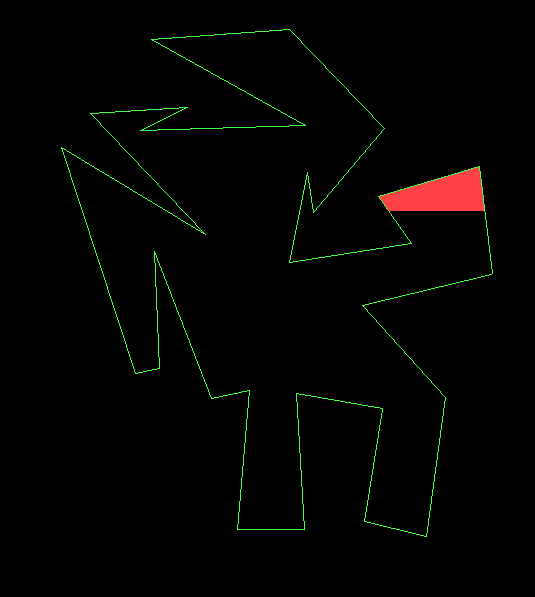
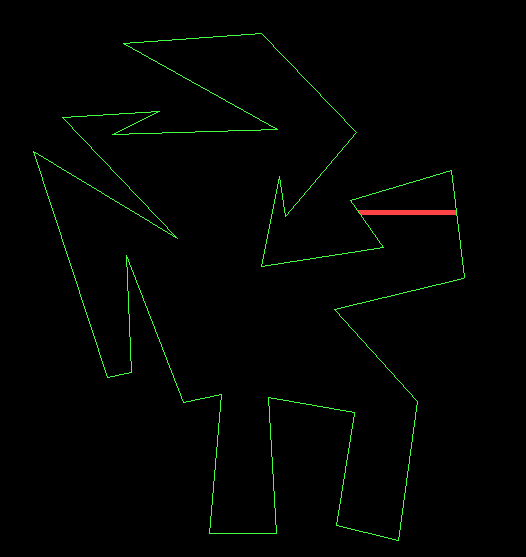
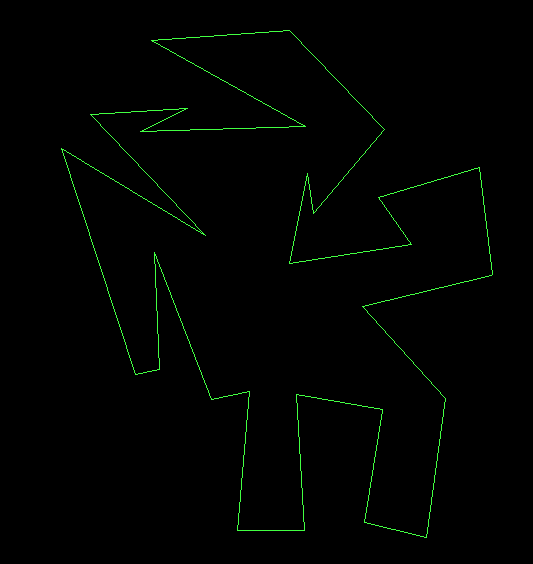
Что говорит нам о том, что нестрогое неравенство здесь действительно нужно, как и утверждалось на лекции. Придти к этому выводу можно и аналитически, но я бы хотел показать это и практически, что, собственно, и сделал.

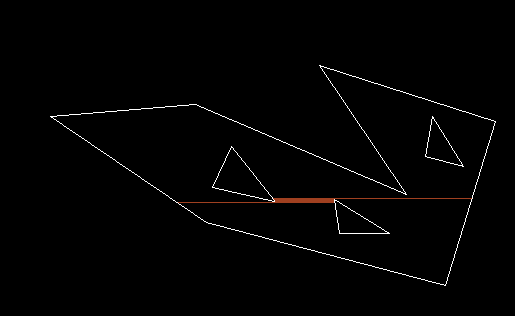
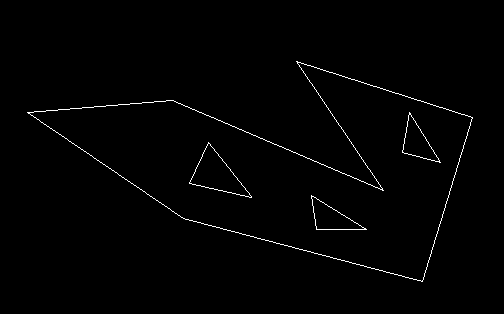


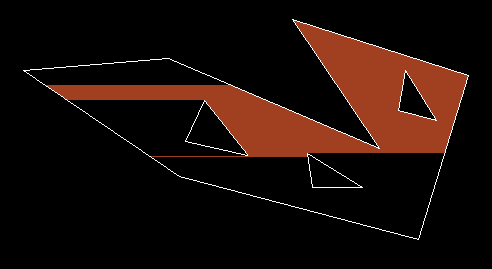
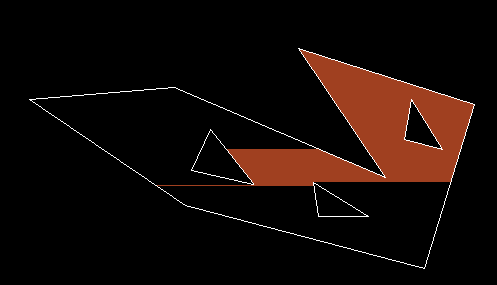
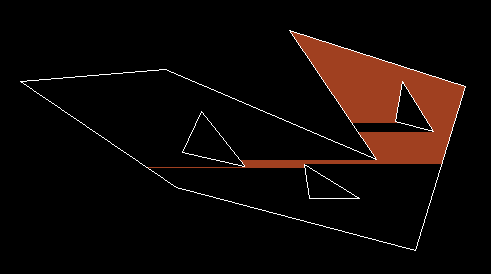
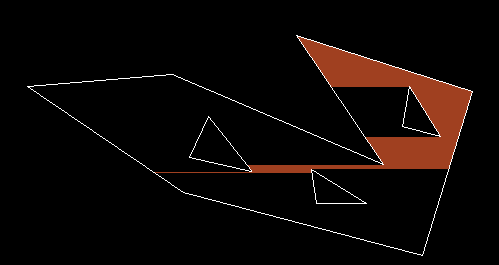
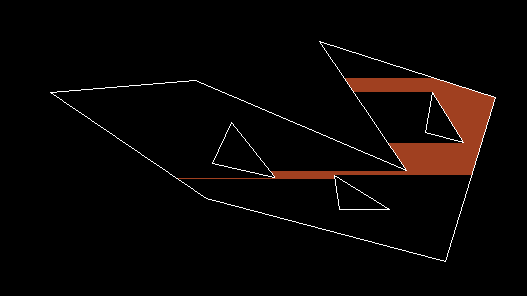
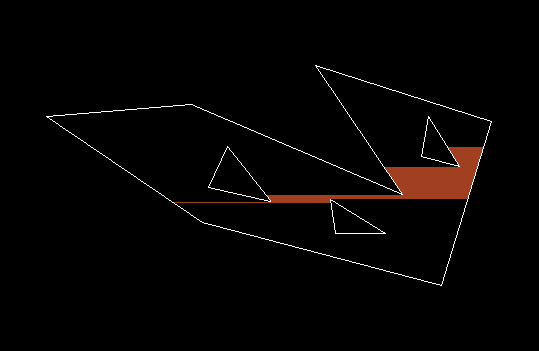
Выполнение с задержкой:

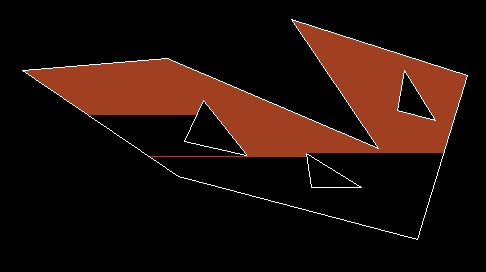
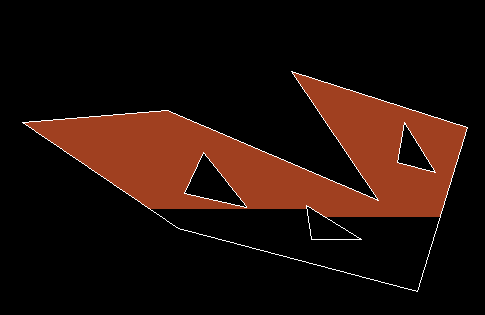
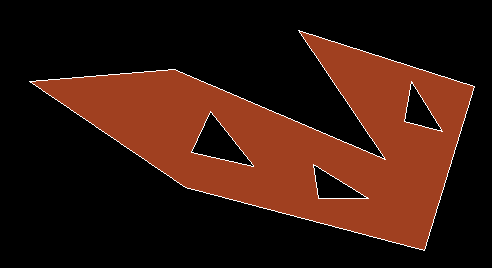
 







# Исследование временных характеристик

