|  |  |
| --- | --- |
| 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](#_Toc38969430)

[Техническое задание 3](#_Toc38969431)

[Теоретическая часть 3](#_Toc38969432)

[Алгоритм со списком рёбер и флагом 3](#_Toc38969433)

[Обрисовка контура 4](#_Toc38969434)

[Заполнение 4](#_Toc38969435)

[Практическая часть 5](#_Toc38969436)

[Программная реализация алгоритма на ЯП Python 5](#_Toc38969437)

[Пользовательский интерфейс 6](#_Toc38969438)

[Демонстрация работы алгоритма 10](#_Toc38969439)

[Исследование временных характеристик 18](#_Toc38969440)

# 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 растрового 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 пункта 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 на лекции простым алгоритмом затравочного заполнения является уменьшение размера стека, так как теперь мы рассматриваем «необходимый минимум» без невостребованной информации, так как теперь мы храним только один затравочный пиксел для 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).

# Практическая часть

## Программная реализация алгоритма на ЯП Python

def leadRoundEdge(img, edge):

if edge[0][1] == edge[1][1]:

return

if edge[0][1] > edge[1][1]:

edge[1], edge[0] = edge[0], edge[1]

stepX = (edge[1][0] - edge[0][0])/(edge[1][1] - edge[0][1])

curX = edge[0][0]

curY = edge[0][1]

while curY < edge[1][1]:

if img.get(int(curX) + 1, curY) != noteColorCheck:

img.put(noteColor, (int(curX) + 1, curY))

else:

img.put(noteColor, (int(curX), curY))

curX += stepX

curY += 1

def leadRoundFigure(img, edgesArray):

for figure in range(len(edgesArray)):

arrEnd = len(edgesArray[figure]) - 1

for i in range(arrEnd):

leadRoundEdge(img, edgesArray[figure][i])

leadRoundEdge(img, edgesArray[figure][arrEnd])

def rasterScanWithFlag(img, edgesArray, sides):

leadRoundFigure(img, edgesArray)

for curY in range(sides[0], sides[2] + 1):

curColor = curColorBackground

invColor = curColorLines

curPointScanString = sides[3]

for curX in range(sides[3], sides[1] + 3):

if img.get(curX, curY) == noteColorCheck:

img.put(curColor, (curPointScanString, curY, curX, curY + 1))

curColor, invColor = invColor, curColor

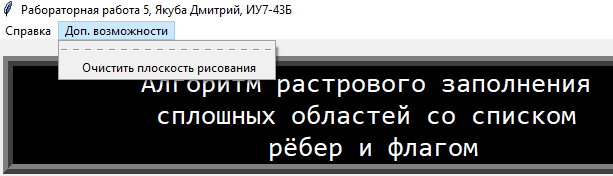
curPointScanString = curX

img.put(curColor, (curPointScanString, curY, curX, curY + 1))

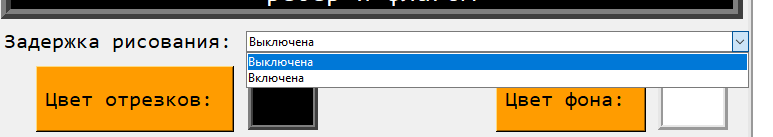
## Пользовательский интерфейс



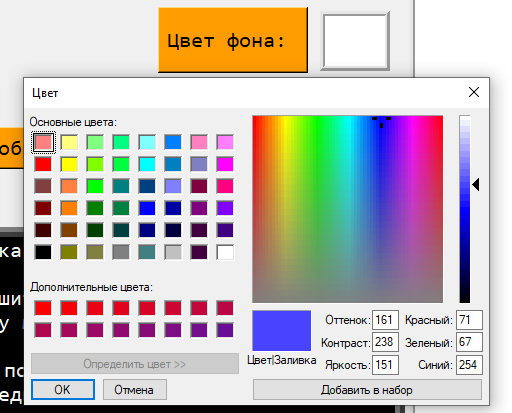
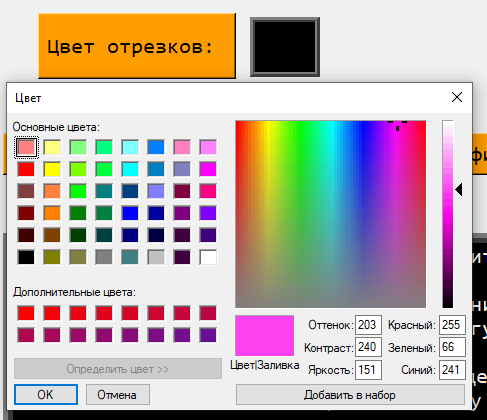
Предусмотрена очистка плоскости рисования

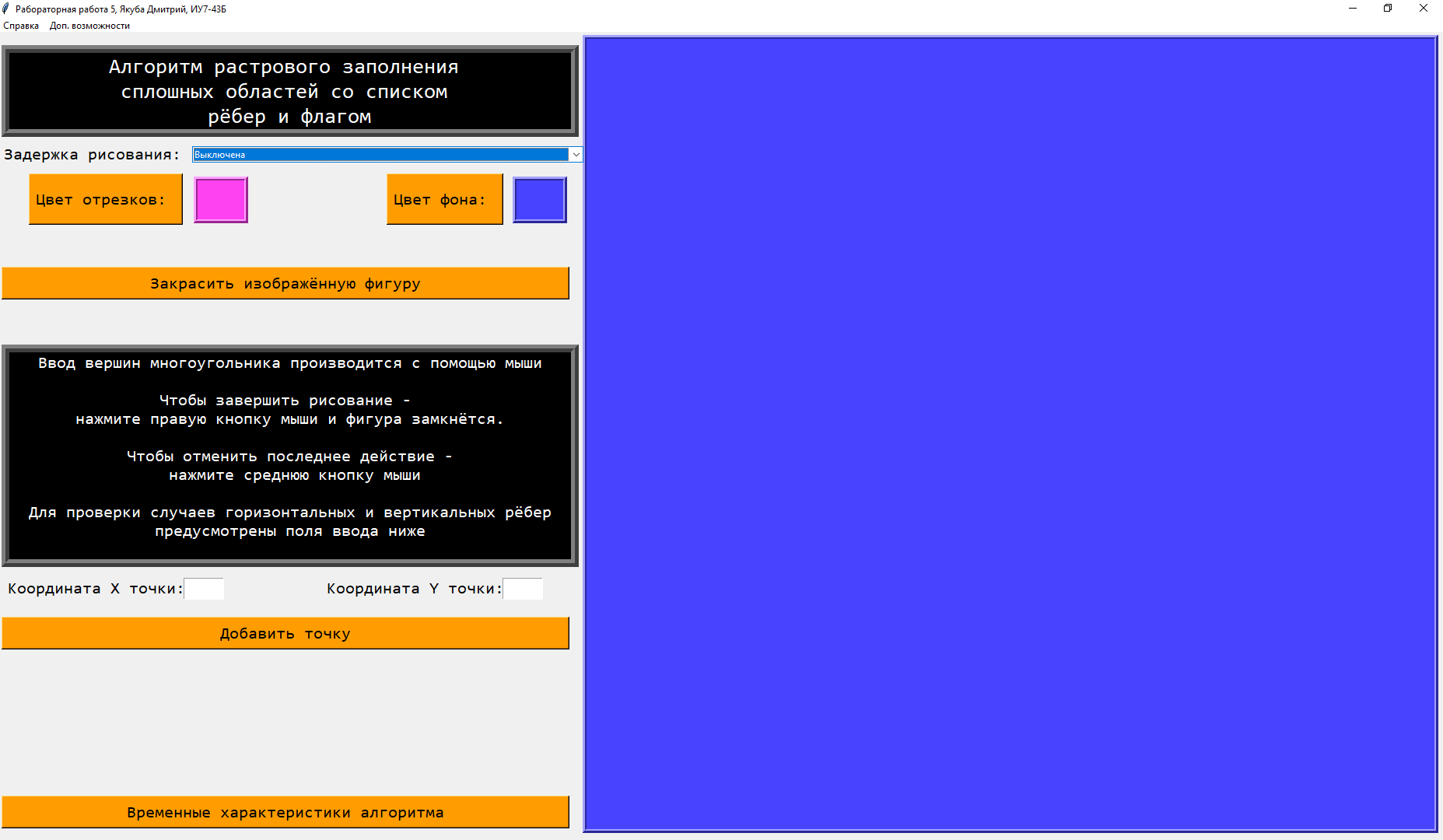


Предусмотрен выбор режима выполнения работы программы

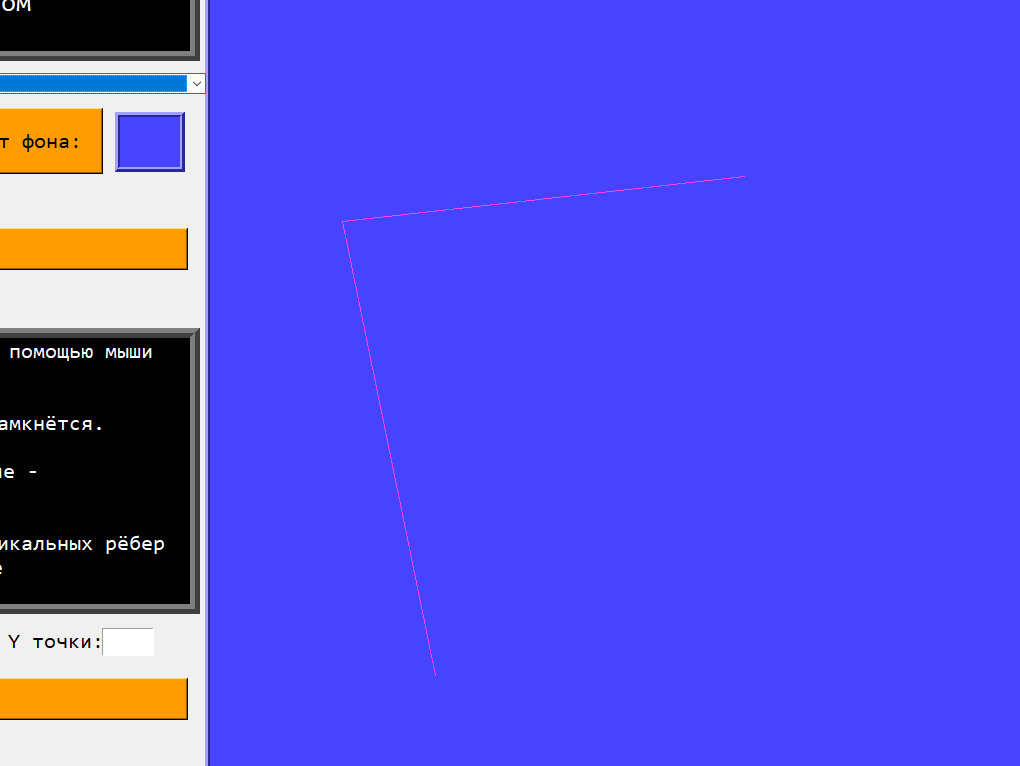


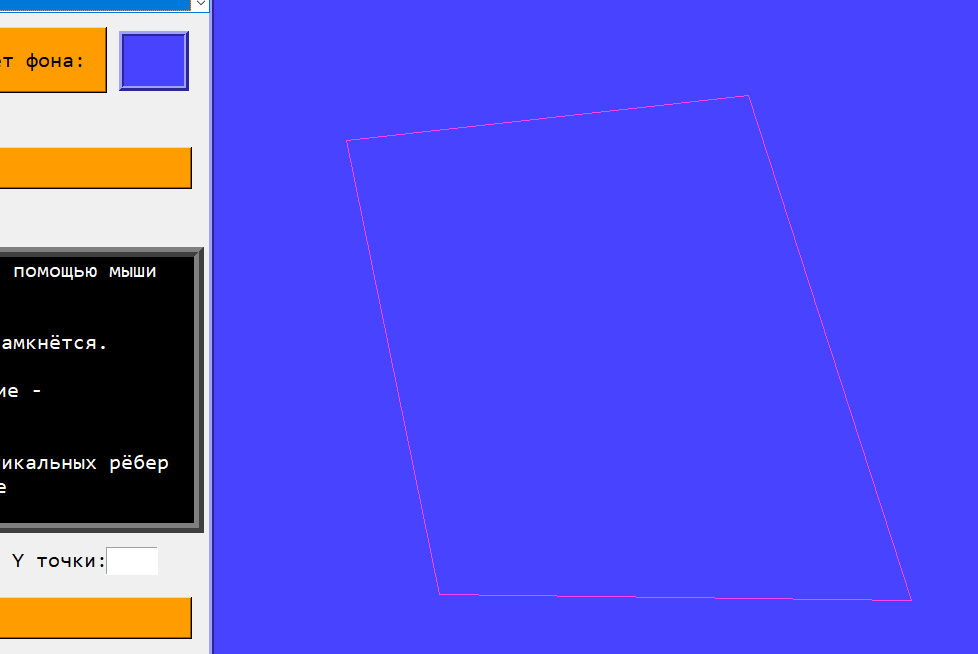
Предусмотрен выбор цвета отрезков и фона плоскости рисования

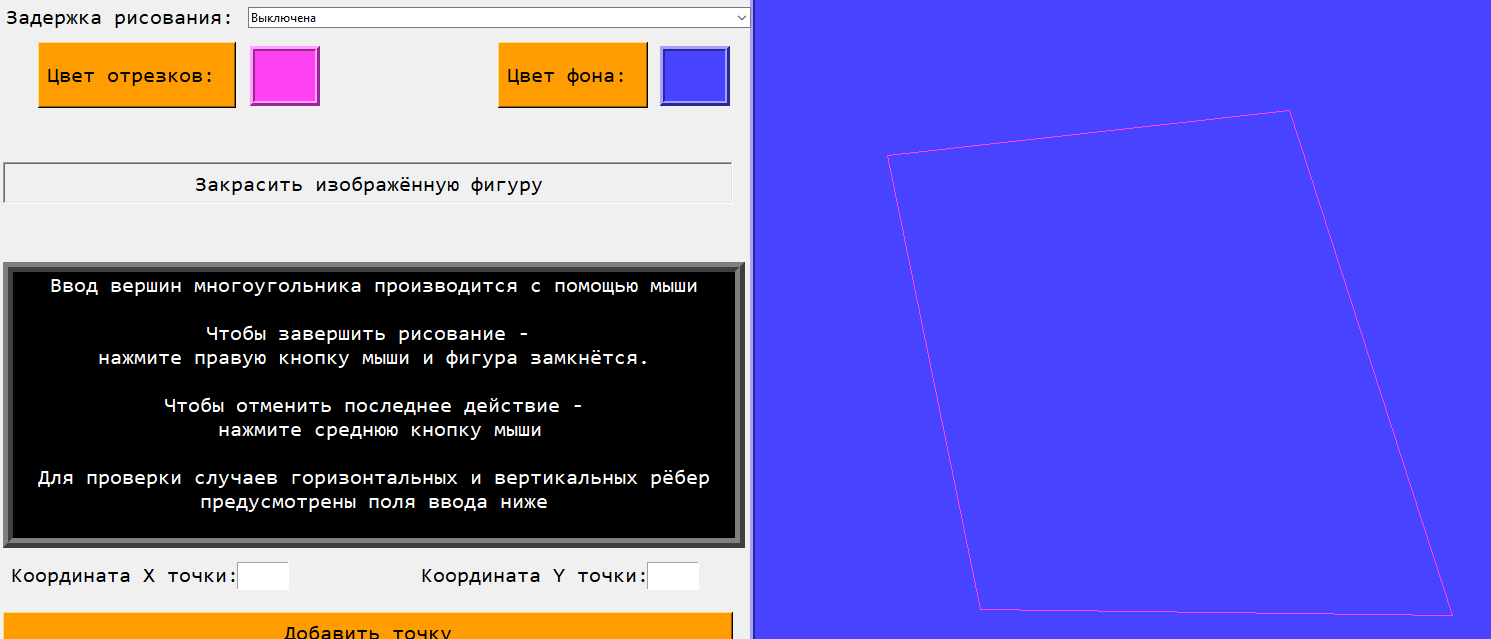




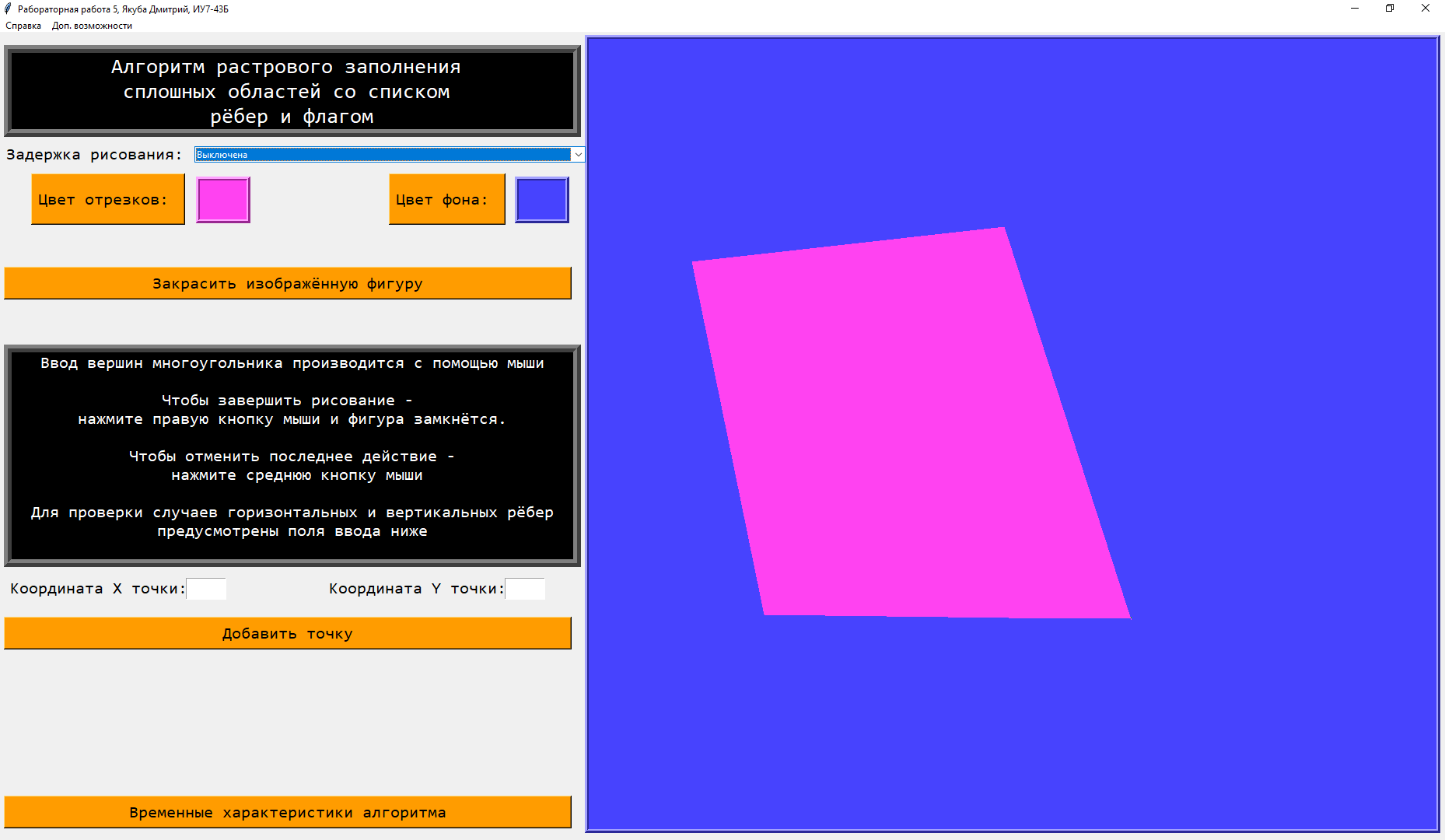
Ввод вершин производится посредством клика левой кнопкой мыши в точке, соответствующей вершине многоугольника. Отрисовка текущих рёбер происходит по нажатию кнопки мыши. Отмена последнего действия – средняя кнопка мыши. Замыкание фигуры – правая кнопка мыши.

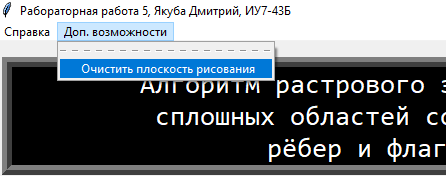


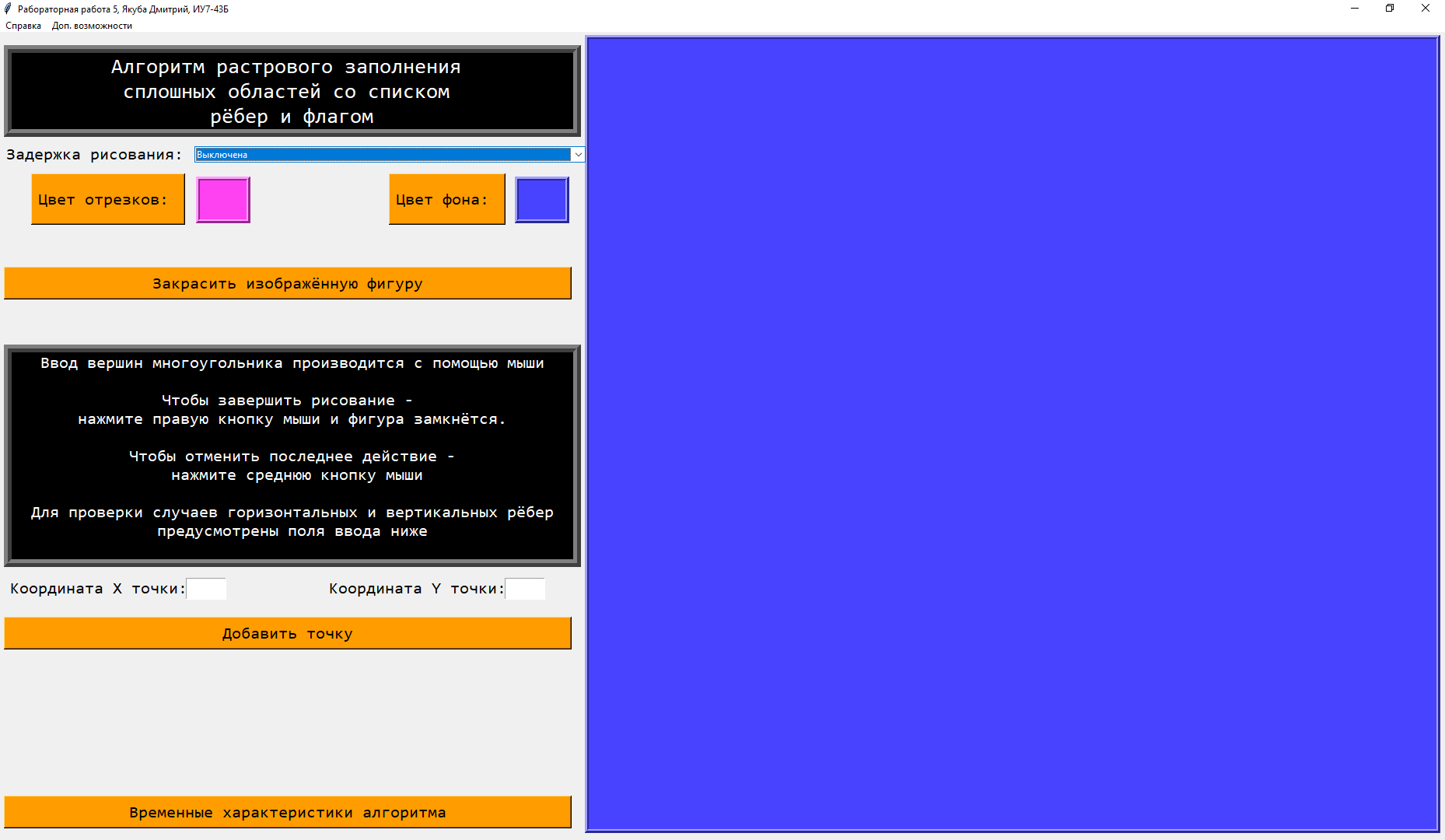




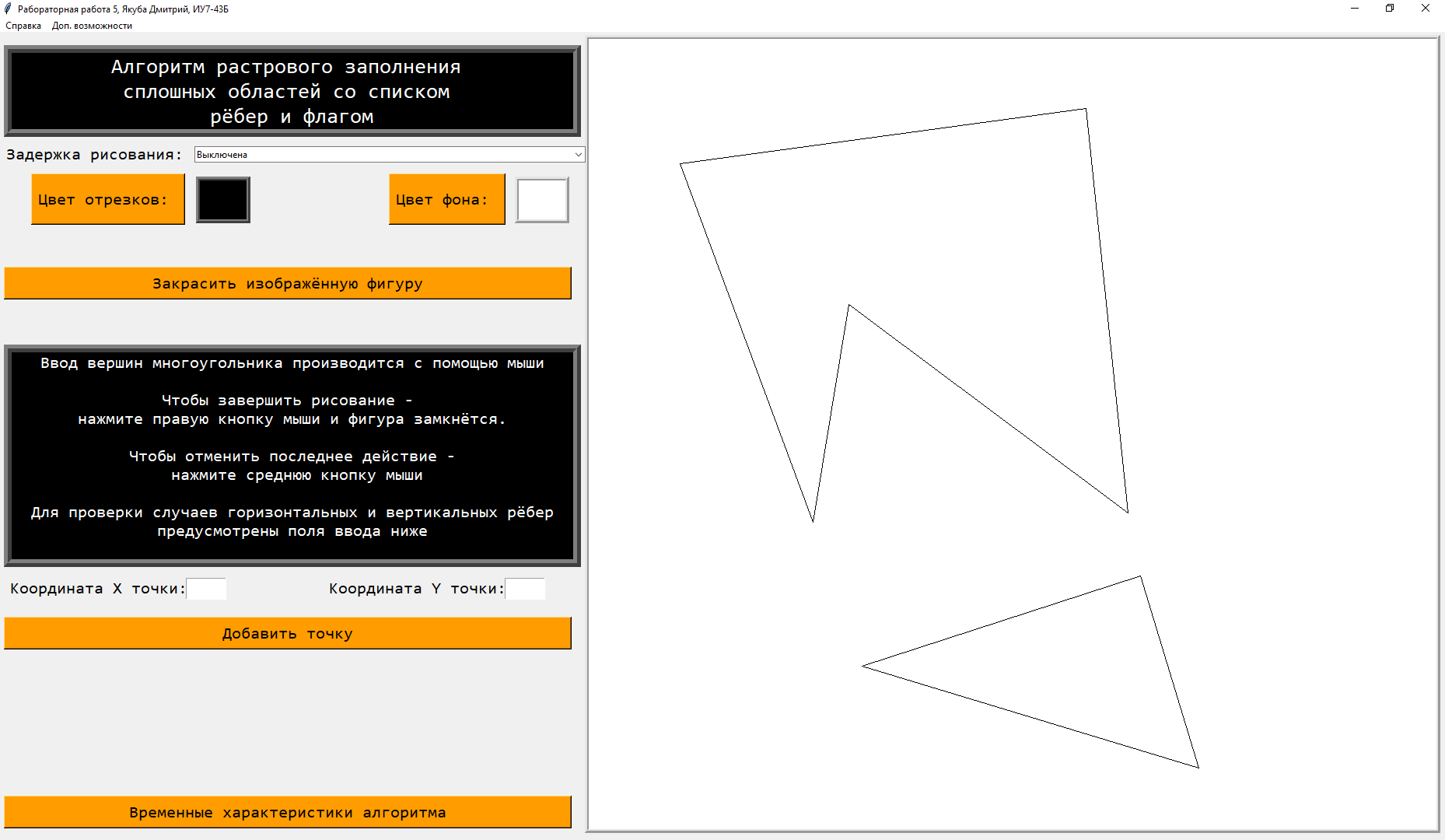
Закраска заданной фигуры происходит по нажатию кнопки «закрасить изображённую фигуру»

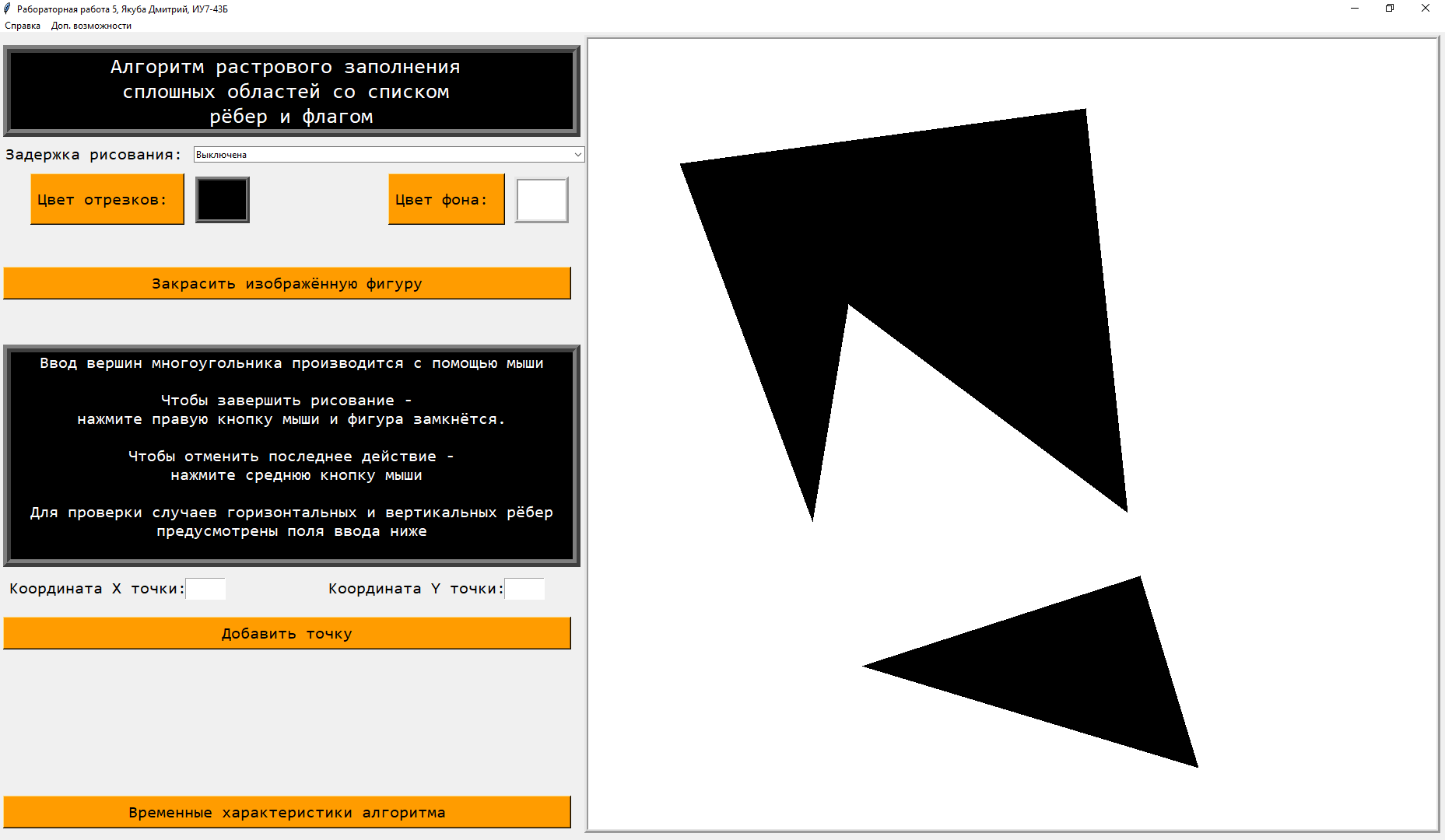


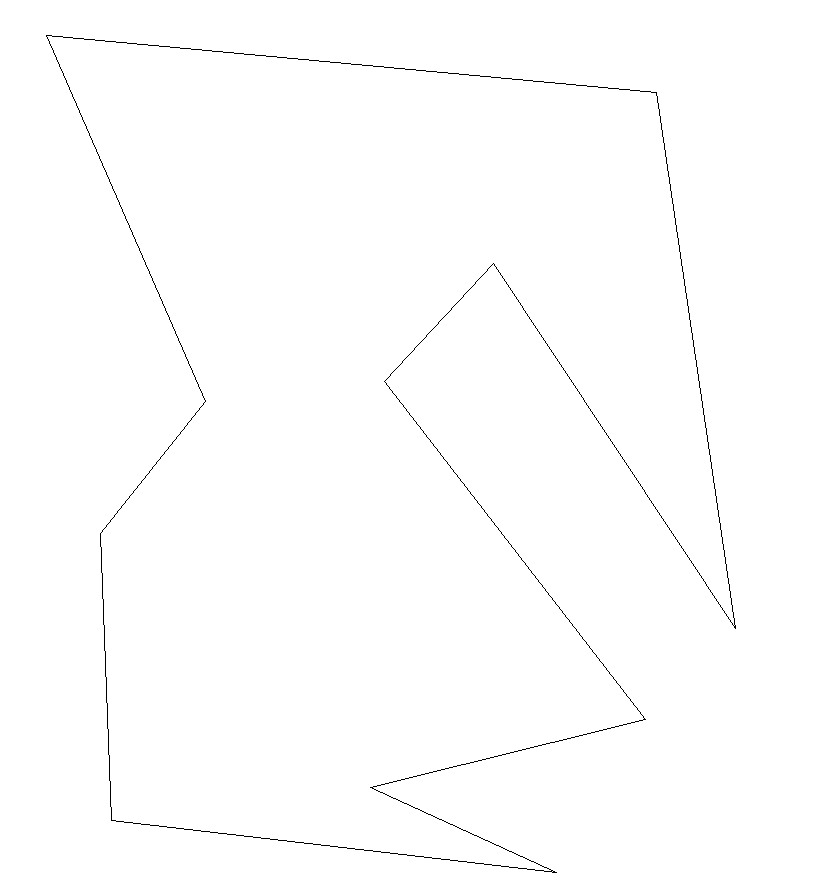
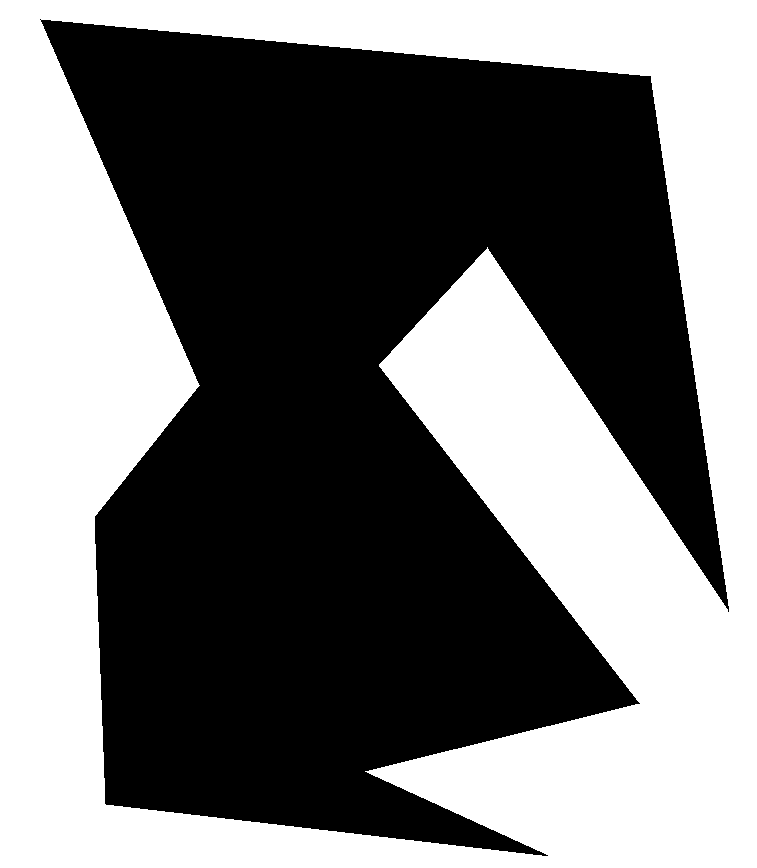


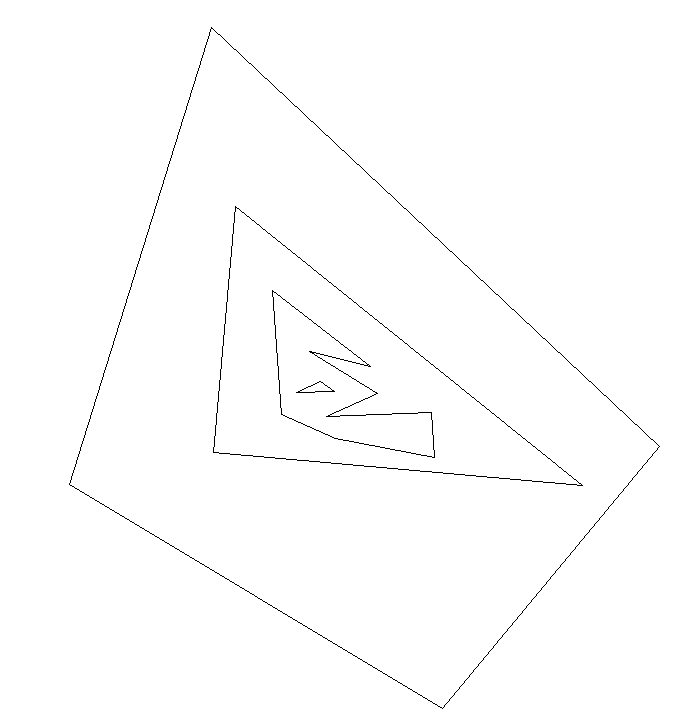
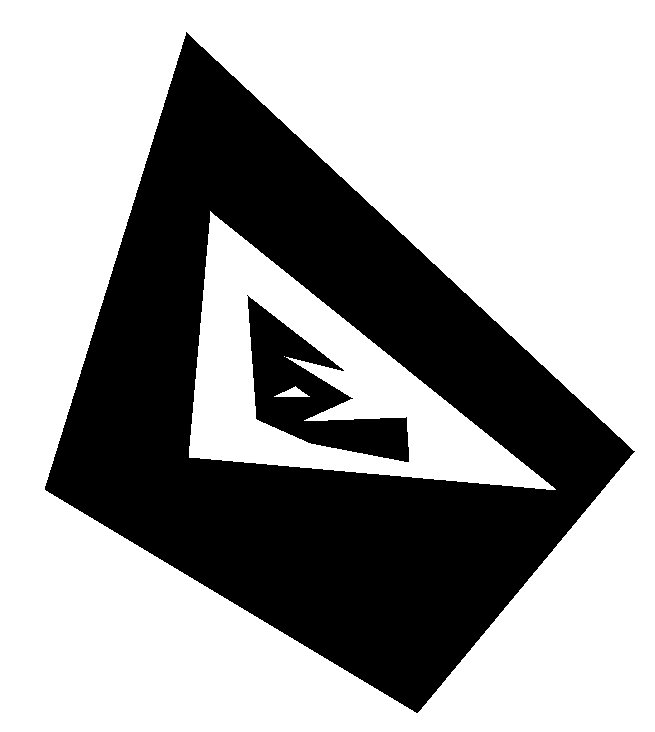


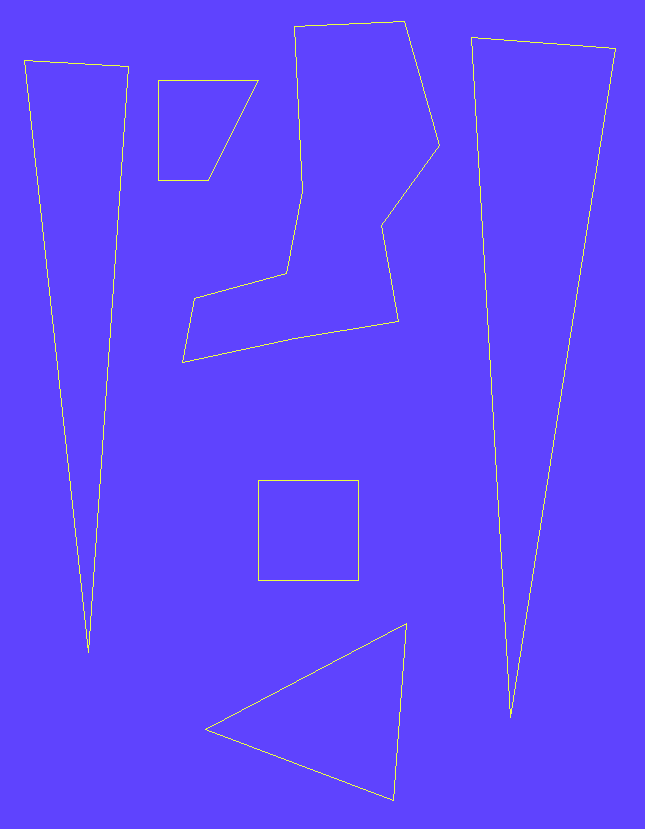
## Демонстрация работы алгоритма



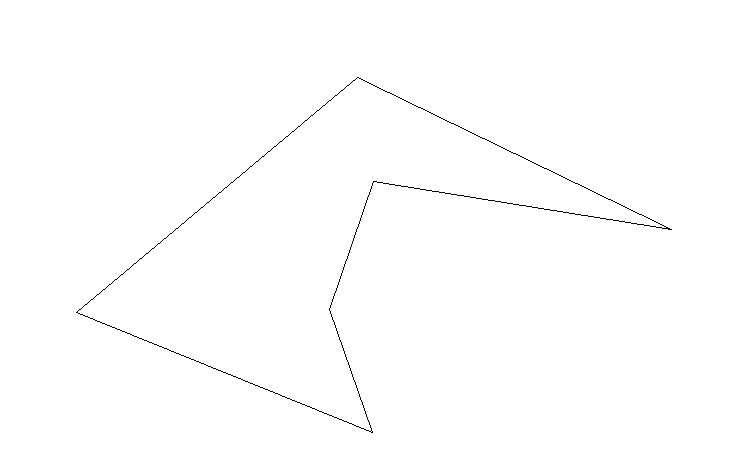


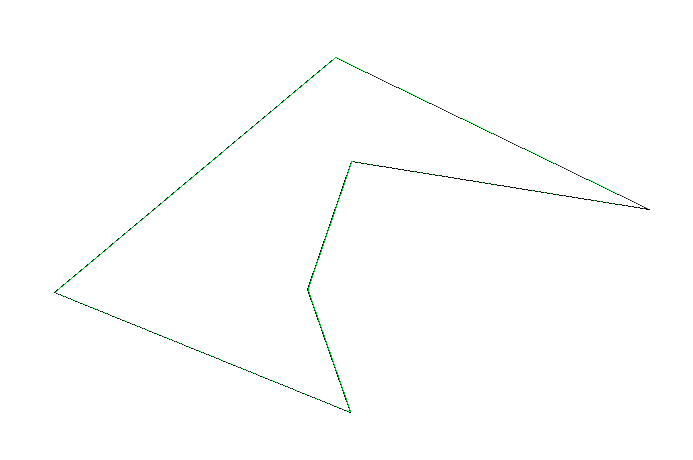
 

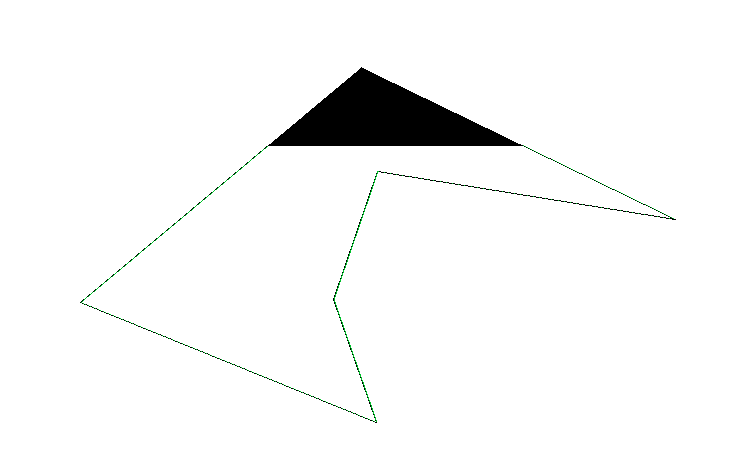
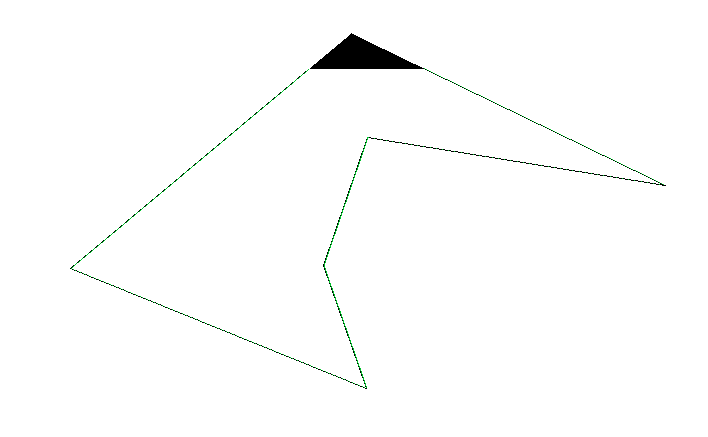
 

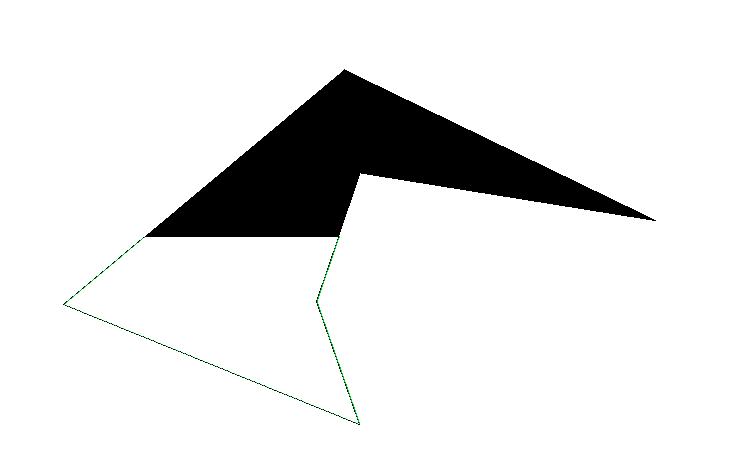
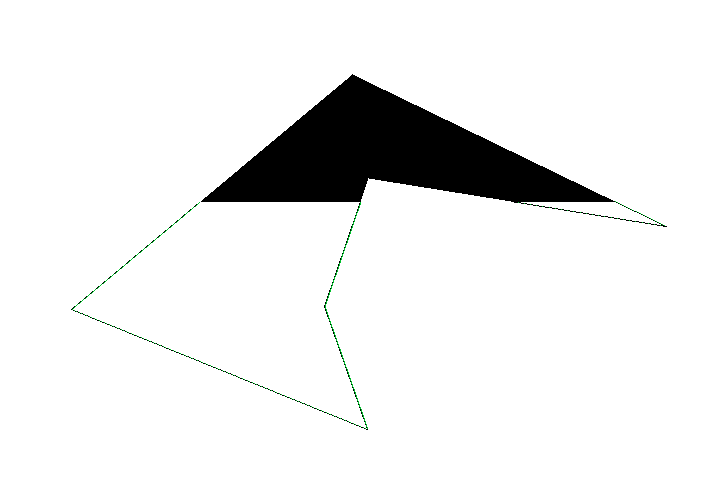


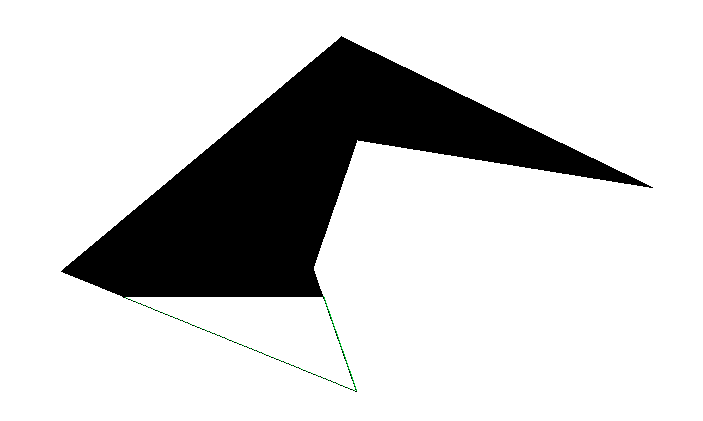
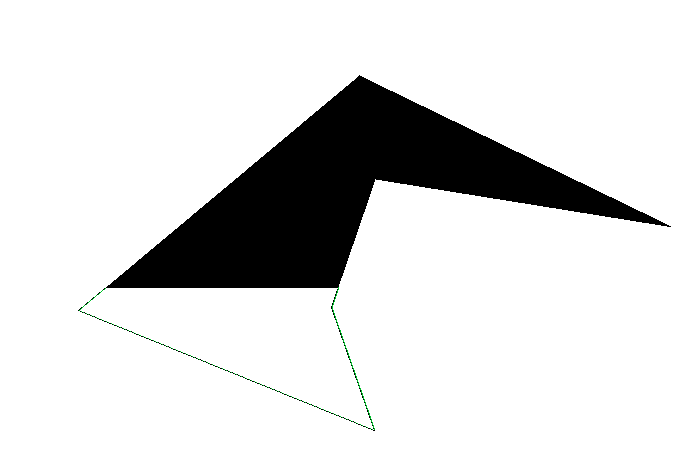
Ниже предоставлены снимки работы программы с задержкой

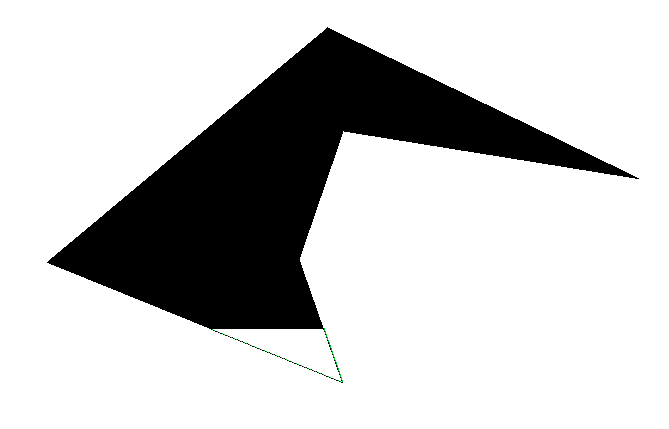


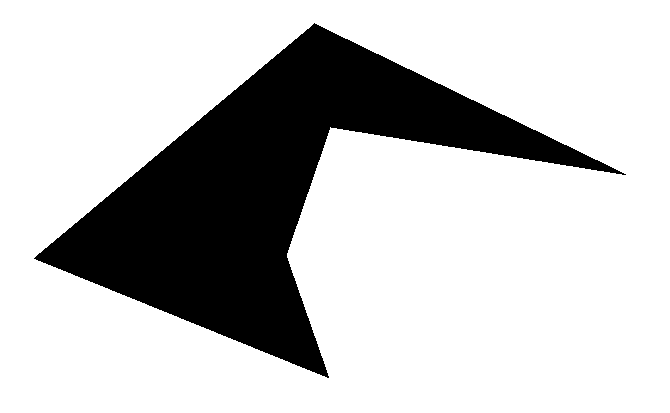




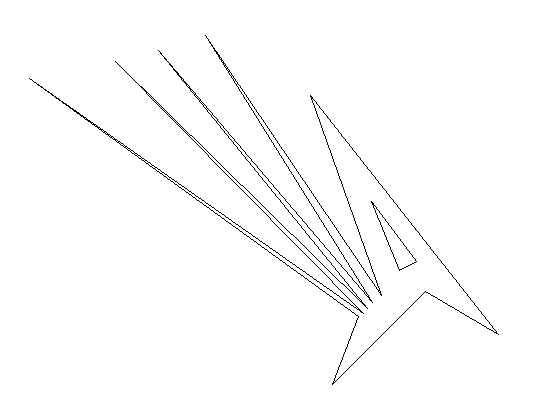


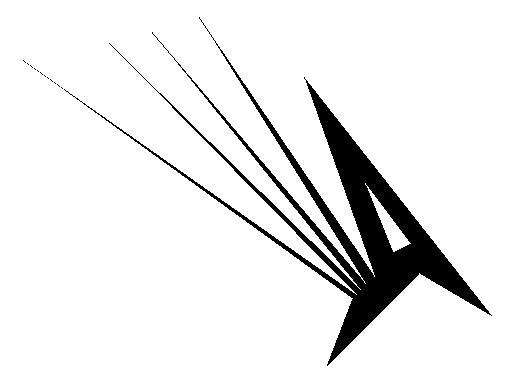
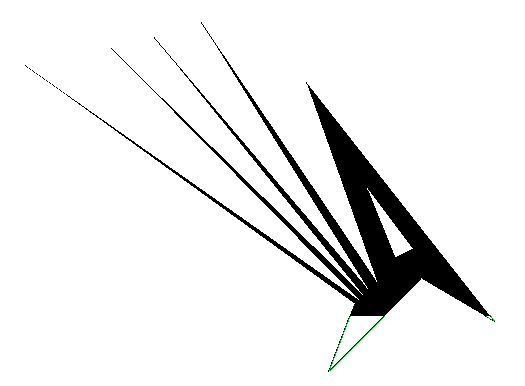
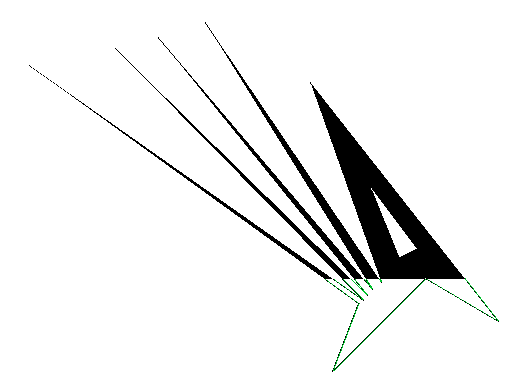
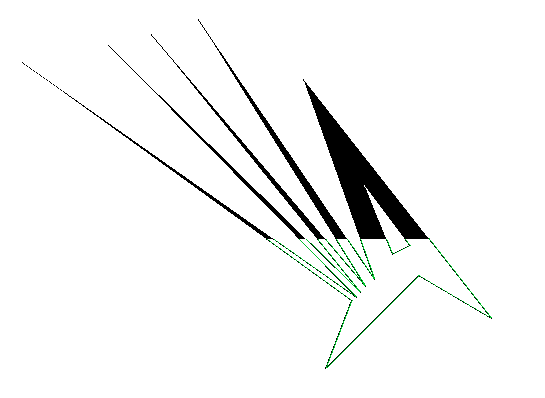
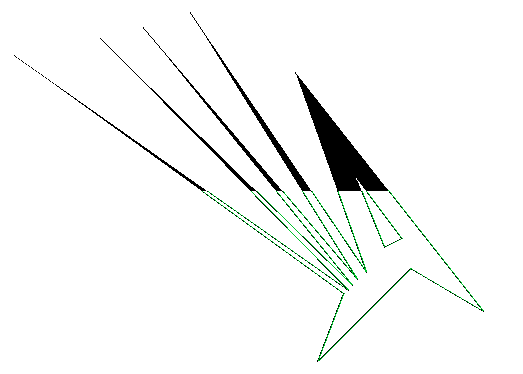
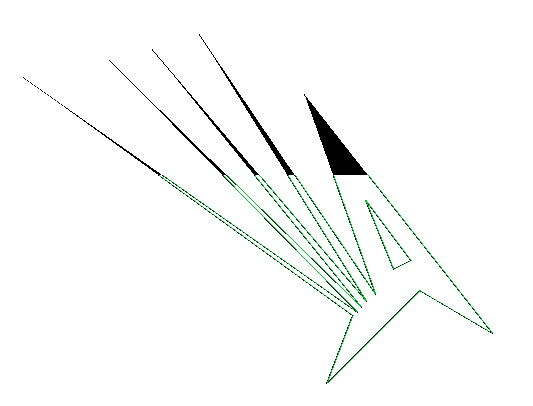
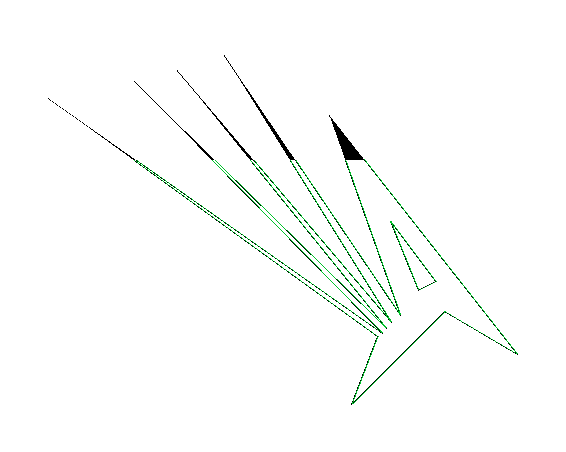
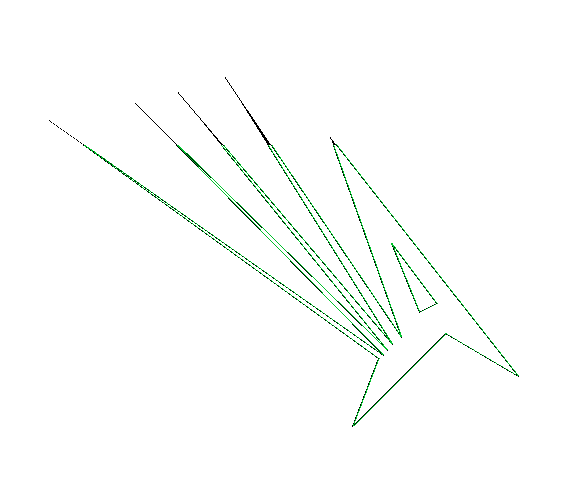
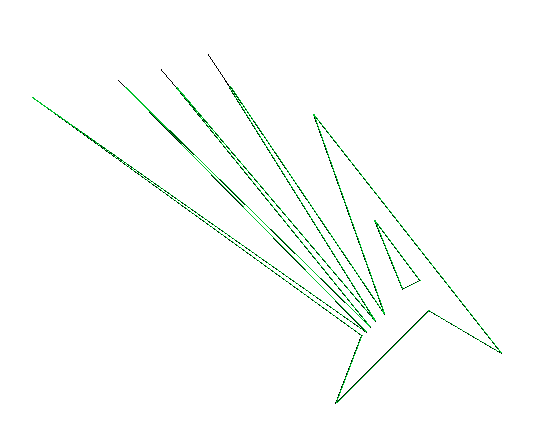
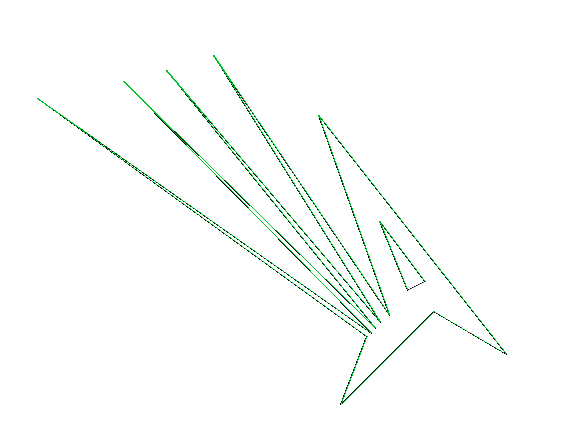




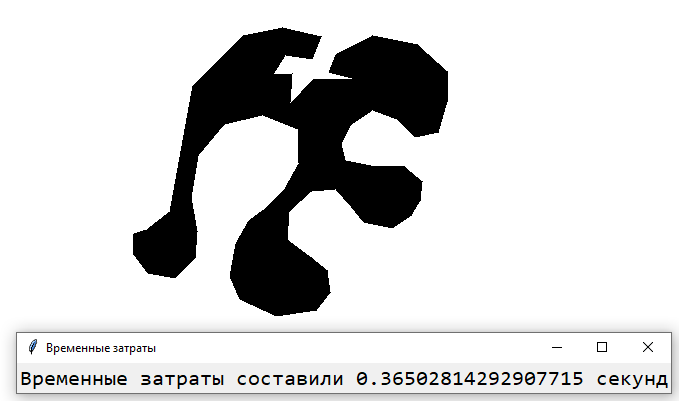


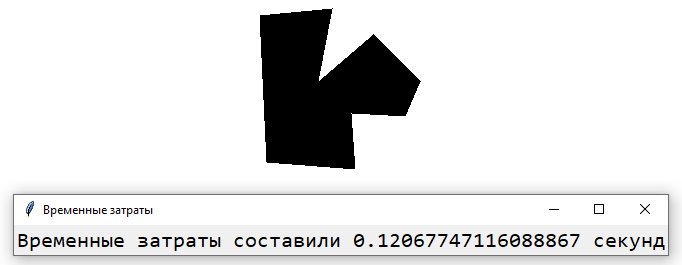
И на более сложный вариант:

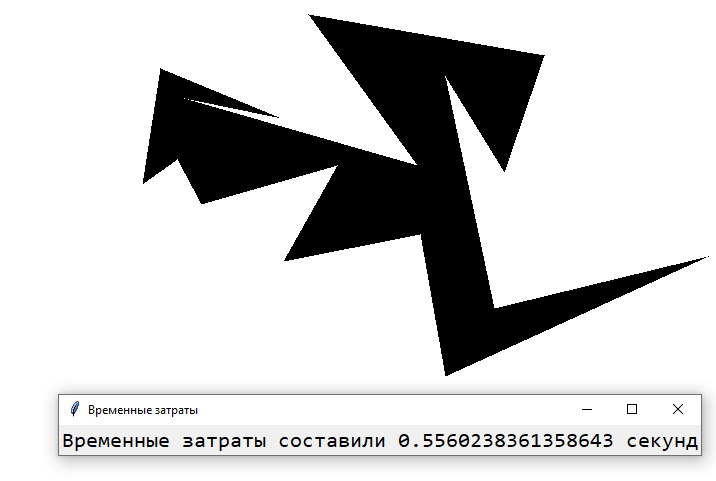


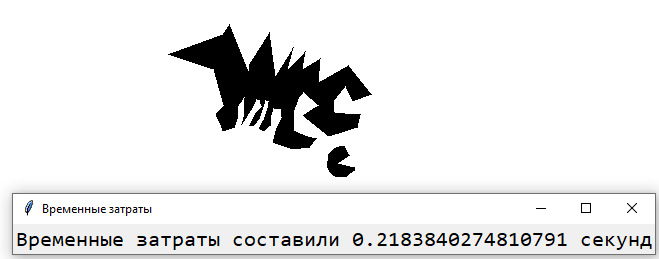


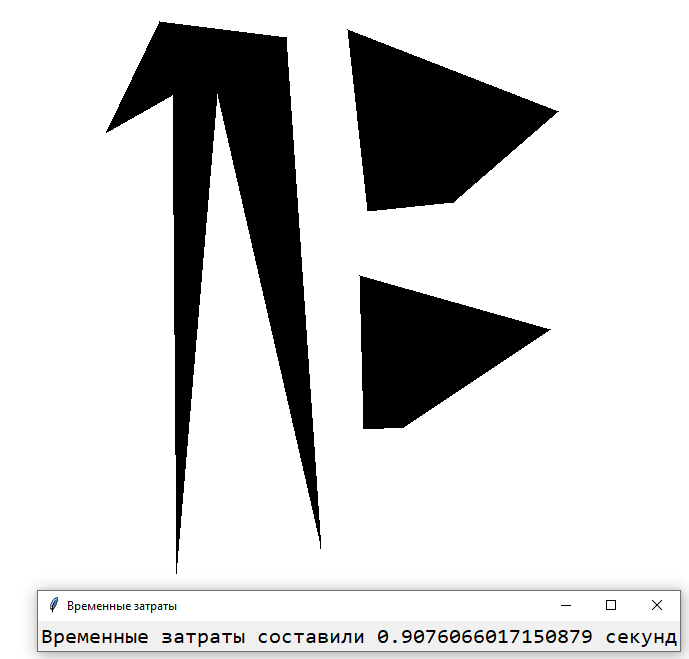
# Исследование временных характеристик

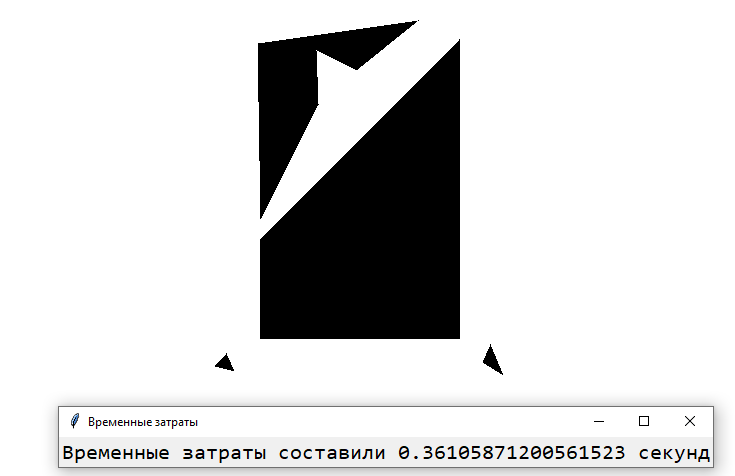












Как видно из предоставленных выше снимков, чем больше размер обрабатываемой плоскости, тем дольше длится работа алгоритма. При этом стоит отметить, что при повышении количества рёбер, разница не такая большая, чем при увеличении размера обрабатываемой плоскости.