|  |  |
| --- | --- |
| **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**

**по дисциплине «Компьютерная графика»**

|  |  |
| --- | --- |
| **Тема:** Реализация и исследование построчного затравочного заполнения сплошных областей.  **Студент:** Челядинов Илья  **Группа:** ИУ7-43Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель:** Куров А. В. |  |

Москва.

2020 г.

**Цель работы:**

Реализация и исследование построчного алгоритма заполнения с затравкой.

**Задание:**

Реализовать алгоритм построчного заполнения с затравкой.

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).

Продемонстрировать возможность заполнения с помощью затравочного алгоритма произвольной области, ограниченной замкнутой кривой линией.

**Теоретический материал**

Для работы алгоритмов заполнения с затравкой:

1. Должна быть задана область для заполнения (произвольная, не обязательно многоугольные области, могут быть области, ограниченные кривой линией).
2. Должна быть задана затравочная точка (или затравка) - точка, расположенная внутри области.

Основная идея алгоритмов заполнения с затравкой: поскольку затравочная точка (или затравка) — это точка, расположенная внутри некоторой области, на основе знания ее координат путем анализа соседних пикселей, пытаемся найти новые затравочные точки.

Область может задаваться:

1. Путем задания цвета границы, то есть имеем дело с гранично-определенными областями.
2. Путем указания цвета пикселей, расположенных внутри области, то есть имеем дело с внутренне-определенными областями.

Области могут быть:

1. 4-х связными (4 направления: 2 горизонтальных и 2 вертикальных)
2. 8-ми связными (8 направлений: 2 горизонтальных, 2 вертикальных и 4 диагональных)

Мы рассматриваем заполнение гранично-определенных 4-ех связных областей.

В простом алгоритме заполнения с затравкой перебираются все затравочные пикселы (пока стек не пуст, извлекаем пиксел из стека, меняем его цвет на цвет заполнения и анализируем 4 соседних пиксела. Если цвет соседних пикселей не равен цвету границы или цвету заполнения, помещаем их в стек).

Алгоритм не является достаточно эффективным.

1. В стек будут заноситься все затравочные пикселы, их может быть много (в зависимости от площади) => требуется большой объем памяти
2. Некоторые затравочные пикселы могут вноситься в стек не по одному разу (по 2, иногда и по 3 раза).
3. Основное время тратится на ввод/вывод: считывание цвета пиксела, изменение цвета пиксела. При большой площади области это может занимать довольно много времени.
4. Считывание информации стека также может занимать много времени, так как некоторые пикселы могут быть уже закрашены и их количество может быть достаточно большим.

Увеличить эффективность мы можем, если не вносить в стек все пикселы в качестве затравочных, а помещать один для некоторой группы затравочных пикселей (для непрерывного интервала пикселей). Так работает построчный алгоритм с затравкой.

Непрерывный интервал пикселей — это группа примыкающих друг к другу пикселей, расположенных на одной сканирующей строке, еще не закрашенных и не являющимися граничными, но ограниченные граничными или уже заполненными пикселами.

Построчный алгоритм заполнения с затравкой:

1. Ввод исходных данных: информация о границах заполняемой области, координаты затравочного пиксела, цвет границы, цвет заполнения.
2. Занесение затравочного пиксела в стек.
3. Пока стек не пуст, выполнить следующие действия

3.1. Извлечение пиксела из стека (x,y)

3.2. Закраска пикселей текущей строки (y) влево и вправо от затравочного от затравочного до встречи с граничным пикселей

(пока Цвет(x,y) <> цвет границы:

Цвет(x,y) = цвет закраски;

x = x - 1; - при движении влево (x = x + 1; - при движении вправо))

3.3. Если Цвет(x,y) = цвет границы, то в качестве Xл = x + 1 (Xпр = x - 1)

3.4. Поиск новых затравочных пикселей в интервале Xл<=x<=Xпр на двух соседних строках по отношению к текущей (Yв = y + 1, Yн = y - 1)

Примечание: на заданном интервале поиска необходимо найти самый правый затравочный пиксел и поместить его в стек

Анализ эффективности:

1. Цвет каждого пиксела меняется только один раз.
2. Цвет пиксела анализируется максимум 3 раза: 2 раза на строчках, расположенных на границах сверху и снизу, и у первого затравочного пиксела, в остальных случаях – 3 раза.
3. Количество обрабатываемых пикселей: обрабатываем пиксели внутри выбранной области, а также пиксели, расположенные на границе с областью закраски за пределами фигуры.
4. Выигрываем по памяти по сравнению с простым алгоритмом заполнения с затравкой, так как не каждый затравочный пиксел размещаем в стеке

**Исходный код.  
def** fill\_area(stack, pause):  
 draw\_tuple = get\_color\_tuple(draw\_color)  
 fill\_tuple = get\_color\_tuple(fill\_color)  
  
 **while** stack:  
 current\_point = stack.pop()  
 img.put(fill\_color, current\_point)  
  
 x, y = current\_point[0] + 1, current\_point[1]  
 **while** img.get(x, y) != draw\_tuple **and** img.get(x, y) != fill\_tuple:  
 *# img.put(fill\_color, (x, y))* x += 1  
 rx = x - 1  
 img.put(fill\_color, (current\_point[0] + 1, y, rx + 1, y + 1))  
  
 x = current\_point[0] - 1  
 **while** img.get(x, y) != draw\_tuple **and** img.get(x, y) != fill\_tuple:  
 *# img.put(fill\_color, (x, y))* x -= 1  
 lx = x + 1  
 img.put(fill\_color, (lx, y, current\_point[0], y + 1))  
  
 **for** i **in** [1, -1]:  
 x = lx  
 y = current\_point[1] + i  
  
 **while** x <= rx:  
 flag = 0  
 **while** img.get(x, y) != draw\_tuple **and** img.get(x, y) != fill\_tuple **and** x <= rx:  
 flag = 1  
 x += 1  
  
 **if** flag:  
 stack.append([x - 1, y])  
  
 flag = 0  
 xi = x  
 **while** (img.get(x, y) == draw\_tuple **or** img.get(x, y) == fill\_tuple) **and** x < rx:  
 x += 1  
  
 **if** x == xi:  
 x += 1  
 **if** pause:  
 time.sleep(0.1)  
 canvas.update()

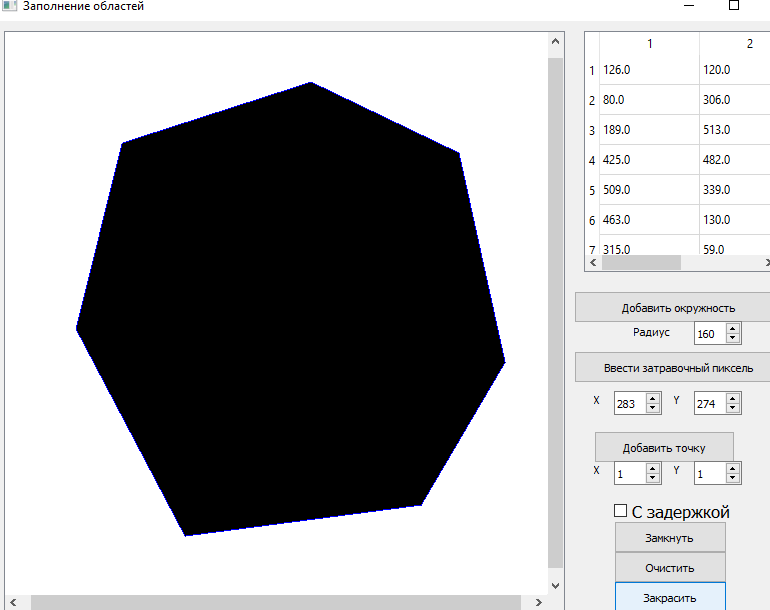
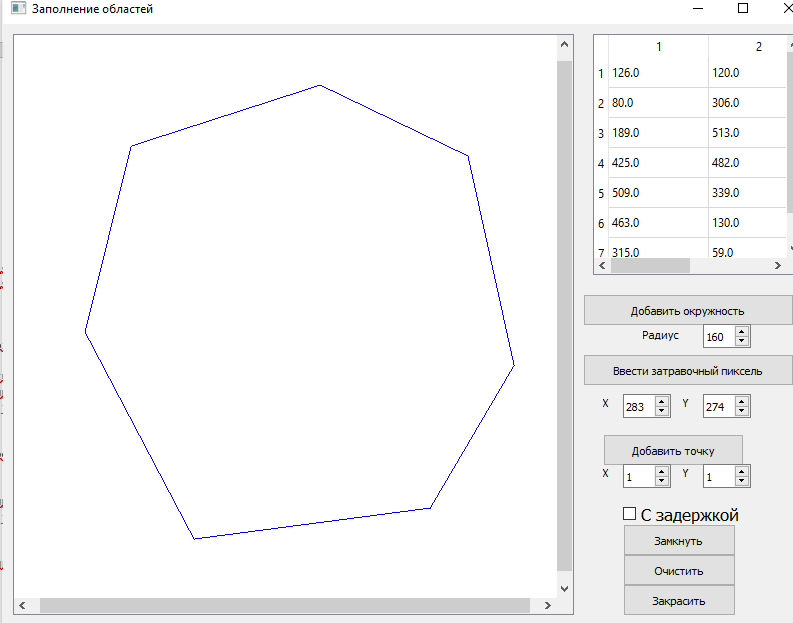
**Примеры работы программы.**

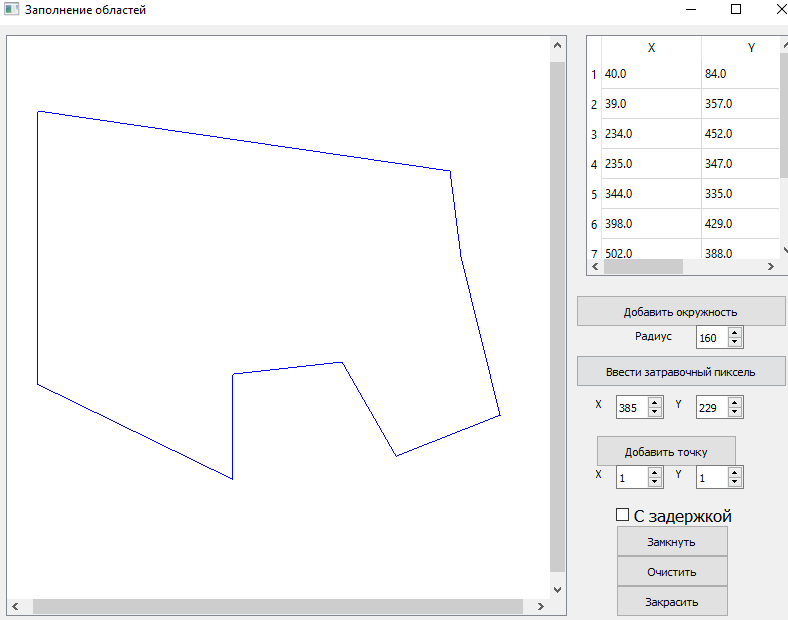
Заполнение произвольного многоугольника c единичной площадью:

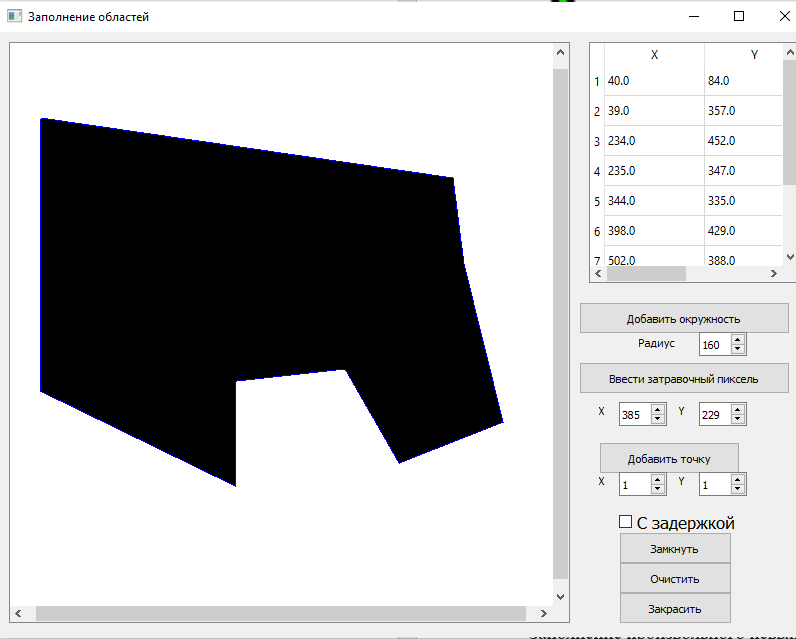


Время выполнения: 0.027

Заполнение произвольного выпуклого многоугольника:

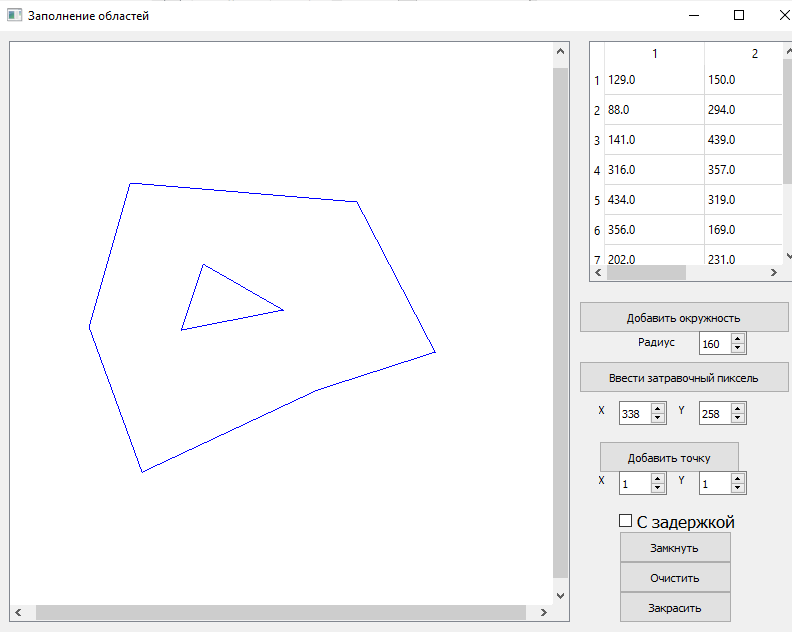
Время выполнения: 0.522

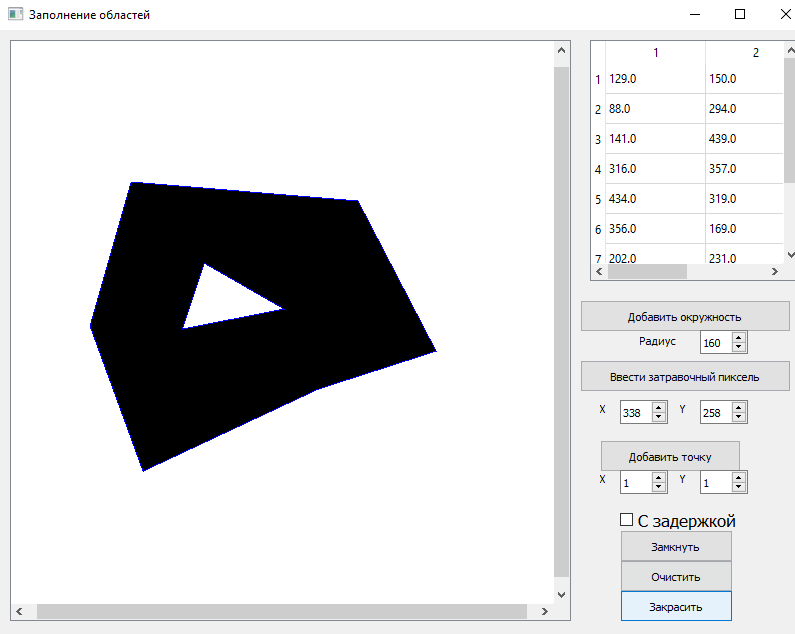
Заполнение произвольного невыпуклого многоугольника: 



Время выполнения: 0.494

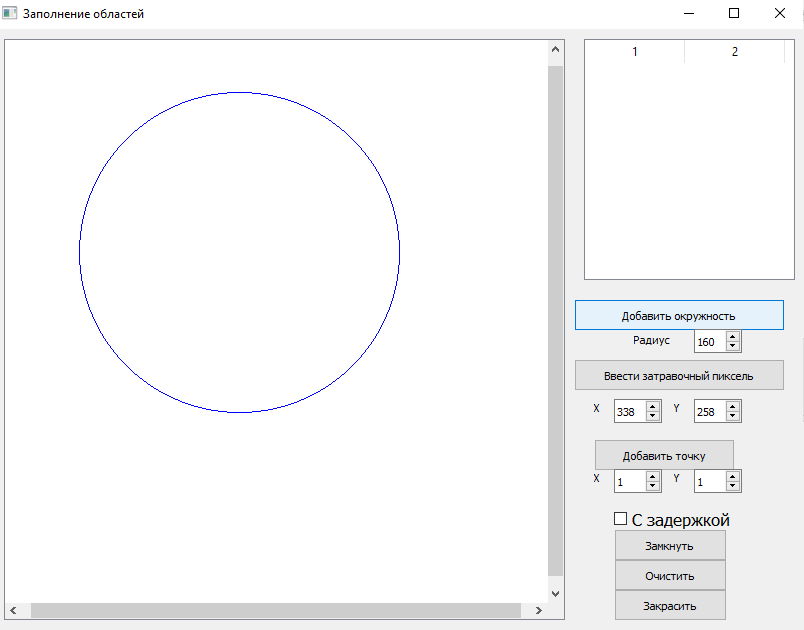
Заполнение произвольного многоугольника с отверстием:

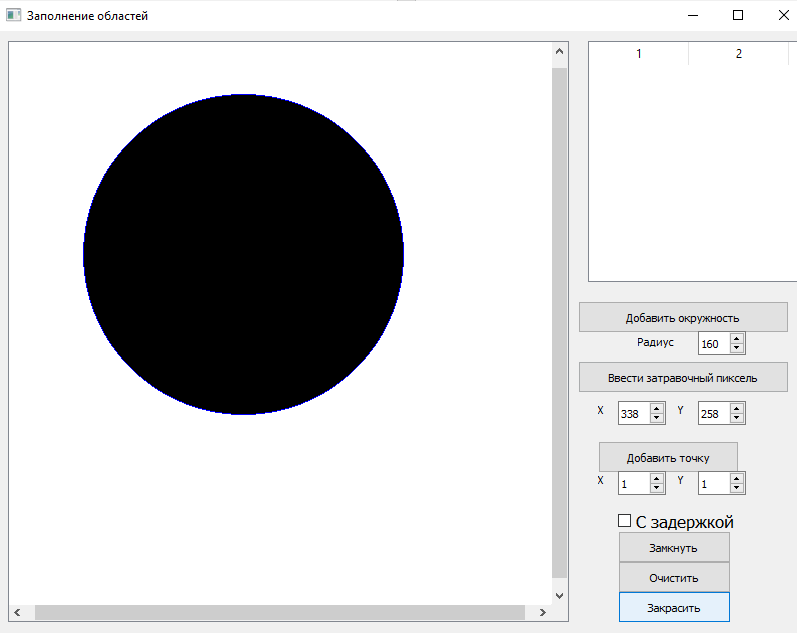




Время выполнения: 0.677

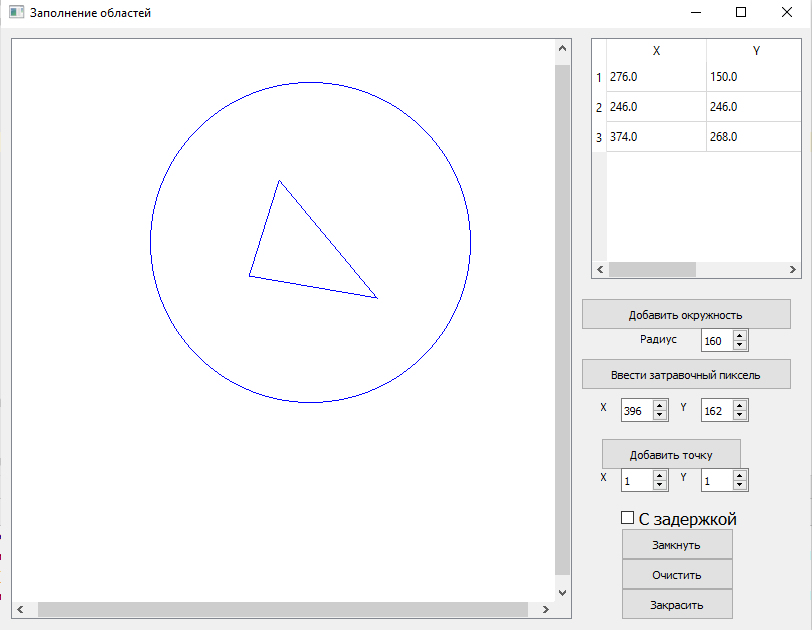
Заполнение произвольного эллипса:

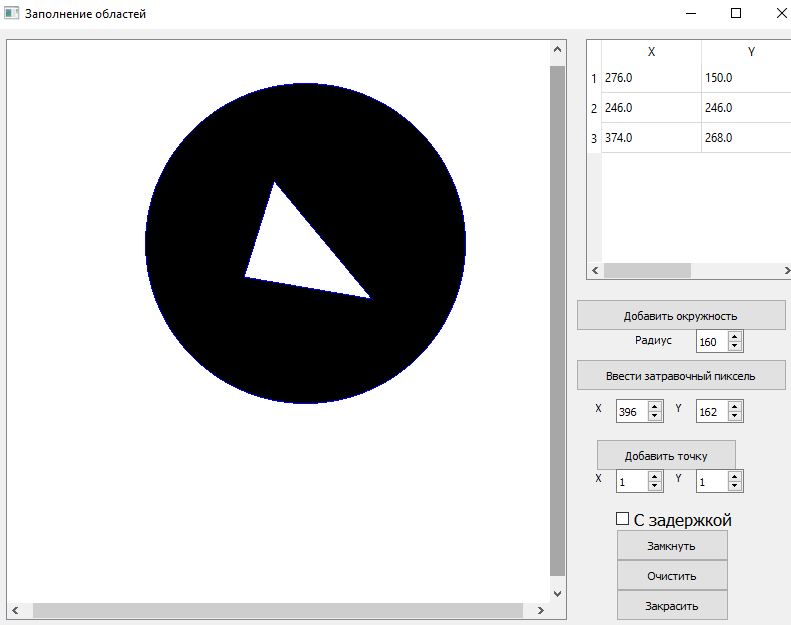




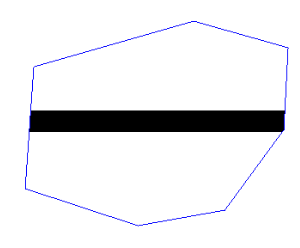
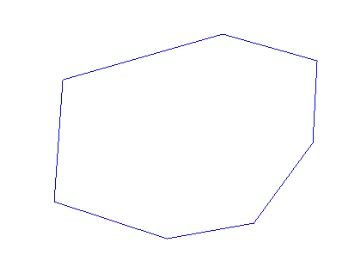
Время выполнения: 0.389

Заполнение произвольного эллипса с отверстием:

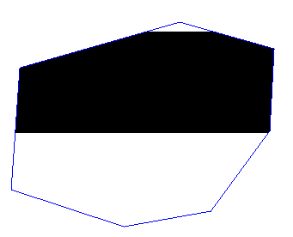
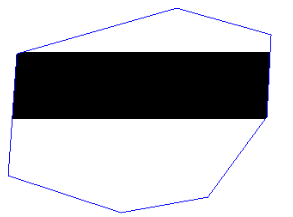


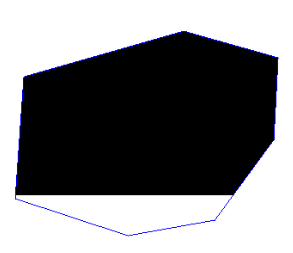
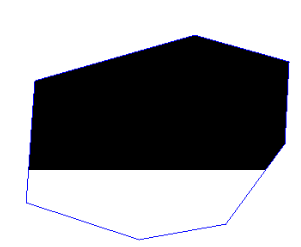


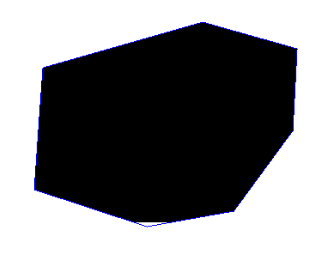
Время выполнения: 0.556











Время выполнения: 5.554