|  |  |
| --- | --- |
| **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 (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04 Программная инженерия**

**Отчет**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 5 |

**Название:**

Реализация и исследование алгоритмов растрового заполнения сплошных областей

**Дисциплина:** Компьютерная графика

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ7-43Б |  | 29.04.2020 | С.С. Кононенко |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | А.В. Куров |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2020

**1. Цель работы**

Реализация и исследование одного из алгоритмов растрового заполнения области.

**2. Техническое задание**

Реализовать один из алгоритмов заполнения сплошной области. 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. 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. Обеспечить замер времени выполнения алгоритма.

**3. Теоретический материал**

**Растровая развертка сплошных областей** – генерация сплошных областей из простых описаний ребер или вершин. В методах растровой развертки пытаются определить в порядке сканирования строк, лежит ли точка внутри многоугольника или контура. Алгоритмы обычно идут от «верха» многоугольника к «низу» (в моей работе «движение» будет происходить от «низа» к «верху», так как работа программы ведется в компьютерных координатах).

**Алгоритм с упорядоченным списком ребер**

Процесс реализации алгоритма можно разбить на два этапа:

* Подготовка (сортировка) данных;
* Преобразование отсортированных данных в растровую форму.

В моей работе я использовал самую совершенную версию данного алгоритма — алгоритм с использованием упорядоченного **списка активных ребер** (САР). В этом случае значительно сокращается потребность в памяти, а точки пересечения со строками сканирования вычисляются в пошаговом режиме. В процессе подготовки данных в алгоритме с упорядоченным списком ребер, использующем список активных ребер необходимо:

* Определить для каждого ребра многоугольника наивысшую сканирующую строку, пересекаемую данным ребром;
* Затем ребро многоугольника сохраняется в Y-группе, соответствующей этой сканирующей строке;
* Формируется связанный список, в который заносятся следующие значения:
* Начальное значение абсцисс точек пересечения - **x**;
* Число сканирующих строк, пересекаемых ребром многоугольника - **y**;
* Шаг приращения по оси абсцисс при переходе от одной строки сканирования к другой - **dx**;
* В процесе преобразования подготовленных данных в растровую форму для каждой строки сканирования осуществляется проверка соответствующей Y-группы на наличие новых ребер, в случае их обнаружения, соответствующие значения заносятся в САР.
* После чего координаты абсцисс точек пересечения из связанного списка отсортировываются в порядке возрастания (то есть x1 <= x2) и из него выделяются пары точек, с помощью которых активизируются пиксели на строке сканирования для целых значений **x**.
* Затем для каждого ребра из САР число сканирующих строк, пересекаемых данным многоугольником - **y**, уменьшается на 1.
* Если в результате, **y** становится меньше 0, то данное ребро исключается из САР и вычисляется новое начальное значение координаты абсцисс точек пересечения **xn = x + dx**.
* Затем переходят к новой строке сканирования и этапы реализации алгоритма повторяются.

**4. Реализация алгоритма**

**4.1. Подготовка данных**

Занесение ребра в соответствующуюю Y-группу. Ребро заносится в Y-группу сразу в таком формате, в котором оно будет храниться в списке активных ребер (это метод класса окна, поэтому здесь есть **self** в качестве параметра). Нахождение параметров окаймляющего прямоугольника.

**def** **update\_y\_group**(self, x\_start, y\_start, x\_end, y\_end):

**if** y\_start > y\_end:

x\_end, x\_start = x\_start, x\_end

y\_end, y\_start = y\_start, y\_end

**if** y\_end > self.y\_max:

self.y\_max = y\_end

**if** y\_start < self.y\_min:

self.y\_min = y\_start

y\_proj = y\_end - y\_start **if** y\_end - y\_start **else** **1**

x\_step = -(x\_end - x\_start) / y\_proj

**if** y\_proj != **1**:

**if** y\_end **not** **in** self.y\_groups:

self.y\_groups[y\_end] = [[x\_end, x\_step, y\_proj]]

**else**:

self.y\_groups[y\_end].extend([[x\_end, x\_step, y\_proj]])

**4.2. Преобразование данных в растровую форму**

Проверка списка активных ребер (на предмет удаления из списка).  
  
**def** **check\_active\_edges**(active\_edges):  
 i = **0** **while** i < len(active\_edges):  
 active\_edges[i][**0**] += active\_edges[i][**1**]  
 active\_edges[i][**2**] -= **1** **if** active\_edges[i][**2**] < **1**:  
 active\_edges.pop(i)  
 **else**:  
 i += **1**

Добавление ребер в список активных ребер.

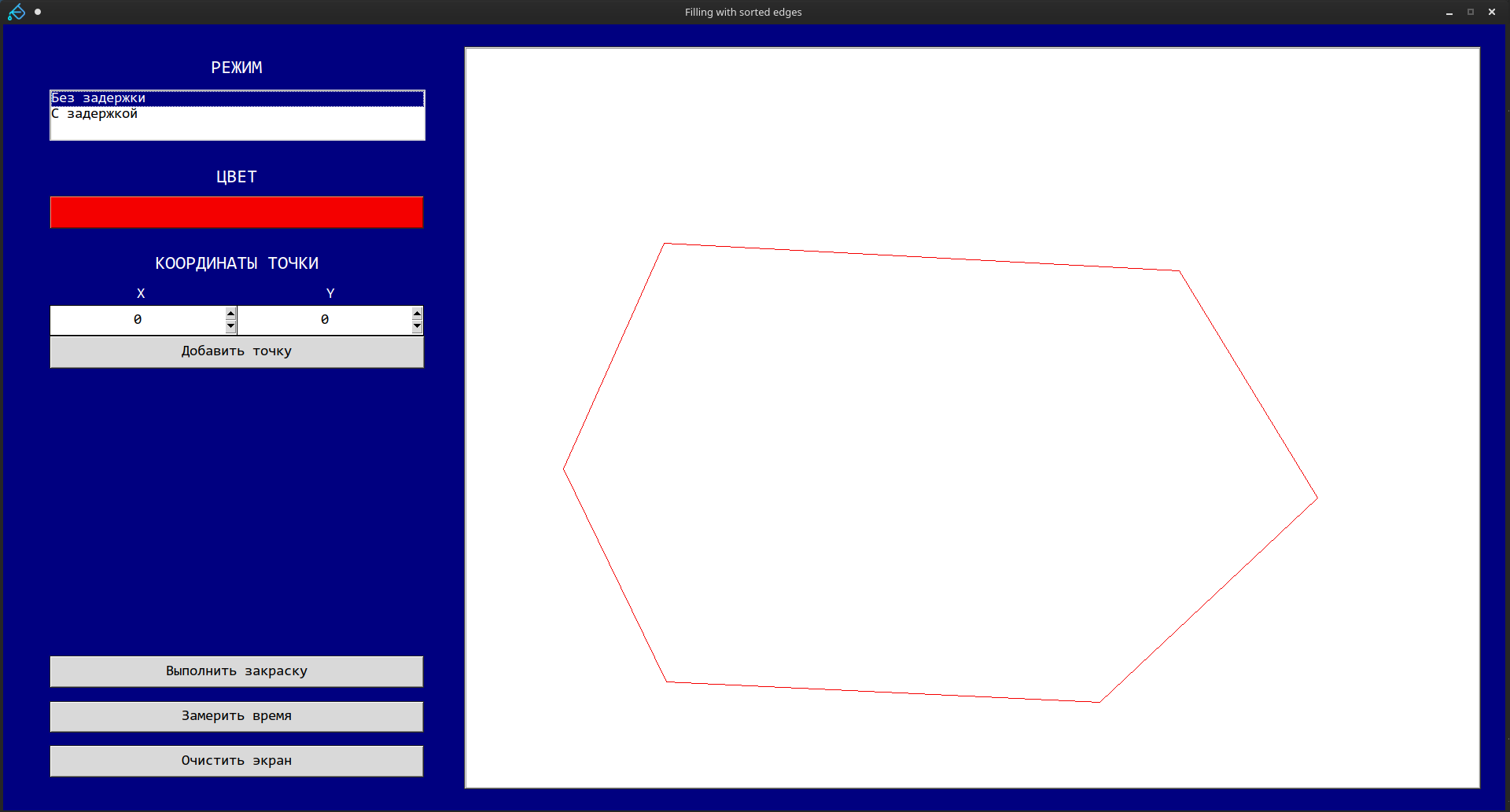
**def** **add\_active\_edges**(y\_groups, active\_edges, y):  
 **if** y **in** y\_groups:  
 **for** edge **in** y\_groups[y]:  
 active\_edges.append(edge)  
 active\_edges.sort(key=**lambda** edge: edge[**0**])

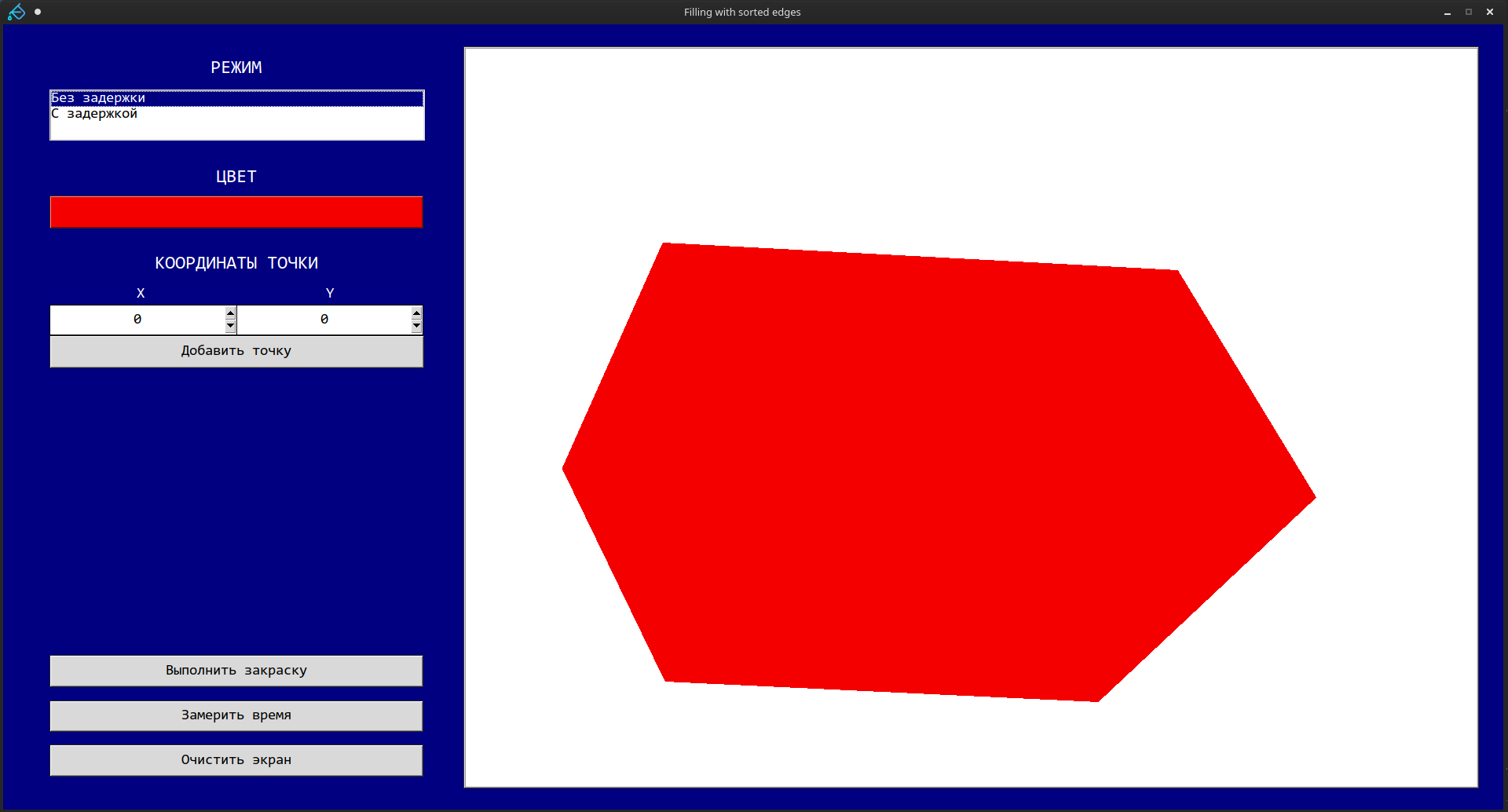
Рисование сканирующей строки.  
  
**def** **draw\_scanline**(root, y):  
 **for** i **in** range(**0**, len(root.active\_edges), **2**):  
 root.image.put(root.color,  
 (int(root.active\_edges[i][**0**]), y, round(root.active\_edges[i+**1**][**0**]) + **1**, y + **1**))

Комплексная функция алгоритма (с учетом задержки).  
  
**def** **fill**(root):  
 delay = root.modelst.get(root.modelst.curselection()[**0**])  
 **if** delay == "С задержкой":  
 delay = True  
 **else**:  
 delay = False  
 y\_end = root.y\_max  
 y\_start = root.y\_min  
 **while** y\_end > y\_start:  
 check\_active\_edges(root.active\_edges)  
 add\_active\_edges(root.y\_groups, root.active\_edges, y\_end)  
 **if** delay:  
 time.sleep(**0.001**)  
 root.canvas.update()  
 draw\_scanline(root, y\_end)  
 y\_end -= **1**

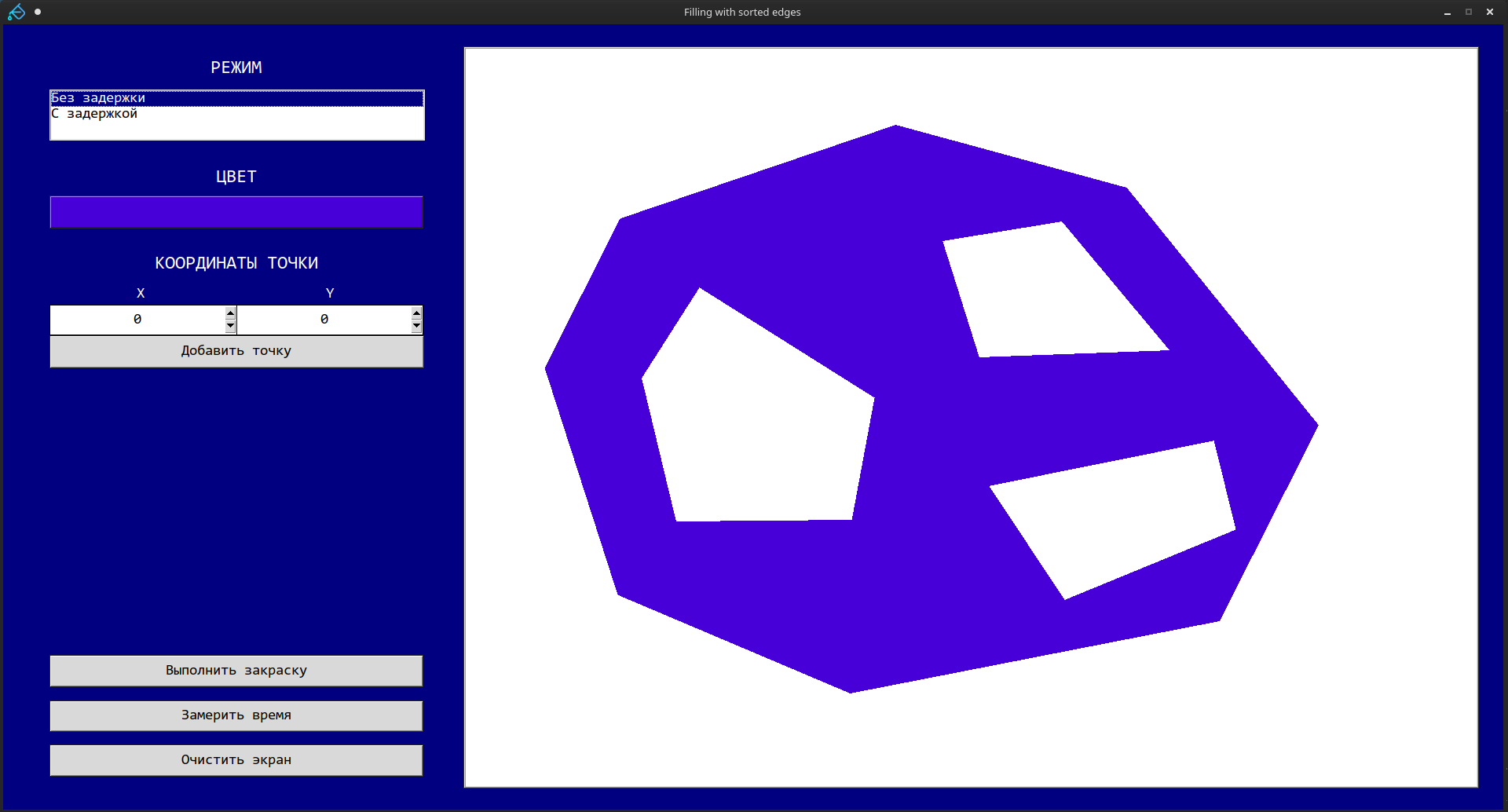
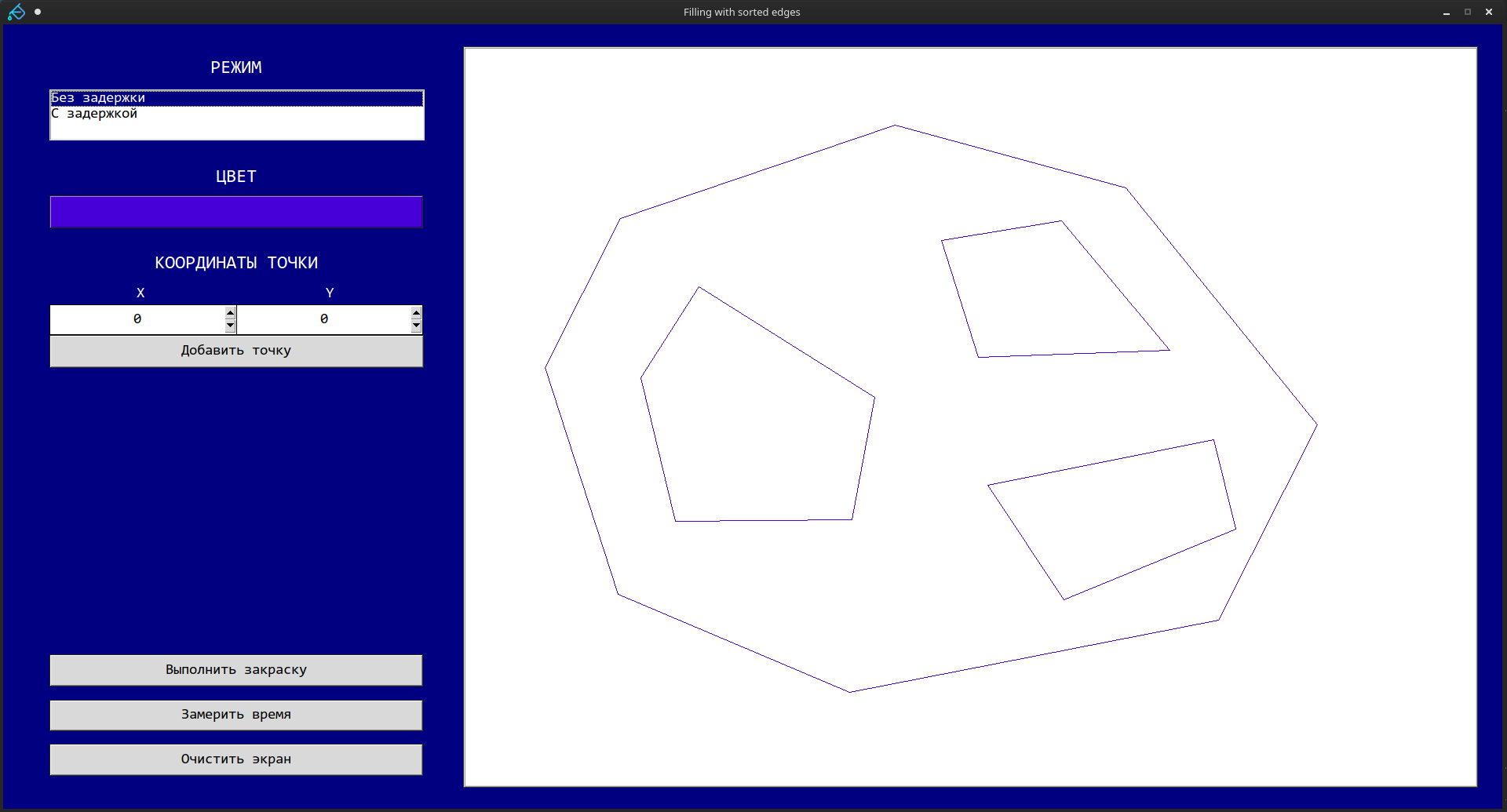
**5. Демонстрация работы программы**

Заполнение произвольного многоугольника.

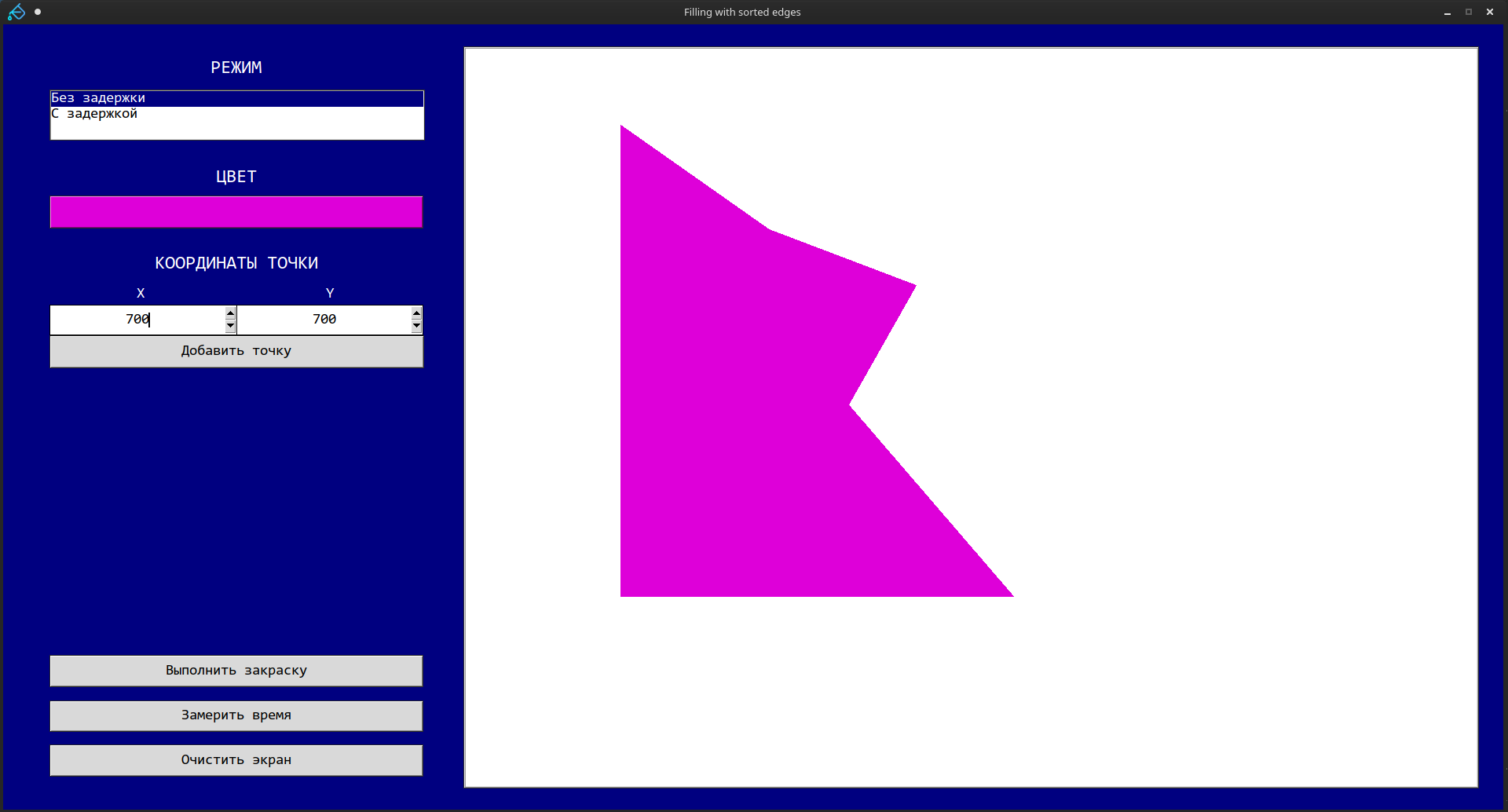
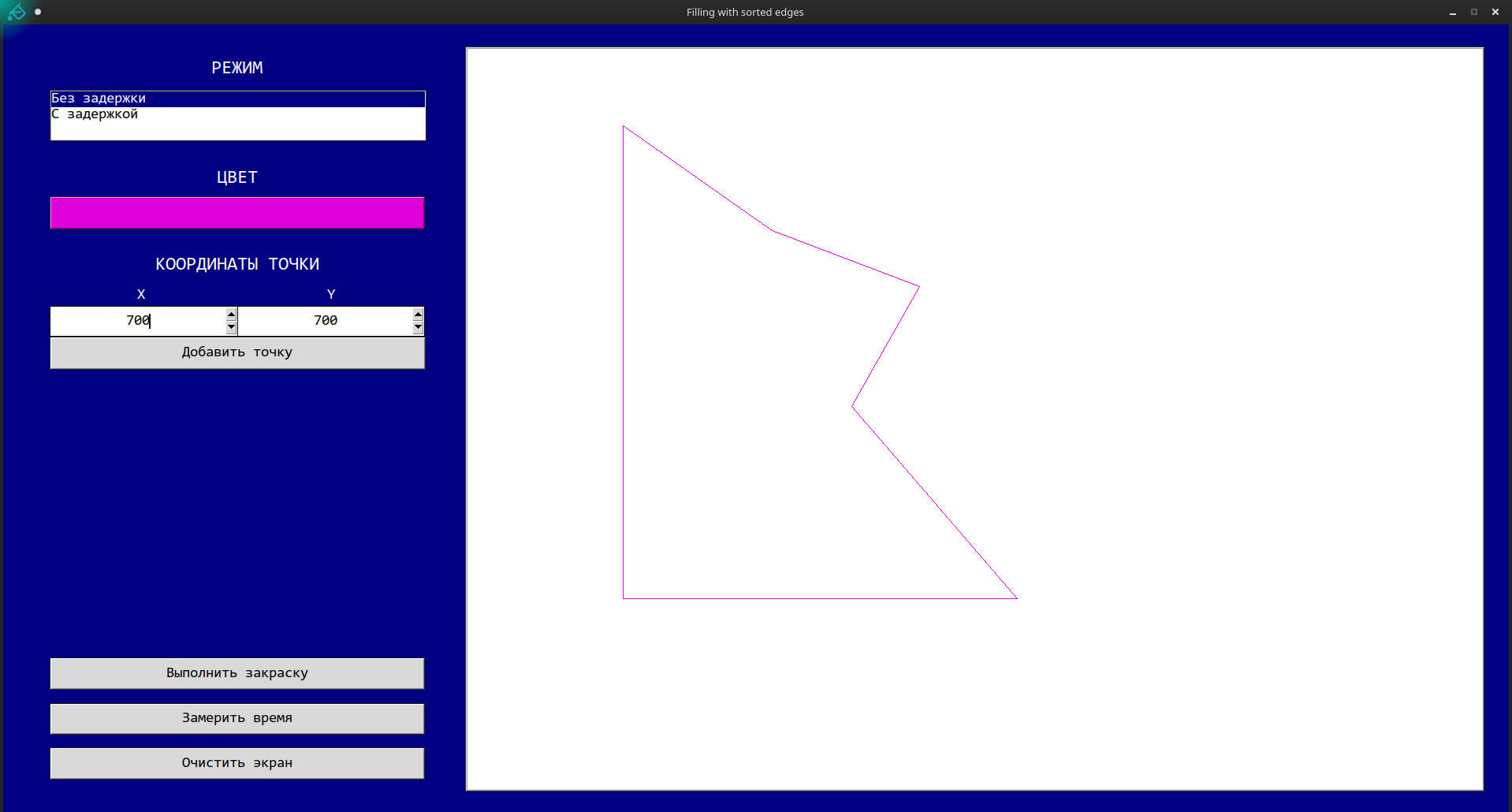


Время, затраченное на закраску: **0.16 мкс.**

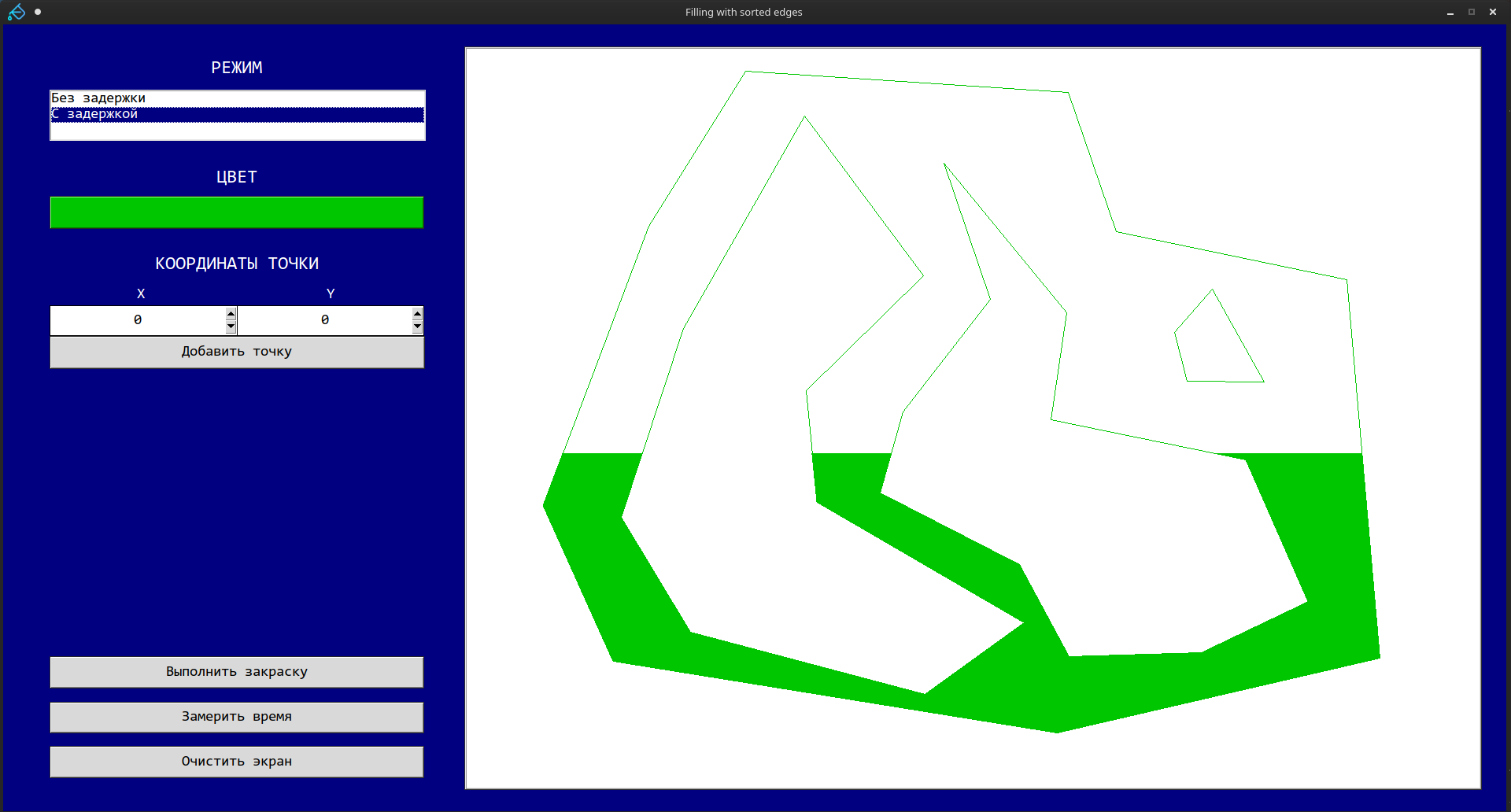
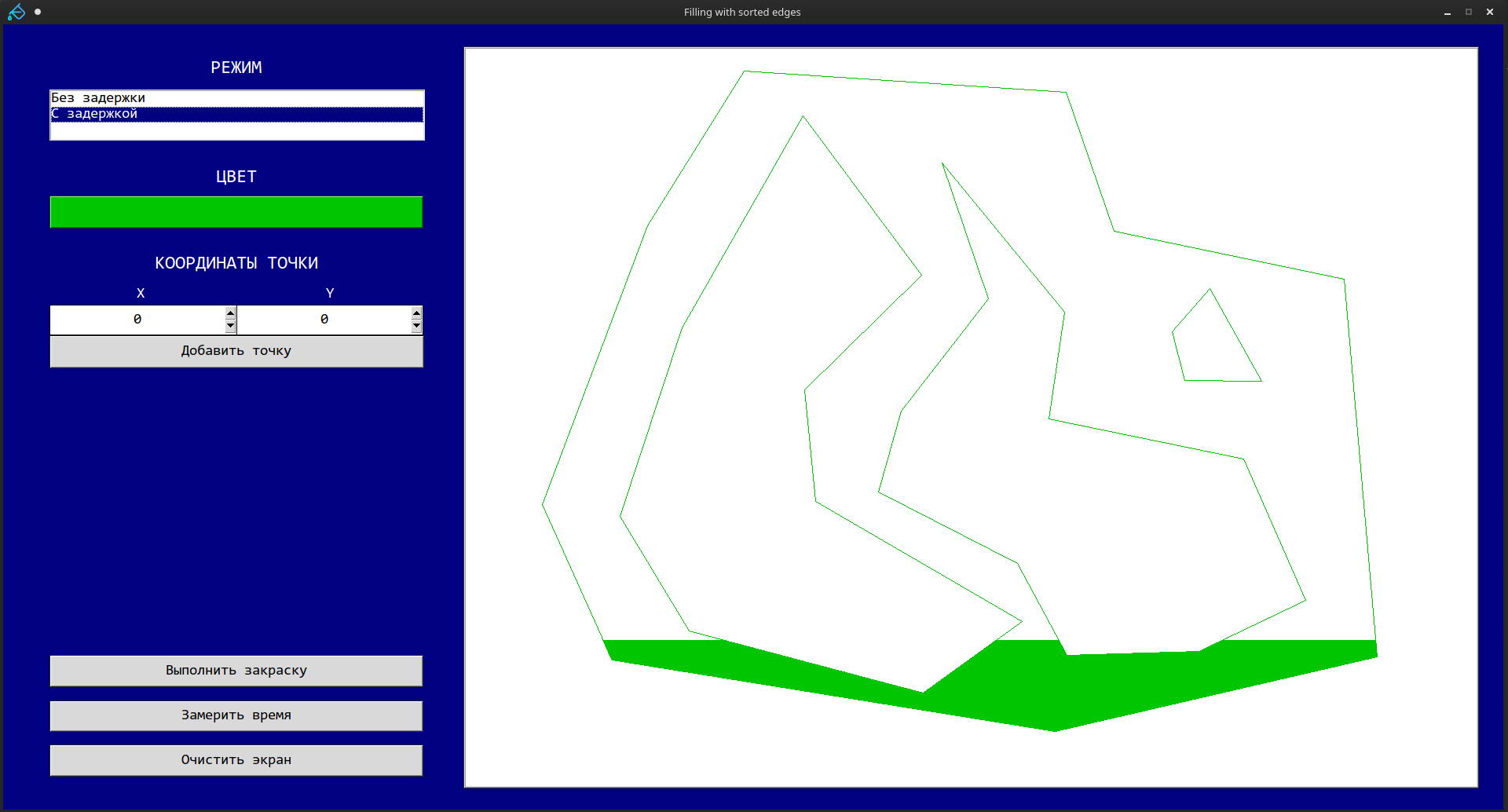
Заполнение произвольного многоугольника с отверстиями.

Время, затраченное на закраску: **0.36 мкс.**

Заполнение многоугольника с горизонтальными и вертикальными ребрами.

Время, затраченное на закраску: **0.11 мкс.**

Поэтапное заполнение многоугольника (режим работы с задержкой).

Как было сказано в теоретической части, повторная закраска не производится, закрашиваются «зоны» между активными ребрами (количество активных ребер всегда кратно двум).