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

|  |  |
| --- | --- |
| **Дисциплина Компьютерная графика**  **Тема Программная реализация и исследование алгоритмов затравочного заполнения сплошных областей**  **Студент Иванов В.А.**  **Группа ИУ7-42Б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель Куров А.В.** |  |

Москва.

2020 г.

**Цель работы**

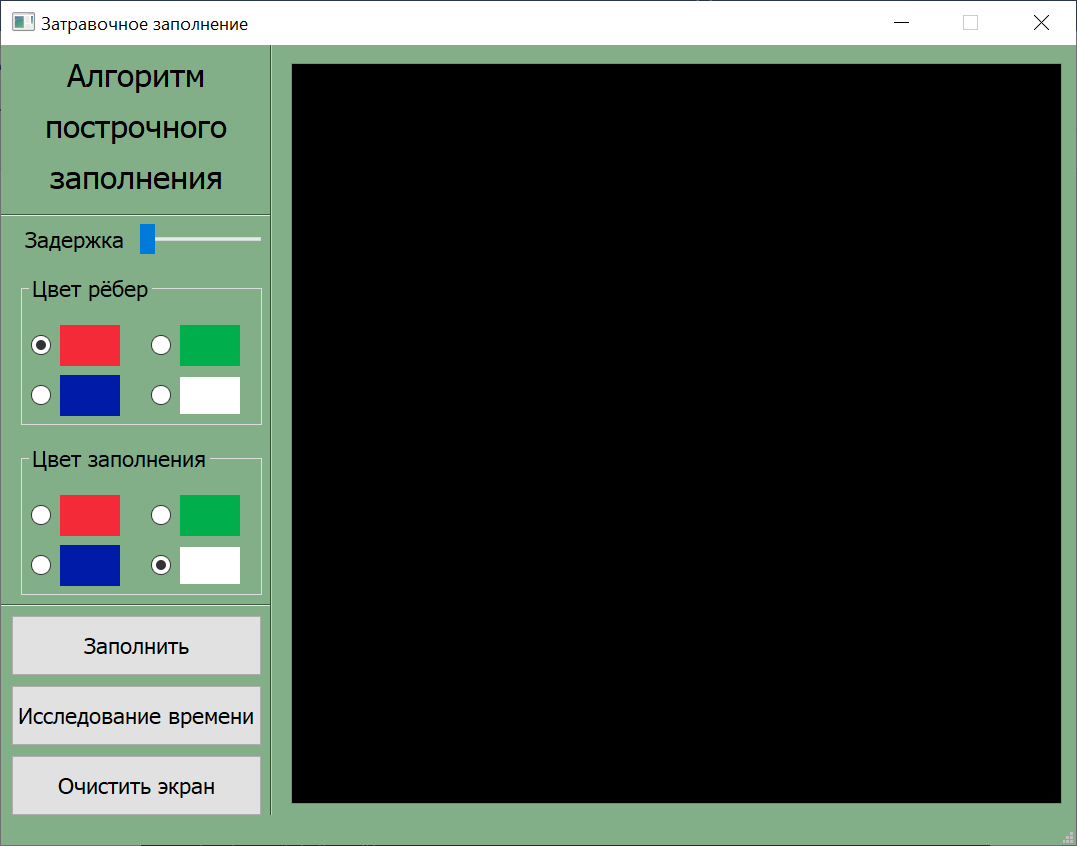
Реализация и исследование алгоритма построчного затравочного заполнения области.

**Описание задания**

1. Реализовать алгоритм
2. Обеспечить возможность ввода с рёбер и затравочной точки помощью мыши (в т.ч. и горизонтальных и вертикальных рёбер)
3. Выполнить замер времени работы алгоритма

**Описание графического интерфейса**

Для использования функционала поставленных задач, был создан графический интерфейс

****

Интерфейс предоставляет возможность:

* Выбора цвета рёбер многоугольника и цвета заполнения
* Вариации времени задержки (от без задержки до 100мс между заполнением каждой сканирующей строки)
* Ввода многоугольника при помощи мыши. Нажатие ЛКМ для ввода произвольного ребра и ПКМ для ввода горизонтального/вертикального ребра.
* Ввод затравочной точки
* Исследование временных характеристик, в зависимости от площади фигуры

**Описание и реализация алгоритма**

Задачей данной лабораторной работы служит заполнение некой гранично-определённой области, используя алгоритм построчного затравочного заполнения. Также рассматривается и возможность существования и внутренних многоугольников, “отверстий”.

Алгоритм создан для оптимизации простого затравочного алгоритма. Идея алгоритма основана на построчном заполнении указанной области, причём в памяти требуется хранить лишь одну позицию из каждой непрерывной строки заполняемых пикселей. Сами позиции хранятся в стеке, который опустошается в процессе работы.

Алгоритм имеет следующие шаги:

1. Извлечь из стека очередной затравочный пиксель (x, y)
2. Заполнить строку y до граничных пикселей вправо и влево, при этом запомнив координату x крайнего левого и правого заполняемого пикселя: x\_лев и x\_пра.
3. Просмотреть строки y-1, y+1 от x\_лев до x\_пра на наличие незаполненных строк. В случае, если такие строки найдены, добавить крайнюю правую позицию этих строк в стек.

Далее алгоритм продолжается уже для другого элемента стека. Изначально в стеке находится лишь затравочная позиция, а выполнение ведётся до полного опустошения стека (что означает, что новых не закрашенных строк области нет).

Преимуществом над простым алгоритмом затравочного заполнения является то, что в стеке будет находится гораздо меньше уже закрашенных пикселей, что позволяет минимизировать количество обращений к графической памяти. Таким образом каждый пиксель заполняемой области будет обработан алгоритмом лишь единожды.

Программная реализация:

Основная функция:

def string\_fill(self, x0, y0, delay=0):  
 stack = Stack()  
 stack.push(x0, y0)  
 while stack.is\_not\_empty():  
 x, y = stack.pop()  
 if self.is\_border\_pixel(x, y) or self.is\_fill\_pixel(x, y):  
 continue  
 self.draw\_fill\_pixel(x, y)  
  
 right\_x = self.fill\_line\_right(x + 1, y)  
 left\_x = self.fill\_line\_left(x - 1, y)  
  
 self.add\_new\_points(stack, left\_x, right\_x, y + 1)  
 self.add\_new\_points(stack, left\_x, right\_x, y - 1)  
  
 if delay != 0: # Задержка  
 self.make\_delay(delay)  
 self.update\_image()

Заполнение строки влево и вправо:

def fill\_line\_right(self, x, y):  
 while not self.is\_border\_pixel(x, y) and x < self.max\_x:  
 self.draw\_fill\_pixel(x, y)  
 x += 1  
 right\_x = x - 1  
 return right\_x  
  
def fill\_line\_left(self, x, y):  
 while not self.is\_border\_pixel(x, y) and x > self.min\_x:  
 self.draw\_fill\_pixel(x, y)  
 x -= 1  
 left\_x = x + 1  
 return left\_x

Поиск новых точек для заполнения:

def add\_new\_points(self, stack, left\_x, right\_x, y):  
 if not self.min\_y <= y <= self.max\_y:  
 return  
 x = left\_x  
 while x <= right\_x:  
 flag = False  
 # Переход к самому правому пикселю незаполненной области  
 while x <= right\_x and not self.is\_border\_pixel(x, y) \  
 and not self.is\_fill\_pixel(x, y):  
 flag = True  
 x += 1  
  
 # Если пиксель найден, то заносится в стек  
 if flag:  
 stack.push(x - 1, y)  
  
 # Переход к следующей незаполненной области  
 temp\_x = x  
 while (self.is\_border\_pixel(x, y) or self.is\_fill\_pixel(x, y)) \  
 and x < right\_x:  
 x += 1  
 if temp\_x == x:  
 x += 1

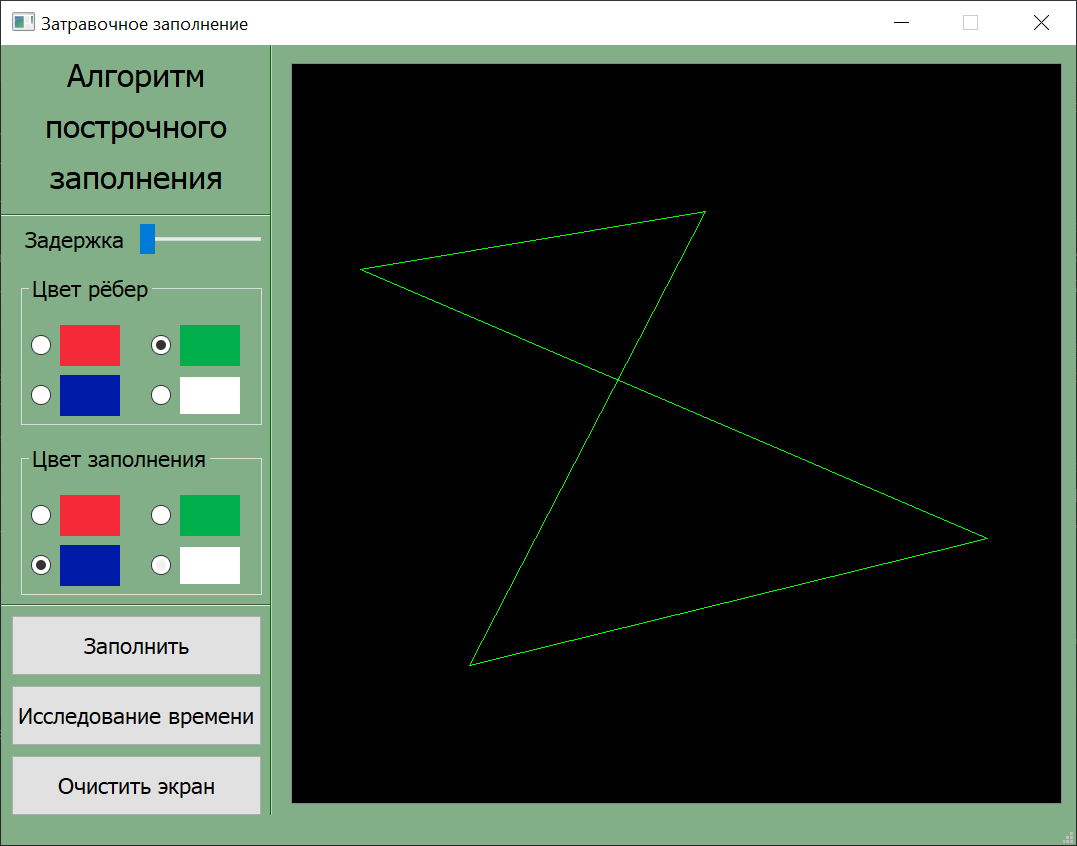
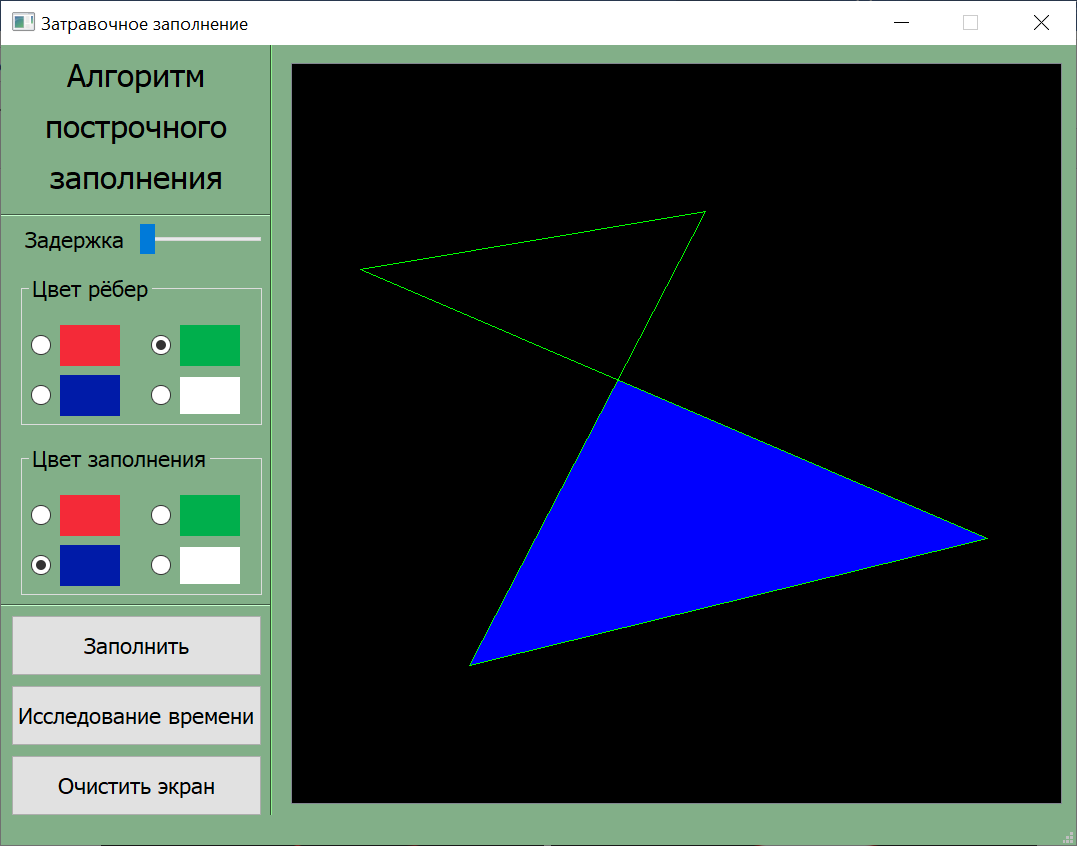
Стек был реализован с использованием динамического списка list():

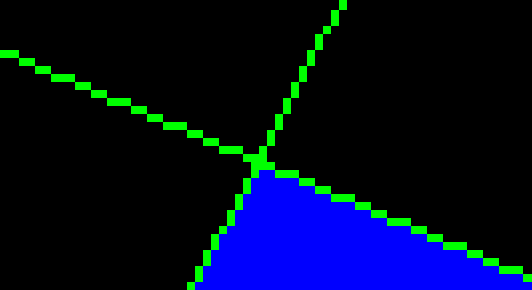
class Stack(object):  
 point\_list = list()  
 def pop(self):  
 return self.point\_list.pop()  
 def push(self, x, y):  
 self.point\_list.append((x, y))  
 def is\_not\_empty(self):  
 return len(self.point\_list) != 0

Примечание: в программе также предусмотрен случай выхода области заполнения за пределы экрана. Такое может быть, когда точкой затравки выбрана область снаружи замкнутой области или при заполнении незамкнутой области. Для этого производится сравнение x, y с крайними точками экрана.

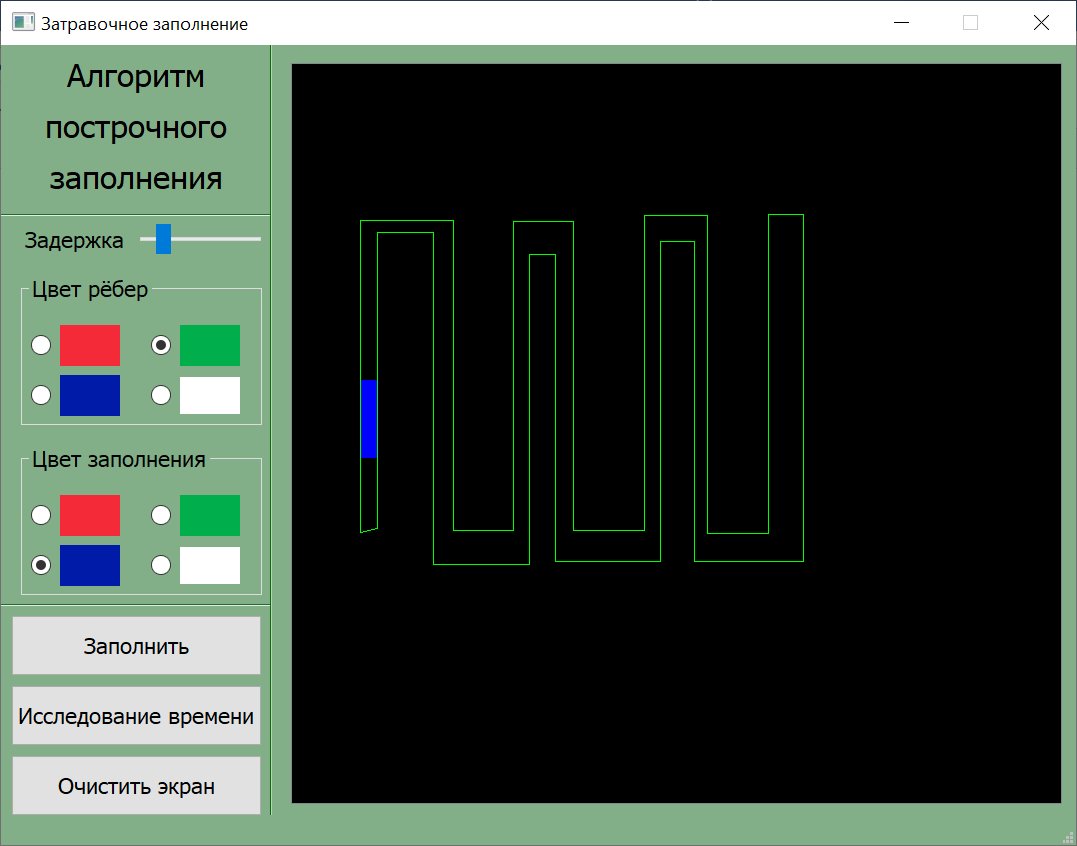
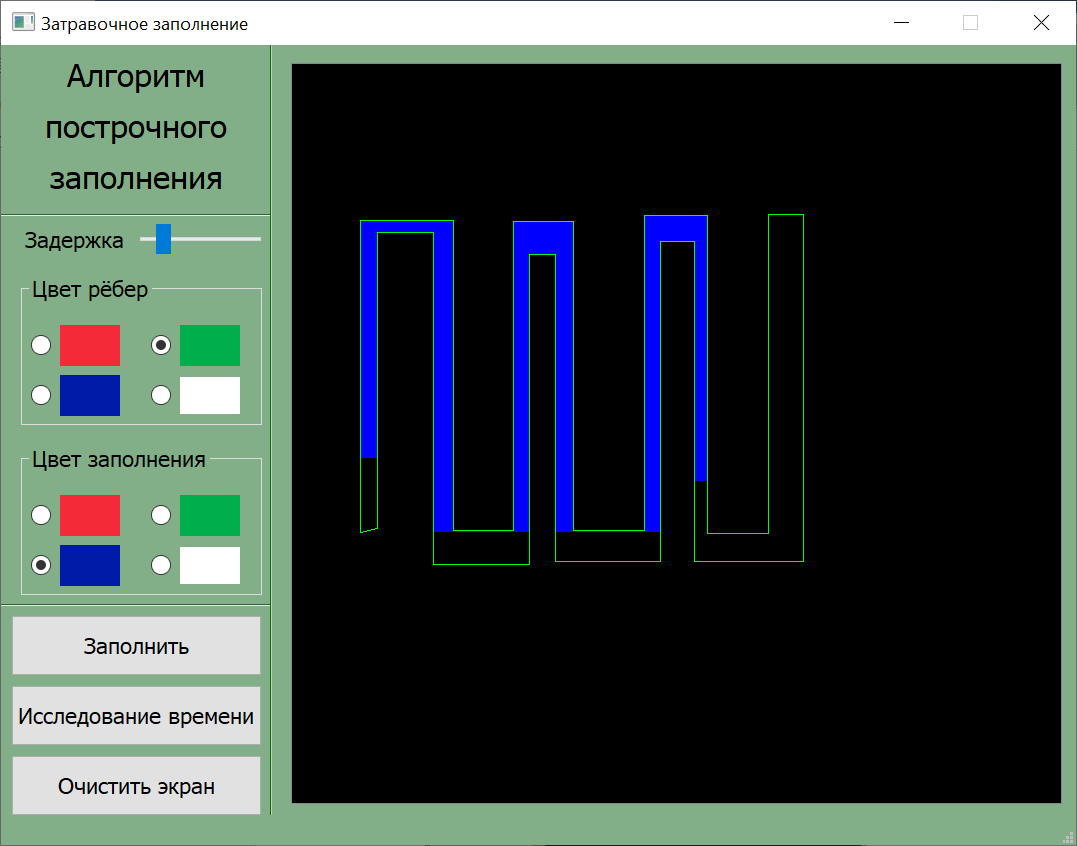
**Визуальные характеристики**

Ниже приведены примеры работы программы в различных ситуациях:



Использование задержки:

Задержка наглядно демонстрирует процесс заполнения.

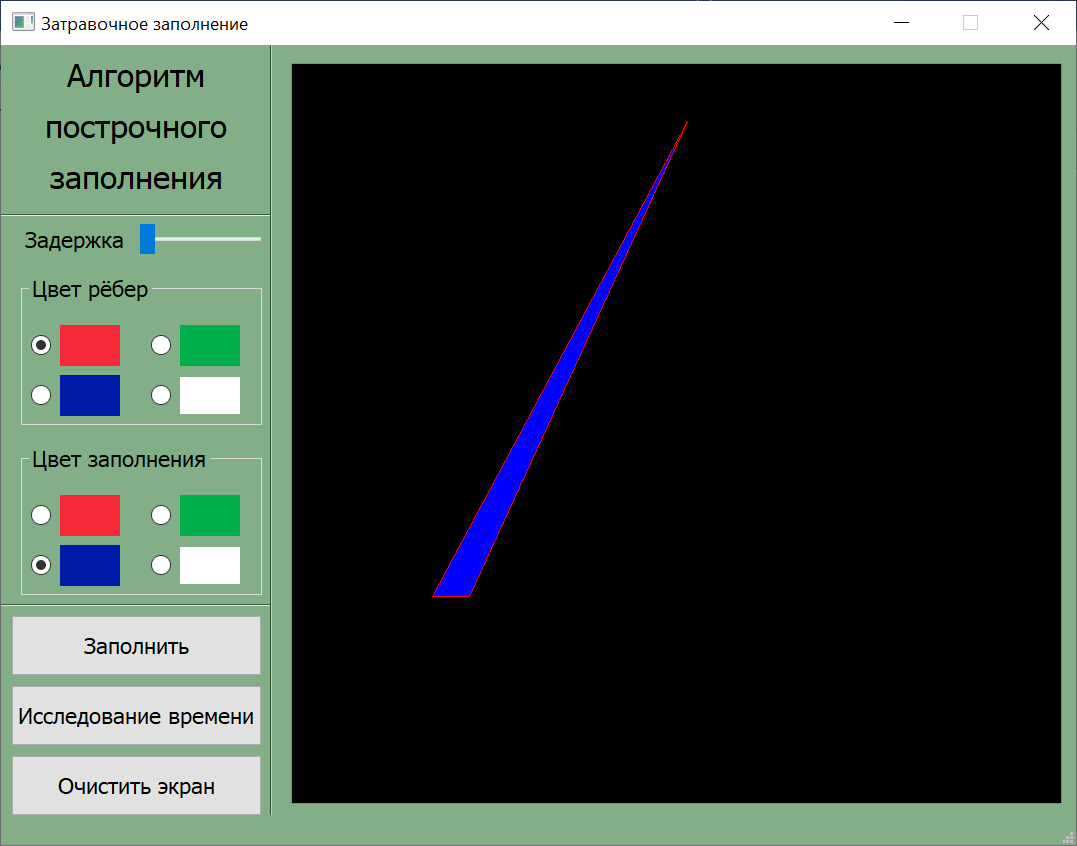
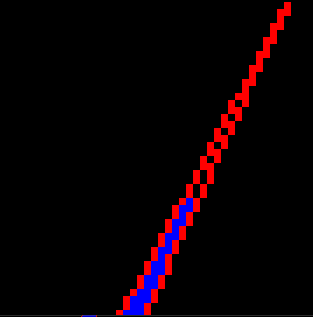
Примеры с краевыми случаями:

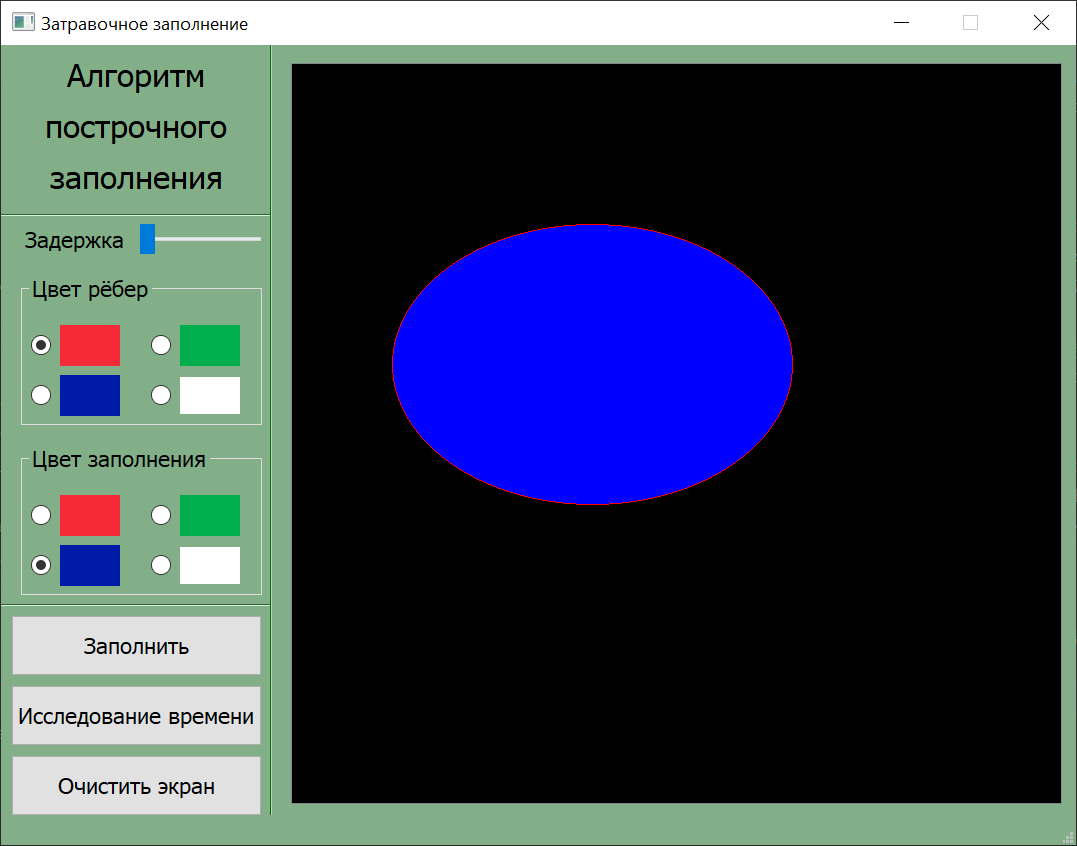
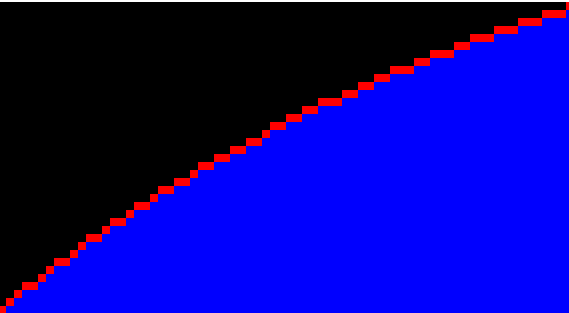
- “Узкие” строки

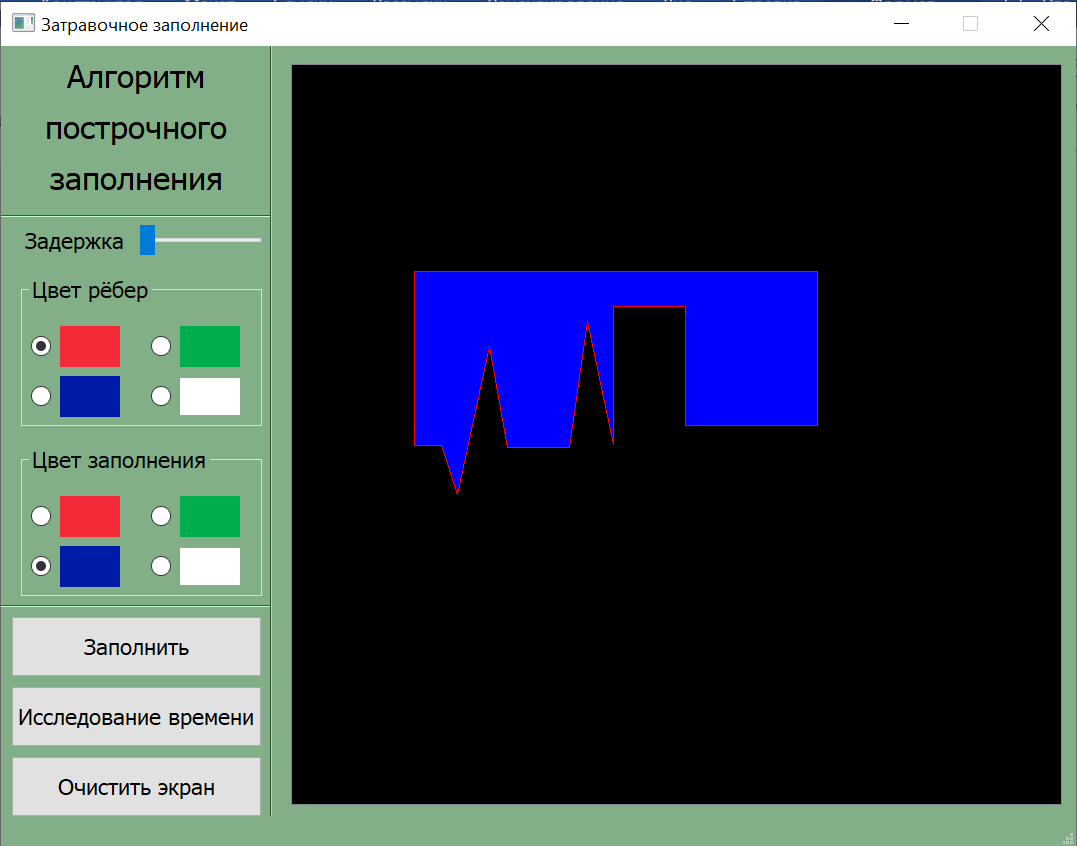
- Кривая линия, ограничивающая область

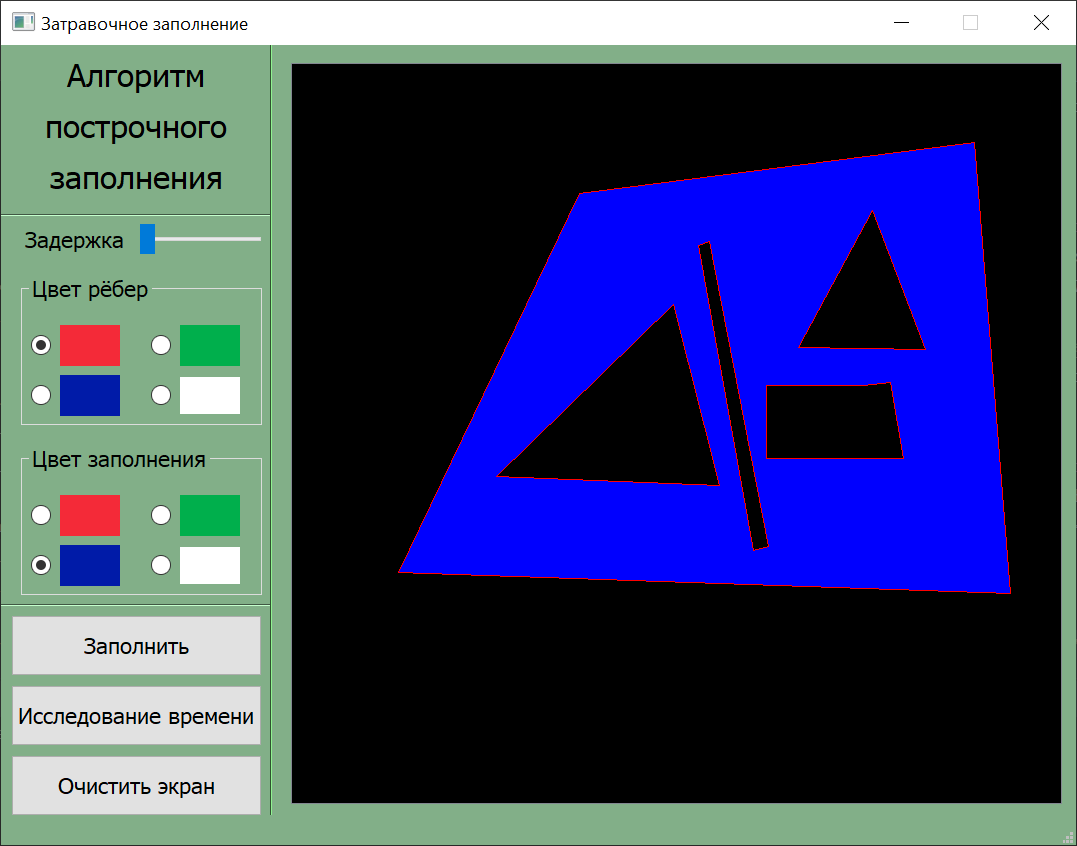
- Вертикальные и горизонтальные линии

- Отверстия внутри фигуры

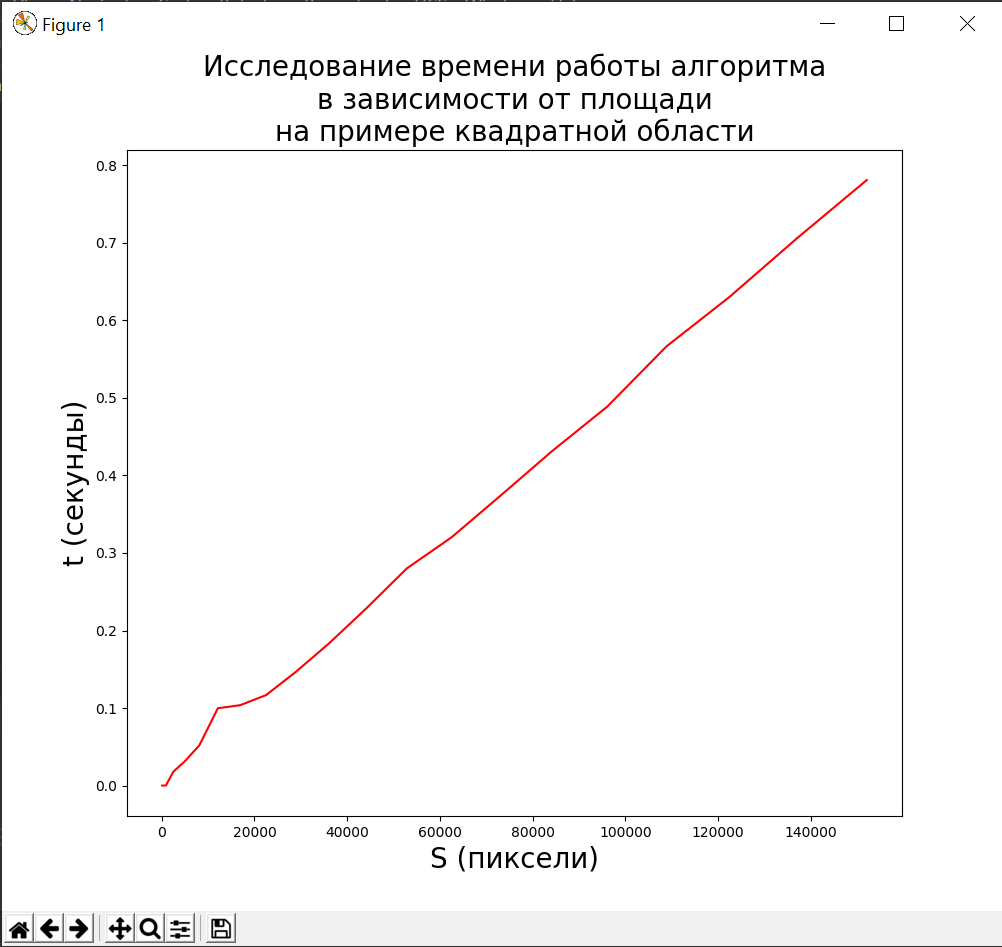
 





**Анализ временных характеристик**

В программе реализовано средство замера времени работы алгоритма в зависимости от площади фигуры. В качестве закрашиваемой фигуры был выбран квадрат. Снизу приведена площадь, вычисленная как .



Из графика становится видно, что данный алгоритм демонстрирует линейную зависимость от площади фигуры. Это объясняется тем, что вне зависимости от формы, каждый пиксель области будет обработан однократно.

**Заключение**

В ходе лабораторной работы были изучены различные алгоритмы затравочного заполнения областей и был реализован алгоритм построчного заполнения. Проведено исследование временных характеристик.