|  |  |
| --- | --- |
| 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 № 5**

|  |  |
| --- | --- |
| **Дисциплина Компьютерная графика**  **Тема Программная реализация и исследование алгоритмов растрового заполнения сплошных областей**  **Студент Иванов В.А.**  **Вариант 9 (Алгоритм с упорядоченным списком ребер (лучший вариант из возможных)**  **Группа ИУ7-42Б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель Куров А.В.** |  |

Москва.

2020 г.

**Цель работы**

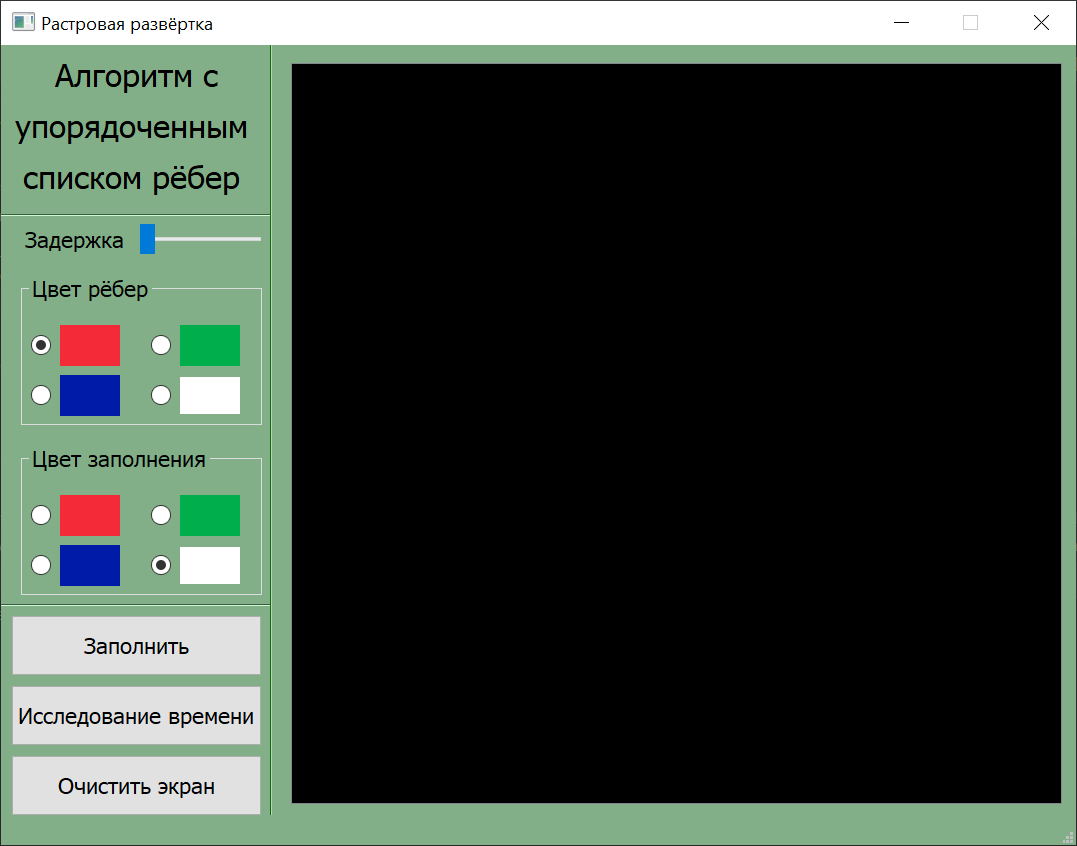
Реализация и исследование одного из алгоритмов (Алгоритм с упорядоченным списком ребер) растрового заполнения области

**Описание задания**

1. Реализовать алгоритм
2. Обеспечить возможность ввода с 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.
* Исследование временных характеристик, в зависимости от площади фигуры

**Описание и реализация алгоритма**

Задачей данной лабораторной работы служит заполнение некой области, заданной множеством рёбер, образующих многоугольник. Также рассматривается и возможность существования и внутренних многоугольников, “отверстий”.

Примечание: для наглядности, работа алгоритма была выбрана таким образом, чтобы не заполнять пиксели рёбер

В моём варианте требовалось реализовать алгоритм с упорядоченным списком ребер. Суть алгоритма заключается в следующем:

1. Определить координаты точек пересечения сканирующей строки с рёбрами фигуры.
2. Составить из точек, лежащих на одной координате ординат пары точек. При этом следует учитывать, что в вершинах многоугольника два ребра будут иметь общую точку пересечения с сканирующей строкой
3. Произвести заливку путём соединения пар точек линиями.

Алгоритм может быть оптимизирован по памяти, если использовать y-группы и список активных рёбер. Оптимизация состоит в том, что будут обрабатываться только рёбра, имеющие пересечения с данной сканирующей строкой, а также в том, что храниться будут только точки пересечения данной сканирующей строки.

Перед началом заливки, по каждому ребру определяется такая структура, как активное ребро. Оно содержит:

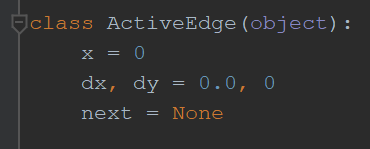
x – точка пересечения текущей сканирующей строки с ребром

dx – изменение x при переходе к следующей строке

dy – количество сканирующих строк, пересекающих ребро, и ещё не обработанных

next – указатель на следующее активное ребро

Реализация в программе:



Эти структуры распределяются по y-группам. Ребро помещается в y-группу, равную наивысшей сканирующей строке, пересекающей его.

Реализация в программе:

def create\_y\_group(edge\_arr):  
 y\_group = {}  
 for edge in edge\_arr:  
 active\_edge = ActiveEdge()  
 active\_edge.x = edge.x1  
 active\_edge.dy = edge.y1 - edge.y2  
 if active\_edge.dy != 0:  
 active\_edge.dx = (edge.x2 - edge.x1) / active\_edge.dy  
 else:  
 active\_edge.dy = -2  
 active\_edge.dx = 0  
  
 if edge.y1 in y\_group.keys():  
 y\_group[edge.y1].append(active\_edge)  
 else:  
 y\_group[edge.y1] = [active\_edge]

return y\_group

На вход подаётся множество рёбер edge\_arr, элементы которого состоят из значений точек (x1, y1) и (x2, y2). После чего производится проход по каждому ребру: из него образуется структура активного ребра и помещается в соответствующую y-группу.

Когда y-группа сформирована, алгоритм переходит к отрисовке. Для экономии памяти, пары X-ов определяются непосредственно из САР. Для каждой сканирующей строки y:

1. В САР помещаются новые рёбра из y-групп. При этом вставка в САР происходит так, чтобы не нарушить его упорядоченность (по возрастанию x)
2. Начинается проход по всем точкам рёбер.
   1. В случае, если очередная пара точек не совпадает производится отрисовка линии.

Отдельно требуется рассмотреть случай, когда ребро имеет уклон, при котором на одной строчке расположено сразу несколько пикселей этого ребра (то есть при |dx| > 1). В таком случае для правой точки требуется перейти к самой правому пикселю ступеньки, а для левой – к самому левому.

После чего производится отрисовка линии и переход к следующей паре пикселей. Отдельно рассматривается ситуация, когда левый пиксель пары являлся одной из вершин

* 1. В случае, если две точки пары совпали производится анализ. В случае, если точка является локальным экстремумом, следует пропустить обе точки. Иначе, следует пропустить только одну точку, а от другой построить линию до точки, идущей далее по списку.

1. Переход к следующей y-строке. Все элементы САР уменьшаются на 1 по dy, и значение x корректируется на dx. Из списка исключаются элементы с dy < 0.
2. В случае, если есть задержка производится высвечивание уже отрисованного изображения и задержка на указанное время

Реализация в программе. САР обозначен как lae (List of Active Edges):

def fill\_sorted\_list(self, delay=0):  
 if not len(self.edge\_arr):  
 return # В случае, если нет рёбер  
 # Список активных рёбер  
 lae = None  
  
 # Формирование y-группы  
 y\_group = create\_y\_group(self.edge\_arr)  
 y = max(y\_group.keys())  
 y\_min = find\_min\_y(self.edge\_arr)  
  
 while y > y\_min:  
 # Добавление новых элементов из y-группы  
 if y in y\_group.keys():  
 for edge in y\_group[y]:  
 lae = lae\_add(lae, edge)  
  
 # Проход во всем точкам пересечения  
 temp\_edge = lae  
 while temp\_edge is not None and temp\_edge.next is not None:  
 next\_edge = temp\_edge.next  
 if not is\_match(temp\_edge, next\_edge):  
 # В случае, если точки не совпадают  
 if abs(temp\_edge.dx) > 1:  
 x1 = round\_down(temp\_edge.x + abs(temp\_edge.dx/2))  
 else:  
 x1 = round(temp\_edge.x)  
  
 if abs(next\_edge.dx) > 1:  
 x2 = round\_up(next\_edge.x - abs(next\_edge.dx/2))  
 else:  
 x2 = round(next\_edge.x)  
  
 self.draw\_line(x1+1, x2-1, y)  
  
 temp\_edge = next\_edge.next  
 if temp\_edge is not None and is\_match(temp\_edge, next\_edge):  
 if not is\_extremum(temp\_edge.dy, next\_edge.dy):  
 temp\_edge = temp\_edge.next  
 elif is\_horisontal(temp\_edge) or is\_horisontal(next\_edge):  
 temp\_edge = temp\_edge.next  
 else:  
 # В случае, если точки совпали  
 if is\_horisontal(temp\_edge) or is\_horisontal(next\_edge):  
 temp\_edge = next\_edge.next  
 if temp\_edge is not None:  
 temp\_edge = temp\_edge.next  
 elif is\_extremum(temp\_edge.dy, next\_edge.dy):  
 temp\_edge = next\_edge.next  
 else:  
 temp\_edge = next\_edge  
 # Переход к следующей строке  
 y -= 1   
 lae = lae\_step(lae)  
  
 if delay != 0: # Задержка  
 self.make\_delay(delay)

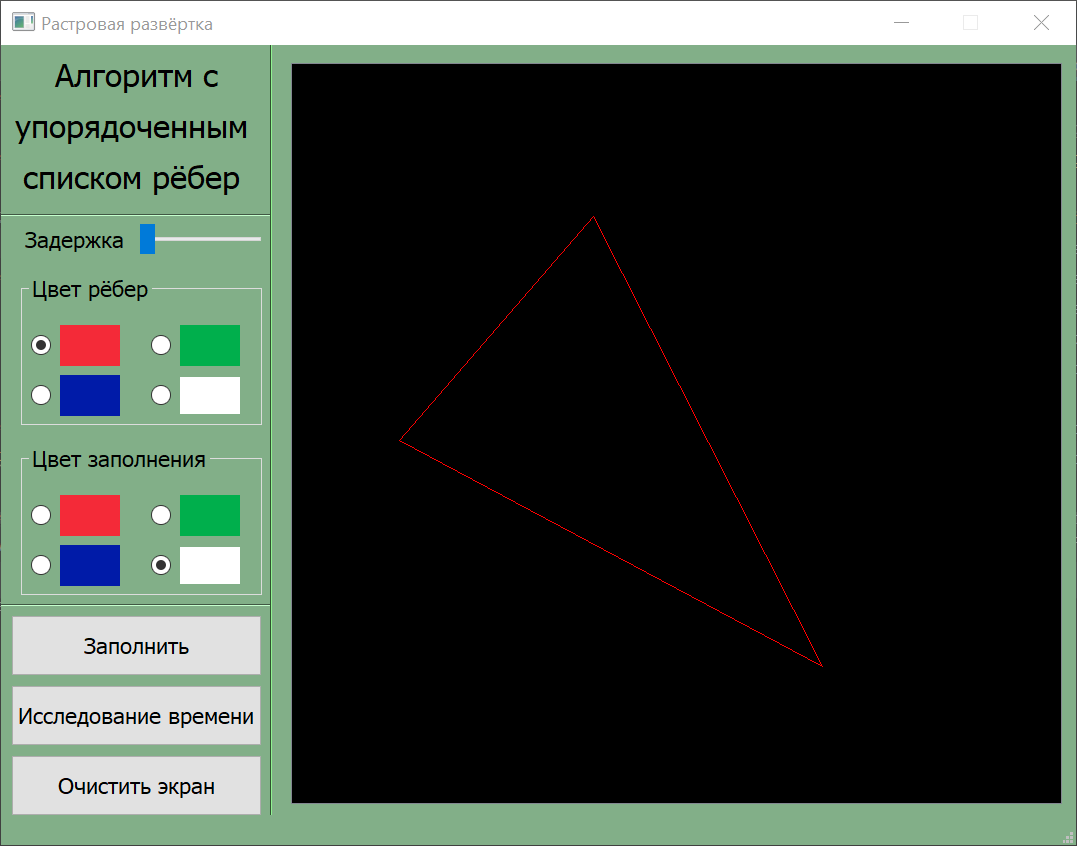
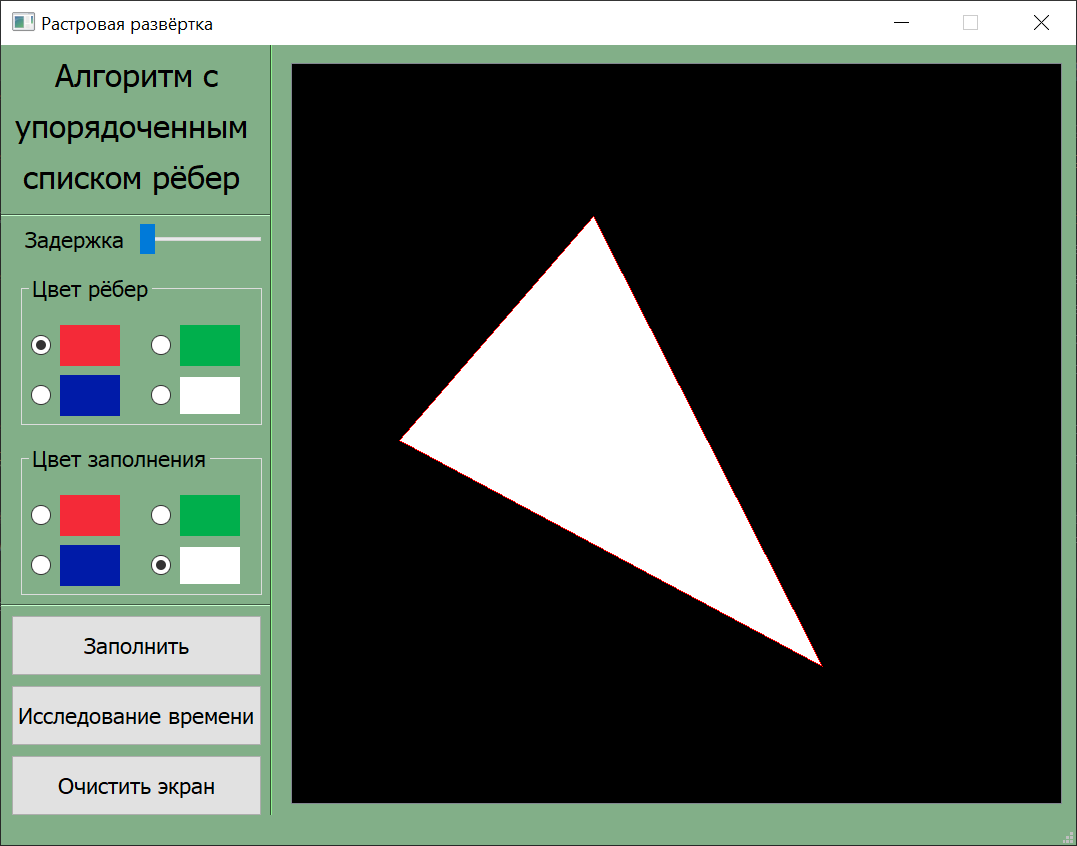
Прикладные функции:

|  |  |
| --- | --- |
| Добавление ребра в САР:  def lae\_add(lae, edge):  if lae is None or edge.x < lae.x or \  (edge.x == lae.x and edge.dx < lae.dx):  edge.next = lae  return edge   temp = lae  while temp.next is not None:  if edge.x < temp.next.x\  or (edge.x == temp.next.x and edge.dx < temp.next.dx):  switch = temp.next  temp.next = edge  edge.next = switch  return lae  else:  temp = temp.next  temp.next = edge  return lae | Обновление САР при переходе к новой строке:  def lae\_step(lae):  temp = lae  while temp is not None:  temp.dy -= 1  temp.x += temp.dx  temp = temp.next   while lae.dy < 0:  lae = lae.next  temp = lae  while temp.next is not None:  if temp.next.dy < 0:  temp.next = temp.next.next  else:  temp = temp.next   return lae |

**Визуальные характеристики**

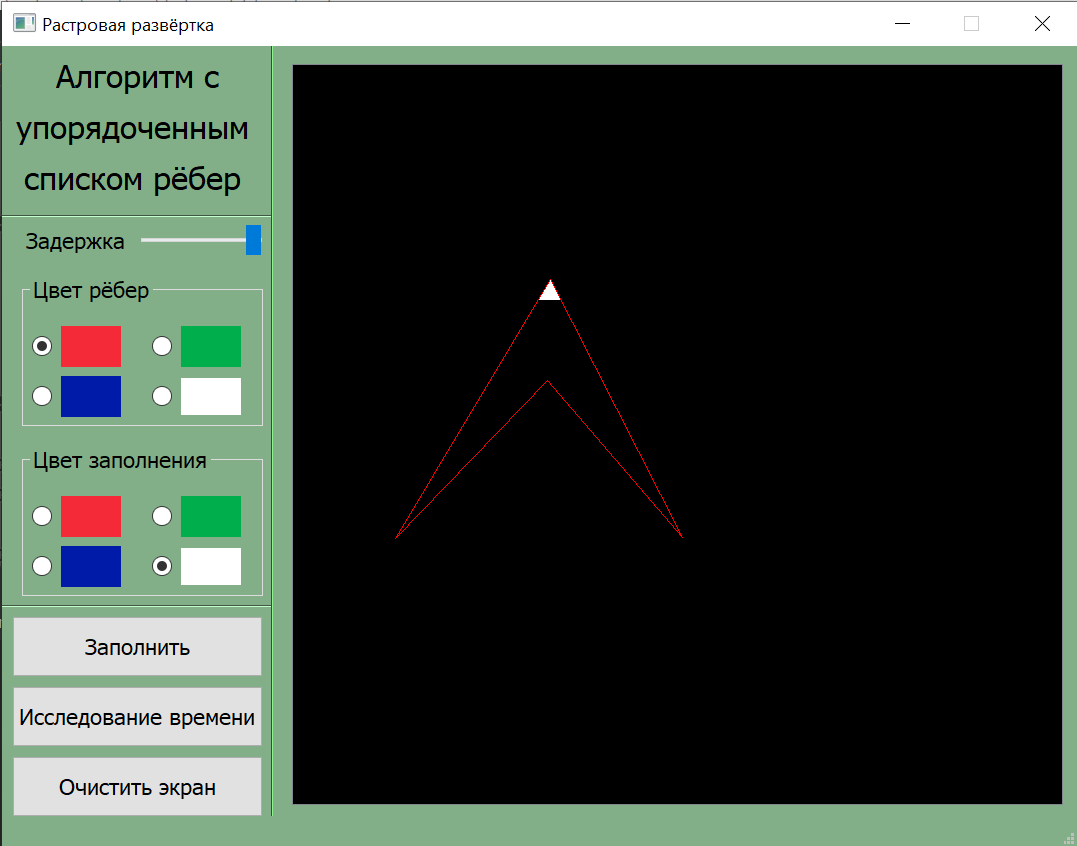
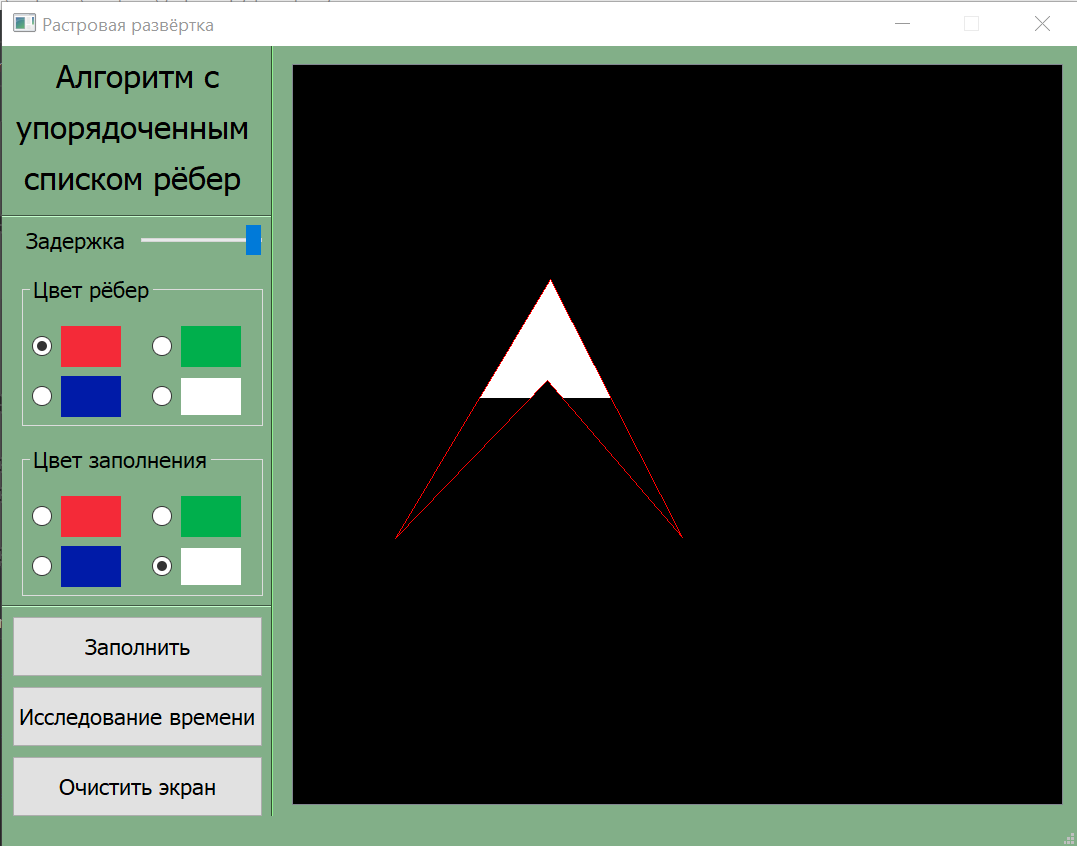
Ниже приведены примеры работы программы в различных ситуациях:

Как упоминалось ранее, алгоритм построен так, что заполняет пиксели, лежащие строго внутри указанной области, то есть не перекрашивая введённые рёбра. Например:

Использование задержки:

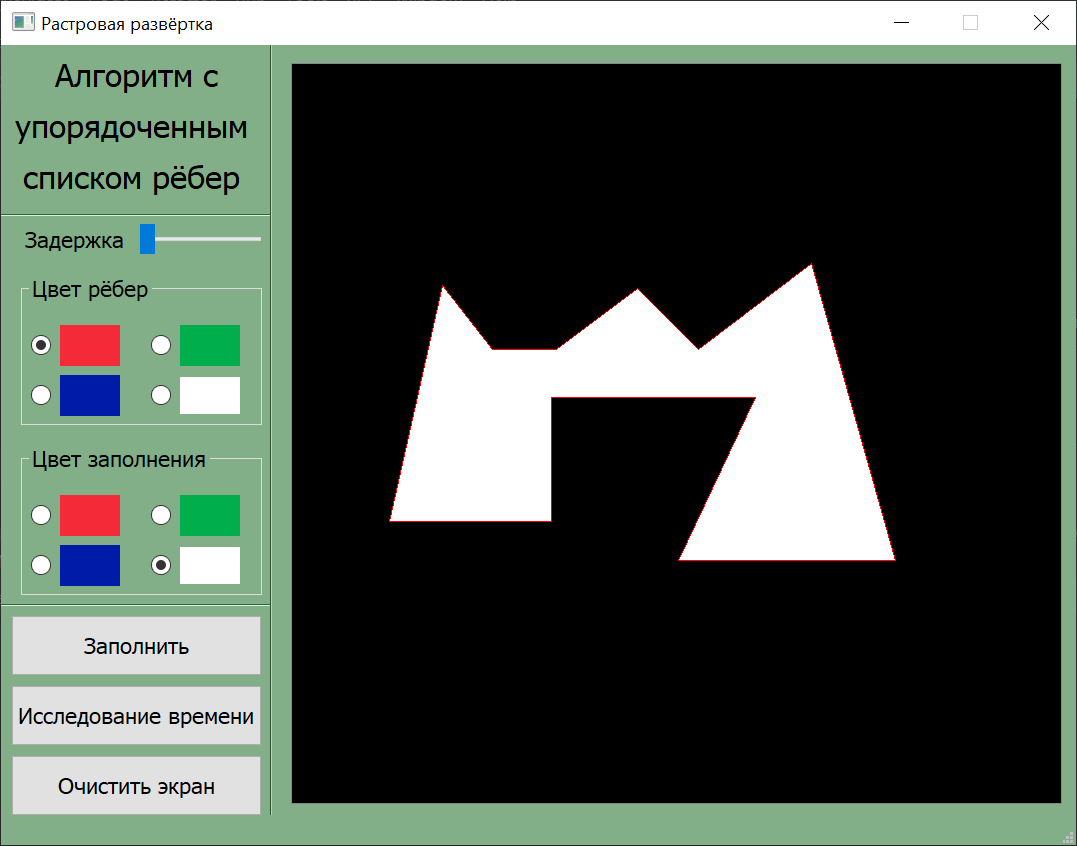
Задержка наглядно демонстрирует процесс заполнения.

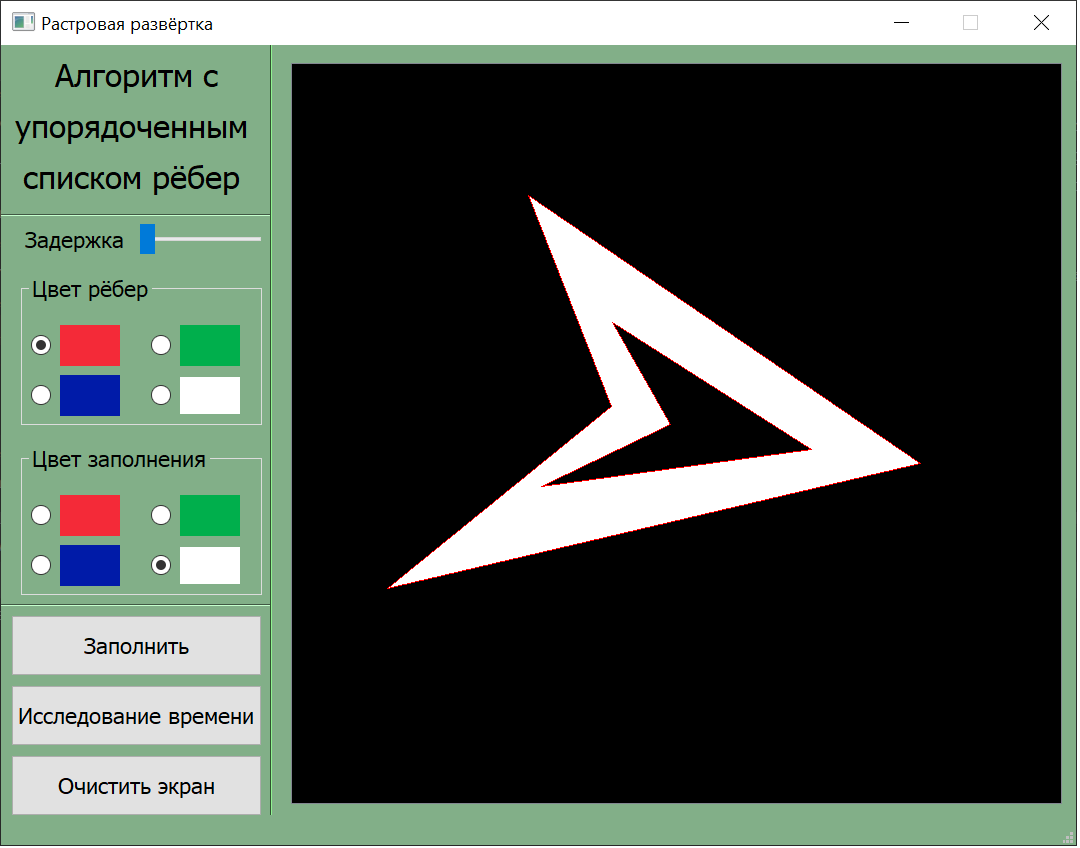
Примеры с краевыми случаями:

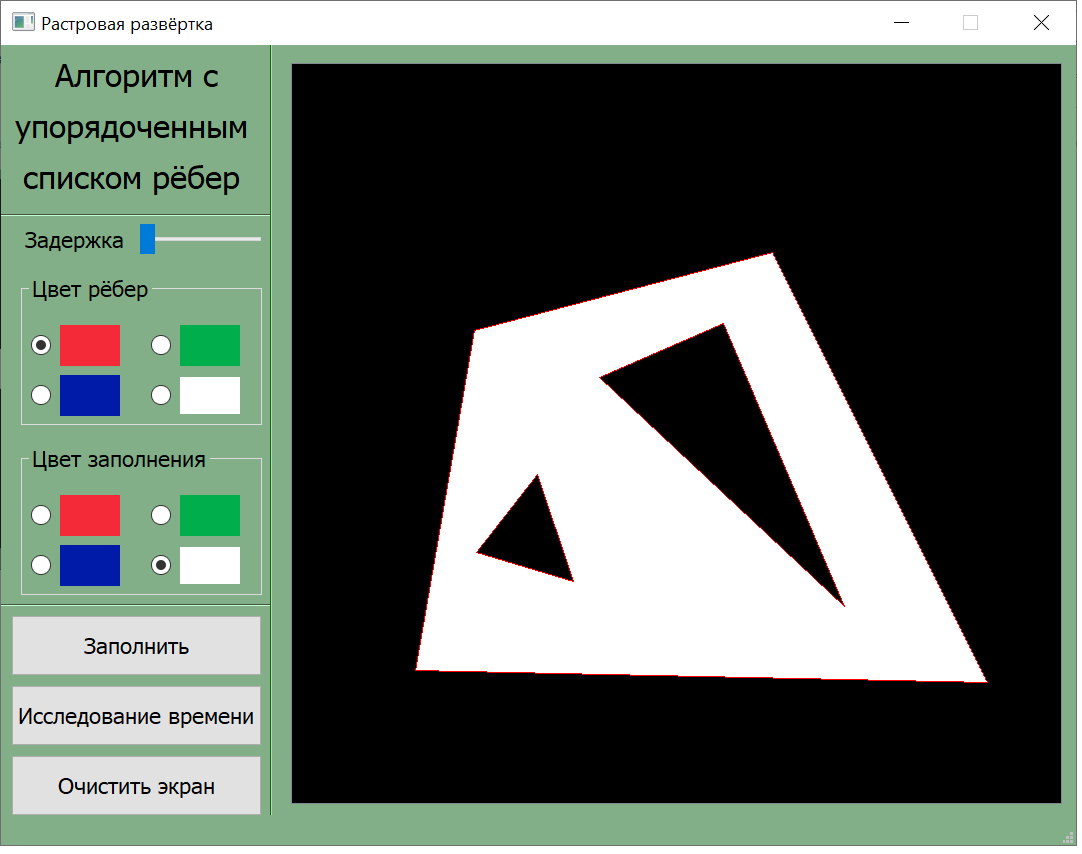
- Прохождение строчек через вершины-экстремумы

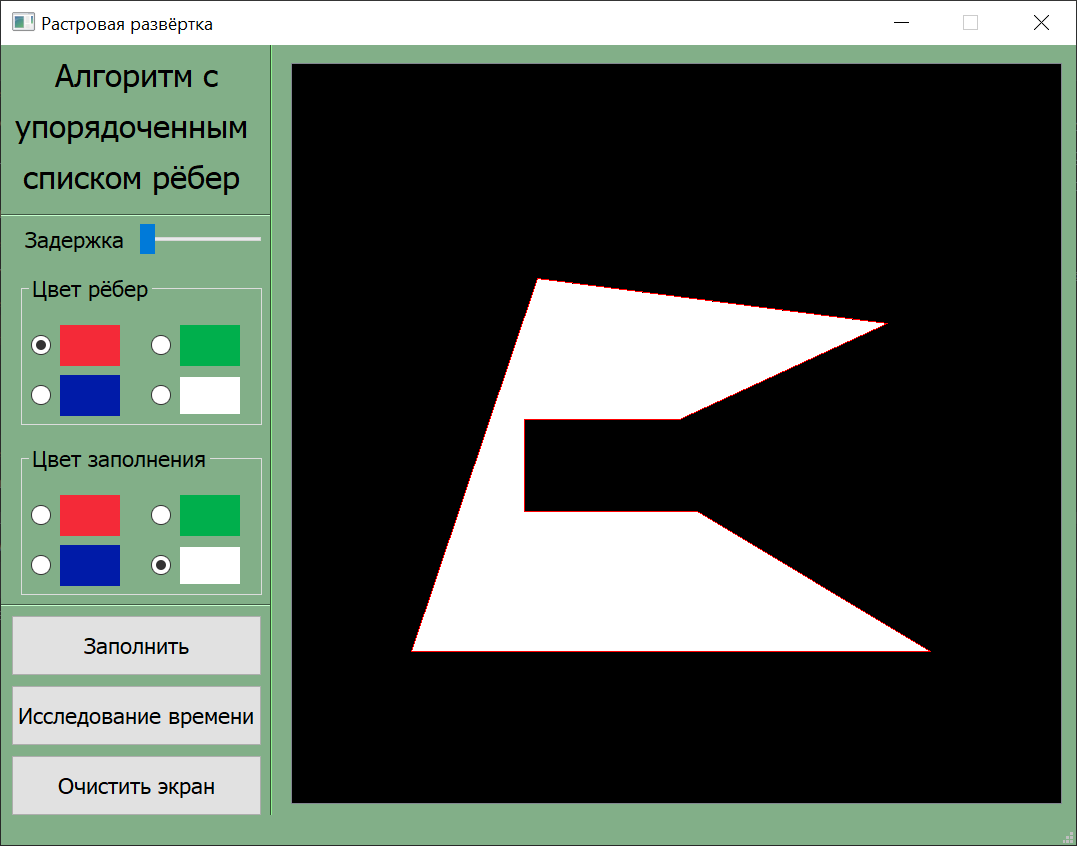
- Вертикальные и горизонтальные линии

- Отверстия внутри фигуры



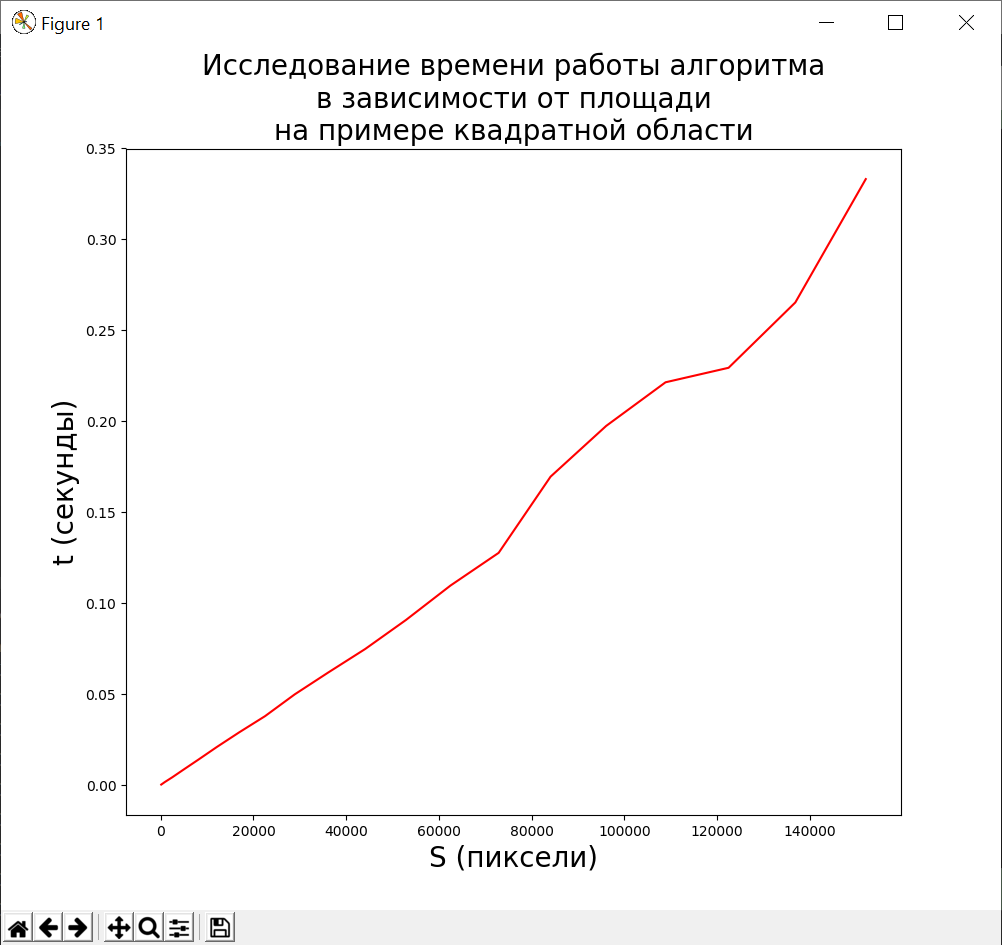






**Анализ временных характеристик**

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.

**Заключение**

В ходе лабораторной работы были изучены различные алгоритмы растрового заполнения областей и был реализован алгоритм с упорядоченным списком рёбер. Выявлены и проработаны аспекты его реализации. Также проведено исследование временных характеристик.