|  |  |
| --- | --- |
| 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** Сусликов Д. В.  **Группа ИУ 7-45**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |  |

Москва.

2020 г.

**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).

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

**Теоретический материал:** главное отличие алгоритма построчного затравочного заполнения от простого заключается в том, размер стека минимизирован и не хранит не нужную информацию.

**Описание алгоритма построчного затравочного заполнения:**

Ввод исходных данных (информация о границе области, цвет границы, цвет закращивания)

Вычерчивание границ области

*Занесение затравочного пиксела в стек*

**Push** затравка (х, у)

**Пока** стек не пуст

*Извлечение пиксела из стека*

**Pop** Пиксел (х, у)

*сохраняем х-координату затравочного пиксела*

Врем\_х = х

*заполняем интервал справа от затравки*

х = х + 1

**Пока** Пиксел (х, у) < > Гран\_—значение

Пиксел (х, у) = Нов\_значение *(закрашивание пиксела)*

x = х + 1

*сохраняем крайний справа пиксел*

Хправ = х – 1

*восстанавливаем х-координату затравки*

х = Врем\_х

*заполняем интервал слева от затравки*

х = х – 1

**Пока** Пиксел (х, у) < > Гран\_значение

Пиксел (х, у} = Нов значение

x = х - 1

*сохраняем крайний слева пиксел*

Хлев =х+ 1

х = Xлев

y = y + 1

**Пока** x <= Xправ

*ищем затравку на строке выше*

Флаг = 0

**Пока** (Пиксел (x, y) < > Гран\_значение **И**

Пиксел (x, y) < > Нов\_значение **И**x < Xправ)

**Если** Флаг = 0 **тогда** Флаг = 1

x = x + 1

*помещаем в стек крайний справа пиксел*

**Если** Флаг = 1 **тогда**

**Если** (x = Xправ **И** Пиксел(x, y) < > Гран\_значение

**И** Пиксел (x, y) < > Нов\_значение) **тогда**

**Push** Пиксел (x, y)

**Иначе**

**Push** Пиксел (x – 1, y)

Флаг = 0

*продолжим проверку, если интервал был прерван*

Врем\_х = х

**Пока** ((Пиксел (x, y) = Гран\_значение **ИЛИ**

Пиксел(x, y) = Нов\_значение) **И** x < Xправ)

x = x + 1

*удостоверимся, что координата пиксела увеличена*

**Если** x = Врем\_х **тогда** х = х + 1

*Аналогично всё для строки ниже*

*(нужно лишь опуститься на y = y – 2)*

**Алгоритма построчного затравочного заполнения:**

def algorithm(self):

        time\_start = time.time()

        # вычерчивание границ области

        self.draw\_edges()

        pix = QPixmap()

        my\_painter = QPainter()

        my\_painter.begin(self.image)

        # ввод исходных данных

        ends = self.ends\_colour

        my\_painter.setPen(self.pen)

        stack = []

        # занесение затравочного пиксела в стек

        push(stack, [self.x\_seed, self.y\_seed])

        while stack:

            # извлечение пиксела из стека

            current\_pixel = pop(stack)

            x = current\_pixel[0]

            y = current\_pixel[1]

            my\_painter.drawPoint(x, y)

            # сохраняем х-координату затравочного пиксела

            x\_temp = x

            # заполняем интервал справа от затравки

            x += 1

            while self.image.pixelColor(x, y) != ends:

                my\_painter.drawPoint(x, y)

                x += 1

            # сохраняем крайний справа пиксел

            x\_right = x - 1

            # восстанавливаем x-координату затравки

            x = x\_temp

            # заполняем интервал слева от затравки

            x -= 1

            while self.image.pixelColor(x, y) != ends:

                my\_painter.drawPoint(x, y)

                x -= 1

            # сохраняем крайний справа пиксел

            x\_left = x + 1

            x = x\_left

            y += 1

            while x <= x\_right:

                # ищем затравку на строке выше

                flag = 0

                while self.image.pixelColor(x, y) != ends and \

                    self.image.pixelColor(x, y) != self.pen\_colour\

                        and x < x\_right:

                    if flag == 0:

                        flag = 1

                    x += 1

                # помещаем в стек крайний справа пиксел

                if flag == 1:

                    if x == x\_right and self.image.pixelColor(x, y) != ends\

                        and self.image.pixelColor(x, y) != self.pen\_colour:

                        push(stack, [x, y])

                    else:

                        push(stack, [x - 1, y])

                    flag = 0

                 # продолжим проверку, если интервал был прерван

                x\_temp = x

                while self.image.pixelColor(x, y) == ends or \

                    self.image.pixelColor(x, y) == self.pen\_colour\

                        and x < x\_right:

                    x += 1

                # удостоверимся, что координата пиксела увеличена

                if x == x\_temp:

                    x += 1

            # проверяем строку ниже

            # (аналогично строке выше, только нужно опуститься по y)

            x = x\_left

            # переходим на нижнюю строку

            y -= 2

            while x <= x\_right:

                flag = 0

                while self.image.pixelColor(x, y) != ends and \

                    self.image.pixelColor(x, y) != self.pen\_colour\

                        and x < x\_right:

                    if flag == 0:

                        flag = 1

                    x += 1

                if flag == 1:

                    if x == x\_right and self.image.pixelColor(x, y) != ends\

                        and self.image.pixelColor(x, y) != self.pen\_colour:

                        push(stack, [x, y])

                    else:

                        push(stack, [x - 1, y])

                    flag = 0

                x\_temp = x

                while self.image.pixelColor(x, y) == ends or \

                    self.image.pixelColor(x, y) == self.pen\_colour\

                        and x < x\_right:

                    x += 1

                if x == x\_temp:

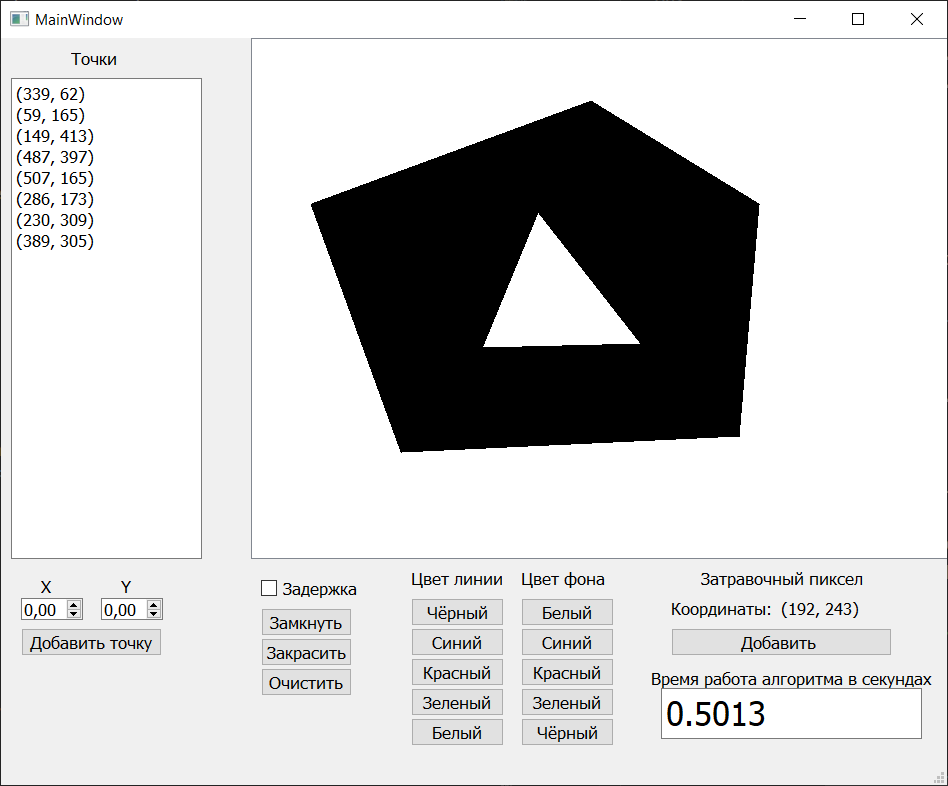
                    x += 1

        time\_process = float(time.time() - time\_start)

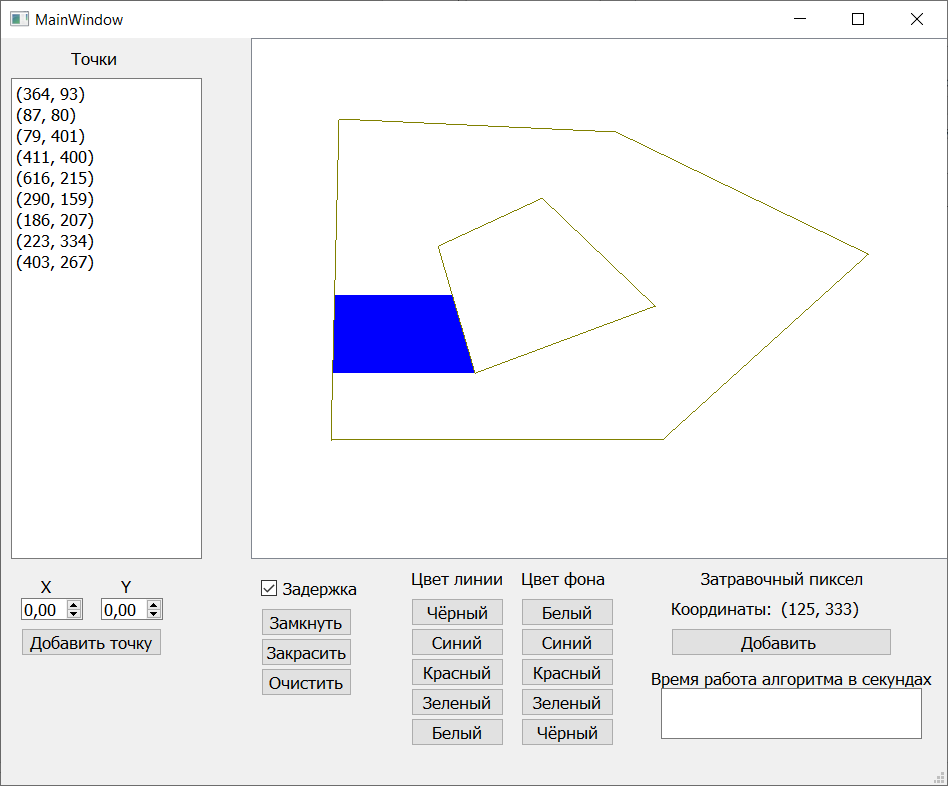
        self.timing\_text.setText(str(round(time\_process, 4)))

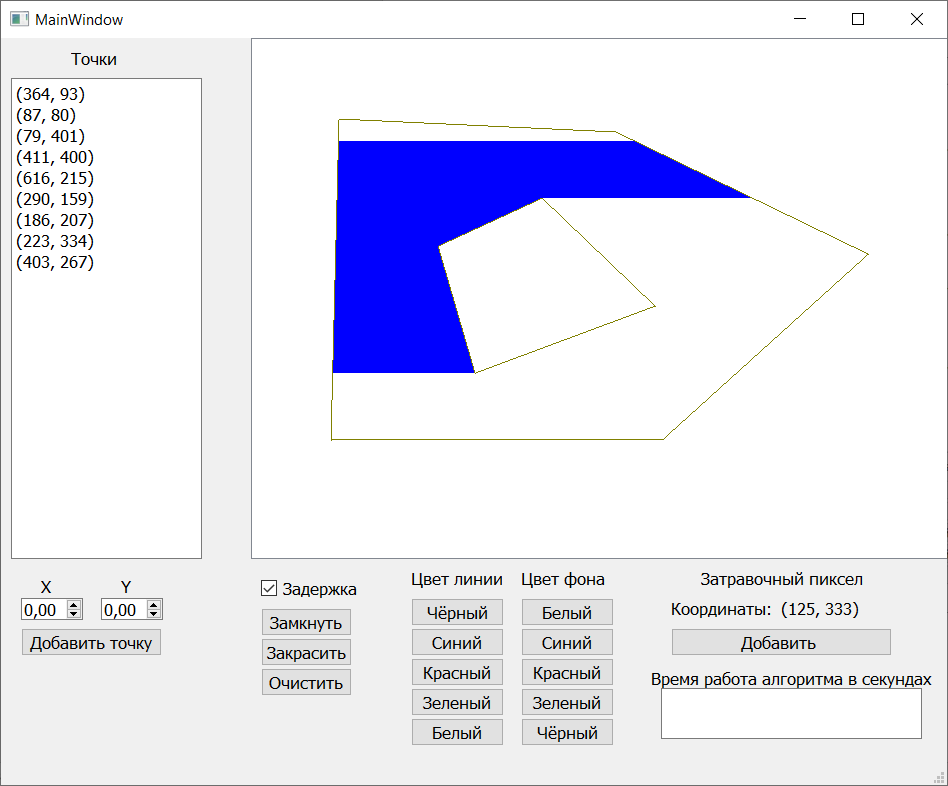
        pix.convertFromImage(self.image)

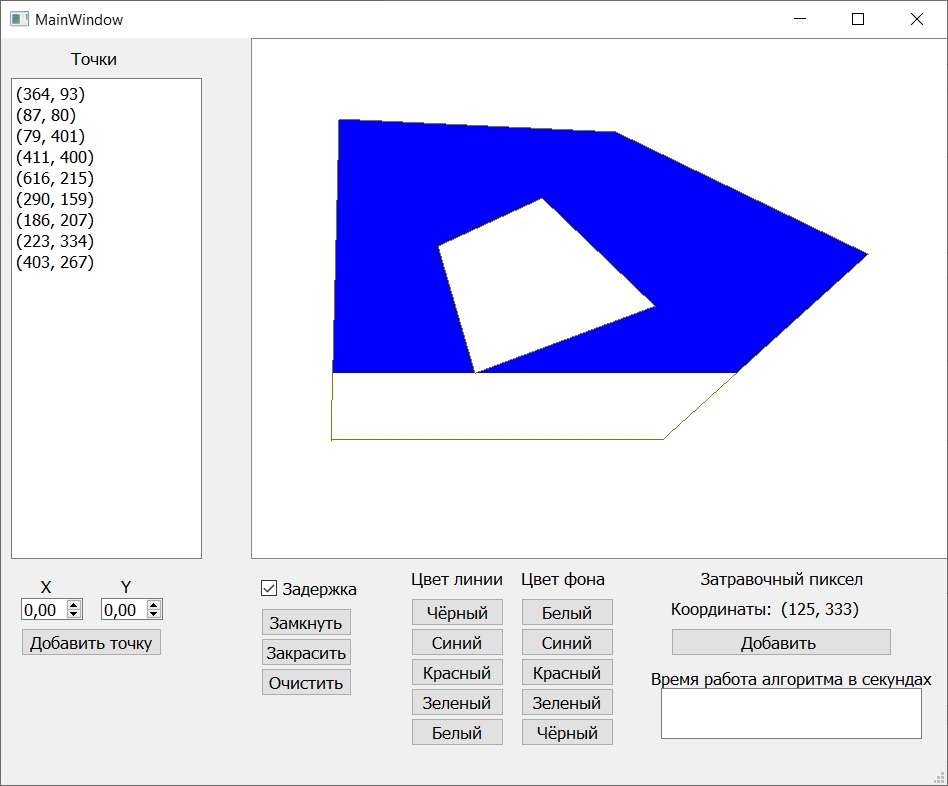
        self.scene.addPixmap(pix)

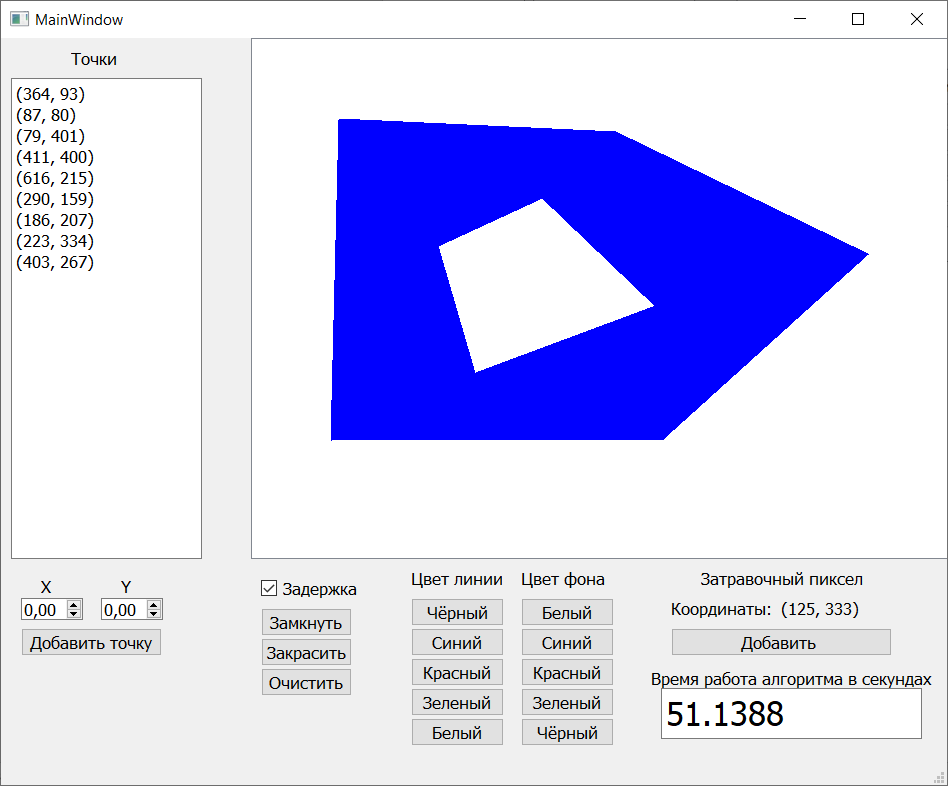
Пример программы:

Пример работы алгоритма с задержкой









Видео-пример работы программы по ссылке: https://youtu.be/lGKAMmlR8oo