

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Дисциплина « Компьютерная графика»**

**Лабораторная работа №6**

**по теме:**

**«РЕАЛИЗАЦИЯ И ИССЛЕДОВАНИЕ АЛГОРИТМА ПОСТРОЧНОГО ЗАТРАВОЧНОГО ЗАПОЛНЕНИЯ СПЛОШНЫХ ОБЛАСТЕЙ.»**

**Работу выполнил:**

студент группы ИУ7-43Б

Сукочева А.

**Работу проверил:**

Куров А. В.

2020 г.

**Цель работы:**

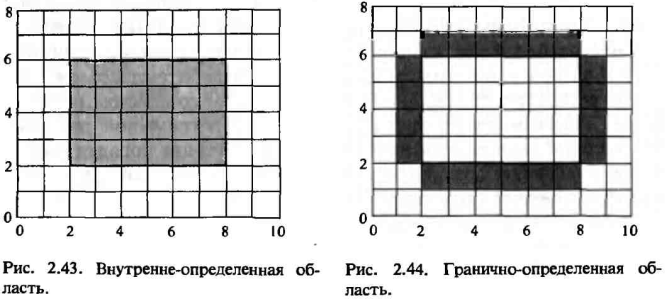
Реализация и исследование алгоритма построчного затравочного заполнения.

**Задание:**

Реализовать заполнение произвольной области, ограниченной замкнутой кривой линией с помощью затравочного алгоритма.

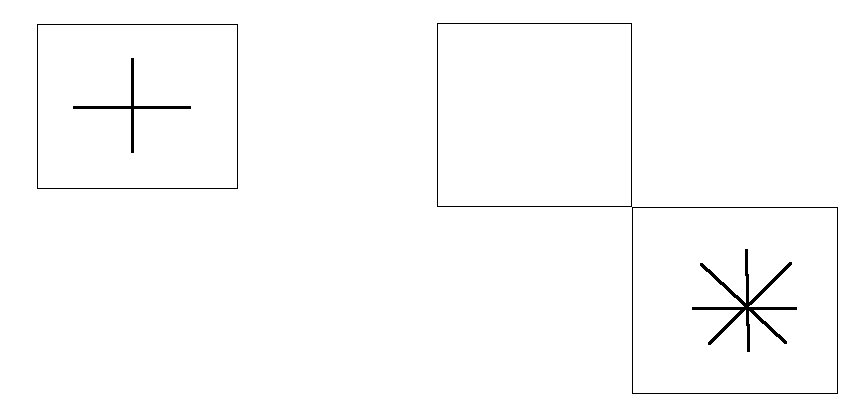
**Теоретический материал:**

Область может задаваться путем границы, т.е. имеем дело с гранично-определенными областями. Область может задаваться путем указания цвета пикселей, расположенных внутри области, т.е. имеем дело с внутренне-определенными областями.



Области заполнения могут быть 4-х связными и 8-ми связными. Если область 4-х связная, то любого пикселя в этой области можно достичь с помощью комбинации возможного движения только в 4х направлениях (←↑→↓). Для 8-ми связных областей появляются 4 движения дополнительно (↙️↖️↗️↘️).

(4-х связная - 4 направления; 8-ми связная - 8 направлений.)



Непрерывный интервал пикселей - группа примыкающих друг к другу пикселей (ограниченная уже заполненными или граничными пикселами)

**Требования:**

1. 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.
2. Пользователь должен иметь возможность задания цвета заполнения.
3. Работа программы должна предусматривать два режима – с задержкой и без задержки.
4. Режим с задержкой должен позволить проследить выполняемую последовательность действий.
5. Обеспечить замер времени выполнения алгоритма (без задержки, с выводом на экран только окончательного результата).

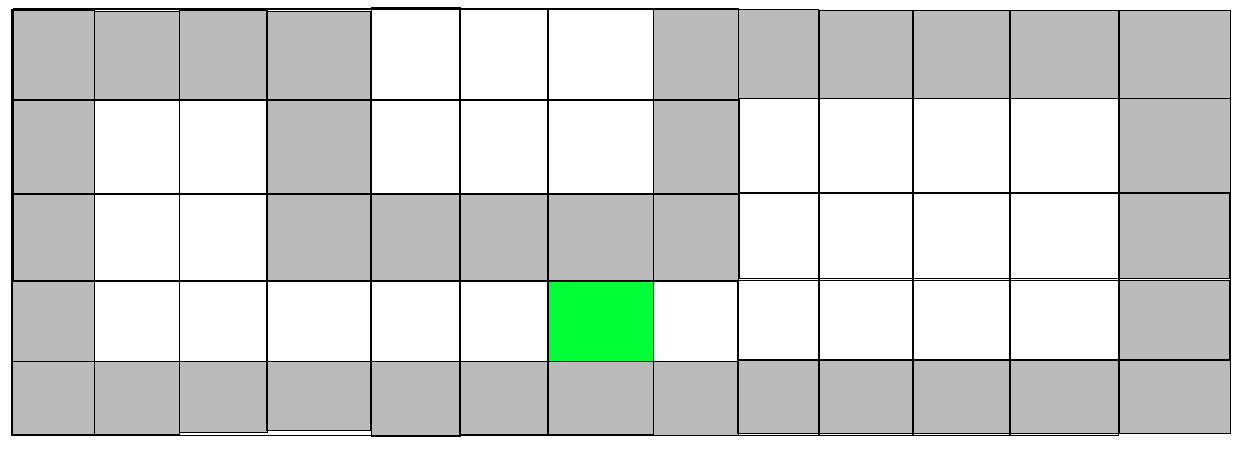
**Алгоритм:**

В алгоритме заполнения с затравкой предполагается, что известен хотя бы один пиксель из внутренней области многоугольника. Этот пиксель называется затравочным пикселем. Алгоритм пытается найти и закрасить все другие пиксели, принадлежащие внутренней области многоугольника. В данном алгоритме целесообразно использовать стек. В стек будут помещаться новые затравочные пиксели, причем для любого непрерывного интервала на сканирующей строке будет храниться только один пиксель. Данный алгоритм применяется к гранично-определенным областям.

Построчный алгоритм заполнения с затравкой:

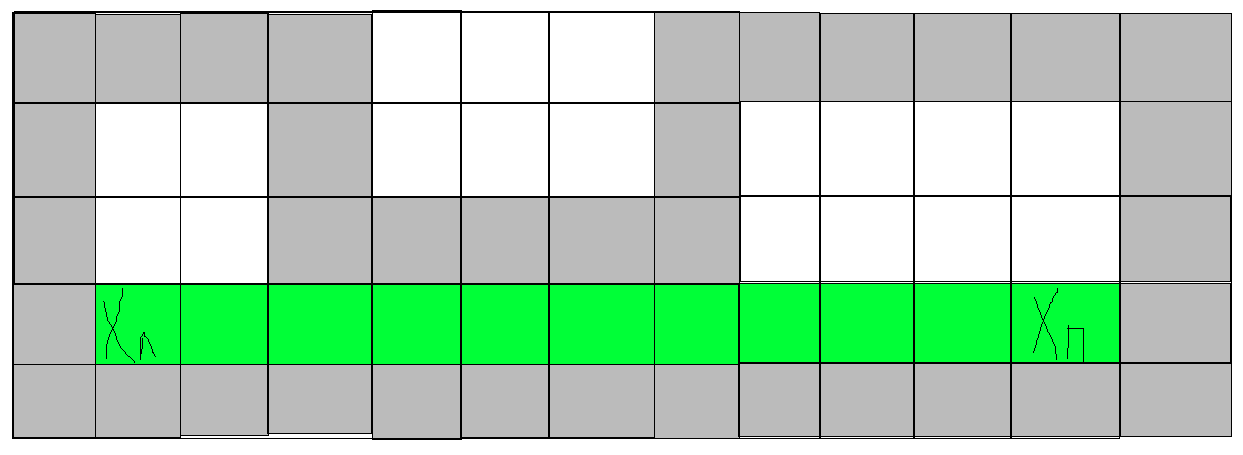
1. Определяется затравочный пиксель, который заносится в стек и пока стек не пуст, пиксель извлекается из стека.
2. Пиксель закрашивается, и закрашиваются все пиксели на сканирующей строке влево и вправо от текущего пикселя (Пока не встретим границу).
3. В переменных Хл и Хп запоминаются координаты крайнего левого и крайнего правого пикселей интервала закрашенной строки. В диапазоне Хл ≤ Х ≤ Хп проверяются строки, расположенные непосредственно над и под текущей строкой. Определяется, есть ли на них еще не закрашенные пиксели. Если есть незакрашенные пиксели, то в указанном диапазоне крайний правый пиксель в каждом интервале отмечается как затравочный и помещается в стек.

Допустим на дана такая область:

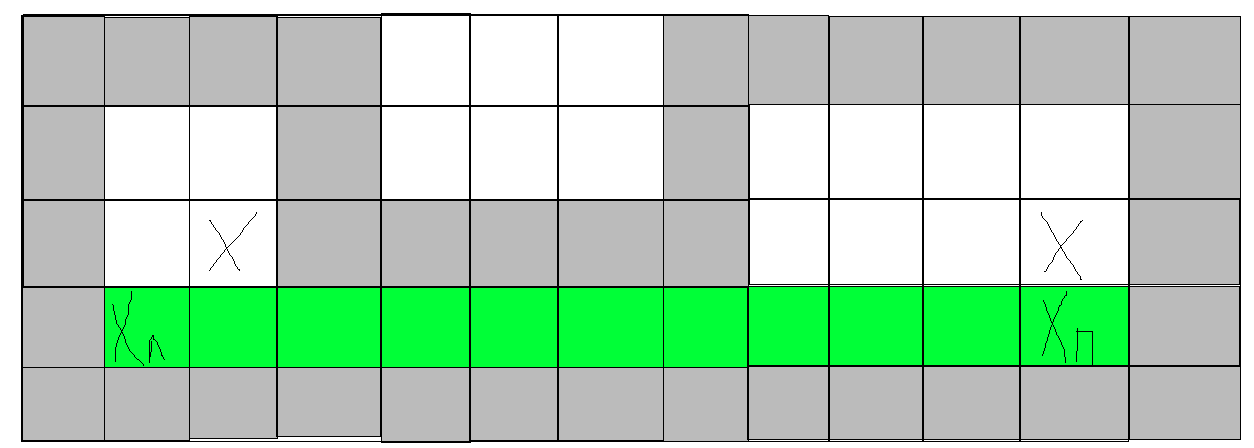


Зеленым цветом показан затравочный пиксель.

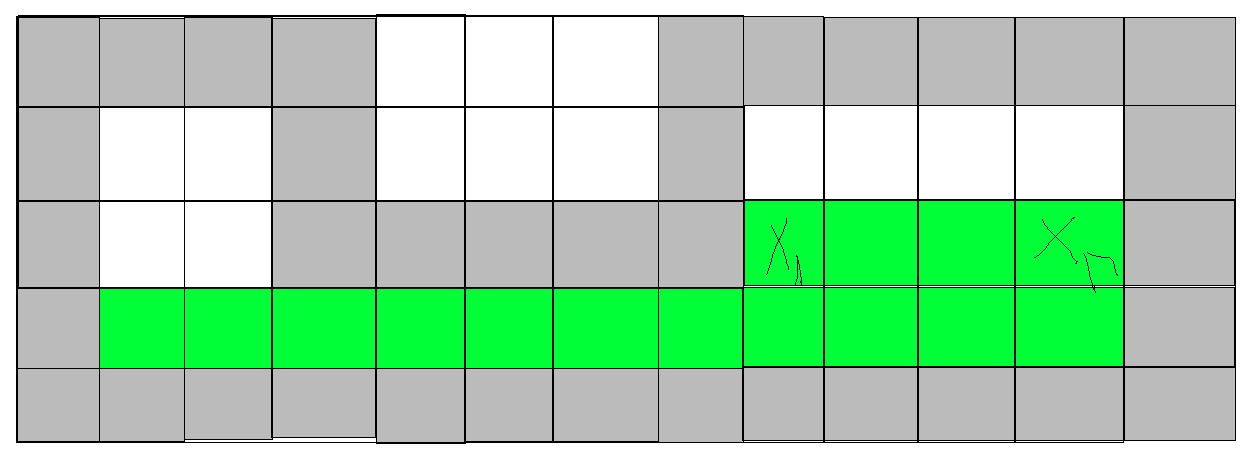
Алгоритм закрашивает все пиксели слева и справа и запоминает Хл и Хп.

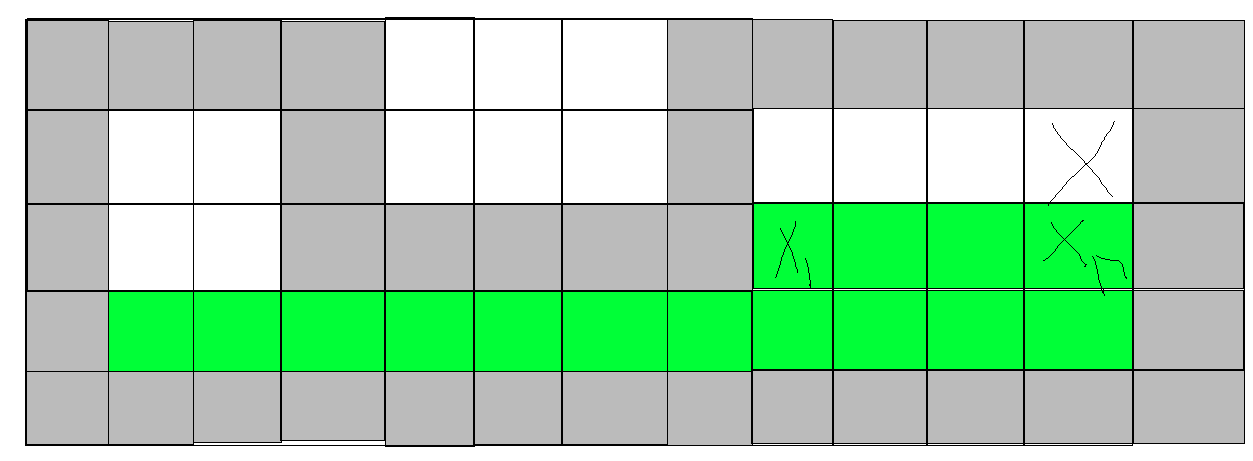


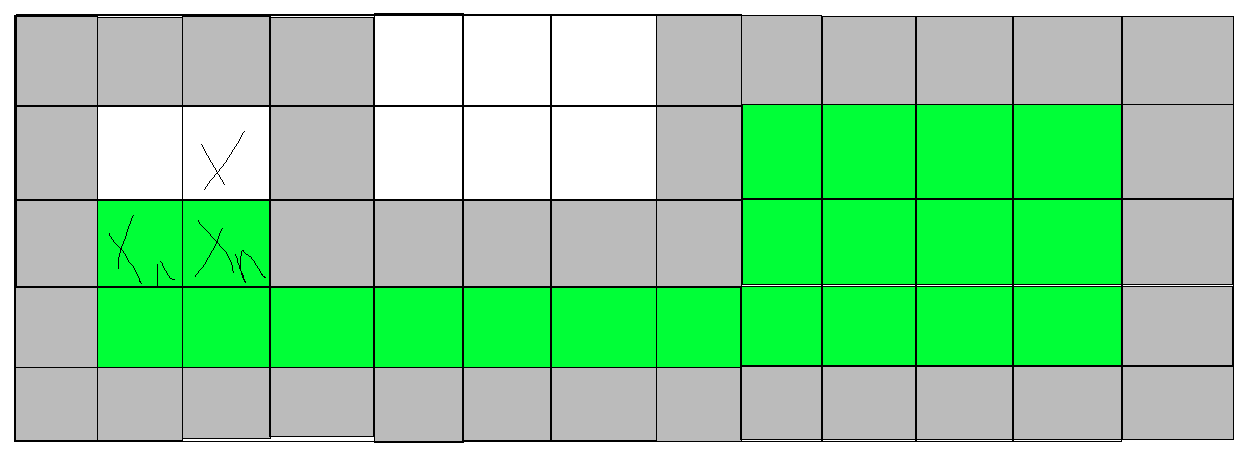
Далее в диапазоне Хл ≤ Х ≤ Хп проверяется строка, расположенная выше и ищутся новые з.п.

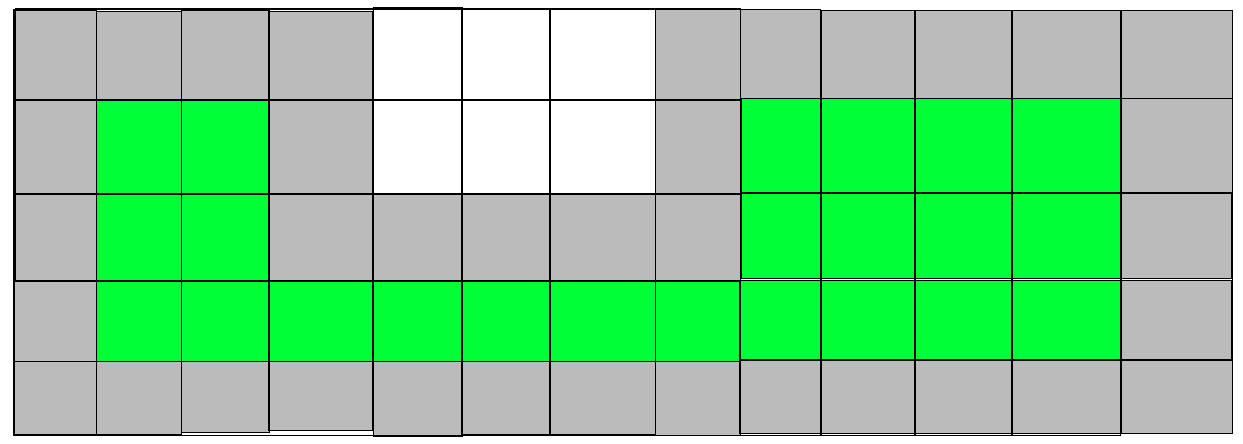


Алгоритм находит 2 новых з.п. и помещает их в стек. На строке ниже алгоритм не находит новых з.п. Далее он достает новый з.п. из стека и повторяет вышеизложенные действия.

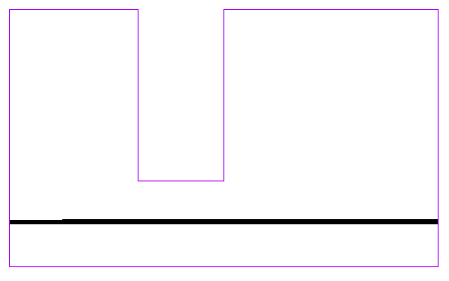
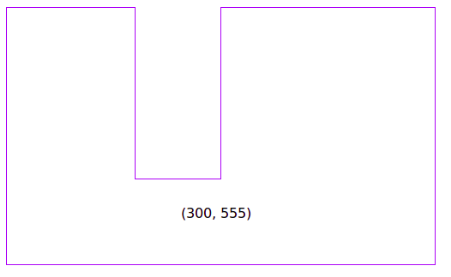


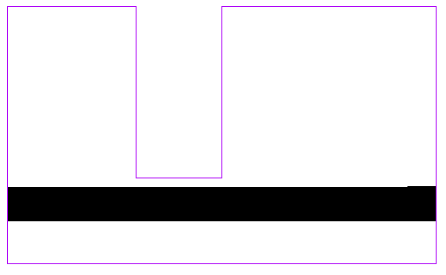


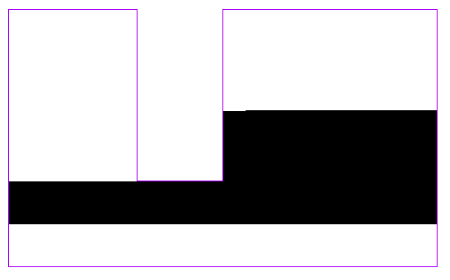




Теперь стек пуст и алгоритм завершает свою работу.









**Программа:**

Я разбила главную задачу на несколько функций.

Функция **fill** создает стек и вызывает остальные функции, которые в свою очередь занимаются отрисовкой и поиском новых затравочных пикселей.

**def fill(canvas\_class, start\_point):**

**# Создаем стек и кладем в него затравочную точку.**

**stack = stack\_class(start\_point)**

**# Пока что стек не пуст**

**while not stack.is\_empty():**

**# Достаем из стека новый затравочный пиксель.**

**x, y = stack.pop()**

**# Отрисовываем его.**

**canvas\_class.draw\_pixel((x, y))**

**# Заполняем все пиксели справа от затравочной точки до того момента,**

**# Пока что не встретим пиксель с цветом границы.**

**# В переменную x\_right запоминаем крайний парвый пиксель.**

**x\_right = fill\_right(canvas\_class, x + 1, y)**

**# Заполняем все пиксели слева от затравочной точки до того момента,**

**# Пока что не встретим пиксель с цветом границы.**

**# В переменную x\_left запоминаем крайний левый пиксель.**

**x\_left = fill\_left(canvas\_class, x - 1, y)**

**# На строке выше в диапазоне от x\_left <= x <= x\_right**

**# Ищем новые затравочные пиксели и помещаем их в стек.**

**find\_pixel(stack, canvas\_class, x\_right, x\_left, y + 1)**

**# На строке ниже в диапазоне от x\_left <= x <= x\_right**

**# Ищем новые затравочные пиксели и помещаем их в стек.**

**find\_pixel(stack, canvas\_class, x\_right, x\_left, y - 1)**

Функции **fill\_right** и **fill\_left** закрашивают пиксели соответственно справа и слева от затравочного пикселя, пока что не встретят границу.

**def fill\_right(canvas\_class, x, y):**

**# Пока что цвет пиксела не равен цвету границы**

**while canvas\_class.img.get(round(x), round(y)) != canvas\_class.color\_line[0]:**

**# Отрисовываем пиксель**

**canvas\_class.draw\_pixel((x, y))**

**# Увеличиваем x (Идем вправо)**

**x += 1**

**# Возвращаем x - 1**

**# Т.к. после цикла пиксель с координатами (x, y)**

**# Будет равен цвету границы, а нам нужно взять**

**# Крайний справа пиксель, т.е. пиксель,**

**# Который находится справа от границы.**

**return x - 1**

**def fill\_left(canvas\_class, x, y):**

**# Пока что цвет пиксела не равен цвету границы**

**while canvas\_class.img.get(round(x), round(y)) != canvas\_class.color\_line[0]:**

**# Отрисовываем пиксель**

**canvas\_class.draw\_pixel((x, y))**

**# Уменьшаем x (Идем влево)**

**x -= 1**

**# Возвращаем x + 1**

**# Т.к. нам нужен крайний слева пиксель.**

**return x + 1**

Функция **find\_pixel** ищет новые затравочные пиксели.

**def find\_pixel(stack, canvas\_class, x\_right, x, y):**

**# Ищем новый затравочный пиксель.**

**while x <= x\_right:**

**# Флаг - признак нахождения нового затравочного пикселя.**

**flag = False**

**# Пока цвет текущего пикселя не равен цвету заполнения и не равен граничному цвету и x <= x\_right**

**while canvas\_class.compare\_color\_line(x, y) and canvas\_class.compare\_color\_fill(x, y) and x <= x\_right:**

**# Нашли затравочный пиксель.**

**if flag == False:**

**flag = True**

**x += 1**

**# Если нашли новый пиксель, то помещаем его в стек.**

**if flag:**

**if x == x\_right and canvas\_class.compare\_color\_line(x, y) and canvas\_class.compare\_color\_fill(x, y):**

**stack.push([x, y])**

**else:**

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

**flag = False**

**# Продолжаем проверку (Если интервал был прерван)**

**x\_temp = x**

**while not canvas\_class.compare\_color\_line(x, y) or not canvas\_class.compare\_color\_fill(x, y) and x < x\_right:**

**x += 1**

**if x == x\_temp:**

**x += 1**

**Результат выполнения:**

Интерфейс немного обновлен (По сравнению с предыдущей л.р.):

Теперь при наведении мыши на холст отображаются координаты x и y точки, на которую указывает мышка в данный момент (Это нужно для более простого определения координат затравочной точки).

