## МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Вятский государственный университет» Факультет автоматики и вычислительной техники Кафедра электронных вычислительных машин

### РЕАЛИЗАЦИЯ АЛГОРИТМОВ РАСТРОВОЙ ГРАФИКИ ДЛЯ ЗАПОЛНЕНИЯ СПОЛШНЫХ ОБЛАСТЕЙ

Отчет по лабораторной работе №8, 9 дисциплины «Компьютерная графика»

Выполнил студент группы ИВТ-21	/Рзаев А.Э./
Проверил старший преподаватель	_/Вожегов Д.В./

#### 1 Постановка задачи

Написать программу, реализующую алгоритм заливки многоугольника любой формы.

Проверить правильность работы программы, нарисовав с помощью функций модуля GRAPH невыпуклый многоугольник (с дырами внутри), закрасьте его заданным цветом, указав координаты принадлежащей многоугольнику точки.

Написать и отладить программу, реализующую алгоритм построчного заполнения выпуклого многоугольника, заданного координатами вершин и цветом границы (можно воспользоваться алгоритмом заполнения треугольника, прочитанным на лекции).

#### 2 Краткие теоретические сведения

Алгоритмы заполнения с затравкой.

Данные алгоритмы ищут и закрашивают связную компоненту области, содержащую затравочный пиксел. Под связностью понимают четырех или восьми связность в зависимости от постановки задачи. Каждый из пикселов четырехсвязной области достижим из любого другого пиксела этой области с помощью последовательности перемещений на пиксел из четырехэлементного набора горизонтальных и вертикальных направлений, восьмисвязной - из восьмиэлементного (добавляются еще и диагональные направления). Алгоритмы для восьмисвязных областей применимы к четырехсвязным, но не наоборот.

Процесс заполнения областей иногда называют заливкой, так как можно представить, что в точке затравки находится источник, заливающий всю область определенным цветом. Область может быть определена внутренне (все внутренние пикселы - одного цвета, внешние - другого) и гранично (все пикселы на границе или вне области - одного цвета не такого, как внутри).

Схема данного алгоритма приведена в приложении А, код программы, реализующей данную программу приведён в приложении Б.

#### Построчный алгоритм заливки.

Применяется идея построчного сканирования и используется стек для хранения одного затравочного пиксела для любого непрерывного интервала на сканирующей строке. Непрерывный интервал - это группа примыкающих друг к другу пикселов, ограниченная уже закрашенными или граничными пикселами. Если непрерывный интервал пикселов принадлежит внутренней части области, пикселы на и под этим интервалом либо граничные, либо тоже находятся внутри области и могут служить затравочными для своих строк.

Схема алгоритма приведена в приложении В, листинг кода программы приведён в приложении Г.

Экранные формы обеих программ приведены в приложении Д.

### 3 Вывод

В данной лабораторной работе были изучены 2 алгоритма заливки: алгоритм с затравкой и построчный алгоритма заливки. Были рассмотрены как плюсы, так и минусы обоих алгоритмов.

Алгоритм заливки с затравкой:

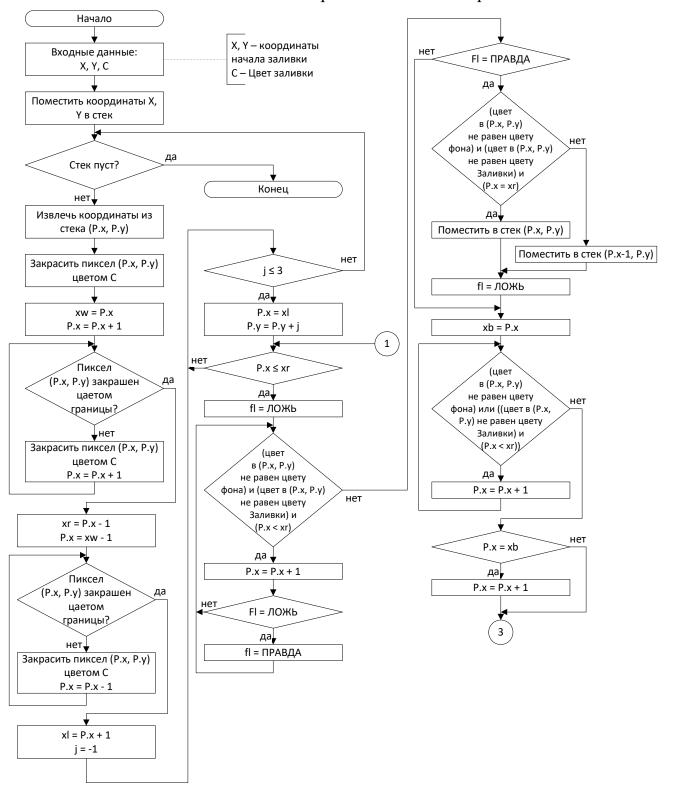
- 1. Способен заливать произвольные области (даже с "дырами");
- 2. Сравнительно медленный (из-за сложного анализатора).

Алгоритм с построчной заливкой:

- 1. Быстрый алгоритм (по сравнению с алгоритмом с затравкой);
- 2. Для заливки сложных полигонов (невыпуклых и с "дырами") требует добавления анализатора.

# Приложение А (обязательное)

### Схема алгоритма заливки с затравкой

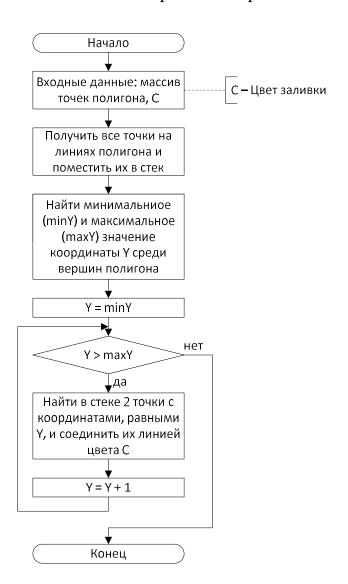


# Приложение Б (обязательное)

#### Листинг программы для затравочного алгоритма

```
void FillTriangle(int x1, int y1, int x2, int y2, int x3, int y3) {
      if (y2 < y1) {
           std::swap(x1, x2);
           std::swap(y1, y2);
      if (y3 < y2) {
           std::swap(x2, x3);
           std::swap(y2, y3);
      }
      for (int y = y1; y < y2; ++y) {
            int xl = std::round(x1 - 1.0 * (y1 - y) * (x1 - x2) / (y1 - y2)),
                 xr = std::round(x1 - 1.0 * (y1 - y) * (x1 - x3) / (y1 - y3));
            DrawLineBresenhem8(x1, y, xr, y);
      for (int y = y2; y < y3; ++y) {
           int x1 = std::round(x3 - 1.0 * (y3 - y) * (x3 - x2) / (y3 - y2)),
                xr = std::round(x3 - 1.0 * (y3 - y) * (x3 - x1) / (y3 - y1));
           DrawLineBresenhem8(x1, y, xr, y);
}
void FillPolygon(const std::vector<pii>& points) {
      int x0 = points.front().first, y0 = points.front().second;
      for (Uint32 i = 1; i < points.size() - 1; ++i) {
           FillTriangle(x0, y0, points[i].first, points[i].second, points[i +
1].first, points[i + 1].second);
}
```

# Приложение В (обязательное) Схема алгоритма построчной заливки



# Приложение Г (обязательное)

#### Листинг программы для построчного алгоритма

```
void FillPolygon(int x0, int y0) {
      std::stack<pii> stack;
      stack.emplace(x0, y0);
      while (!stack.empty()) {
            pii p = stack.top(); stack.pop();
            int x = p.first, y = p.second;
            pixels[linear(x, y)] = FG COLOR;
            int xw = x; x += 1;
            while (check(x, y) && pixels[linear(x, y)] != FG_COLOR) {
                  pixels[linear(x, y)] = FG_COLOR;
                  x += 1;
            int xr = x - 1;
            x = xw; x -= 1;
            while (check(x, y) \&\& pixels[linear(x, y)] != FG_COLOR) {
                  pixels[linear(x, y)] = FG COLOR;
                  x -= 1;
            int xl = x + 1;
            for (int j : { y - 1, y + 1 }) {
                  x = x1; y = j;
                  if (!check(x, y)) {
                        continue;
                  }
                  while (x \le xr) {
                        bool fl = false;
                        while (x < xr \&\& pixels[linear(x, y)] != FG COLOR) {
                              x += 1;
                              fl = true;
                        if (fl) {
                              if (x == xr \&\& pixels[linear(x, y)] != FG COLOR) {
                                    stack.emplace(x, y);
                              }
                              else {
                                    stack.emplace(x - 1, y);
                               }
                              fl = false;
                        int xb = x;
                        while (x < xr \&\& pixels[linear(x, y)] == FG COLOR) x += 1;
                        if (x == xb) x += 1;
                  }
            }
      }
}
```

Приложение Д Экранные формы программ

