МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФГБОУ ВО АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Институт цифровых технологий, электроники и физики

Кафедра вычислительной техники и электроники (ВТиЭ)

Лабораторная работа № 1

**Алгоритмы растрирования**

Выполнил студент 595 гр.

\_\_\_\_\_\_\_\_\_\_\_\_\_ Лаптев А.В.

Проверил:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Калачев А.В.

Лабораторная работа защищена

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г.

Барнаул 2023

**Задание 1.**

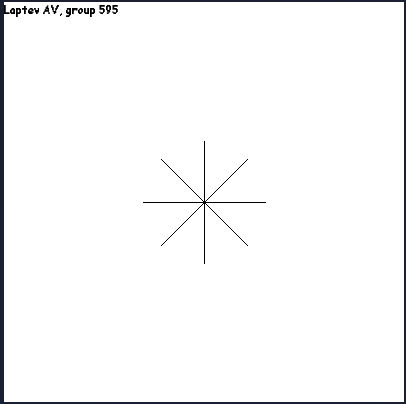
Реализовать в виде отдельной функции алгоритм Брезенхема для отображения отрезка с заданными координатами концов. При реализации алгоритма для управления состоянием пикселов разрешается использовать только функцию SetPixel. В качестве теста корректности разработанной функции программа должна быть способна отображать:

* Звездочку с заданным количеством лучей (N);
* Набор из N отрезков со случайными координатами концов;
* Отрезок с заданными координатами.

Режим работы и значения параметров должны передаваться в программу в командной строке.

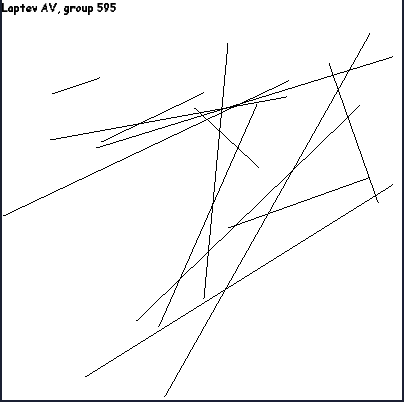
# Тест работоспособности программы

Звездочка с заданным количеством лучей (N):



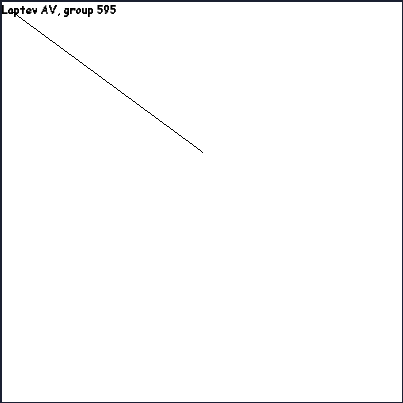
*Рис.1. Звезда с 8 лучами.*

Набор из N отрезков со случайными координатами концов:



*Рис.3. 13 отрезков со случайными координатами.*

Отрезок с заданными координатами:



*Рис.5. Отрезок с заданными координатами.*

**Задание 2**

Реализуйте в простейшей windows-программе алгоритм заполнения многоугольников YX в виде функции FillPolygon(), воспринимающей в качестве параметров массив вершин многоугольника, цвет внутренности и цвет границы многоугольника. Программа должна работать в двух режимах:

* визуализации двух многоугольников
  + выпуклый с 7 вершинами;
  + невыпуклый с 10 вершинами;
* последовательной визуализации многоугольников, описание которых считывается программой из внешнего текстового файла, указываемого в командной строке.

Файл с описаниями многоугольников имеет следующий формат.

<N>

<x1> <y1>

<x2> <y2>

…

<xN> <yN>

<K>

<x1> <y1>

<x2> <y2>

…

<xK> <yK>

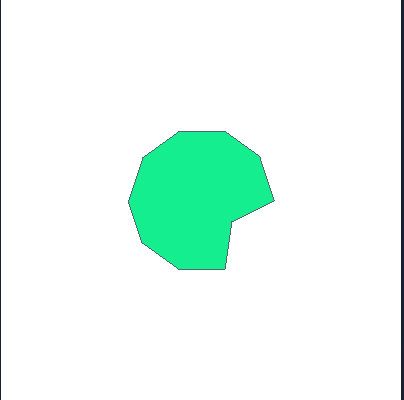
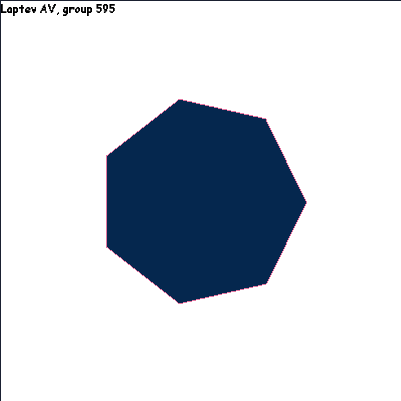
…

Здесь N, K — количество вершин в соответствующих многоугольниках, xi ,yi

— координаты вершин многоугольников. В файле может содержаться произвольное количество описаний многоугольников. Количество вершин в многоугольниках не может превосходить 500.

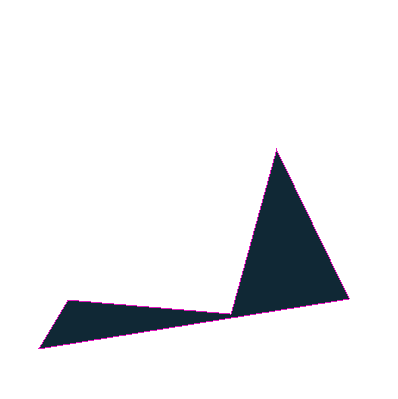
# Тест работоспособности программы

На рис.6 представлен результат отрисовки двух многоугольников. Цвета и координаты задаются в программе.



*Рис.6. Визуализация двух многоугольников.*

Если прописать путь к файлу с точками, то будет отрисован многоугольник, заданный в файле со случайными цветами заполнения.



*Рис.7. Визуализация многоугольников, заданных в файле.*

**Вывод**

В ходе выполнения данной лабораторной работы мной были получены навыки написания алгоритмов растрирования.

**Приложение А**

import os

from random import randint

from PIL import Image, ImageDraw, ImageFont

import math

def starBresenham(N, x0, y0):

    # Рисование звезды из линий с использованием алгоритма Брезенхема

    x1 = randint(x0, width)

    y1 = y0

    oneSegment(x0, y0, x1, y1, colorLine=(0, 0, 0))

    r = math.trunc(math.sqrt((x1 - x0) \*\* 2 + (y1 - y0) \*\* 2))

    for i in range(N - 1):

        x1 = x0 + round(r \* math.cos((360.0 / N) \*

                        (i + 1.0) \* math.pi / 180.0))

        y1 = y0 + round(r \* math.sin((360.0 / N) \*

                        (i + 1.0) \* math.pi / 180.0))

        oneSegment(x0, y0, x1, y1, colorLine=(0, 0, 0))

def randomCoordsBresenham(N):

    # Рисование случайных прямых с использованием алгоритма Брезенхема

    for i in range(N):

        x0 = randint(0, width)

        y0 = randint(0, height)

        x1 = randint(0, width)

        y1 = randint(0, height)

        oneSegment(x0, y0, x1, y1, colorLine=(0, 0, 0))

def oneSegment(x0, y0, x1, y1, colorLine):

    # Рисование единичной прямой по алгоритму Брезенхема

    listPoints = []

    deltaX = abs(x1 - x0)

    deltaY = abs(y1 - y0)

    error = 0

    if (x1 - x0 >= 0 and y1 - y0 >= 0) or (x1 - x0 < 0 and y1 - y0 < 0):

        dx = 1

        dy = 1

    else:

        dx = 1

        dy = -1

    if deltaX == 0 and deltaY == 0:

        draw.point((x0, y0), colorLine)

        listPoints.append([x0, y0])

        return

    if deltaX >= deltaY:

        if x1 - x0 >= 0 and (y1 - y0 >= 0 or y1 - y0 < 0):

            # Вправо вниз, меньше 45 и вправо вверх меньше 45

            y = y0

            for x in range(x0, x1 + 1):

                draw.point((x, y), colorLine)

                listPoints.append([x, y])

                error += deltaY

                if error >= deltaX:

                    y += dy

                    error -= deltaX

        else:

            # Влево вниз, меньше 45 и влево вверх меньше 45

            y = y1

            for x in range(x1, x0 + 1):

                draw.point((x, y), colorLine)

                listPoints.append([x, y])

                error += deltaY

                if error >= deltaX:

                    y += dy

                    error -= deltaX

    else:

        dx, dy = dy, dx

        if (x1 - x0 >= 0 or x1 - x0 < 0) and y1 - y0 >= 0:

            # Вправо вниз, больше 45 и влево вниз, больше 45

            x = x0

            for y in range(y0, y1 + 1):

                draw.point((x, y), colorLine)

                listPoints.append([x, y])

                error += deltaX

                if error >= deltaY:

                    x += dx

                    error -= deltaY

        else:

            # Вправо вверх, больше 45 и влево вверх, больше 45

            x = x1

            for y in range(y1, y0 + 1):

                draw.point((x, y), colorLine)

                listPoints.append([x, y])

                error += deltaX

                if error >= deltaY:

                    x += dx

                    error -= deltaY

    return listPoints

def selectBresenham():

    # Выбор режима работы по построению прямых по методу Брезенхема

    print("Введите номер теста, который желаете выполнить:")

    print("1. Звезда с количеством лучей N")

    print("2. Набор из N отрезков со случайными координатами")

    print("3. Отрезок с заданными координатами")

    mode = int(input("Ваш вариант: "))

    while mode < 1 or mode > 3:

        print("\nНеправильный номер теста!")

        print("Введите номер теста, который желаете выполнить:")

        print("1. Звезда с количеством лучей N")

        print("2. Набор из N отрезков со случайными координатами")

        print("3. Отрезок с заданными координатами")

        mode = int(input("Ваш вариант: "))

    if mode == 1:

        N = int(input("Введите количество лучей: "))

        while N < 1:

            print("\nНеверное количество лучей!")

            N = int(input("Введите количество лучей: "))

        x0 = int(input("Введите координату центра звезды по x: "))

        y0 = int(input("Введите координату центра звезды по y: "))

        while x0 < 0 or y0 < 0 or x0 > width or y0 > height:

            print("\nНеверные координаты центра! Введите новые координаты")

            x0 = int(input("Введите координату центра звезды по x: "))

            y0 = int(input("Введите координату центра звезды по y: "))

        starBresenham(N, x0, y0)

        graphName = "star"

    elif mode == 2:

        N = int(input("Введите количество отрезков: "))

        while N < 1:

            print("\nНеверное количество отрезков!")

            N = int(input("Введите количество отрезков: "))

        randomCoordsBresenham(N)

        graphName = "randomSegment"

    elif mode == 3:

        x0 = int(input("Введите координату начала отрезка по x: "))

        y0 = int(input("Введите координату начала отрезка по y: "))

        x1 = int(input("Введите координату конца отрезка по x: "))

        y1 = int(input("Введите координату конца отрезка по y: "))

        while x0 < 0 or x0 > width or y0 < 0 or y0 > height or x1 < 0 or x1 > width or y1 < 0 or y1 > height:

            print("\nНеверные координаты начала или конца отрезка")

            x0 = int(input("Введите координату начала отрезка по x: "))

            y0 = int(input("Введите координату начала отрезка по y: "))

            x1 = int(input("Введите координату конца отрезка по x: "))

            y1 = int(input("Введите координату конца отрезка по y: "))

        oneSegment(x0, y0, x1, y1, colorLine=(0, 0, 0))

        graphName = "segment"

        image.save("graph\_" + graphName + ".png", "PNG")

    image.show()

def generatePolygon(N):

    # Генерируем выпуклый 7-угольник и невыпуклый 10-угольник

    centerX = round(width / 2)

    centerY = round(height / 2)

    polygon = []

    x1 = randint(centerX, width)

    y1 = round(centerY)

    r = math.trunc(math.sqrt((x1 - centerX) \*\* 2 + (y1 - centerY) \*\* 2))

    polygon.append([x1, y1])

    if N == 7:

        for i in range(N - 1):

            x1 = centerX + round(r \* math.cos((360.0 / N) \*

                                              (i + 1.0) \* math.pi / 180.0))

            y1 = centerY + round(r \* math.sin((360.0 / N) \*

                                              (i + 1.0) \* math.pi / 180.0))

            polygon.append([x1, y1])

    elif N == 10:

        i = 0

        x1 = centerX + round(0.5 \* r \* math.cos((360.0 / N) \*

                                                (i + 1.0) \* math.pi / 180.0))

        y1 = centerY + round(0.5 \* r \* math.sin((360.0 / N) \*

                                                (i + 1.0) \* math.pi / 180.0))

        polygon.append([x1, y1])

        for i in range(1, N - 1):

            x1 = centerX + round(r \* math.cos((360.0 / N) \*

                                              (i + 1.0) \* math.pi / 180.0))

            y1 = centerY + round(r \* math.sin((360.0 / N) \*

                                              (i + 1.0) \* math.pi / 180.0))

            polygon.append([x1, y1])

    return polygon

def fillTwoPolygons():

    # Закраска двух многоугольников

    N = 7

    fillPolygon(generatePolygon(N), colorSolid=(randint(0, 255), randint(0, 255), randint(0, 255)),

                colorBorder=(randint(0, 255), randint(0, 255), randint(0, 255)))

    graphName = "7-polygon"

    image.save("graph\_" + graphName + ".png", "PNG")

    image.show()

    draw.rectangle([0, 0, width, height], fill=(

        255, 255, 255))  # Очищаем рисунок

    N = 10

    fillPolygon(generatePolygon(N), colorSolid=(randint(0, 255), randint(0, 255), randint(0, 255)),

                colorBorder=(randint(0, 255), randint(0, 255), randint(0, 255)))

    graphName = "10-polygon"

    image.save("graph\_" + graphName + ".png", "PNG")

    image.show()

def fillPolygons(inputFile):

    # Последовательный вывод многоугольников

    linesFile = inputFile.readlines()

    i = 0

    while i < len(linesFile):

        coords = []

        draw.rectangle([0, 0, width, height], fill=(255, 255, 255))

        countTopsPolygons = int(linesFile[i].split('\n')[0])

        for j in range(i + 1, countTopsPolygons + i + 1):

            strCoords = linesFile[j].split('\n')[0]

            coords.append([int(strCoords.split()[0]),

                          int(strCoords.split()[1])])

        i += countTopsPolygons + 1

        fillPolygon(coords, colorSolid=(randint(0, 255), randint(0, 255), randint(0, 255)),

                    colorBorder=(randint(0, 255), randint(0, 255), randint(0, 255)))

        image.show()

def fillPolygon(topsPolygon, colorSolid, colorBorder):

    # Закрашивание многоугольника

    sortTops = []

    bordersListY = []

    for i in range(len(topsPolygon)):

        # Построение границ многоугольника по координатам вершин

        if i != len(topsPolygon) - 1:

            x0 = topsPolygon[i][0]

            y0 = topsPolygon[i][1]

            x1 = topsPolygon[i + 1][0]

            y1 = topsPolygon[i + 1][1]

        else:

            x0 = topsPolygon[i][0]

            y0 = topsPolygon[i][1]

            x1 = topsPolygon[0][0]

            y1 = topsPolygon[0][1]

        sortTops.append([x0, y0, x1, y1])

        for j in range(len(oneSegment(x0, y0, x1, y1, colorBorder))):

            # Добавляем в массив все координаты ребер многоугольника

            bordersListY.append([oneSegment(

                x0, y0, x1, y1, colorBorder)[j], i])

    bordersListX = sorted(bordersListY)  # Соритруем массив по координате x

    # Сортируем массив по координате y

    bordersListY = sorted(bordersListX, key=lambda x: x[0][1])

    cnt = 0

    flag = 0

    diffTopsYBefore = 0

    diffTopsYAfter = 0

    differenceI = 0

    for i in range(len(bordersListY) - 1):

        if bordersListY[i][0][1] == bordersListY[i + 1][0][1]:

            difference = abs(

                bordersListY[i][0][0] - bordersListY[i + 1][0][0])

            if difference > 1:

                # Это обычный отрезок

                cnt += 1

            elif difference == 1:

                if bordersListY[i][1] != bordersListY[i + 1][1] and flag == 0:

                    cnt += 1

            else:

                # Это вершина

                for j in range(len(topsPolygon)):

                    # Проходим по списку вершин в поисках нужной

                    if (bordersListY[i][0][0] == topsPolygon[j][0]) and (bordersListY[i][0][1] == topsPolygon[j][1]):

                        """ Как только нашли вершину, лпределяем ее местоположение относительно соседних

                            Если эта вершина выше (ниже) обеих соседних, то считаем ее отрезком, иначе границей """

                        flag = 1

                        differenceI = bordersListY[i][1] - \

                            bordersListY[i + 1][1]

                        if differenceI > 0:

                            flag1 = 1

                        else:

                            flag1 = -1

                        if j == 0:

                            diffTopsYBefore = bordersListY[i][0][1] - \

                                topsPolygon[len(topsPolygon) - 1][1]

                            diffTopsYAfter = bordersListY[i][0][1] - \

                                topsPolygon[j + 1][1]

                        elif j == len(topsPolygon) - 1:

                            diffTopsYBefore = bordersListY[i][0][1] - \

                                topsPolygon[j - 1][1]

                            diffTopsYAfter = bordersListY[i][0][1] - \

                                topsPolygon[0][1]

                        else:

                            diffTopsYBefore = bordersListY[i][0][1] - \

                                topsPolygon[j - 1][1]

                            diffTopsYAfter = bordersListY[i][0][1] - \

                                topsPolygon[j + 1][1]

                        if ((diffTopsYBefore > 0) and (diffTopsYAfter >= 0)) or ((diffTopsYBefore < 0) and (diffTopsYAfter < 0)):

                            cnt += 1

                        break

                if flag == 0:

                    cnt += 1

        else:

            cnt = 0

            flag = 0

        if cnt % 2 == 1:

            oneSegment(bordersListY[i][0][0], bordersListY[i][0][1],

                       bordersListY[i + 1][0][0], bordersListY[i + 1][0][1], colorSolid)

    for i in range(len(sortTops)):

        # Отрисовываем границу уже после заливки, чтобы она была видна на изображении

        oneSegment(sortTops[i][0], sortTops[i][1],

                   sortTops[i][2], sortTops[i][3], colorBorder)

def selectFillPolygon():

    # Выбор режима для закраски многоугольников по методу YX

    print("Выберите режим работы программы:")

    print("1. Визуализация двух многоугольников")

    print("2. Последовательная визуализация многоугольников")

    selectMode = int(input("Ваш вариант: "))

    while selectMode < 1 or selectMode > 2:

        print("\nНеверный режим! Введите режим еще раз")

        selectMode = int(input("Ваш вариант: "))

    if selectMode == 1:

        fillTwoPolygons()

    else:

        charactersPolygons = input(

            "Введите имя файла с характеристиками многоугольников: ")

        try:

            filename = os.path.abspath('Polygons/' + charactersPolygons)

            with open(filename) as inputFile:

                fillPolygons(inputFile)

        except FileNotFoundError:

            print("\nНет такого файла! Введите другое имя")

            charactersPolygons = input(

                "Введите имя файла с характеристиками многоугольников: ")

            filename = os.path.abspath('Polygons/' + charactersPolygons)

            with open(filename) as inputFile:

                fillPolygons(inputFile)

if \_\_name\_\_ == '\_\_main\_\_':

    width = 400

    height = 400

    image = Image.new("RGB", (width, height), (255, 255, 255))

    draw = ImageDraw.Draw(image)

    fnt = ImageFont.truetype("pixel.ttf", 14)

    draw.text((0, 0), "Laptev AV, group 595", fill=(0, 0, 0), font=fnt)

    print("Выберите задание для выполнения:")

    print("1. Алгоритм Брезенхема")

    print("2. Алгоритм YX")

    yourVariant = int(input("Ваш вариант: "))

    while yourVariant < 1 or yourVariant > 2:

        print("\nВы ввели некорректный вариант! Введите новый вариант")

        yourVariant = int(input("Ваш вариант: "))

    if yourVariant == 1:

        selectBresenham()

    else:

        selectFillPolygon()