|  |  |
| --- | --- |
| 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 № 3**

|  |  |
| --- | --- |
| **Тема** Реализация и исследование алгоритмов построения отрезков  **Студент** Воякин А. Я.  **Группа ИУ7-44Б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |  |

Москва.

2020 г.

**Цель работы:** реализация алгоритмов построения отрезков по методу цифрового дифференциального анализатора (ЦДА), алгоритмов Брезенхема (действительного, целочисленного и с устранением ступенчатости), Ву и исследование их характеристик и сравнение полученных результатов.

**Техническое задание:**

1. Рисование отдельных отрезков и сравнение их визуальных характеристик:

1\* Алгоритм цифрового дифференциального анализатора;

2\* Алгоритмы Брезенхема;

3\* Алгоритм Ву;

4\* Библиотечный алгоритм;

1. Исследование визуальных характеристик для отрезка, расположенного во всем спектре изменения углов;
2. Исследование временных характеристик (результат оформить в виде гистограммы);
3. Исследование ступенчатости;

**Теоретический материал:**

Общие требования:

1. Отрезок должен выглядеть как отрезок прямой, начинаться и заканчиваться в заданных точках

2. Интенсивность (яркость) вдоль отрезка должна быть постоянной. Отрезки, имеющие разные углы наклона, должны быть одной интенсивности. Восприятие человека зависит не только от интенсивности свечения объекта, но и от расстояния между светящимися объектами //чтобы удовлетворить этому требованию, надо высвечивать точки с переменной интенсивностью от расстояния – потребует дополнительных вычислений, без особой нужды не используется

3. Алгоритмы (особенно нижнего уровня) должны работать быстро

Все алгоритмы имеют пошаговый характер – на очередном шаге высвечиваем пиксель, и производим вычисления, используемые в следующем шаге.

**Алгоритм цифрового дифференциального анализатора:**

def cda(self, x\_start, y\_start, x\_end, y\_end, draw=True, steps=False):

# Проверка вырожденности орезка в точку.

if x\_start == x\_end and y\_start == y\_end and draw:

self.draw\_pixel(round(x\_start), round(y\_start))

return

# Вычисление длины отрезка.

dx = x\_end - x\_start

dy = y\_end - y\_start

if fabs(dx) > fabs(dy):

l = fabs(dx)

else:

l = fabs(dy)

dx /= l

dy /= l

# Задание координатам текущей точки начальных значений.

x = x\_start

y = y\_start

i = 1

step = 1

max\_step = 1

while i <= l + 1:

if draw:

self.draw\_pixel(round(x), round(y))

if steps and i <= l:

if (round(x + dx) == round(x) and round(y + dy) != round(y)) or (round(x + dx) != round(x) and round(y + dy) == round(y)):

step += 1

else:

if max\_step < step:

max\_step = step

step = 1

x += dx

y += dy

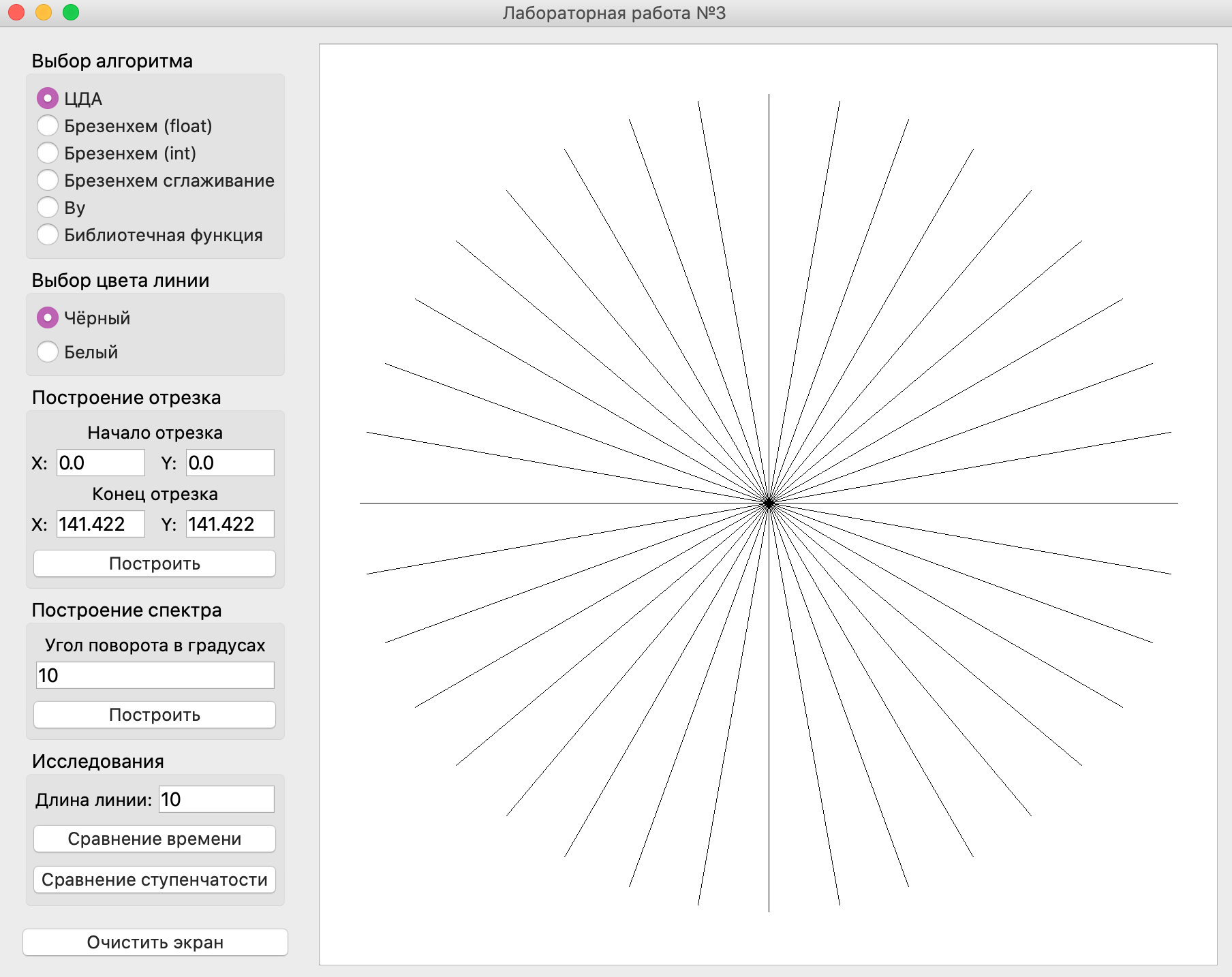
i += 1

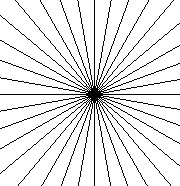
if steps:

return max(max\_step, step)

Недостаток алгоритма: Округление в цикле. Из-за этого он может медленно работать.

Демонстрацию работы алгоритмов буду показывать на спектре с углом в 10 градусов.

****

****

**Алгоритм Брезенхема с действ. коэф-ми:**

def bres\_float(self, x\_start, y\_start, x\_end, y\_end, draw=True, steps=False):

# Проверка вырожденности орезка в точку.

if x\_start == x\_end and y\_start == y\_end:

if draw:

self.draw\_pixel(round(x\_start), round(y\_start))

return

# Вычисление приращений.

dx = x\_end - x\_start

dy = y\_end - y\_start

# Вычисление шага изменения каждой координаты пиксела.

sx = sign(dx)

sy = sign(dy)

# Вычисление модулей приращения координат.

dx = fabs(dx)

dy = fabs(dy)

# Анализ вычисленного значения m и обмен местами dX и dY при m > 1.

if dy >= dx:

dx, dy = dy, dx

fl = 1 # флаг, определяющий факт обмена местами координат.

else:

fl = 0

# Вычисление модуля тангенса угла наклона отрезка.

m = dy / dx

# Инициализация начального значения ошибки.

f = m - 0.5

# Инициализация начальных значений координат текущего пиксела.

x = round(x\_start)

y = round(y\_start)

# Цикл от i=1 до i=dX+1 с шагом 1.

i = 1

step = 1

max\_step = 1

x\_buff = x

y\_buff = y

while i <= dx + 1:

# Высвечивание точки с координатами (X,Y).

if draw:

self.draw\_pixel(x, y)

# Вычисление координат и ошибки для следующего пиксела.

if f >= 0:

if fl == 1:

x += sx

else:

y += sy

f -= 1

if f <= 0:

if fl == 1:

y += sy

else:

x += sx

f += m

i += 1

if steps and i <= dx:

if (x\_buff == x and y\_buff != y) or (x\_buff != x and y\_buff == y):

step += 1

else:

if max\_step < step:

max\_step = step

step = 1

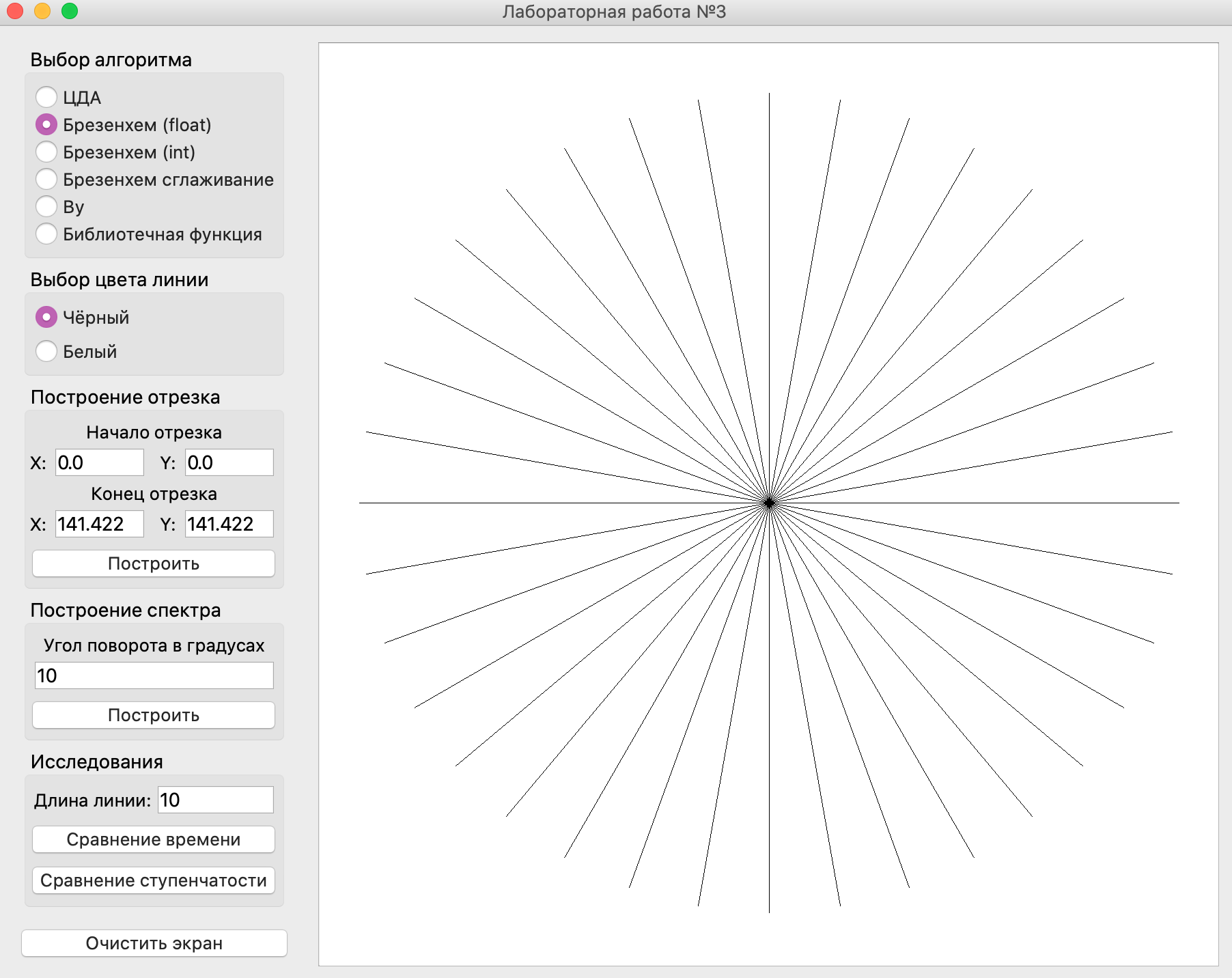
x\_buff = x

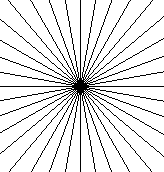
y\_buff = y

if steps:

return max(max\_step, step)

Недостатком рассмотренного варианта алгоритма Брезенхема является использование вычислений с плавающей точкой и использование операции деления при определении тангенса угла наклона отрезка. Сократить время работы алгоритма можно при использовании целочисленной арифметики и отказа от операции деления.





**Алгоритм Брезенхема с целыми коэф-ми:**

def bres\_int(self, x\_start, y\_start, x\_end, y\_end, draw=True, steps=False):

# Проверка вырожденности орезка в точку.

if x\_start == x\_end and y\_start == y\_end:

if draw:

self.draw\_pixel(round(x\_start), round(y\_start))

return

# Вычисление приращений.

dx = x\_end - x\_start

dy = y\_end - y\_start

# Вычисление шага изменения каждой координаты пиксела.

sx = sign(dx)

sy = sign(dy)

# Вычисление модулей приращения координат.

dx = fabs(dx)

dy = fabs(dy)

# Анализ вычисленного значения m и обмен местами dX и dY при m > 1.

if dy >= dx:

dx, dy = dy, dx

fl = 1 # флаг, определяющий факт обмена местами координат.

else:

fl = 0

# Инициализация начального значения ошибки.

f = 2 \* dy - dx

# Инициализация начальных значений координат текущего пиксела.

x = round(x\_start)

y = round(y\_start)

# Цикл от i=1 до i=dX+1 с шагом 1.

i = 1

step = 1

max\_step = 1

x\_buff = x

y\_buff = y

while i <= dx + 1:

# Высвечивание точки с координатами (X,Y).

if draw:

self.draw\_pixel(x, y)

# Вычисление координат и ошибки для следующего пиксела.

if f >= 0:

if fl == 1:

x += sx

else:

y += sy

f -= 2 \* dx

if f <= 0:

if fl == 1:

y += sy

else:

x += sx

f += 2 \* dy

i += 1

if steps and i <= dx:

if (x\_buff == x and y\_buff != y) or (x\_buff != x and y\_buff == y):

step += 1

else:

if max\_step < step:

max\_step = step

step = 1

x\_buff = x

y\_buff = y

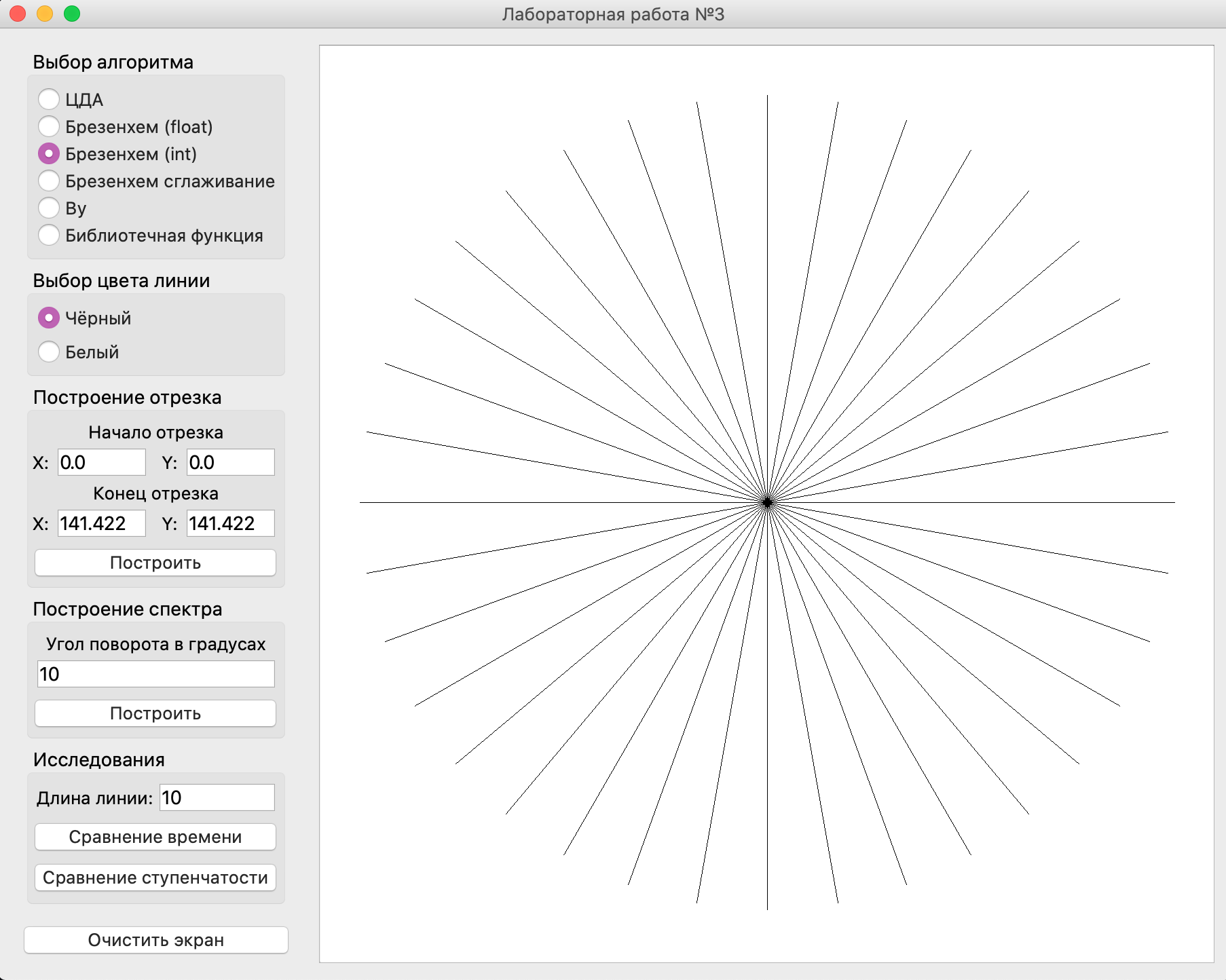
if steps:

return max(max\_step, step)

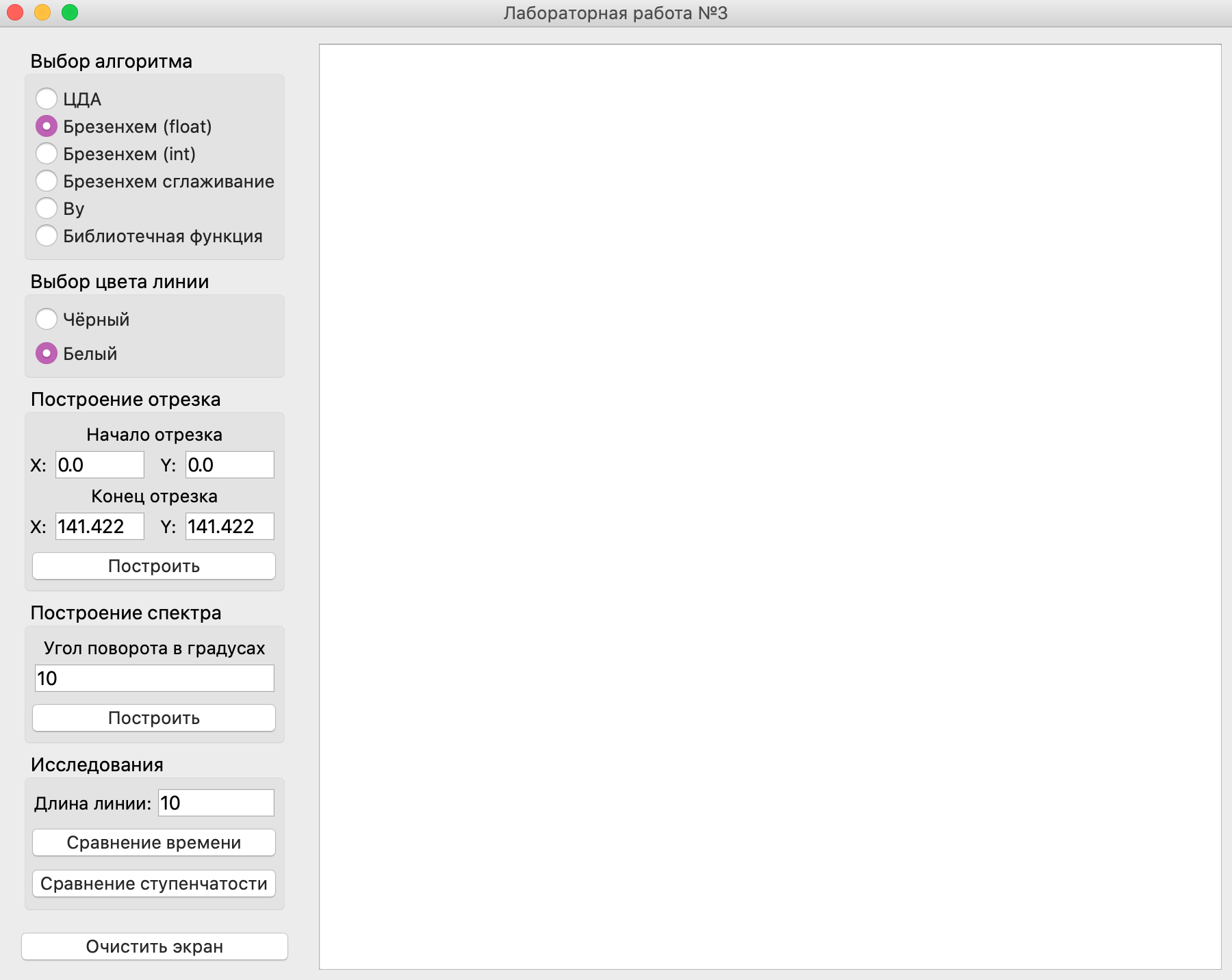
Результат работы алгоритма Брезенхема с целыми коэффициентами полностью идентичен результату работы алгоритма с вещественными коэффициентами.

Это можно проверить, высветив линию одним алгоритмом чёрным цветом, а затем высветив линию другим алгоритмом цветом фона. Все пикселы должны исчезнуть.

Рисуем спектр чёрным цветом с Брезенхем (int):



Рисуем спектр белым цветом с Брезенхем (float):



Чёрных пикселей не видно, значит результат алгоритмов идентичен.

**Алгоритм Брезенхема построения отрезка с устранением ступен-ти:**

Используется при отображении ребер многоугольника, который закрашивается. Идея состоит в сглаживании резких переходов от ступени к ступени. Сглаживание основывается на том, что каждый пиксель высвечивается со своим уровнем интенсивности. Уровень выбирается пропорционально площади части пикселя. 1 пиксель – квадрат с единичной стороной, а не математическая точка.

def bres\_smooth(self, x\_start, y\_start, x\_end, y\_end, draw=True, steps=False):

I = 255

# Проверка вырожденности отрезка. Если отрезок вырожден, то высвечивание отдельного пиксела.

if x\_start == x\_end and y\_start == y\_end:

if draw:

self.draw\_pixel(round(x\_start), round(y\_start))

return

# Вычисление приращений.

dx = x\_end - x\_start

dy = y\_end - y\_start

# Вычисление шага изменения каждой координаты пиксела.

sx = sign(dx)

sy = sign(dy)

# Вычисление модулей приращения координат.

dx = fabs(dx)

dy = fabs(dy)

# Вычисление модуля тангенса угла наклона отрезка

m = dy / dx

# Анализ вычисленного значения m и обмен местами dX и dY при m > 1.

if m >= 1:

dx, dy = dy, dx

m = 1 / m

fl = 1 # флаг, определяющий факт обмена местами координат.

else:

fl = 0

# Инициализация начального значения ошибки.

f = I / 2

# Инициализация начальных значений координат текущего пиксела.

x = round(x\_start)

y = round(y\_start)

# Вычисление скорректированного значения тангенса угла наклона m и коэффициента W.

m \*= I

W = I - m

# Высвечивание пиксела с координатами (X,Y) интенсивностью E(f).

if draw:

self.draw\_pixel(x, y, round(f))

step = 1

max\_step = 1

x\_buff = x

y\_buff = y

# Цикл от i=1 до i=dX с шагом 1.

i = 1

while i <= dx:

if f < W:

if fl == 0:

x += sx

else:

y += sy

f += m

else:

x += sx

y += sy

f -= W

if draw:

self.draw\_pixel(x, y, round(f))

if steps:

if (x == x\_buff and y != y\_buff) or (x != x\_buff and y == y\_buff):

step += 1

else:

if max\_step < step:

max\_step = step

step = 1

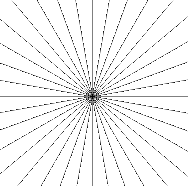
x\_buff = x

y\_buff = y

i += 1

if steps:

return max(max\_step, step)



Данный алгоритм хорошо подойдет для отображения многоугольников, в которых плоская сторона будет снаружи, а сторона со «ступеньками» внутри, для дальнейшего закрашивания многоугольника.

**Алгоритм Ву:**

def wu(self, x\_start, y\_start, x\_end, y\_end, draw=True, steps=False):

dx = x\_end - x\_start

dy = y\_end - y\_start

Imax = 255

m = 1

step = 1

max\_step = 1

if fabs(dy) > fabs(dx):

if y\_start > y\_end:

x\_start, x\_end = x\_end, x\_start

y\_start, y\_end = y\_end, y\_start

if dy != 0:

m = dx / dy

for y in range(round(y\_start), round(y\_end) + 1):

d1 = x\_start - floor(x\_start)

d2 = 1 - d1

if draw:

# Нижняя точка.

self.draw\_pixel(int(x\_start), y, round(fabs(d2) \* Imax))

# Верхняя точка.

self.draw\_pixel(int(x\_start) + 1, y, round(fabs(d1) \* Imax))

if steps and y < round(y\_end):

if int(x\_start) == int(x\_start+m):

step += 1

else:

if max\_step < step:

max\_step = step

step = 1

x\_start += m

else:

if x\_start > x\_end:

x\_start, x\_end = x\_end, x\_start

y\_start, y\_end = y\_end, y\_start

if dx != 0:

m = dy / dx

for x in range(round(x\_start), round(x\_end) + 1):

d1 = y\_start - floor(y\_start)

d2 = 1 - d1

if draw:

# Нижняя точка.

self.draw\_pixel(x, int(y\_start), round(fabs(d2) \* Imax))

# Верхняя точка.

self.draw\_pixel(x, int(y\_start) + 1, round(fabs(d1) \* Imax))

if steps and x < round(x\_end):

if int(y\_start) == int(y\_start+m):

step += 1

else:

if max\_step < step:

max\_step = step

step = 1

y\_start += m

if steps:

return max(max\_step, step)

В алгоритме Ву на каждом шаге устанавливается

не одна, а две точки. Например, если основной осью является Х, то рассматриваются

точки с координатами (х, у) и (х, у+1). В зависимости от величины ошибки, которая

показывает как далеко ушли пиксели от идеальной линии по неосновной оси,

распределяется интенсивность между этими двумя точками. Чем больше удалена точка от

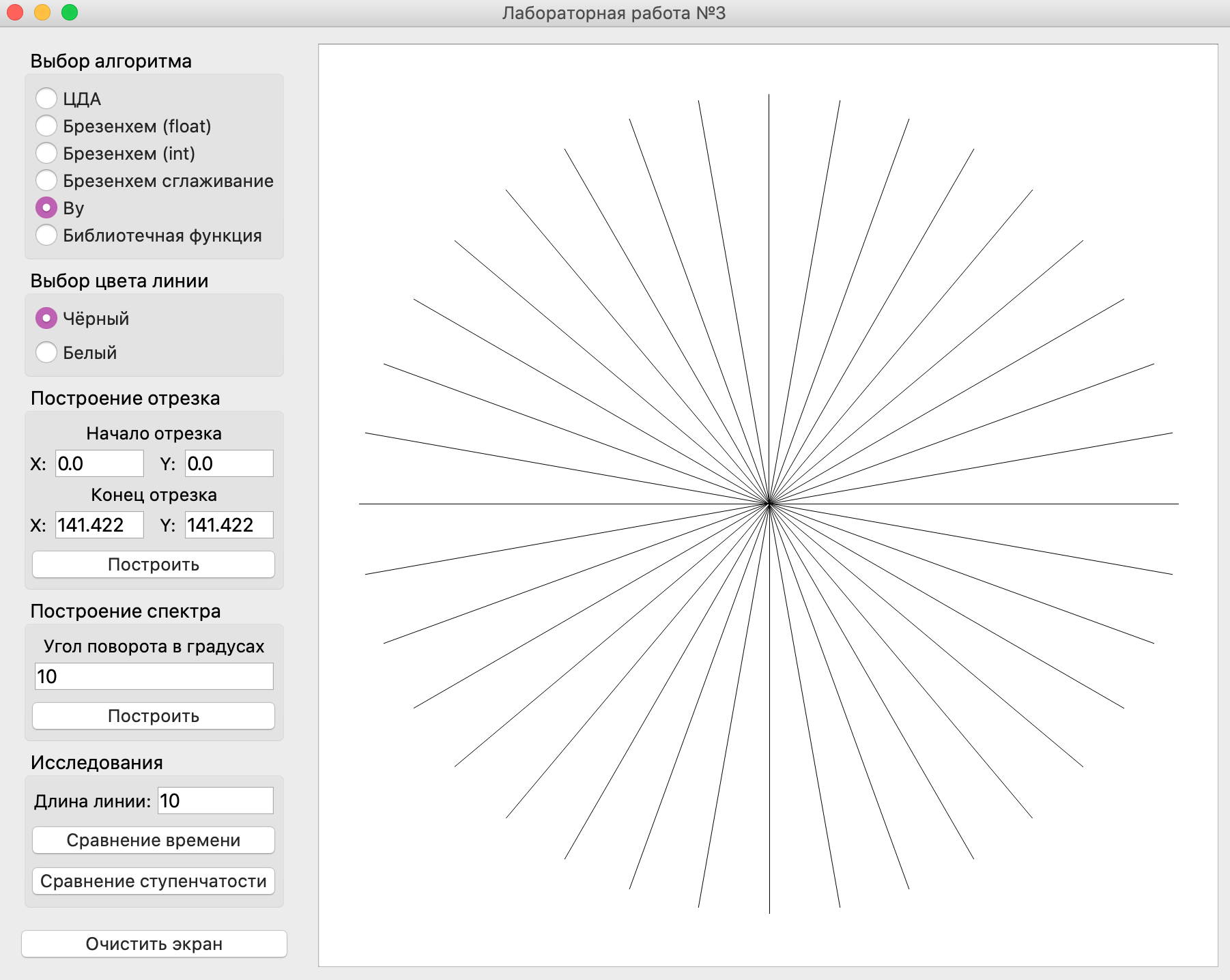
идеальной линии, тем меньше ее интенсивность. Значения интенсивности двух пикселей

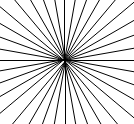
всегда дают в сумме единицу, то есть это интенсивность одного пикселя, в точности

попавшего на идеальную линию. Такое распределение придаст линии одинаковую

интенсивность на всем ее протяжении, создавая при этом иллюзию, что точки

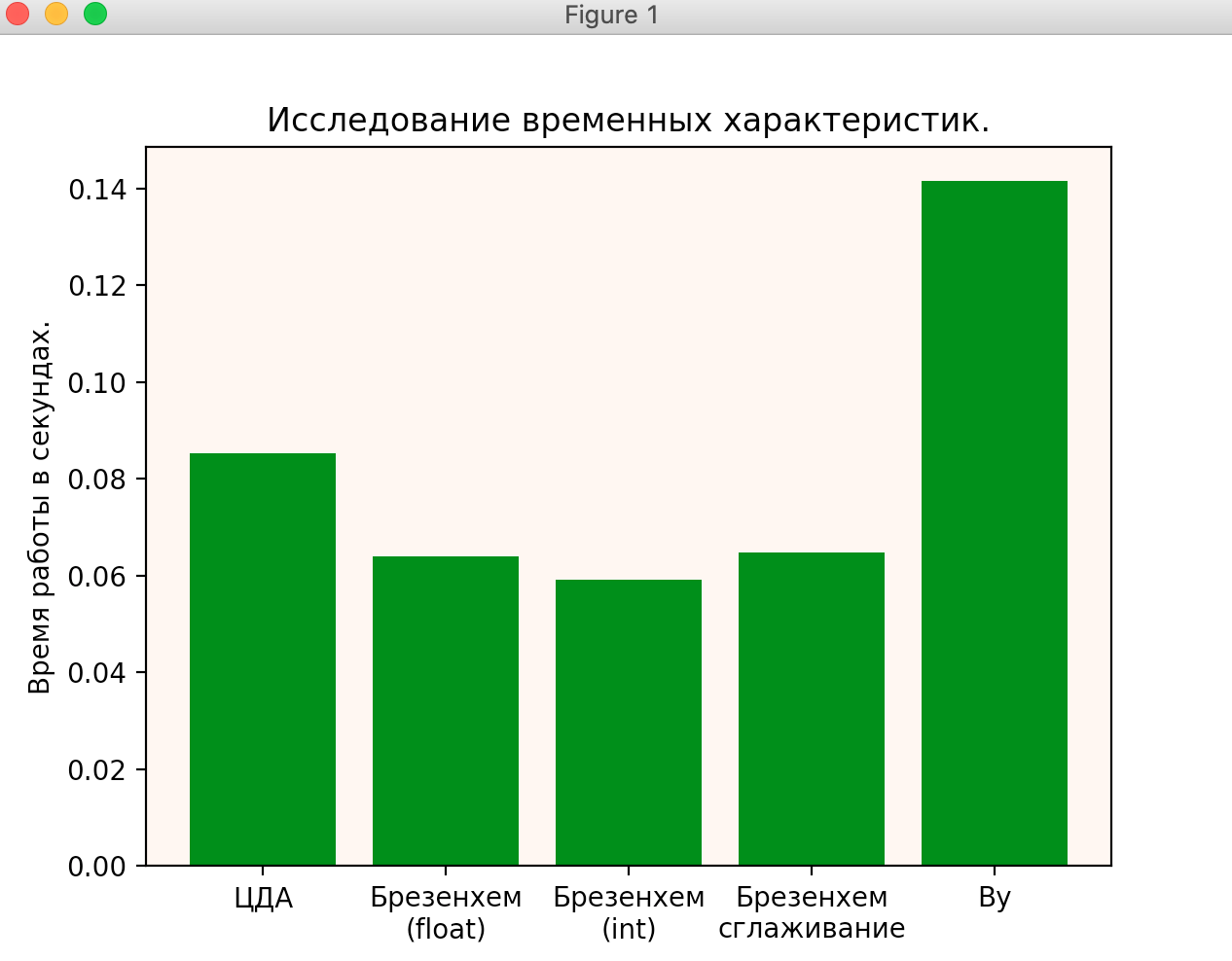
расположены вдоль линии не по две, а по одной.





**Дополнительные задания.**

**Исследование временных характеристик:**



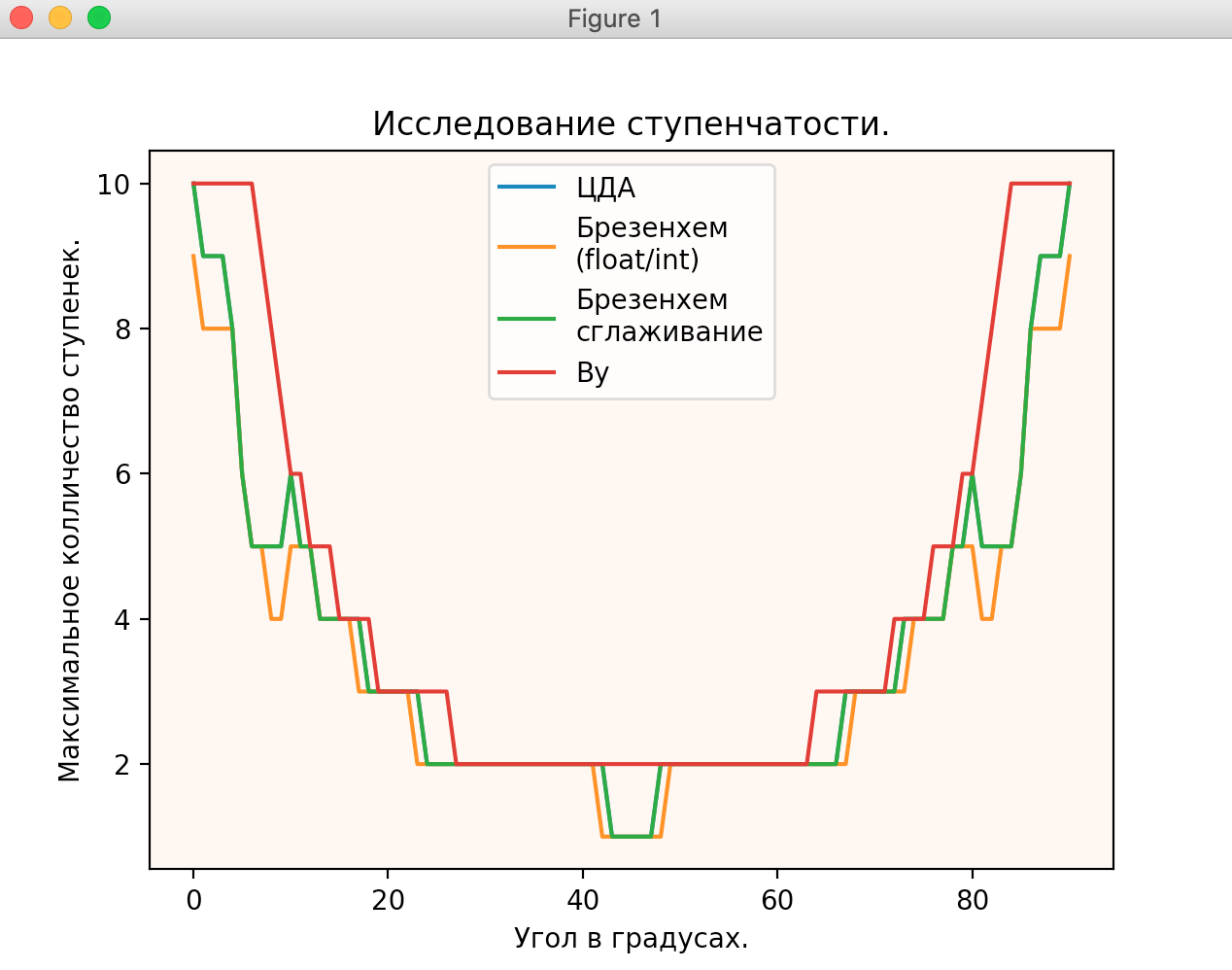
Результат получен при построении спектра линий длинной 10 с углом в 1 градус. Спектр был построен каждым алгоритмом по 10 раз.

**Анализ результата:**

Алгоритм Брезенхема с целыми коэффициентами работает немного быстрее чем алгоритм с действительными коэффициентами. Разница не очень большая, т.к. современные процессоры стали очень быстро выполнять вычисления с плавающей запятой. Лет 10 назад разрыв был бы намного существеннее.

Алгоритм Ву сложнее остальных т.к. он высвечивает не одну, а две точки. Значение интенсивности двух пикселей всегда дают в сумме единицу, то есть это интенсивность одного пикселя, в точности попавшего на идеальную линию. Такое распределение придаст линии одинаковую интенсивность на всем ее протяжении, создавая при этом иллюзию, что точки расположены вдоль линии не по две, а по одной.

**Исследование ступенчатости:**

****

Результат был получен исследованием отрезков, расположенных в пределах первого октанта. При этом ступеньку образуют те пиксели, у которых значение одной из координат остаётся неизменным при изменяющейся другой координате.

**Анализ результата:**

Результаты алгоритма Брезенхема с целыми коэффициентами совпадают с результатом алгоритма Брезенхема с вещественными коэффициентами.

Результат алгоритма ЦДА совпадает с результатом алгоритма Брезенхема со сглаживанием, но это касается только максимального количества ступенек, визуально алгоритмы совершенно разные.

Как видно количество ступенек алгоритма Ву превышает количество ступенек всех других алгоритмов, но это не говорит о его не качественности. Наоборот, визуально это самый лучший алгоритм из представленных, это достигается грамотной работой с интенсивностью высвечивания точек.

**Полный код программы**

import sys

from PyQt5 import QtWidgets, QtGui, QtCore

import design

from time import time

from math import pi, cos, sin, radians, fabs, floor

import matplotlib.pyplot as plt

class Visual(QtWidgets.QMainWindow, design.Ui\_MainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setupUi(self)

self.graphicsView.scale(0.5, -0.5)

self.scene = QtWidgets.QGraphicsScene()

self.graphicsView.setScene(self.scene)

h = self.graphicsView.height()

w = self.graphicsView.width()

self.scene.setSceneRect(-w/2, -h/2, w-2, h-2)

self.pen = QtGui.QPen(QtCore.Qt.black)

self.pen.setWidth(0)

# Связи кнопок и функций.

self.pushButtonScreenClean.clicked.connect(self.clean\_screen)

self.radioButtonBlack.clicked.connect(self.set\_black)

self.radioButtonWhite.clicked.connect(self.set\_white)

self.pushButtonDrawLine.clicked.connect(self.draw\_line)

self.pushButtonDrawSpectrum.clicked.connect(self.draw\_spectrum)

self.pushButtonTimeComp.clicked.connect(self.time\_compare)

self.pushButtonStepComp.clicked.connect(self.step\_comp)

def clean\_screen(self):

self.scene.clear()

def set\_black(self):

self.pen.setColor(QtCore.Qt.black)

def set\_white(self):

self.pen.setColor(QtCore.Qt.white)

def add\_line(self, x\_start, y\_start, x\_end, y\_end, draw=True):

if self.radioButtonCDA.isChecked():

self.cda(x\_start, y\_start, x\_end, y\_end, draw)

elif self.radioButtonBresFloat.isChecked():

self.bres\_float(x\_start, y\_start, x\_end, y\_end, draw)

elif self.radioButtonBresInt.isChecked():

self.bres\_int(x\_start, y\_start, x\_end, y\_end, draw)

elif self.radioButtonBresSmooth.isChecked():

self.bres\_smooth(x\_start, y\_start, x\_end, y\_end, draw)

elif self.radioButtonWu.isChecked():

self.wu(x\_start, y\_start, x\_end, y\_end, draw)

else:

self.lib\_func(x\_start, y\_start, x\_end, y\_end)

def draw\_line(self):

try:

x\_start = float(self.lineEditStartX.text())

y\_start = float(self.lineEditStartX.text())

x\_end = float(self.lineEditEndX.text())

y\_end = float(self.lineEditEndY.text())

except:

QtWidgets.QMessageBox.critical(self, "", "Координаты начала и конца отрезка должны быть целыми "

"или веществеными числами!")

return

self.add\_line(x\_start, y\_start, x\_end, y\_end)

def draw\_spectrum(self, bo=False, draw=True, length=600):

try:

angle = int(self.lineEditAngle.text())

except:

QtWidgets.QMessageBox.critical(self, "", "Угол поворота должен быть целым числом!")

return

if angle < 1:

QtWidgets.QMessageBox.critical(self, "", "Угол поворота должен быть больше нуля!")

return

t = 0

while t < 2 \* pi - 0.001:

x = length \* cos(t)

y = length \* sin(t)

self.add\_line(0, 0, x, y, draw)

t += radians(angle)

def draw\_pixel(self, x, y, alpha=255):

color = self.pen.color()

QtGui.QColor.setAlpha(color, alpha)

self.pen.setColor(color)

self.scene.addLine(x, y, x, y, self.pen)

def lib\_func(self, x\_start, y\_start, x\_end, y\_end):

self.scene.addLine(round(x\_start), round(y\_start), round(x\_end), round(y\_end), self.pen)

def cda(self, x\_start, y\_start, x\_end, y\_end, draw=True, steps=False):

# Проверка вырожденности орезка в точку.

if x\_start == x\_end and y\_start == y\_end and draw:

self.draw\_pixel(round(x\_start), round(y\_start))

return

# Вычисление длины отрезка.

dx = x\_end - x\_start

dy = y\_end - y\_start

if fabs(dx) > fabs(dy):

l = fabs(dx)

else:

l = fabs(dy)

dx /= l

dy /= l

# Задание координатам текущей точки начальных значений.

x = x\_start

y = y\_start

i = 1

step = 1

max\_step = 1

while i <= l + 1:

if draw:

self.draw\_pixel(round(x), round(y))

if steps and i <= l:

if (round(x + dx) == round(x) and round(y + dy) != round(y)) or (round(x + dx) != round(x) and round(y + dy) == round(y)):

step += 1

else:

if max\_step < step:

max\_step = step

step = 1

x += dx

y += dy

i += 1

if steps:

return max(max\_step, step)

def bres\_float(self, x\_start, y\_start, x\_end, y\_end, draw=True, steps=False):

# Проверка вырожденности орезка в точку.

if x\_start == x\_end and y\_start == y\_end:

if draw:

self.draw\_pixel(round(x\_start), round(y\_start))

return

# Вычисление приращений.

dx = x\_end - x\_start

dy = y\_end - y\_start

# Вычисление шага изменения каждой координаты пиксела.

sx = sign(dx)

sy = sign(dy)

# Вычисление модулей приращения координат.

dx = fabs(dx)

dy = fabs(dy)

# Анализ вычисленного значения m и обмен местами dX и dY при m > 1.

if dy >= dx:

dx, dy = dy, dx

fl = 1 # флаг, определяющий факт обмена местами координат.

else:

fl = 0

# Вычисление модуля тангенса угла наклона отрезка.

m = dy / dx

# Инициализация начального значения ошибки.

f = m - 0.5

# Инициализация начальных значений координат текущего пиксела.

x = round(x\_start)

y = round(y\_start)

# Цикл от i=1 до i=dX+1 с шагом 1.

i = 1

step = 1

max\_step = 1

x\_buff = x

y\_buff = y

while i <= dx + 1:

# Высвечивание точки с координатами (X,Y).

if draw:

self.draw\_pixel(x, y)

# Вычисление координат и ошибки для следующего пиксела.

if f >= 0:

if fl == 1:

x += sx

else:

y += sy

f -= 1

if f <= 0:

if fl == 1:

y += sy

else:

x += sx

f += m

i += 1

if steps and i <= dx:

if (x\_buff == x and y\_buff != y) or (x\_buff != x and y\_buff == y):

step += 1

else:

if max\_step < step:

max\_step = step

step = 1

x\_buff = x

y\_buff = y

if steps:

return max(max\_step, step)

def bres\_int(self, x\_start, y\_start, x\_end, y\_end, draw=True, steps=False):

# Проверка вырожденности орезка в точку.

if x\_start == x\_end and y\_start == y\_end:

if draw:

self.draw\_pixel(round(x\_start), round(y\_start))

return

# Вычисление приращений.

dx = x\_end - x\_start

dy = y\_end - y\_start

# Вычисление шага изменения каждой координаты пиксела.

sx = sign(dx)

sy = sign(dy)

# Вычисление модулей приращения координат.

dx = fabs(dx)

dy = fabs(dy)

# Анализ вычисленного значения m и обмен местами dX и dY при m > 1.

if dy >= dx:

dx, dy = dy, dx

fl = 1 # флаг, определяющий факт обмена местами координат.

else:

fl = 0

# Инициализация начального значения ошибки.

f = 2 \* dy - dx

# Инициализация начальных значений координат текущего пиксела.

x = round(x\_start)

y = round(y\_start)

# Цикл от i=1 до i=dX+1 с шагом 1.

i = 1

step = 1

max\_step = 1

x\_buff = x

y\_buff = y

while i <= dx + 1:

# Высвечивание точки с координатами (X,Y).

if draw:

self.draw\_pixel(x, y)

# Вычисление координат и ошибки для следующего пиксела.

if f >= 0:

if fl == 1:

x += sx

else:

y += sy

f -= 2 \* dx

if f <= 0:

if fl == 1:

y += sy

else:

x += sx

f += 2 \* dy

i += 1

if steps and i <= dx:

if (x\_buff == x and y\_buff != y) or (x\_buff != x and y\_buff == y):

step += 1

else:

if max\_step < step:

max\_step = step

step = 1

x\_buff = x

y\_buff = y

if steps:

return max(max\_step, step)

def bres\_smooth(self, x\_start, y\_start, x\_end, y\_end, draw=True, steps=False):

I = 255

# Проверка вырожденности отрезка. Если отрезок вырожден, то высвечивание отдельного пиксела.

if x\_start == x\_end and y\_start == y\_end:

if draw:

self.draw\_pixel(round(x\_start), round(y\_start))

return

# Вычисление приращений.

dx = x\_end - x\_start

dy = y\_end - y\_start

# Вычисление шага изменения каждой координаты пиксела.

sx = sign(dx)

sy = sign(dy)

# Вычисление модулей приращения координат.

dx = fabs(dx)

dy = fabs(dy)

# Вычисление модуля тангенса угла наклона отрезка

m = dy / dx

# Анализ вычисленного значения m и обмен местами dX и dY при m > 1.

if m >= 1:

dx, dy = dy, dx

m = 1 / m

fl = 1 # флаг, определяющий факт обмена местами координат.

else:

fl = 0

# Инициализация начального значения ошибки.

f = I / 2

# Инициализация начальных значений координат текущего пиксела.

x = round(x\_start)

y = round(y\_start)

# Вычисление скорректированного значения тангенса угла наклона m и коэффициента W.

m \*= I

W = I - m

# Высвечивание пиксела с координатами (X,Y) интенсивностью E(f).

if draw:

self.draw\_pixel(x, y, round(f))

step = 1

max\_step = 1

x\_buff = x

y\_buff = y

# Цикл от i=1 до i=dX с шагом 1.

i = 1

while i <= dx:

if f < W:

if fl == 0:

x += sx

else:

y += sy

f += m

else:

x += sx

y += sy

f -= W

if draw:

self.draw\_pixel(x, y, round(f))

if steps:

if (x == x\_buff and y != y\_buff) or (x != x\_buff and y == y\_buff):

step += 1

else:

if max\_step < step:

max\_step = step

step = 1

x\_buff = x

y\_buff = y

i += 1

if steps:

return max(max\_step, step)

def wu(self, x\_start, y\_start, x\_end, y\_end, draw=True, steps=False):

dx = x\_end - x\_start

dy = y\_end - y\_start

Imax = 255

m = 1

step = 1

max\_step = 1

if fabs(dy) > fabs(dx):

if y\_start > y\_end:

x\_start, x\_end = x\_end, x\_start

y\_start, y\_end = y\_end, y\_start

if dy != 0:

m = dx / dy

for y in range(round(y\_start), round(y\_end) + 1):

d1 = x\_start - floor(x\_start)

d2 = 1 - d1

if draw:

# Нижняя точка.

self.draw\_pixel(int(x\_start), y, round(fabs(d2) \* Imax))

# Верхняя точка.

self.draw\_pixel(int(x\_start) + 1, y, round(fabs(d1) \* Imax))

if steps and y < round(y\_end):

if int(x\_start) == int(x\_start+m):

step += 1

else:

if max\_step < step:

max\_step = step

step = 1

x\_start += m

else:

if x\_start > x\_end:

x\_start, x\_end = x\_end, x\_start

y\_start, y\_end = y\_end, y\_start

if dx != 0:

m = dy / dx

for x in range(round(x\_start), round(x\_end) + 1):

d1 = y\_start - floor(y\_start)

d2 = 1 - d1

if draw:

# Нижняя точка.

self.draw\_pixel(x, int(y\_start), round(fabs(d2) \* Imax))

# Верхняя точка.

self.draw\_pixel(x, int(y\_start) + 1, round(fabs(d1) \* Imax))

if steps and x < round(x\_end):

if int(y\_start) == int(y\_start+m):

step += 1

else:

if max\_step < step:

max\_step = step

step = 1

y\_start += m

if steps:

return max(max\_step, step)

def time\_compare(self):

try:

length = int(self.lineEditLineLen.text())

except:

QtWidgets.QMessageBox.critical(self, "", "Длина линии должна быть целым числом!")

return

if length < 1:

QtWidgets.QMessageBox.critical(self, "", "Длина линии должна быть больше нуля!")

return

angle = self.lineEditAngle.text()

self.lineEditAngle.setText('10')

data = []

self.radioButtonCDA.toggle()

start = time()

for i in range(5):

self.draw\_spectrum(draw=True, length=length)

data.append(time() - start)

self.radioButtonBresFloat.toggle()

start = time()

for i in range(5):

self.draw\_spectrum(draw=True, length=length)

data.append(time() - start)

self.radioButtonBresInt.toggle()

start = time()

for i in range(5):

self.draw\_spectrum(draw=True, length=length)

data.append(time() - start)

self.radioButtonBresSmooth.toggle()

start = time()

for i in range(5):

self.draw\_spectrum(draw=True, length=length)

data.append(time() - start)

self.radioButtonWu.toggle()

start = time()

for i in range(5):

self.draw\_spectrum(draw=True, length=length)

data.append(time() - start)

self.lineEditAngle.setText(angle)

fig, ax = plt.subplots()

ax.bar(['ЦДА', 'Брезенхем\n(float)', 'Брезенхем\n(int)', 'Брезенхем\nсглаживание', 'Ву'], data, color='green')

ax.set\_facecolor('seashell')

plt.title("Исследование временных характеристик.")

plt.ylabel("Время работы в секундах.")

self.scene.clear()

plt.show()

def step\_comp(self):

try:

length = int(self.lineEditLineLen.text()) - 1

except:

QtWidgets.QMessageBox.critical(self, "", "Длина линии должна быть целым числом!")

return

if length < 1:

QtWidgets.QMessageBox.critical(self, "", "Длина линии должна быть больше нуля!")

return

cda = []

bres\_float = []

bres\_int = []

bres\_smooth = []

wu = []

t = 0

angles = [i for i in range(91)]

while t <= pi / 2 + 0.01:

x = length \* cos(t)

y = length \* sin(t)

cda.append(self.cda(0, 0, x, y, False, True))

bres\_float.append(self.bres\_float(0, 0, x, y, False, True))

bres\_int.append(self.bres\_int(0, 0, x, y, False, True))

bres\_smooth.append(self.bres\_smooth(0, 0, x, y, False, True))

wu.append(self.wu(0, 0, x, y, False, True))

t += radians(1)

fig, ax = plt.subplots()

ax.plot(angles, cda, label='ЦДА')

ax.plot(angles, bres\_float, label='Брезенхем\n(float/int)')

ax.plot(angles, bres\_smooth, label='Брезенхем\nсглаживание')

ax.plot(angles, wu, label='Ву')

ax.set\_facecolor('seashell')

plt.title("Исследование ступенчатости.")

plt.legend()

plt.ylabel("Максимальное колличество ступенек.")

plt.xlabel("Угол в градусах.")

plt.show()

def sign(x):

if x < 0:

return -1

elif x > 0:

return 1

return 0

def main():

app = QtWidgets.QApplication(sys.argv)

window = Visual()

window.show()

app.exec\_()

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

main()