|  |  |
| --- | --- |
| 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** Челядинов Илья  **Группа ИУ7-43**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |  |

Москва.

2020 г.

**Цель работы:**

Анализ и практическое применение методов построения отрезков

**Техническое задание:**

Организовать интерфейс для построения отрезков данными методами:

* Алгоритм цифрового дифференциального анализатора;
* Алгоритмы Брезенхема;
* Алгоритм Ву;

Добавить возможность выбора цвета фона и линии

Построение линии по координатам начала и конца

Регулировании длины линии

Исследование визуальных характеристик для отрезка, расположенного во всем спектре изменения углов;

**Теоретический материал:**

Общие требования:

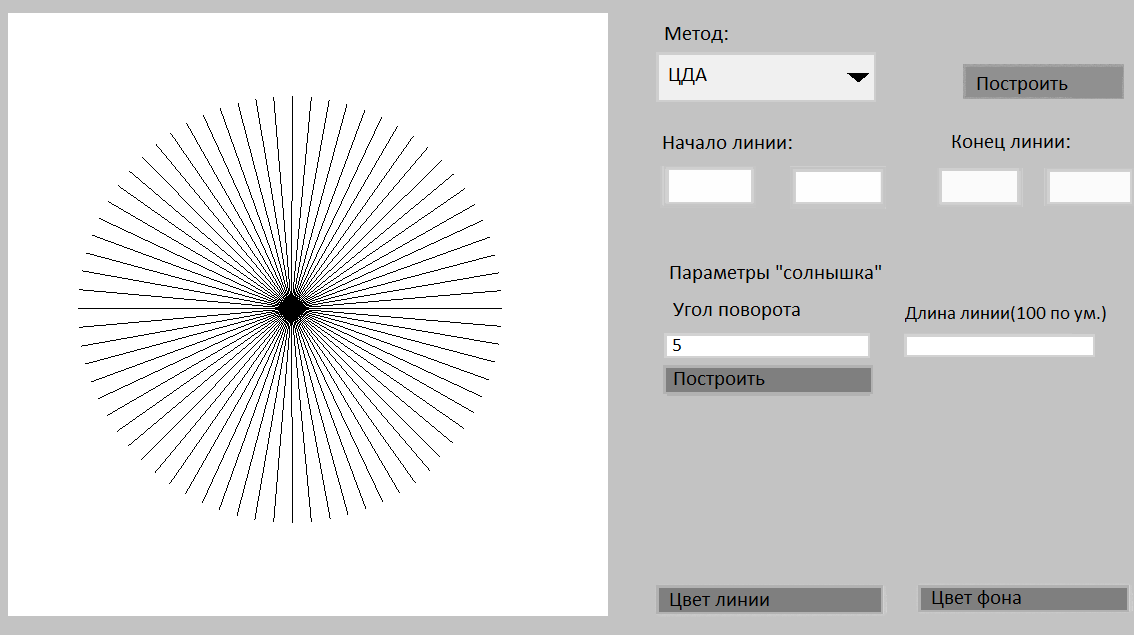
1. Отрезок должен выглядеть как отрезок прямой, начинаться и заканчиваться в заданных точках

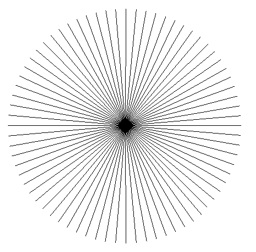
2. Предоставить реализация оптимизированных алгоритмов

**Алгоритм цифрового дифференциального анализатора:**

У этого алгоритма есть существенный минус, который сильно замедляет его работу – округление в цикле. При большом количестве итераций данный метод не эффективен

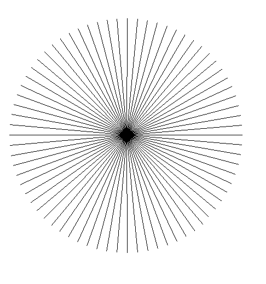
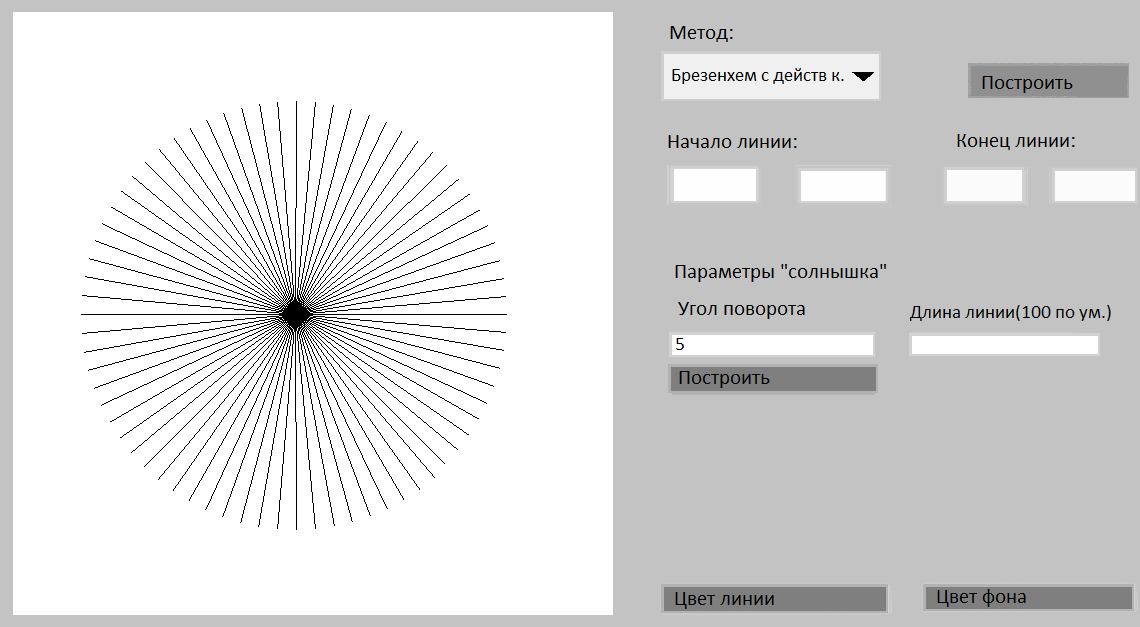
dx = abs(pf[0] - ps[0])  
dy = abs(pf[1] - ps[1])  
  
**if** dx:  
 tg = dy / dx  
**else**:  
 tg = 0  
  
**if** dx > dy:  
 steep = dx  
**else**:  
 steep = dy  
sx = (pf[0] - ps[0]) / steep  
sy = (pf[1] - ps[1]) / steep  
  
x = ps[0]  
y = ps[1]  
stairs = []  
st = 1  
**while** abs(x - pf[0]) > 1 **or** abs(y - pf[1]) > 1:  
 canvas.create\_line(round(x), round(y), round(x + 1), round(y + 1), fill=fill)  
 **if** (abs(int(x) - int(x + sx)) >= 1 **and** tg > 1) **or** (abs(int(y) - int(y + sy)) >= 1 >= tg):  
 stairs.append(st)  
 st = 0  
 **else**:  
 st += 1  
 x += sx  
 y += sy





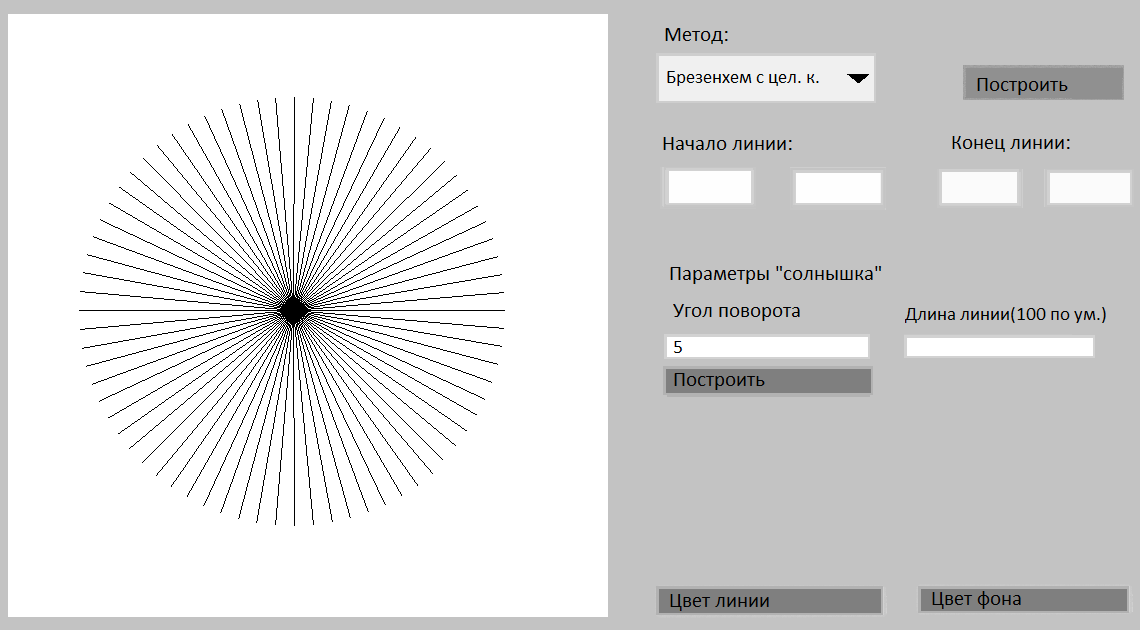
**Алгоритм Брезенхема с действительными коэффициентами:**

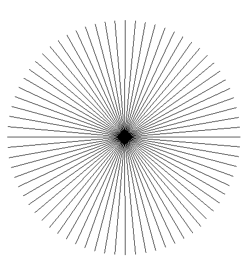
dx = pf[0] - ps[0]  
dy = pf[1] - ps[1]  
sx = sign(dx)  
sy = sign(dy)  
dy = abs(dy)  
dx = abs(dx)  
  
**if** dy >= dx:  
 dx, dy = dy, dx  
 steep = 1 *# шагаем по y***else**:  
 steep = 0  
  
tg = dy / dx *# tангенс угла наклона*e = tg - 1 / 2 *# начальное значение ошибки*x = ps[0] *# начальный икс*y = ps[1] *# начальный игрек*stairs = []  
st = 1  
**while** x != pf[0] **or** y != pf[1]:  
 canvas.create\_line(x, y, x + 1, y + 1, fill=fill)  
 *# выбираем пиксель* **if** e >= 0:  
 **if** steep == 1: *# dy >= dx* x += sx  
 **else**: *# dy < dx* y += sy  
 e -= 1 *# отличие от целого* stairs.append(st)  
 st = 0  
 **if** e <= 0:  
 **if** steep == 0: *# dy < dx* x += sx  
 **else**: *# dy >= dx* y += sy  
 st += 1  
 e += tg *# отличие от целого*



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

dx = pf[0] - ps[0]  
dy = pf[1] - ps[1]  
sx = sign(dx)  
sy = sign(dy)  
dy = abs(dy)  
dx = abs(dx)  
**if** dy >= dx:  
 dx, dy = dy, dx  
 steep = 1  
**else**:  
 steep = 0  
e = 2 \* dy - dx *# отличие от вещественного (e = tg - 1 / 2) tg = dy / dx*x = ps[0]  
y = ps[1]  
stairs = []  
st = 1  
**while** x != pf[0] **or** y != pf[1]:  
 canvas.create\_line(x, y, x + 1, y + 1, fill=fill)  
 **if** e >= 0:  
 **if** steep == 1:  
 x += sx  
 **else**:  
 y += sy  
 stairs.append(st)  
 st = 0  
 e -= 2 \* dx *# отличие от вещественного (e -= 1)* **if** e <= 0:  
 **if** steep == 0:  
 x += sx  
 **else**:  
 y += sy  
 st += 1  
 e += 2 \* dy *# difference (e += tg)*

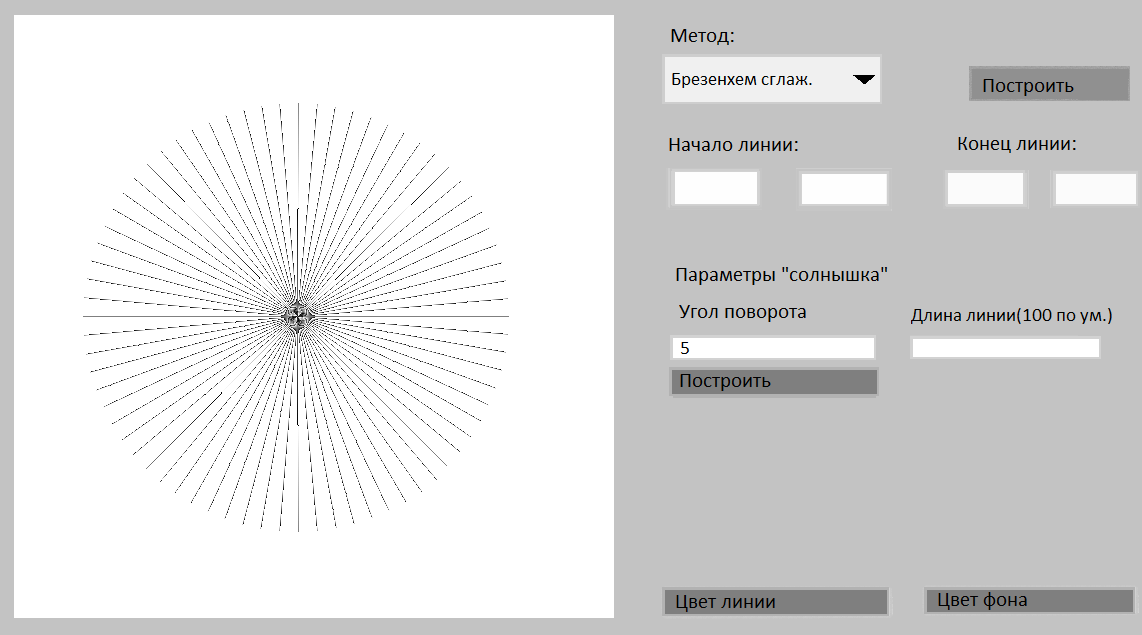
Подчеркнутая строчка – отличие от вещественного метода.

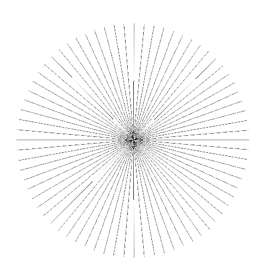


**Алгоритм Брезенхема построения отрезка с устранением ступенчатости:**

Метод основан на изменении интенсивности точек при переходе от ступени к ступени, поэтому линии кажутся немного более тонкими и происходит сглаживание. Мера интенсивности каждой точки выбирается пропорционально площади части пикселя. Часто применяется при прорисовке закрашенного многоугольника.

I = 100  
fill = get\_rgb\_intensity(canvas, fill, I)  
dx = pf[0] - ps[0]  
dy = pf[1] - ps[1]  
sx = sign(dx)  
sy = sign(dy)  
dy = abs(dy)  
dx = abs(dx)  
**if** dy >= dx:  
 dx, dy = dy, dx  
 steep = 1 *#***else**:  
 steep = 0 *#*tg = dy / dx \* I *# тангенс угла наклона (умножаем на инт., чтобы не приходилось умножать внутри цикла*e = I / 2 *# интенсивность для высвечивания начального пикселя*w = I - tg *# пороговое значение*x = ps[0]  
y = ps[1]  
stairs = []  
st = 1  
**while** x != pf[0] **or** y != pf[1]:  
 canvas.create\_line(x, y, x + 1, y + 1, fill=fill[round(e) - 1])  
 **if** e < w:  
 **if** steep == 0: *# dy < dx* x += sx *# -1 if dx < 0, 0 if dx = 0, 1 if dx > 0* **else**: *# dy >= dx* y += sy *# -1 if dy < 0, 0 if dy = 0, 1 if dy > 0* st += 1 *# stepping* e += tg  
 **elif** e >= w:  
 x += sx  
 y += sy  
 e -= w  
 stairs.append(st)  
 st = 0



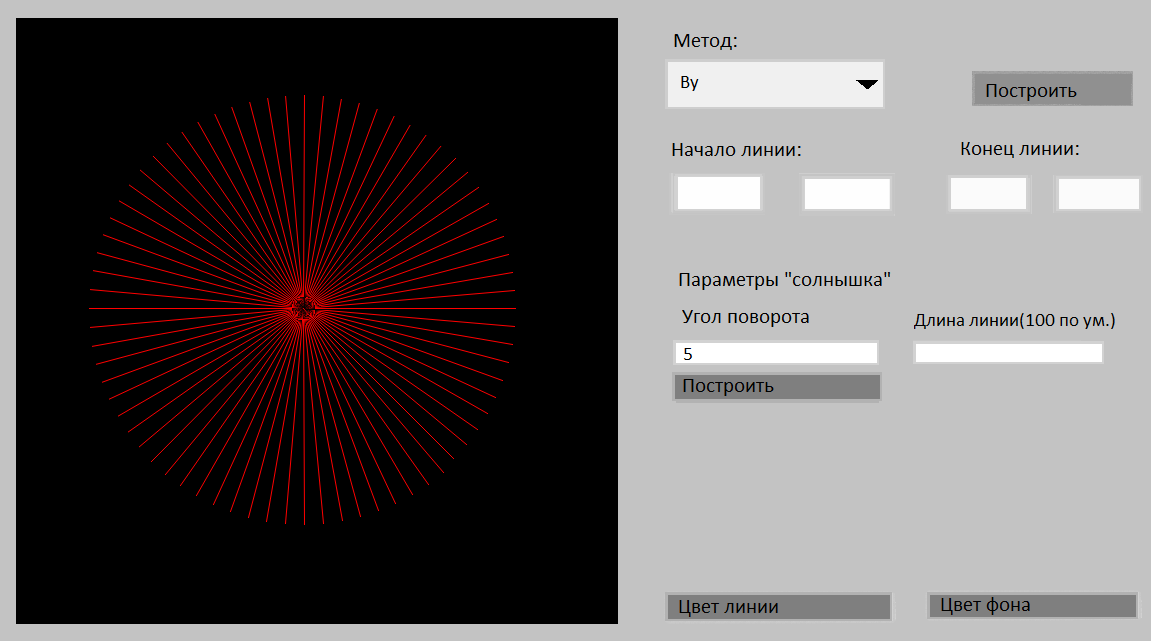


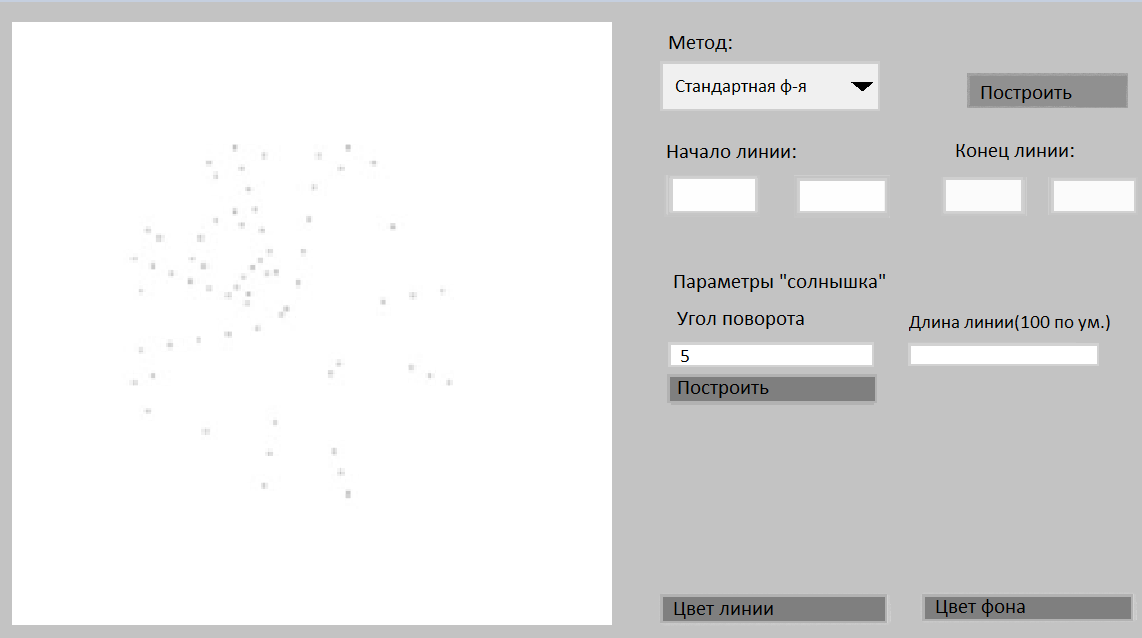
**Алгоритм Ву:**

x1 = ps[0]  
x2 = pf[0]  
y1 = ps[1]  
y2 = pf[1]  
  
I = 100  
stairs = []  
  
fills = get\_rgb\_intensity(canvas, fill, **"white"**, I)  
**if** x1 == x2 **and** y1 == y2:  
 canvas.create\_line(x1, y1, x1 + 1, y1 + 1, fill = fills[100])  
  
steep = abs(y2 - y1) > abs(x2 - x1)  
  
**if** steep:  
 x1, y1 = y1, x1  
 x2, y2 = y2, x2  
**if** x1 > x2:  
 x1, x2 = x2, x1  
 y1, y2 = y2, y1  
  
dx = x2 - x1  
dy = y2 - y1  
  
**if** dx == 0:  
 tg = 1  
**else**:  
 tg = dy / dx  
  
*# first endpoint*xend = round(x1)  
yend = y1 + tg \* (xend - x1)  
xpx1 = xend  
y = yend + tg  
  
*# second endpoint*xend = int(x2 + 0.5)  
xpx2 = xend  
st = 0  
  
*# main loop***if** steep:  
 **for** x **in** range(xpx1, xpx2):  
 canvas.create\_line(int(y), x + 1, int(y) + 1, x + 2,  
 fill = fills[int((I - 1) \* (abs(1 - y + int(y))))])  
 canvas.create\_line(int(y) + 1, x + 1, int(y) + 2, x + 2,  
 fill = fills[int((I - 1) \* (abs(y - int(y))))])  
  
 **if** (abs(int(x) - int(x + 1)) >= 1 **and** tg > 1) **or** \  
 (**not** 1 > abs(int(y) - int(y + tg)) >= tg):  
 stairs.append(st)  
 st = 0  
 **else**:  
 st += 1  
 y += tg  
**else**:  
 **for** x **in** range(xpx1, xpx2):  
 *#print((I - 1)\*round(abs(1 - y + floor(y))))* canvas.create\_line(x + 1, int(y), x + 2, int(y) + 1,  
 fill = fills[round((I - 1) \* (abs(1 - y + floor(y))))])  
 *#print((I - 1)\*round(abs(y - floor(y))))* canvas.create\_line(x + 1, int(y) + 1, x + 2, int(y) + 2,  
 fill = fills[round((I - 1) \* (abs(y - floor(y))))])  
  
 **if** (abs(int(x) - int(x + 1)) >= 1 **and** tg > 1) **or** \  
 (**not** 1 > abs(int(y) - int(y + tg)) >= tg):  
 stairs.append(st)  
 st = 0  
 **else**:  
 st += 1  
 y += tg

Выберем новые цвета для линий и фона

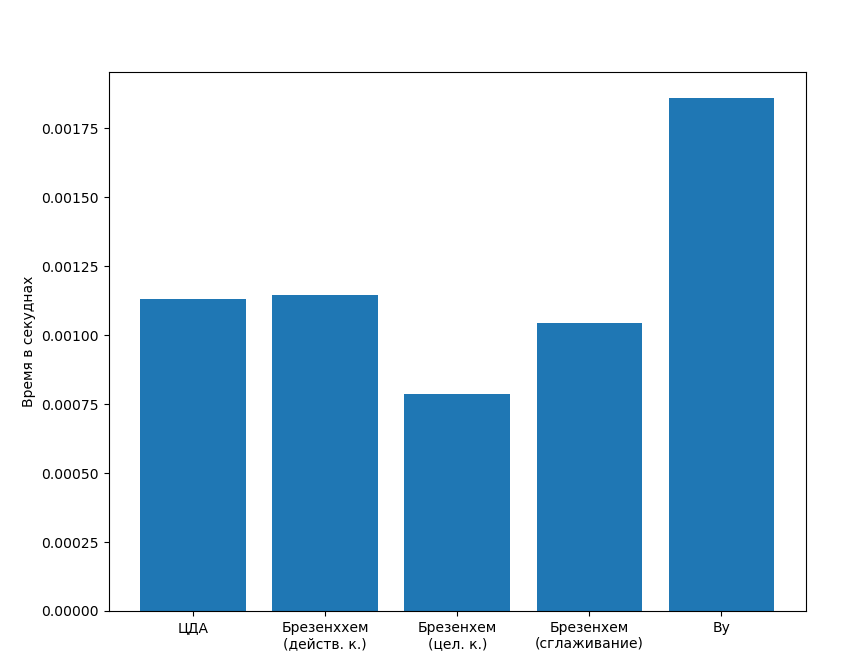
Построим “солнышко”





Судя по картинке, библиотечный алгоритм основан на алгоритме Брезенхема (одном из) или цда. Ву работает медленно.

Время работы алгоритмов (значения в сек.). Результаты замеров приведены путем вычисления среднего арифметического из 20 запусков



По представленным замерам можно сделать вывод, что алгоритм Ву очень медленный, поэтому его использование в графических редакотрах нецелесообразно.