**СОПРОВОДИТЕЛЬНАЯ ДОКУМЕНТАЦИЯ**

**Лабораторная работа 2**

**«Базовые растровые алгоритмы»**

**Цель работы:**

Цель лабораторной работы заключается в разработке приложения для визуализации базовых растровых алгоритмов. Алгоритмы включают в себя:

- Цифровой дифференциальный анализатор (ЦДА)

- Пошаговый алгоритм

- Алгоритм Брезенхема для окружности

- Алгоритм Брезенхема для линии

**Использованные средства разработки**:

Для разработки приложения был выбран язык программирования Python, а в качестве библиотеки для визуализации результатов - Matplotlib. Также были использованы элементы интерфейса Matplotlib Widgets.

**Реализация алгоритмов:**

1. ЦДА (Цифровой дифференциальный анализатор):

def draw\_dda\_line(x1, y1, x2, y2, color="b."):

Данный алгоритм использует алгоритм ЦДА для отрисовки отрезка между двумя точками. Интерфейс предоставляет слайдеры для ввода координат точек и кнопку "Обновить" для визуализации.

1. Пошаговый алгоритм:

def draw\_steps\_line(x, k, b, steps, color="b."):

Пошаговый алгоритм рисует линию с использованием уравнения прямой. Интерфейс аналогичен предыдущему с добавлением соответствующих слайдеров.

1. Брезенхем для окружности:

def draw\_b\_circle(x1, y1, x2, y2, color="b."):

Этот алгоритм использует алгоритм Брезенхема для рисования окружности. Слайдеры используются для ввода координат центра и конечной точки радиуса.

1. Брезенхем для линии:

def draw\_b\_line(x1, y1, x2, y2, color="b."):

Алгоритм Брезенхема для отрисовки линии между двумя точками. Интерфейс аналогичен предыдущим.

**Интерфейс приложения:**

Слайдеры и кнопки:

Интерфейс включает слайдеры для ввода координат точек и параметров алгоритмов, а также кнопки "Обновить" для визуализации изменений.

**Возможности взаимодействия:**

ЦДА (Цифровой дифференциальный анализатор):

- Ввод координат точек с помощью слайдеров.

- Обновление графика по нажатию кнопки "Обновить".

Пошаговый алгоритм:

- Ввод параметров уравнения прямой (k, b) и координаты точки с помощью слайдеров.

- Обновление графика по нажатию кнопки "Обновить".

Брезенхем для окружности:

- Ввод координат центра и конечной точки радиуса с помощью слайдеров.

- Обновление графика по нажатию кнопки "Обновить".

Брезенхем для линии:

- Ввод координат точек с помощью слайдеров.

- Обновление графика по нажатию кнопки "Обновить".

**Заключение:**

В ходе выполнения лабораторной работы было создано приложение на языке программирования Python, предназначенное для демонстрации базовых алгоритмов растеризации и сглаживания. В рамках данного приложения реализованы алгоритмы цифрового дифференциального анализатора (ЦДА), пошагового алгоритма, алгоритма Брезенхема для линий, а также алгоритма Брезенхема для окружностей.

Программа обладает удобным графическим интерфейсом, позволяющим в режиме реального времени изменять параметры линий и окружностей, а также просматривать результаты работы каждого алгоритма. Для этого использованы элементы управления библиотеки Matplotlib, такие как ползунки и кнопки, что делает взаимодействие с программой интуитивно понятным.

Кроме того, чтобы улучшить визуальное восприятие изображения, была добавлена возможность сглаживания при помощи антиалиасинга, используя соответствующие параметры библиотеки Matplotlib.

import matplotlib.pyplot as plt  
import math  
from matplotlib.widgets import Button  
from matplotlib.widgets import Slider  
  
  
def draw\_dda\_line(x1, y1, x2, y2, color="b."):  
 ax = plt.subplot(2, 4, 1)  
 plt.title("ЦДА", fontsize=11)  
 ax.plot(x1, y1, color)  
 ax.grid()  
 ax.set\_xlabel("x (point)")  
 ax.set\_ylabel("y (point)")  
 plt.plot(x1, y1, ':')  
  
 dx = x2 - x1  
 dy = y2 - y1  
  
 # calculate steps required for generating pixels  
 steps = abs(dx) if abs(dx) > abs(dy) else abs(dy)  
  
 # calculate increment in x & y for each step  
 if steps != 0:  
 x\_inc = float(dx / steps)  
 y\_inc = float(dy / steps)  
 else:  
 x\_inc, y\_inc = 0,1  
  
 for i in range(0, int(steps + 1)):  
 # draw pixels  
 plt.plot(x1, y1, color)  
 x1 += x\_inc  
 y1 += y\_inc  
 plt.axis('square')  
  
  
def draw\_steps\_line(x, k, b, steps, color="b."):  
 ax = plt.subplot(2, 4, 2)  
 ax.grid()  
 plt.title("Пошаговый", fontsize=11)  
 ax.set\_xlabel("x (point)")  
 ax.set\_ylabel("y (point)")  
  
 y = float()  
 for i in range(0, int(steps + 1)):  
 # draw pixels  
 y = x \* k + b  
 y = round(y)  
 plt.plot(int(x), y, color)  
 x += 1  
 plt.axis('square')  
  
  
def draw\_b\_circle(x1, y1, x2, y2, color="b."):  
 ax = plt.subplot(2, 4, 3)  
 ax.grid()  
 plt.title("Брезенхем окружность", fontsize=11)  
 ax.set\_xlabel("x (point)")  
 ax.set\_ylabel("y (point)")  
  
 px = []  
 py = []  
 r = math.sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2))  
 disp\_x = x1  
 disp\_y = y1  
 x = 0  
 y = r  
 delta = (2 - 2 \* r)  
 while y >= 0:  
 px.append(disp\_x + x)  
 py.append(disp\_y + y)  
 px.append(disp\_x + x)  
 py.append(disp\_y - y)  
 px.append(disp\_x - x)  
 py.append(disp\_y + y)  
 px.append(disp\_x - x)  
 py.append(disp\_y - y)  
  
 error = 2 \* (delta + y) - 1  
 if (delta < 0) and (error <= 0):  
 x += 1  
 delta = delta + (2 \* x + 1)  
 continue  
 if (delta > 0) and (error > 0):  
 y -= 1  
 delta = delta - (1 + 2 \* y)  
 continue  
 x += 1  
 delta = delta + (2 \* (x - y))  
 y -= 1  
 plt.plot(px, py, color)  
 plt.axis('square')  
  
  
def draw\_b\_line(x1, y1, x2, y2, color="b."):  
 ax = plt.subplot(2, 4, 4)  
 plt.title("Брезенхем линия", fontsize=11)  
 ax.set\_xlabel("x (point)")  
 ax.set\_ylabel("y (point)")  
 plt.plot(x2, y2, color)  
 ax.grid()  
  
 # start algor Brezenhem line  
 dx = x2 - x1  
 dy = y2 - y1  
  
 sign\_x = 1 if dx > 0 else -1 if dx < 0 else 0  
 sign\_y = 1 if dy > 0 else -1 if dy < 0 else 0  
  
 if dx < 0:  
 dx = -dx  
 if dy < 0:  
 dy = -dy  
  
 if dx > dy:  
 pdx, pdy = sign\_x, 0  
 es, el = dy, dx  
 else:  
 pdx, pdy = 0, sign\_y  
 es, el = dx, dy  
  
 x, y = x1, y1  
  
 error, t = el / 2, 0  
  
 plt.plot(x, y, color)  
  
 while t < el:  
 error -= es  
 if error < 0:  
 error += el  
 x += sign\_x  
 y += sign\_y  
 else:  
 x += pdx  
 y += pdy  
 t += 1  
 plt.plot(x, y, color)  
 plt.axis('square')  
  
  
def submit\_dda\_line(x, y, x\_end, y\_end):  
 def clicked(event):  
 ax2 = plt.subplot(2, 4, 1)  
 ax2.cla()  
 draw\_dda\_line(int(x.val), int(y.val), int(x\_end.val), int(y\_end.val))  
 plt.draw()  
  
 return clicked  
  
  
def submit\_steps\_line(x, k, b, steps):  
 def clicked(event):  
 ax2 = plt.subplot(2, 4, 2)  
 ax2.cla()  
 draw\_steps\_line(int(x.val), float(k.val), int(b.val), int(steps.val))  
 plt.axis('square')  
 plt.draw()  
  
 return clicked  
  
  
def submit\_b\_circle(x, y, x\_end, y\_end):  
 def clicked(event):  
 ax2 = plt.subplot(2, 4, 3)  
 ax2.cla()  
 draw\_b\_circle(int(x.val), int(y.val), int(x\_end.val), int(y\_end.val))  
 plt.draw()  
  
 return clicked  
  
  
def submit\_b\_line(x, y, x\_end, y\_end):  
 def clicked(event):  
 ax2 = plt.subplot(2, 4, 4)  
 ax2.cla()  
 draw\_b\_line(int(x.val), int(y.val), int(x\_end.val), int(y\_end.val))  
 plt.draw()  
  
 return clicked  
def draw\_castle\_pitway\_line(x1, y1, x2, y2, color="r."):  
 ax = plt.subplot(2, 4, 6) # Move to the next subplot (right of Bresenham line)  
 plt.title("Кастл-Питвей", fontsize=10)  
 ax.grid()  
 ax.set\_xlabel("x (point)")  
 ax.set\_ylabel("y (point)")  
  
 dx = abs(x2 - x1)  
 dy = abs(y2 - y1)  
 x, y = x1, y1  
 sx = 1 if x1 < x2 else -1  
 sy = 1 if y1 < y2 else -1  
  
 if dx > dy:  
 err = dx / 2.0  
 while x != x2:  
 plt.plot(x, y, color)  
 err -= dy  
 if err < 0:  
 y += sy  
 err += dx  
 x += sx  
 else:  
 err = dy / 2.0  
 while y != y2:  
 plt.plot(x, y, color)  
 err -= dx  
 if err < 0:  
 x += sx  
 err += dy  
 y += sy  
  
 plt.axis('square')  
def submit\_castle\_pitway\_line(x, y, x\_end, y\_end):  
 def clicked(event):  
 ax2 = plt.subplot(2, 4, 5)  
 ax2.cla()  
 draw\_castle\_pitway\_line(int(x.val), int(y.val), int(x\_end.val), int(y\_end.val))  
 plt.draw()  
  
 return clicked  
  
def main():  
 plt.figure("Базовые растровые алгоритмы", figsize=(16, 8), layout="constrained")  
 plt.subplots\_adjust(left=0.1, right=0.9)  
  
 # input to DDA  
 ax = plt.axes([0.05, 0.35, 0.15, 0.04])  
 dda\_box\_x1 = Slider(ax, label='x1', valinit=100, valmin=0, valmax=500, valstep=1)  
 ax = plt.axes([0.05, 0.3, 0.15, 0.04])  
 dda\_box\_y1 = Slider(ax, label='y1', valinit=0, valmin=0, valmax=500, valstep=1)  
 ax = plt.axes([0.05, 0.25, 0.15, 0.04])  
 dda\_box\_x2 = Slider(ax, label='x2', valinit=0, valmin=0, valmax=500, valstep=1)  
 ax = plt.axes([0.05, 0.2, 0.15, 0.04])  
 dda\_box\_y2 = Slider(ax, label='y2', valinit=100, valmin=0, valmax=500, valstep=1)  
  
 ax = plt.axes([0.1, 0.1, 0.1, 0.04])  
 button\_dda\_line = Button(ax, 'Обновить')  
 button\_dda\_line.on\_clicked(submit\_dda\_line(dda\_box\_x1, dda\_box\_y1, dda\_box\_x2, dda\_box\_y2))  
  
 # input to Steps  
 ax = plt.axes([0.3, 0.35, 0.15, 0.04])  
 steps\_box\_x = Slider(ax, label='x', valinit=0, valmin=0, valmax=500, valstep=1)  
 ax = plt.axes([0.3, 0.30, 0.15, 0.04])  
 steps\_box\_k = Slider(ax, label='k', valinit=-1, valmin=-10, valmax=10, valstep=0.1)  
 ax = plt.axes([0.3, 0.25, 0.15, 0.04])  
 steps\_box\_b = Slider(ax, label='b', valinit=100, valmin=-250, valmax=250, valstep=1)  
 ax = plt.axes([0.3, 0.2, 0.15, 0.04])  
 steps\_box\_steps = Slider(ax, label='Steps', valinit=100, valmin=0, valmax=500, valstep=1)  
  
 ax = plt.axes([0.35, 0.1, 0.1, 0.04])  
 button\_steps\_line = Button(ax, 'Обновить')  
 button\_steps\_line.on\_clicked(submit\_steps\_line(steps\_box\_x, steps\_box\_k, steps\_box\_b, steps\_box\_steps))  
  
 # input to Brezenhem circle  
 ax = plt.axes([0.55, 0.35, 0.15, 0.04])  
 circle\_box\_x = Slider(ax, label='x1', valinit=100, valmin=0, valmax=500, valstep=1)  
 ax = plt.axes([0.55, 0.30, 0.15, 0.04])  
 circle\_box\_y = Slider(ax, label='y1', valinit=0, valmin=0, valmax=500, valstep=1)  
 ax = plt.axes([0.55, 0.25, 0.15, 0.04])  
 circle\_box\_x\_end = Slider(ax, label='x2', valinit=0, valmin=0, valmax=500, valstep=1)  
 ax = plt.axes([0.55, 0.2, 0.15, 0.04])  
 circle\_box\_y\_end = Slider(ax, label='y2', valinit=100, valmin=0, valmax=500, valstep=1)  
  
 ax = plt.axes([0.6, 0.1, 0.1, 0.04])  
 button\_b\_circle = Button(ax, 'Обновить')  
 button\_b\_circle.on\_clicked(submit\_b\_circle(circle\_box\_x, circle\_box\_y, circle\_box\_x\_end, circle\_box\_y\_end))  
  
 # input to Brezenhem line  
 ax = plt.axes([0.8, 0.35, 0.15, 0.04])  
 line\_box\_x1 = Slider(ax, label='x1', valinit=100, valmin=0, valmax=500, valstep=1)  
 ax = plt.axes([0.8, 0.30, 0.15, 0.04])  
 line\_box\_y1 = Slider(ax, label='y1', valinit=0, valmin=0, valmax=500, valstep=1)  
 ax = plt.axes([0.8, 0.25, 0.15, 0.04])  
 line\_box\_x2 = Slider(ax, label='x2', valinit=0, valmin=0, valmax=500, valstep=1)  
 ax = plt.axes([0.8, 0.2, 0.15, 0.04])  
 line\_box\_y2 = Slider(ax, label='y2', valinit=100, valmin=0, valmax=500, valstep=1)  
  
 ax = plt.axes([0.85, 0.1, 0.1, 0.04])  
 button\_b\_line = Button(ax, 'Обновить')  
 button\_b\_line.on\_clicked(submit\_b\_line(line\_box\_x1, line\_box\_y1, line\_box\_x2, line\_box\_y2))  
  
 # DDA line  
 draw\_dda\_line(int(dda\_box\_x1.val), int(dda\_box\_y1.val), int(dda\_box\_x2.val), int(dda\_box\_y2.val))  
 # Steps line  
 draw\_steps\_line(int(steps\_box\_x.val), int(steps\_box\_k.val), int(steps\_box\_b.val), int(steps\_box\_steps.val))  
 # Brezenhem circle  
 draw\_b\_circle(int(circle\_box\_x.val), int(circle\_box\_y.val), int(circle\_box\_x\_end.val),  
 int(circle\_box\_y\_end.val))  
 # Brezenhem line  
 draw\_b\_line(int(line\_box\_x1.val), int(line\_box\_y1.val), int(line\_box\_x2.val), int(line\_box\_y2.val))  
  
 plt.show()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()