**`Минобрнауки России**

**Юго-Западный государственный университет**

**Кафедра программной инженерии**

ЛАБОРАТОРНАЯ РАБОТА

по дисциплине «Компьютерная графика»

по теме:

«Растеризация отрезков прямых»

Выполнили: студенты группы ПО-31б

Василевский В.Я.

Пирогов Д.Ю.

Проверил: старший преподаватель

Ефремов В.В.

Курск 2025 г.

**Вариант – 4**

**Цель работы**: изучение алгоритмов растеризации отрезков, создание программы для визуализации работы алгоритмов.

**Задание:**

1. Разработать программу, реализующую алгоритмы растеризации отрезков с их последующей прорисовкой. Вывести рисунок в соответствии с вариантом. Для вывода отрезков использовать:

* алгоритм ЦДА; − алгоритм Брезенхема;
* целочисленный алгоритм Брезенхема;
* встроенные средства языка программирования.

1. Проиллюстрировать разницу между результатами или её отсутствие.
2. Ответить на контрольные вопросы.
3. Оформить отчёт.

Вариант:



**Ссылка на репозиторий:**https://github.com/Vados-rb26/CompGraphLabs.git

**Программа на python:**

import tkinter as tk  
from tkinter import messagebox, filedialog  
from math import sqrt, floor  
from PIL import Image, ImageDraw  
import re  
  
  
def sign(x):  
 if x > 0:  
 return 1  
 elif x < 0:  
 return -1  
 else:  
 return 0  
  
  
def DigitalDifferentialAnalyzer(x1, y1, x2, y2, Canvas, color):  
 if abs(x1 - x2) < 0.1 and abs(y1 - y2) < 0.1:  
 Canvas.create\_rectangle(x1, y1, x1 + 1, y1 + 1, fill=color, outline=color)  
 return  
  
 dx = x2 - x1  
 dy = y2 - y1  
  
 if abs(dx) >= abs(dy):  
 l = abs(dx)  
 else:  
 l = abs(dy)  
  
 if l < 0.1:  
 return  
  
 dx = dx / l  
 dy = dy / l  
  
 x = x1 + 0.5 \* sign(dx)  
 y = y1 + 0.5 \* sign(dy)  
  
 for i in range(int(l) + 1):  
 Canvas.create\_rectangle(floor(x), floor(y), floor(x) + 1, floor(y) + 1, fill=color, outline=color)  
 x = x + dx  
 y = y + dy  
  
  
def BrezenhemFloat(x1, y1, x2, y2, Canvas, color):  
 if abs(x1 - x2) < 0.1 and abs(y1 - y2) < 0.1:  
 Canvas.create\_rectangle(x1, y1, x1 + 1, y1 + 1, fill=color, outline=color)  
 return  
  
 sx = sign(x2 - x1)  
 sy = sign(y2 - y1)  
  
 dx = abs(x2 - x1)  
 dy = abs(y2 - y1)  
  
 x = x1  
 y = y1  
 flag = 0  
  
 if dy > dx:  
 temp = dx  
 dx = dy  
 dy = temp  
 flag = 1  
  
 if dx < 0.1:  
 return  
  
 f = dy / dx - 0.5  
  
 steps = int(dx) + 1  
  
 for i in range(steps):  
 Canvas.create\_rectangle(int(x), int(y), int(x) + 1, int(y) + 1, fill=color, outline=color)  
  
 if f >= 0:  
 if flag == 1:  
 x = x + sx  
 else:  
 y = y + sy  
 f = f - 1  
  
 if flag == 1:  
 y = y + sy  
 else:  
 x = x + sx  
  
 f = f + dy / dx  
  
  
def BrezenhemInteger(x1, y1, x2, y2, Canvas, color):  
 x1, y1, x2, y2 = int(round(x1)), int(round(y1)), int(round(x2)), int(round(y2))  
  
 if x1 == x2 and y1 == y2:  
 Canvas.create\_rectangle(x1, y1, x1 + 1, y1 + 1, fill=color, outline=color)  
 return  
  
 sx = sign(x2 - x1)  
 sy = sign(y2 - y1)  
  
 dx = abs(x2 - x1)  
 dy = abs(y2 - y1)  
  
 x = x1  
 y = y1  
 flag = 0  
  
 if dy > dx:  
 temp = dx  
 dx = dy  
 dy = temp  
 flag = 1  
  
 if dx == 0:  
 return  
  
 f = 2 \* dy - dx  
  
 for i in range(dx + 1):  
 Canvas.create\_rectangle(x, y, x + 1, y + 1, fill=color, outline=color)  
  
 if f >= 0:  
 if flag == 1:  
 x = x + sx  
 else:  
 y = y + sy  
 f = f - 2 \* dx  
  
 if flag == 1:  
 y = y + sy  
 else:  
 x = x + sx  
  
 f = f + 2 \* dy  
  
  
def DrawLineBuiltin(x1, y1, x2, y2, Canvas, color):  
 Canvas.create\_line(x1, y1, x2, y2, fill=color, width=1)  
  
  
def DigitalDifferentialAnalyzer\_pil(x1, y1, x2, y2, draw, color):  
 if abs(x1 - x2) < 0.1 and abs(y1 - y2) < 0.1:  
 draw.point((x1, y1), fill=color)  
 return  
  
 dx = x2 - x1  
 dy = y2 - y1  
  
 if abs(dx) >= abs(dy):  
 l = abs(dx)  
 else:  
 l = abs(dy)  
  
 if l < 0.1:  
 return  
  
 dx = dx / l  
 dy = dy / l  
  
 x = x1 + 0.5 \* sign(dx)  
 y = y1 + 0.5 \* sign(dy)  
  
 for i in range(int(l) + 1):  
 draw.point((floor(x), floor(y)), fill=color)  
 x = x + dx  
 y = y + dy  
  
  
def BrezenhemFloat\_pil(x1, y1, x2, y2, draw, color):  
 if abs(x1 - x2) < 0.1 and abs(y1 - y2) < 0.1:  
 draw.point((x1, y1), fill=color)  
 return  
  
 sx = sign(x2 - x1)  
 sy = sign(y2 - y1)  
  
 dx = abs(x2 - x1)  
 dy = abs(y2 - y1)  
  
 x = x1  
 y = y1  
 flag = 0  
  
 if dy > dx:  
 temp = dx  
 dx = dy  
 dy = temp  
 flag = 1  
  
 if dx < 0.1:  
 return  
  
 f = dy / dx - 0.5  
  
 steps = int(dx) + 1  
  
 for i in range(steps):  
 draw.point((int(x), int(y)), fill=color)  
  
 if f >= 0:  
 if flag == 1:  
 x = x + sx  
 else:  
 y = y + sy  
 f = f - 1  
  
 if flag == 1:  
 y = y + sy  
 else:  
 x = x + sx  
  
 f = f + dy / dx  
  
  
def BrezenhemInteger\_pil(x1, y1, x2, y2, draw, color):  
 x1, y1, x2, y2 = int(round(x1)), int(round(y1)), int(round(x2)), int(round(y2))  
  
 if x1 == x2 and y1 == y2:  
 draw.point((x1, y1), fill=color)  
 return  
  
 sx = sign(x2 - x1)  
 sy = sign(y2 - y1)  
  
 dx = abs(x2 - x1)  
 dy = abs(y2 - y1)  
  
 x = x1  
 y = y1  
 flag = 0  
  
 if dy > dx:  
 temp = dx  
 dx = dy  
 dy = temp  
 flag = 1  
  
 if dx == 0:  
 return  
  
 f = 2 \* dy - dx  
  
 for i in range(dx + 1):  
 draw.point((x, y), fill=color)  
  
 if f >= 0:  
 if flag == 1:  
 x = x + sx  
 else:  
 y = y + sy  
 f = f - 2 \* dx  
  
 if flag == 1:  
 y = y + sy  
 else:  
 x = x + sx  
  
 f = f + 2 \* dy  
  
  
def DrawLineBuiltin\_pil(x1, y1, x2, y2, draw, color):  
 draw.line([(x1, y1), (x2, y2)], fill=color, width=1)  
  
  
current\_segments = []   
current\_mode = "rhombus"   
  
  
def ParseSvgFile(file\_path):  
 segments = []  
  
 try:  
 with open(file\_path, 'r', encoding='utf-8') as file:  
 content = file.read()  
  
 line\_pattern = r'<line[^>]\*x1\s\*=\s\*["\']([^"\']\*)["\'][^>]\*y1\s\*=\s\*["\']([^"\']\*)["\'][^>]\*x2\s\*=\s\*["\']([^"\']\*)["\'][^>]\*y2\s\*=\s\*["\']([^"\']\*)["\'][^>]\*>'  
 lines = re.findall(line\_pattern, content, re.IGNORECASE)  
  
 for line in lines:  
 x1, y1, x2, y2 = map(float, line)  
 segments.append((x1, y1, x2, y2))  
  
 if not segments:  
 path\_pattern = r'<path[^>]\*d\s\*=\s\*["\']([^"\']\*)["\'][^>]\*>'  
 paths = re.findall(path\_pattern, content, re.IGNORECASE)  
  
 for path in paths:  
 commands = re.findall(r'[MLml]\s\*([\d\.]+)\s\*([\d\.]+)', path)  
 if len(commands) >= 2:  
 for i in range(len(commands) - 1):  
 x1, y1 = map(float, commands[i])  
 x2, y2 = map(float, commands[i + 1])  
 segments.append((x1, y1, x2, y2))  
  
 return segments  
  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Не удалось прочитать SVG файл: {str(e)}")  
 return []  
  
  
def SaveAsPbm(image, file\_path):  
 try:  
 bw\_image = image.convert('1')  
 width, height = bw\_image.size  
  
 with open(file\_path, 'w') as f:  
 f.write("P1\n")  
 f.write(f"# Created by Line Rasterization App\n")  
 f.write(f"{width} {height}\n")  
  
 pixels = bw\_image.load()  
 for y in range(height):  
 line = []  
 for x in range(width):  
 pixel = 1 if pixels[x, y] == 255 else 0  
 line.append(str(pixel))  
 f.write(" ".join(line) + "\n")  
  
 return True  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Не удалось сохранить PBM файл: {str(e)}")  
 return False  
  
  
def DrawSegmentsOnCanvas(segments, algorithm\_func, color, offset\_x=0, offset\_y=0):  
 for segment in segments:  
 x1, y1, x2, y2 = segment  
 shifted\_x1 = x1 + offset\_x  
 shifted\_y1 = y1 + offset\_y  
 shifted\_x2 = x2 + offset\_x  
 shifted\_y2 = y2 + offset\_y  
 algorithm\_func(shifted\_x1, shifted\_y1, shifted\_x2, shifted\_y2, Canvas, color)  
  
  
def DrawRhombusOnCanvas(x1, y1, x2, y2, other\_diag\_len, algorithm\_func, color, offset\_x=0, offset\_y=0):  
 shifted\_x1 = x1 + offset\_x  
 shifted\_y1 = y1 + offset\_y  
 shifted\_x2 = x2 + offset\_x  
 shifted\_y2 = y2 + offset\_y  
  
 center\_x = (shifted\_x1 + shifted\_x2) / 2  
 center\_y = (shifted\_y1 + shifted\_y2) / 2  
  
 dx = shifted\_x2 - shifted\_x1  
 dy = shifted\_y2 - shifted\_y1  
 diag1\_len = sqrt(dx \* dx + dy \* dy)  
  
 if diag1\_len < 0.1:  
 return  
  
 perp\_dx = -dy  
 perp\_dy = dx  
 perp\_len = sqrt(perp\_dx \* perp\_dx + perp\_dy \* perp\_dy)  
  
 if perp\_len < 0.1:  
 return  
  
 perp\_dx = perp\_dx / perp\_len \* other\_diag\_len / 2  
 perp\_dy = perp\_dy / perp\_len \* other\_diag\_len / 2  
  
 x3 = center\_x + perp\_dx  
 y3 = center\_y + perp\_dy  
 x4 = center\_x - perp\_dx  
 y4 = center\_y - perp\_dy  
  
 algorithm\_func(shifted\_x1, shifted\_y1, x3, y3, Canvas, color)  
 algorithm\_func(x3, y3, shifted\_x2, shifted\_y2, Canvas, color)  
 algorithm\_func(shifted\_x2, shifted\_y2, x4, y4, Canvas, color)  
 algorithm\_func(x4, y4, shifted\_x1, shifted\_y1, Canvas, color)  
  
  
def LoadSvg():  
 global current\_segments, current\_mode  
  
 file\_path = filedialog.askopenfilename(  
 filetypes=[("SVG files", "\*.svg"), ("All files", "\*.\*")]  
 )  
  
 if not file\_path:  
 return  
  
 segments = ParseSvgFile(file\_path)  
  
 if not segments:  
 messagebox.showwarning("Предупреждение", "Не удалось найти отрезки в SVG файле")  
 return  
  
 current\_segments = segments  
 current\_mode = "svg"  
  
 Canvas.delete("all")  
  
 offsets = [(0, 0), (15, 0), (0, 15), (15, 15)]  
 algorithms = [  
 ("ЦДА", "blue", DigitalDifferentialAnalyzer),  
 ("Брезенхем (вещ.)", "red", BrezenhemFloat),  
 ("Брезенхем (цел.)", "green", BrezenhemInteger),  
 ("Встроенный", "black", DrawLineBuiltin)  
 ]  
  
 for i, (offset\_x, offset\_y) in enumerate(offsets):  
 algorithm\_name, color, algorithm\_func = algorithms[i]  
 DrawSegmentsOnCanvas(segments, algorithm\_func, color, offset\_x, offset\_y)  
 Canvas.create\_text(500, 20 + i \* 20, text=algorithm\_name, fill=color, font=("Arial", 10))  
  
  
def CreateRhombus():  
 global current\_segments, current\_mode  
  
 try:  
 x1 = float(entry\_x1.get())  
 y1 = float(entry\_y1.get())  
 x2 = float(entry\_x2.get())  
 y2 = float(entry\_y2.get())  
 other\_diag\_len = float(entry\_len.get())  
  
 if other\_diag\_len <= 0:  
 messagebox.showerror("Ошибка", "Длина диагонали должна быть положительной")  
 return  
  
 current\_segments = [(x1, y1, x2, y2, other\_diag\_len)] # храним параметры ромба  
 current\_mode = "rhombus"  
  
 Canvas.delete("all")  
  
 offsets = [(0, 0), (15, 0), (0, 15), (15, 15)]  
 algorithms = [  
 ("ЦДА", "blue", DigitalDifferentialAnalyzer),  
 ("Брезенхем (вещ.)", "red", BrezenhemFloat),  
 ("Брезенхем (цел.)", "green", BrezenhemInteger),  
 ("Встроенный", "black", DrawLineBuiltin)  
 ]  
  
 for i, (offset\_x, offset\_y) in enumerate(offsets):  
 algorithm\_name, color, algorithm\_func = algorithms[i]  
 DrawRhombusOnCanvas(x1, y1, x2, y2, other\_diag\_len, algorithm\_func, color, offset\_x, offset\_y)  
 Canvas.create\_text(500, 20 + i \* 20, text=algorithm\_name, fill=color, font=("Arial", 10))  
  
 except ValueError:  
 messagebox.showerror("Ошибка", "Пожалуйста, введите корректные числовые значения")  
  
  
def SaveToPbm():  
 global current\_segments, current\_mode  
  
 file\_path = filedialog.asksaveasfilename(  
 defaultextension=".pbm",  
 filetypes=[("PBM files", "\*.pbm"), ("All files", "\*.\*")]  
 )  
  
 if not file\_path:  
 return  
  
 try:  
 image = Image.new('RGB', (600, 400), 'white')  
 draw = ImageDraw.Draw(image)  
  
 color\_pbm = (0, 0, 0)  
  
 algorithm\_func = DigitalDifferentialAnalyzer\_pil  
  
 if current\_mode == "rhombus" and current\_segments:  
 x1, y1, x2, y2, other\_diag\_len = current\_segments[0]  
 DrawRhombusOnImage(image, draw, x1, y1, x2, y2, other\_diag\_len, algorithm\_func, color\_pbm)  
  
 elif current\_mode == "svg" and current\_segments:  
 DrawSegmentsOnImage(image, draw, current\_segments, algorithm\_func, color\_pbm)  
  
 if SaveAsPbm(image, file\_path):  
 messagebox.showinfo("Успех", f"Изображение сохранено как {file\_path}")  
  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Не удалось сохранить изображение: {str(e)}")  
  
  
def DrawSegmentsOnImage(image, draw, segments, algorithm\_func, color):  
 for segment in segments:  
 x1, y1, x2, y2 = segment  
 algorithm\_func(x1, y1, x2, y2, draw, color)  
  
  
def DrawRhombusOnImage(image, draw, x1, y1, x2, y2, other\_diag\_len, algorithm\_func, color):  
 center\_x = (x1 + x2) / 2  
 center\_y = (y1 + y2) / 2  
  
 dx = x2 - x1  
 dy = y2 - y1  
 diag1\_len = sqrt(dx \* dx + dy \* dy)  
  
 if diag1\_len < 0.1:  
 return  
  
 perp\_dx = -dy  
 perp\_dy = dx  
 perp\_len = sqrt(perp\_dx \* perp\_dx + perp\_dy \* perp\_dy)  
  
 if perp\_len < 0.1:  
 return  
  
 perp\_dx = perp\_dx / perp\_len \* other\_diag\_len / 2  
 perp\_dy = perp\_dy / perp\_len \* other\_diag\_len / 2  
  
 x3 = center\_x + perp\_dx  
 y3 = center\_y + perp\_dy  
 x4 = center\_x - perp\_dx  
 y4 = center\_y - perp\_dy  
  
 algorithm\_func(x1, y1, x3, y3, draw, color)  
 algorithm\_func(x3, y3, x2, y2, draw, color)  
 algorithm\_func(x2, y2, x4, y4, draw, color)  
 algorithm\_func(x4, y4, x1, y1, draw, color)  
  
  
root = tk.Tk()  
root.title("Растеризация отрезков прямых")  
root.geometry("600x500")  
  
title\_label = tk.Label(root, text="Растеризация отрезков прямых (вариант с ромбами)", font=("Arial", 14, "bold"))  
title\_label.pack(pady=5)  
  
input\_frame = tk.Frame(root)  
input\_frame.pack(pady=10)  
  
tk.Label(input\_frame, text="x1").grid(row=0, column=0, padx=5, sticky="e")  
entry\_x1 = tk.Entry(input\_frame, width=8)  
entry\_x1.grid(row=0, column=1, padx=5)  
entry\_x1.insert(0, "200")  
  
tk.Label(input\_frame, text="y1").grid(row=0, column=2, padx=5, sticky="e")  
entry\_y1 = tk.Entry(input\_frame, width=8)  
entry\_y1.grid(row=0, column=3, padx=5)  
entry\_y1.insert(0, "150")  
  
tk.Label(input\_frame, text="x2").grid(row=1, column=0, padx=5, sticky="e")  
entry\_x2 = tk.Entry(input\_frame, width=8)  
entry\_x2.grid(row=1, column=1, padx=5)  
entry\_x2.insert(0, "400")  
  
tk.Label(input\_frame, text="y2").grid(row=1, column=2, padx=5, sticky="e")  
entry\_y2 = tk.Entry(input\_frame, width=8)  
entry\_y2.grid(row=1, column=3, padx=5)  
entry\_y2.insert(0, "150")  
  
tk.Label(input\_frame, text="Длина второй диагонали").grid(row=2, column=0, columnspan=2, padx=5, pady=5, sticky="e")  
entry\_len = tk.Entry(input\_frame, width=8)  
entry\_len.grid(row=2, column=2, padx=5)  
entry\_len.insert(0, "200")  
  
button\_frame = tk.Frame(root)  
button\_frame.pack(pady=10)  
  
btn\_create = tk.Button(button\_frame, text="Создать ромб", command=CreateRhombus, width=12, height=1)  
btn\_create.pack(side=tk.LEFT, padx=5)  
  
btn\_LoadSvg = tk.Button(button\_frame, text="Загрузить SVG", command=LoadSvg, width=12, height=1)  
btn\_LoadSvg.pack(side=tk.LEFT, padx=5)  
  
btn\_save\_pbm = tk.Button(button\_frame, text="Сохранить PBM", command=SaveToPbm, width=12, height=1)  
btn\_save\_pbm.pack(side=tk.LEFT, padx=5)  
  
Canvas = tk.Canvas(root, width=600, height=400, bg="white", relief="solid", bd=1)  
Canvas.pack(pady=10)  
  
root.mainloop()

**Порядок работы:  
1.Запуск приложения**

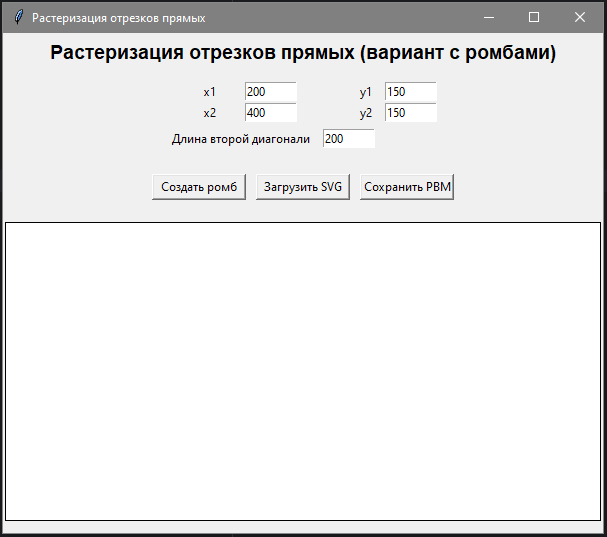
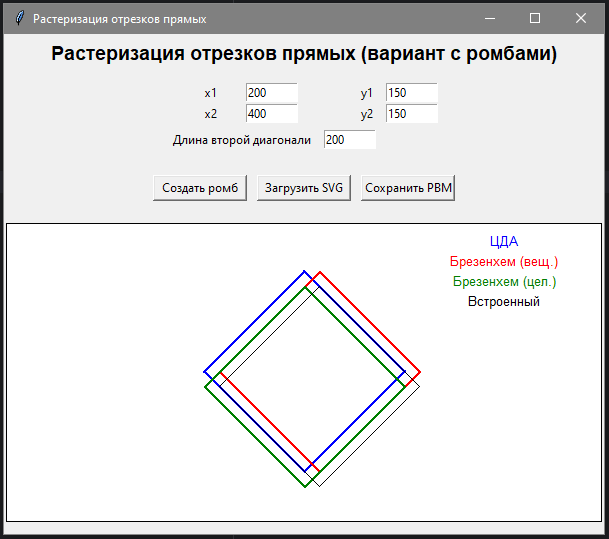
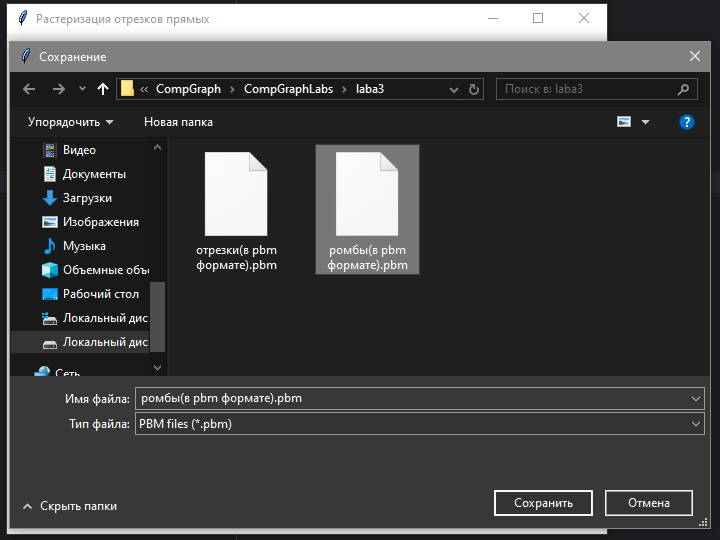


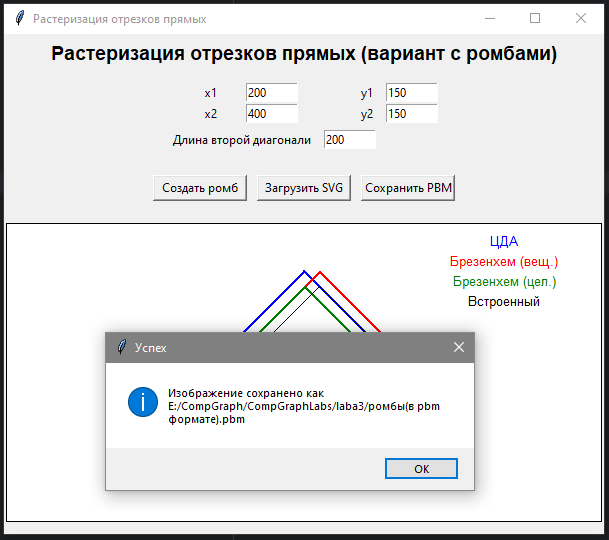
Рис.1 – Запуск приложения

**2.Нажатие на кнопку «Создать» с указанными точками**

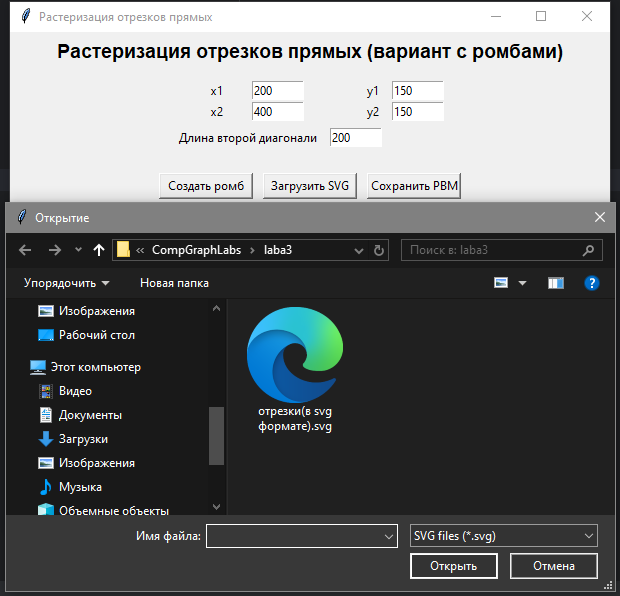
****Рис.2 – Создание ромбов в соответствии с алгоритмами

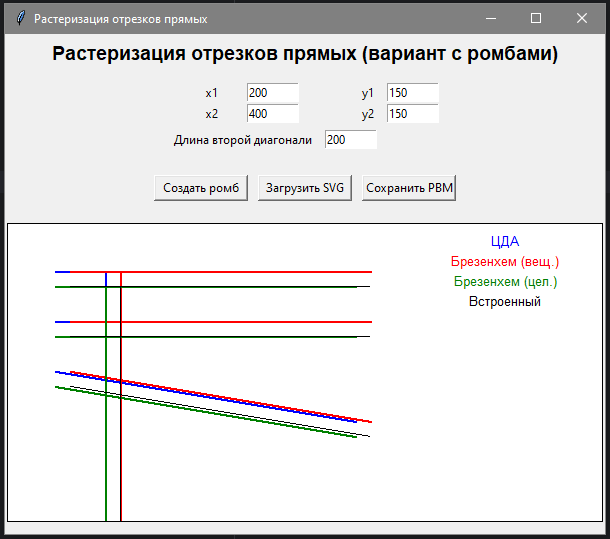
**3.Нажатие на кнопку «Сохранить pbm»**

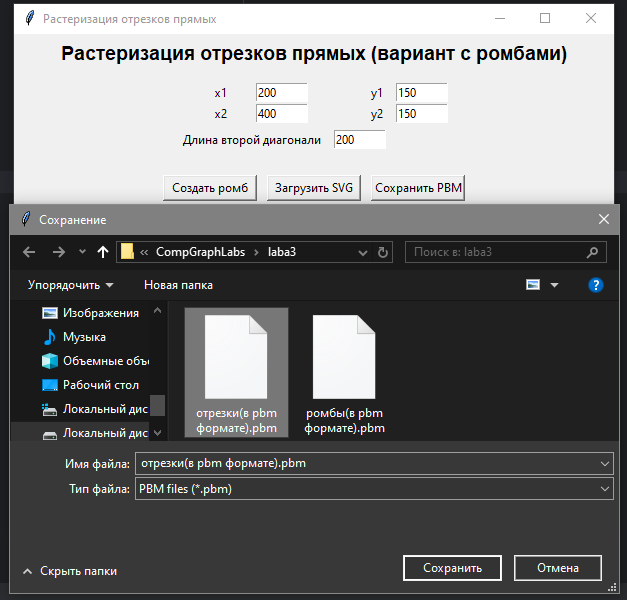
****

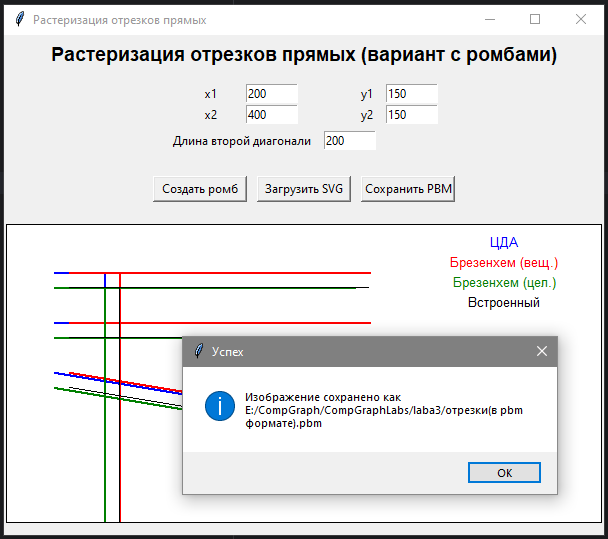
****

**4.Нажатие на кнопку «Загрузить SVG»**

****

****

**5.Сохранение отрезков в pbm формате  
**

****

**Ответы на контрольные вопросы**

**Отвечаю на контрольные вопросы:**

**1. Что такое пиксель**?

Пиксель - это наименьший логический элемент растрового изображения. Каждый пиксель:

- Имеет определенный цвет и яркость

- Занимает определенную позицию в растровой сетке

- Является неделимой единицей изображения на экране

Например, изображение разрешением 1920×1080 состоит из 2,073,600 пикселей.

**2. Что такое растровое изображение?**

Растровое изображение - это изображение, представленное в виде прямоугольной матрицы пикселей. Характеристики:

- Состоит из сетки пикселей фиксированного размера

- Каждый пиксель хранит информацию о цвете

- Имеет фиксированное разрешение (количество пикселей по ширине и высоте)

- При увеличении теряет качество (появляется "пикселизация")

Примеры форматов: JPEG, PNG, BMP, GIF, PBM.

**3. Для чего нужны алгоритмы растеризации отрезков?**

Алгоритмы растеризации отрезков нужны для:

Основная задача: Преобразование математического описания отрезка (координаты концов) в набор пикселей на растровой сетке.

Конкретные применения:

- Отображение векторной графики на растровых дисплеях

- Компьютерная графика и визуализация

- CAD-системы (чертежи, схемы)

- Компьютерные игры

- GIS-системы (географические информационные системы)

- Шрифты и текст на экране

Цели алгоритмов:

- Максимально точно приблизить отрезок к идеальной прямой

- Минимизировать вычислительную сложность

- Обеспечить равномерную толщину линии

- Избегать "ступенчатости" (анти-алиасинг)

**4. Какие ещё алгоритмы растеризации отрезков существуют?**

Основные алгоритмы:

1. ЦДА (Цифровой Дифференциальный Анализатор) - простейший, но с вещественной арифметикой

2. Алгоритм Брезенхема - целочисленный, наиболее популярный

3. Алгоритм Ву (Xiaolin Wu's algorithm) - с анти-алиасингом (сглаживание)

Специализированные алгоритмы:

4. Алгоритм Мичнера - улучшенная версия Брезенхема

5. Алгоритм Кастрова-Питтвея - для толстых линий

6. Алгоритм Гупта-Спролла - высококачественное сглаживание

По типам обрабатываемых примитивов:

- Алгоритмы для окружностей и эллипсов (также Брезенхема)

- Алгоритмы для кривых Безье

- Алгоритмы для полигонов и заливки

Современные подходы:

- Аппаратная растеризация (GPU)

- Алгоритмы с использованием шейдеров

- Растеризация с субпиксельным сглаживанием