**Минобрнауки России**

**Юго-Западный государственный университет**

Кафедра программной инженерии

Лабораторная работа №3

по дисциплине «Компьютерная графика»

по теме:

«Растеризация отрезков прямых»

Выполнили: студенты группы ПО-32б

Разгуляев Е. О., Норов М. С.

Проверил: ст. преподаватель, зав. лабораториями

Ефремов В. В.

Курск 2025 г.

**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.

2. Проиллюстрировать разницу между результатами или её отсутствие.

3. Ответить на контрольные вопросы.

4. Оформить отчёт.

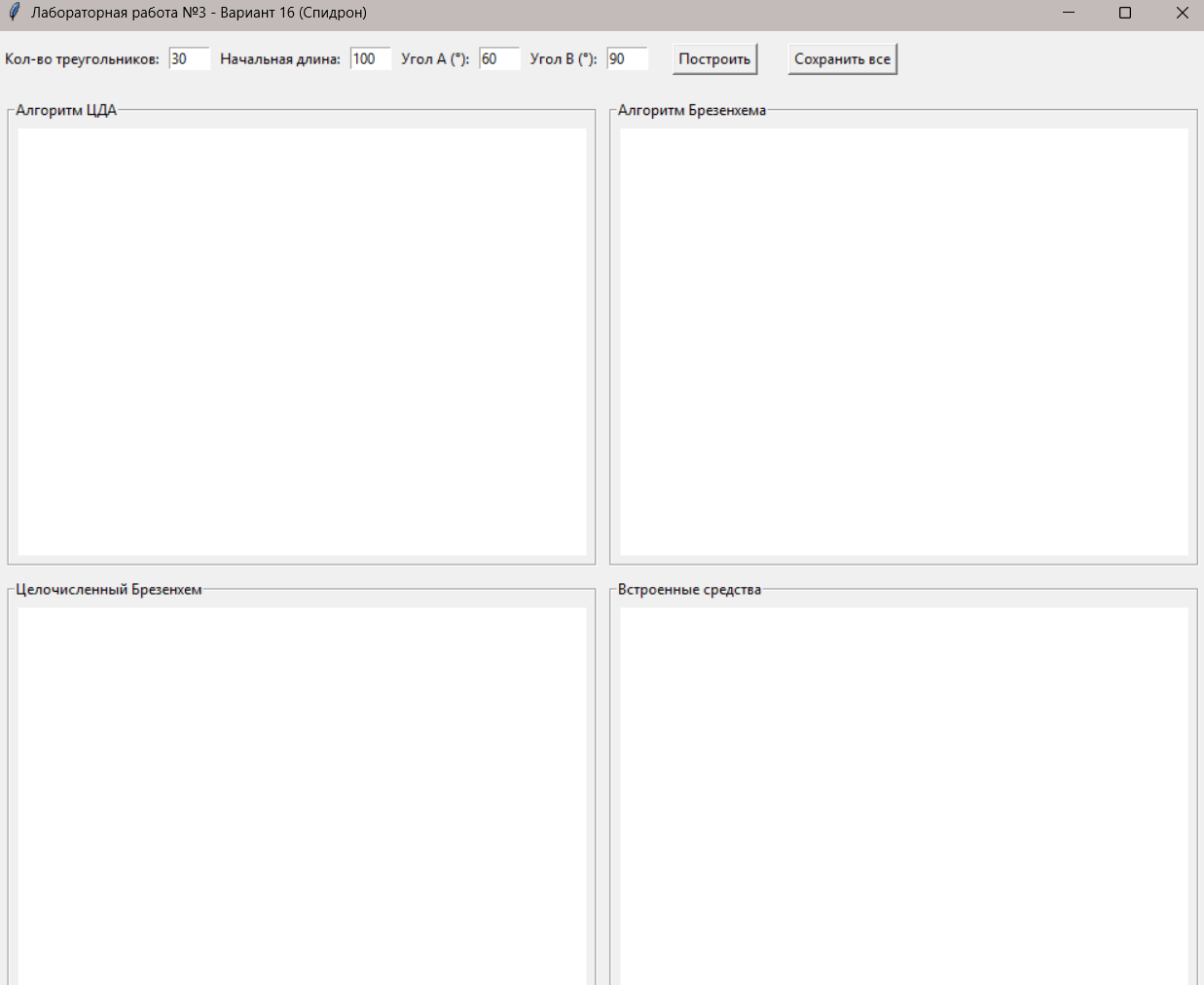
**ХОД РАБОТЫ**

**Вариант 16**

По двум длинам сторон нарисовать спидрон.

[isami972/CG-Razgulyaev-Norov-LR--3](https://github.com/isami972/CG-Razgulyaev-Norov-LR--3)

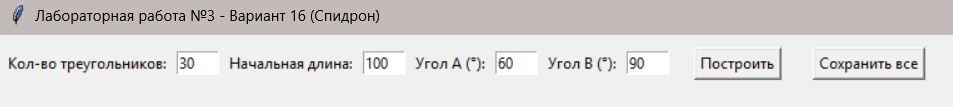
1. Форма:

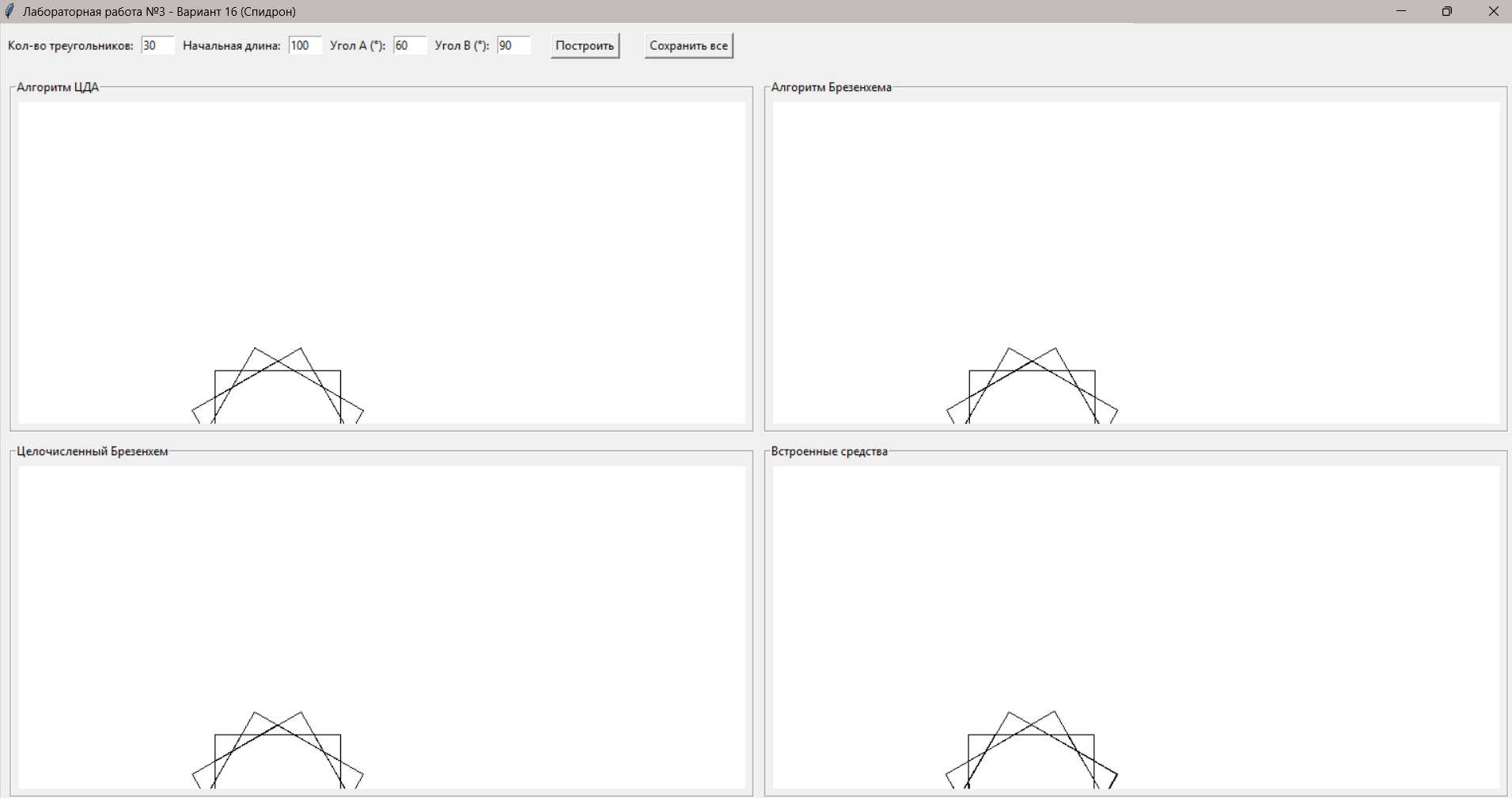


2. Текст программы (весь модуль):

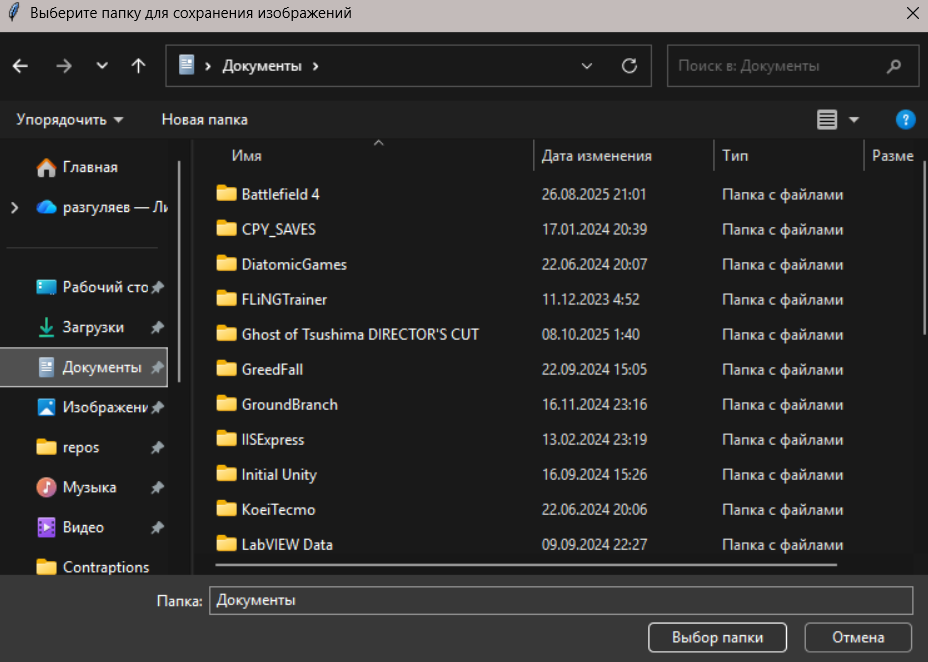
# --- Блок для автоматической установки модулей ---  
import sys  
import subprocess  
try:  
 from PIL import Image, ImageDraw, ImageTk  
except ImportError:  
 print("Библиотека Pillow не найдена. Выполняется установка...")  
 try:  
 subprocess.check\_call([sys.executable, "-m", "pip", "install", "Pillow"])  
 print("\nБиблиотека Pillow успешно установлена. Пожалуйста, перезапустите скрипт.")  
 except Exception as e:  
 print(f"Ошибка при установке Pillow: {e}")  
 print("Пожалуйста, установите библиотеку вручную командой: pip install Pillow")  
 sys.exit()  
# --------------------------------------------------  
  
import tkinter as tk  
from tkinter import messagebox, filedialog  
import math  
  
class Lab3App:  
 def \_\_init\_\_(self, root):  
 self.root = root  
 self.root.title("Лабораторная работа №3 - Вариант 16 (Спидрон)")  
 self.root.geometry("1000x900")  
  
 self.images = {}  
 self.create\_widgets()  
  
 def create\_widgets(self):  
 input\_frame = tk.Frame(self.root, pady=10)  
 input\_frame.pack(fill=tk.X)  
  
 tk.Label(input\_frame, text="Кол-во треугольников:").pack(side=tk.LEFT, padx=5)  
 self.segments\_var = tk.StringVar(value="30")  
 tk.Entry(input\_frame, textvariable=self.segments\_var, width=5).pack(side=tk.LEFT)  
  
 tk.Label(input\_frame, text="Начальная длина:").pack(side=tk.LEFT, padx=5)  
 self.length\_var = tk.StringVar(value="100")  
 tk.Entry(input\_frame, textvariable=self.length\_var, width=5).pack(side=tk.LEFT)  
  
 tk.Label(input\_frame, text="Угол A (°):").pack(side=tk.LEFT, padx=5)  
 self.angle\_a\_var = tk.StringVar(value="60")  
 tk.Entry(input\_frame, textvariable=self.angle\_a\_var, width=5).pack(side=tk.LEFT)  
  
 tk.Label(input\_frame, text="Угол B (°):").pack(side=tk.LEFT, padx=5)  
 self.angle\_b\_var = tk.StringVar(value="90")  
 tk.Entry(input\_frame, textvariable=self.angle\_b\_var, width=5).pack(side=tk.LEFT)  
  
 build\_btn = tk.Button(input\_frame, text="Построить", command=self.draw\_spidron)  
 build\_btn.pack(side=tk.LEFT, padx=20)  
  
 # Одна кнопка сохранения рядом с кнопкой "Построить"  
 save\_btn = tk.Button(input\_frame, text="Сохранить все", command=self.save\_all\_images)  
 save\_btn.pack(side=tk.LEFT, padx=5)  
  
 canvas\_frame = tk.Frame(self.root)  
 canvas\_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)  
  
 self.canvases = {}  
 self.images = {} # Храним изображения для каждого алгоритма  
 algorithm\_names = {  
 "dda": "Алгоритм ЦДА",  
 "bresenham": "Алгоритм Брезенхема",  
 "int\_bresenham": "Целочисленный Брезенхем",  
 "builtin": "Встроенные средства"  
 }  
  
 for i, (algo\_key, algo\_name) in enumerate(algorithm\_names.items()):  
 frame = tk.LabelFrame(canvas\_frame, text=algo\_name, padx=5, pady=5)  
 frame.grid(row=i // 2, column=i % 2, padx=5, pady=5, sticky="nsew")  
 canvas\_frame.grid\_rowconfigure(i // 2, weight=1)  
 canvas\_frame.grid\_columnconfigure(i % 2, weight=1)  
  
 canvas = tk.Canvas(frame, bg='white')  
 canvas.pack(fill=tk.BOTH, expand=True)  
 self.canvases[algo\_key] = canvas  
  
 def save\_all\_images(self):  
 *"""Сохраняет все изображения алгоритмов"""* if not self.images:  
 messagebox.showwarning("Предупреждение", "Сначала постройте спидрон!")  
 return  
  
 # Спрашиваем папку для сохранения  
 folder\_path = filedialog.askdirectory(title="Выберите папку для сохранения изображений")  
  
 if not folder\_path:  
 return  
  
 try:  
 saved\_files = []  
 for algo\_key, image in self.images.items():  
 filename = f"spidron\_{algo\_key}.png"  
 file\_path = f"{folder\_path}/{filename}"  
 image.save(file\_path)  
 saved\_files.append(filename)  
  
 messagebox.showinfo("Успех", f"Все изображения сохранены в папку:\n{folder\_path}\n\nСохраненные файлы:\n" + "\n".join(saved\_files))  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Не удалось сохранить файлы:\n{e}")  
  
 def dda\_algorithm(self, draw, x1, y1, x2, y2, color="black"):  
 dx = x2 - x1  
 dy = y2 - y1  
 steps = max(abs(dx), abs(dy))  
 if steps == 0:  
 draw.point((x1, y1), fill=color)  
 return  
 x\_inc = dx / steps  
 y\_inc = dy / steps  
 x, y = x1, y1  
 for \_ in range(int(steps) + 1):  
 draw.point((round(x), round(y)), fill=color)  
 x += x\_inc  
 y += y\_inc  
  
 def integer\_bresenham\_algorithm(self, draw, x1, y1, x2, y2, color="black"):  
 x1, y1, x2, y2 = int(round(x1)), int(round(y1)), int(round(x2)), int(round(y2))  
 dx = abs(x2 - x1)  
 dy = abs(y2 - y1)  
 sx = 1 if x1 < x2 else -1  
 sy = 1 if y1 < y2 else -1  
 err = dx - dy  
 while True:  
 draw.point((x1, y1), fill=color)  
 if x1 == x2 and y1 == y2: break  
 e2 = 2 \* err  
 if e2 > -dy:  
 err -= dy  
 x1 += sx  
 if e2 < dx:  
 err += dx  
 y1 += sy  
  
 def bresenham\_algorithm(self, draw, x1, y1, x2, y2, color="black"):  
 self.integer\_bresenham\_algorithm(draw, x1, y1, x2, y2, color)  
  
 def generate\_spidron\_vertices(self, start\_point, num\_triangles, start\_len, angle\_a\_deg, angle\_b\_deg):  
 *"""Генерирует вершины для одного рукава спидрона."""* p0 = start\_point  
 p1 = (start\_point[0] + start\_len, start\_point[1])  
  
 vertices = [p0, p1]  
  
 angle\_a\_rad = math.radians(180 - angle\_a\_deg)  
 angle\_b\_rad = math.radians(180 - angle\_b\_deg)  
  
 for i in range(num\_triangles):  
 last\_p = vertices[-1]  
 prev\_p = vertices[-2]  
  
 # Вектор последнего сегмента  
 vx = prev\_p[0] - last\_p[0]  
 vy = prev\_p[1] - last\_p[1]  
  
 # Выбираем угол поворота в зависимости от четности/нечетности шага  
 turn\_angle = angle\_a\_rad if i % 2 == 0 else angle\_b\_rad  
  
 # Поворачиваем вектор  
 new\_vx = vx \* math.cos(turn\_angle) - vy \* math.sin(turn\_angle)  
 new\_vy = vx \* math.sin(turn\_angle) + vy \* math.cos(turn\_angle)  
  
 # Находим новую точку  
 next\_p = (last\_p[0] + new\_vx, last\_p[1] + new\_vy)  
 vertices.append(next\_p)  
  
 return vertices  
  
 def draw\_spidron(self):  
 try:  
 num\_triangles = int(self.segments\_var.get())  
 start\_len = float(self.length\_var.get())  
 angle\_a = float(self.angle\_a\_var.get())  
 angle\_b = float(self.angle\_b\_var.get())  
 except ValueError:  
 messagebox.showerror("Ошибка", "Параметры должны быть числами!")  
 return  
  
 img\_size = (800, 800)  
 start\_point = (200, 400) # Начальная точка смещена для лучшего вида  
  
 vertices = self.generate\_spidron\_vertices(start\_point, num\_triangles, start\_len, angle\_a, angle\_b)  
  
 for algo, canvas in self.canvases.items():  
 image = Image.new("RGB", img\_size, "white")  
 draw = ImageDraw.Draw(image)  
  
 for i in range(len(vertices) - 1):  
 p1 = vertices[i]  
 p2 = vertices[i+1]  
  
 if algo == "dda":  
 self.dda\_algorithm(draw, p1[0], p1[1], p2[0], p2[1])  
 elif algo == "bresenham":  
 self.bresenham\_algorithm(draw, p1[0], p1[1], p2[0], p2[1])  
 elif algo == "int\_bresenham":  
 self.integer\_bresenham\_algorithm(draw, p1[0], p1[1], p2[0], p2[1])  
 elif algo == "builtin":  
 draw.line([p1, p2], fill="black", width=1)  
  
 # Сохраняем изображение для возможности сохранения  
 self.images[algo] = image  
  
 photo = ImageTk.PhotoImage(image)  
 canvas.delete("all")  
 canvas.create\_image(0, 0, anchor=tk.NW, image=photo)  
 canvas.image = photo  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 root = tk.Tk()  
 app = Lab3App(root)  
 root.mainloop()

3. Указываем параметры



4. Нажимаем построить

5. Сохраняем все полученные результаты в форматах png и pbm:



**ВЫВОД**

В ходе выполненной лабораторной работы мы изучили алгоритмы растеризации отрезков, создали программу для визуализации работы алгоритмов.