**Минобрнауки России**

**Юго-Западный государственный университет**

Кафедра программной инженерии

Лабораторная работа №4

по дисциплине «Компьютерная графика»

по теме:

«Растеризация окружностей»

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 окружностей, создание программы для визуализации работы алгоритмов.

**ЗАДАНИЕ**

1. Написать программу (на языке высокого уровня), реализующую алгоритмы растеризации окружности с их последующей прорисовкой. Координаты центра и радиуса должны задаваться пользователем. Реализовать:

* построение окружности по уравнению окружности;
* построение окружности по параметрическому уравнению;
* алгоритм Брезенхема для построения окружности;
* встроенные средства языка программирования.

2. Проиллюстрировать разницу между результатами или её отсутствие.

3. Ответить на контрольные вопросы.

4. Оформить отчёт.

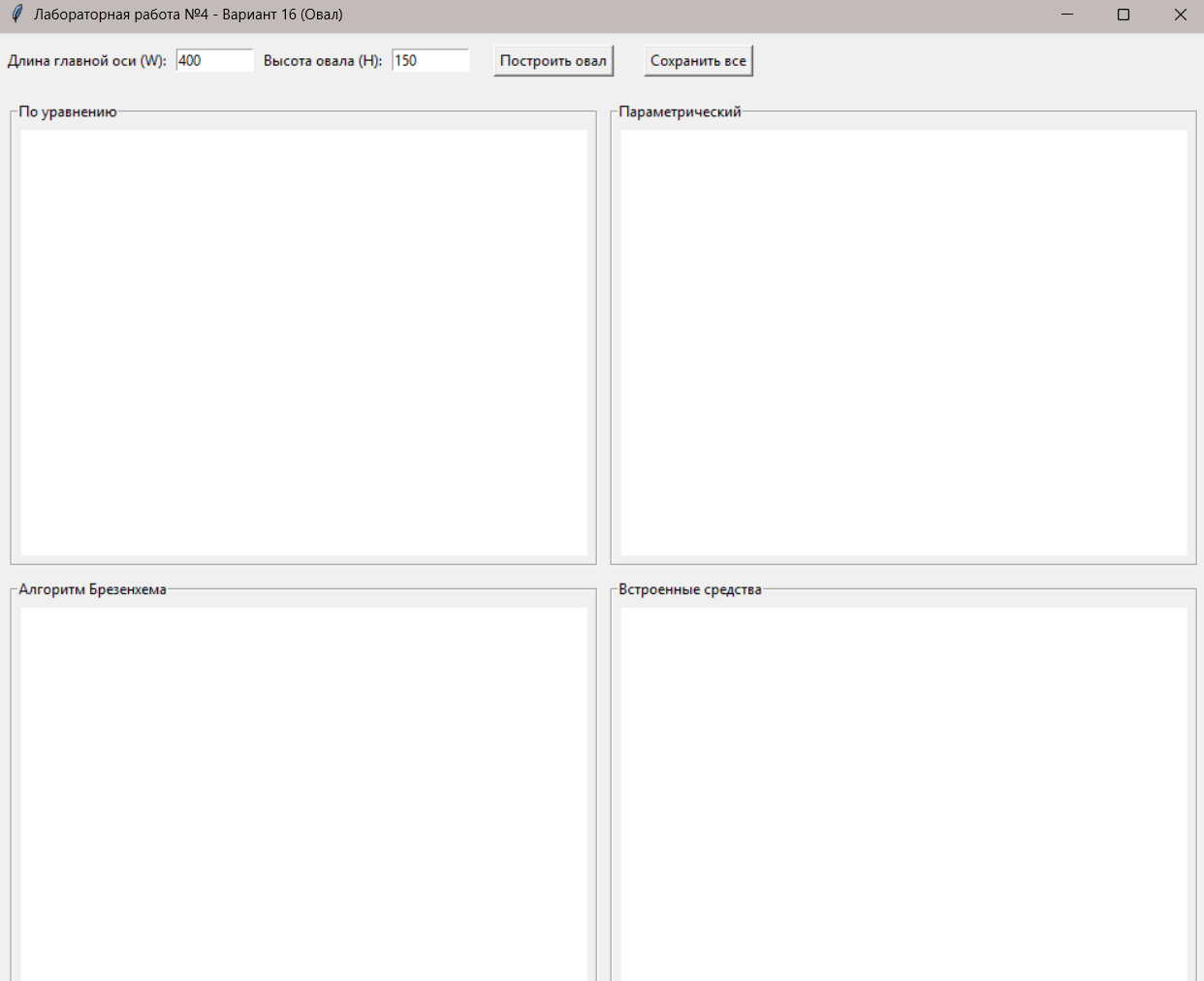
**ХОД РАБОТЫ**

**Ссылка на репозиторий - [isami972/CG-Razgulyaev-Norov-LR-4](https://github.com/isami972/CG-Razgulyaev-Norov-LR-4)**

**Вариант 16**

По радиусу и длине оси вывести овал.

1. Форма:



2. Текст программы (весь модуль):

import tkinter as tk

from tkinter import filedialog, messagebox

import math

# --- Блок для автоматической установки Pillow ---

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()

# -----------------------------------------

class Lab4App:

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

self.root = root

self.root.title("Лабораторная работа №4 - Вариант 16 (Овал)")

self.root.geometry("1000x900")

self.images = {} # Будем хранить все 4 изображения

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="Длина главной оси (W):").pack(side=tk.LEFT, padx=5)

self.axis\_len\_var = tk.StringVar(value="400")

tk.Entry(input\_frame, textvariable=self.axis\_len\_var, width=10).pack(side=tk.LEFT)

tk.Label(input\_frame, text="Высота овала (H):").pack(side=tk.LEFT, padx=5)

self.height\_var = tk.StringVar(value="150")

tk.Entry(input\_frame, textvariable=self.height\_var, width=10).pack(side=tk.LEFT)

build\_btn = tk.Button(input\_frame, text="Построить овал", command=self.draw\_oval)

build\_btn.pack(side=tk.LEFT, padx=20)

# Кнопка сохранения всех 4 изображений

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.algorithm\_names = {

"equation": "По уравнению",

"parametric": "Параметрический",

"bresenham": "Алгоритм Брезенхема",

"builtin": "Встроенные средства"

}

for i, (algo\_key, algo\_name) in enumerate(self.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 4 изображения с разными алгоритмами"""

if not self.images:

messagebox.showwarning("Предупреждение", "Сначала постройте овал!")

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():

algo\_name = self.algorithm\_names[algo\_key].replace(" ", "\_").lower()

filename = f"oval\_{algo\_name}.png"

file\_path = f"{folder\_path}/{filename}"

image.save(file\_path)

saved\_files.append(filename)

messagebox.showinfo("Успех", f"Все 4 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 draw\_arc\_by\_parametric(self, draw, cx, cy, r, start\_angle\_deg, end\_angle\_deg, color="black"):

start\_rad = math.radians(start\_angle\_deg)

end\_rad = math.radians(end\_angle\_deg)

if start\_rad > end\_rad: end\_rad += 2 \* math.pi

step = 0.5 / r if r > 0 else 0.1

angle = start\_rad

while angle <= end\_rad:

x = cx + r \* math.cos(angle)

y = cy - r \* math.sin(angle)

draw.point((round(x), round(y)), fill=color)

angle += step

def draw\_arc\_bresenham(self, draw, cx, cy, r, start\_angle\_deg, end\_angle\_deg, color="black"):

x, y, d = 0, r, 3 - 2 \* r

while x <= y:

points = [(cx+x,cy-y), (cx-x,cy-y), (cx+x,cy+y), (cx-x,cy+y),

(cx+y,cy-x), (cx-y,cy-x), (cx+y,cy+x), (cx-y,cy+x)]

for px, py in points:

angle = math.degrees(math.atan2(cy - py, px - cx))

if angle < 0: angle += 360

if start\_angle\_deg > end\_angle\_deg:

if angle >= start\_angle\_deg or angle <= end\_angle\_deg:

draw.point((px, py), fill=color)

else:

if start\_angle\_deg <= angle <= end\_angle\_deg:

draw.point((px, py), fill=color)

if d < 0: d = d + 4 \* x + 6

else: d, y = d + 4 \* (x - y) + 10, y - 1

x += 1

def draw\_arc\_by\_equation(self, draw, cx, cy, r, start\_angle\_deg, end\_angle\_deg, color="black"):

for x\_offset in range(-r, r + 1):

if r\*r >= x\_offset\*x\_offset:

y\_offset = math.sqrt(r\*r - x\_offset\*x\_offset)

points\_to\_check = [(cx+x\_offset,cy-y\_offset),(cx+x\_offset,cy+y\_offset)]

for x,y in points\_to\_check:

angle = math.degrees(math.atan2(cy-y,x-cx))

if angle<0: angle+=360

if start\_angle\_deg>end\_angle\_deg:

if angle>=start\_angle\_deg or angle<=end\_angle\_deg: draw.point((x,y),fill=color)

else:

if start\_angle\_deg<=angle<=end\_angle\_deg: draw.point((x,y),fill=color)

def draw\_oval(self):

try:

W = int(self.axis\_len\_var.get())

H = int(self.height\_var.get())

if H > W:

messagebox.showerror("Ошибка", "Высота (H) не может быть больше длины (W)!")

return

except ValueError:

messagebox.showerror("Ошибка", "Параметры должны быть целыми числами!")

return

padding = 20

img\_width = W + padding \* 2

img\_height = H + padding \* 2

radius = H / 2

cx\_left = padding + radius

cx\_right = img\_width - padding - radius

cy = img\_height / 2

line\_start\_x = cx\_left

line\_end\_x = cx\_right

line\_top\_y = cy - radius

line\_bottom\_y = cy + radius

# Преобразуем параметры в целые числа один раз

cx\_l, cx\_r, c\_y, r = int(cx\_left), int(cx\_right), int(cy), int(radius)

lsx, lex, lty, lby = int(line\_start\_x), int(line\_end\_x), int(line\_top\_y), int(line\_bottom\_y)

for algo, canvas in self.canvases.items():

image = Image.new("RGB", (img\_width, img\_height), "white")

draw = ImageDraw.Draw(image)

if algo == "builtin":

draw.arc([cx\_l - r, c\_y - r, cx\_l + r, c\_y + r], 90, 270, fill="black", width=1)

draw.arc([cx\_r - r, c\_y - r, cx\_r + r, c\_y + r], 270, 90, fill="black", width=1)

draw.line([(lsx, lty), (lex, lty)], fill="black", width=1)

draw.line([(lsx, lby), (lex, lby)], fill="black", width=1)

elif algo == "parametric":

self.draw\_arc\_by\_parametric(draw, cx\_l, c\_y, r, 90, 270)

self.draw\_arc\_by\_parametric(draw, cx\_r, c\_y, r, 270, 90)

draw.line([(lsx, lty), (lex, lty)], fill="black", width=1)

draw.line([(lsx, lby), (lex, lby)], fill="black", width=1)

elif algo == "bresenham":

self.draw\_arc\_bresenham(draw, cx\_l, c\_y, r, 90, 270)

self.draw\_arc\_bresenham(draw, cx\_r, c\_y, r, 270, 90)

draw.line([(lsx, lty), (lex, lty)], fill="black", width=1)

draw.line([(lsx, lby), (lex, lby)], fill="black", width=1)

elif algo == "equation":

self.draw\_arc\_by\_equation(draw, cx\_l, c\_y, r, 90, 270)

self.draw\_arc\_by\_equation(draw, cx\_r, c\_y, r, 270, 90)

draw.line([(lsx, lty), (lex, lty)], fill="black", width=1)

draw.line([(lsx, lby), (lex, lby)], fill="black", width=1)

# Сохраняем изображение для этого алгоритма

self.images[algo] = image

photo = ImageTk.PhotoImage(image)

canvas.delete("all")

canvas.create\_image(0, 0, anchor=tk.NW, image=photo)

canvas.image = photo

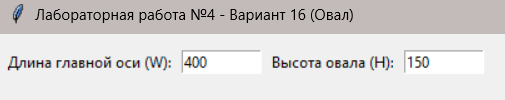
if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

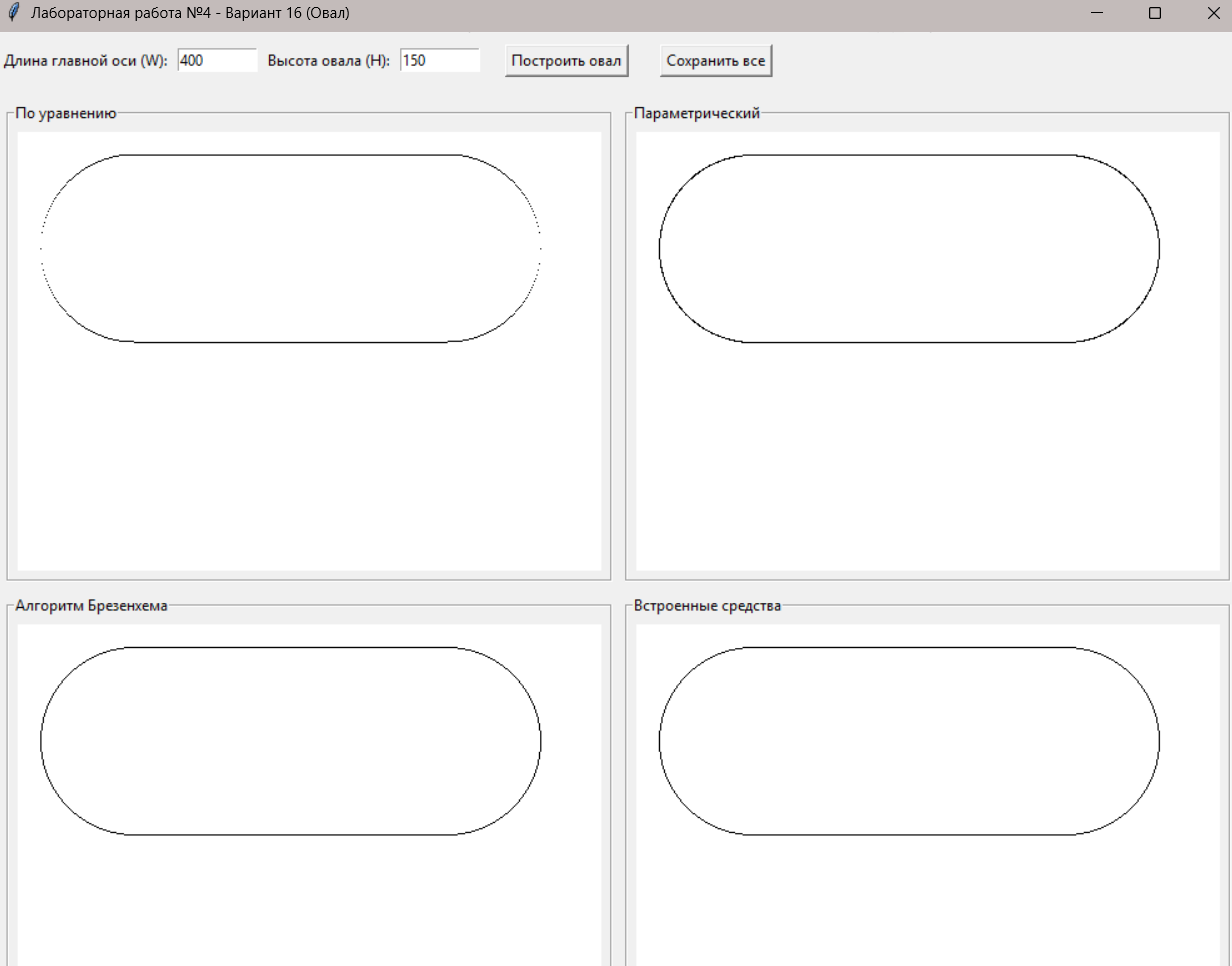
app = Lab4App(root)

root.mainloop()

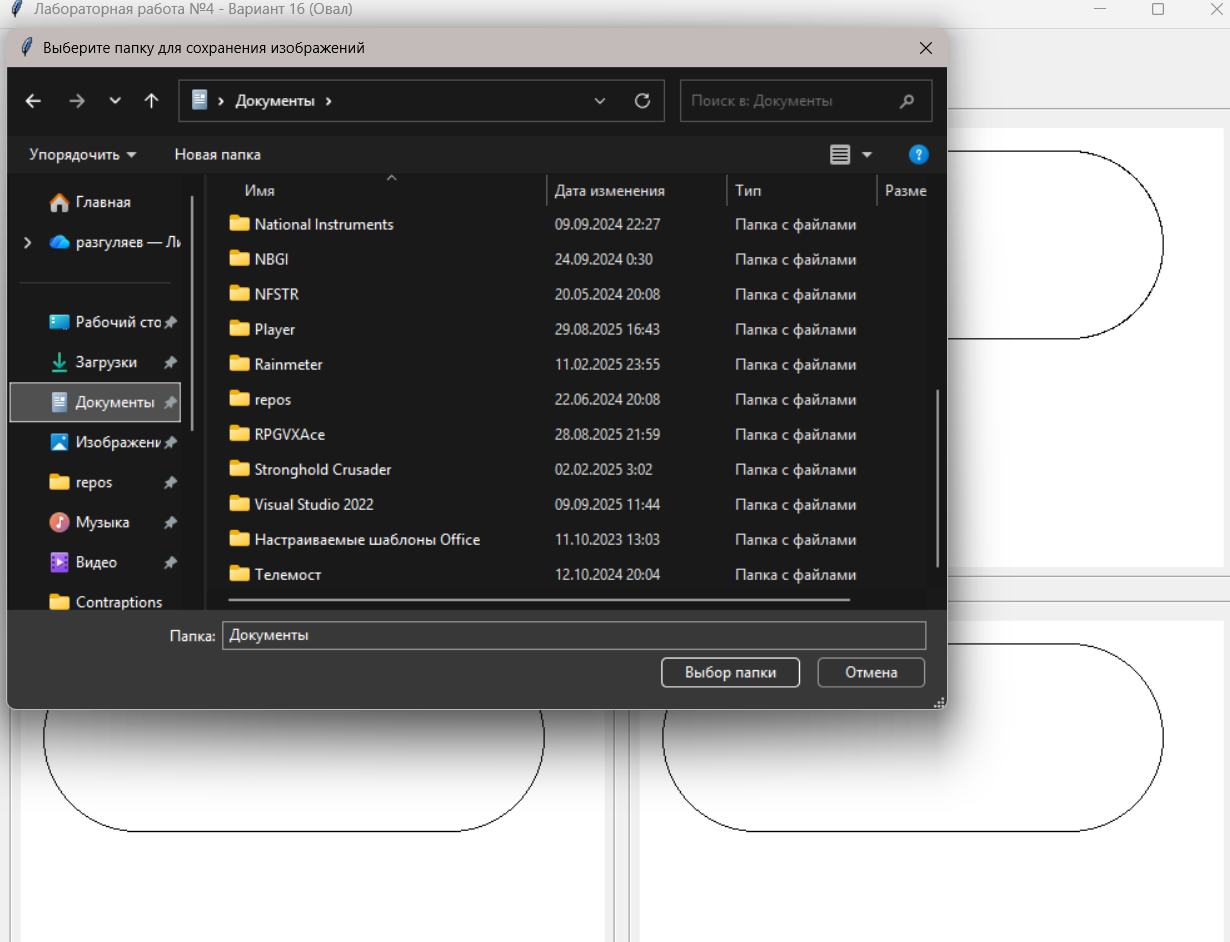
1. Указываем параметр:



2. Результат



3. Сохраняем все полученные результаты в форматах png и ppm:



**ВЫВОД**

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