**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

Чаплыгин Т. А.,

Чеховская Ю. М.

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.

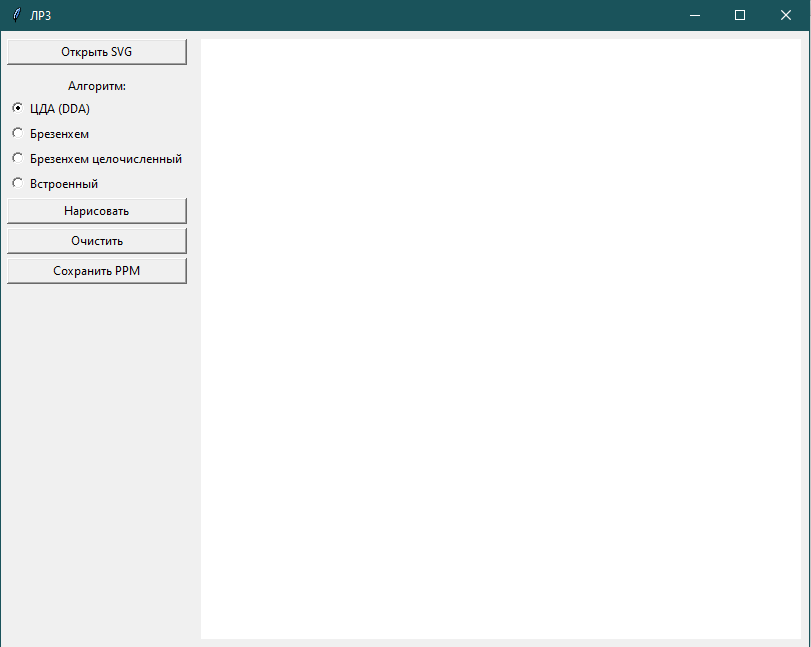
4. Оформить отчёт.

**Ход работы**

**Вариант 11**

По заданным вершинам вывести треугольник и его высоты.

1. Форма:



2. Текст программы:

import tkinter as tk

from tkinter import filedialog, messagebox

from PIL import Image, ImageTk, ImageDraw

import math, re

def clamp\_coord(x, w): return max(0, min(w - 1, int(x)))

def sign(v): return 1 if v > 0 else (-1 if v < 0 else 0)

def set\_pixel(img, x, y, color):

w, h = img.size

if 0 <= x < w and 0 <= y < h:

img.putpixel((x, y), color)

def dda\_line(img, x1, y1, x2, y2, color):

dx, dy = x2 - x1, y2 - y1

L = max(abs(dx), abs(dy))

if L == 0: set\_pixel(img, round(x1), round(y1), color); return

dx\_step, dy\_step = dx / L, dy / L

x, y = x1, y1

for \_ in range(int(L) + 1):

set\_pixel(img, round(x), round(y), color)

x += dx\_step; y += dy\_step

def brez\_float\_line(img, x1, y1, x2, y2, color):

sx, sy = sign(x2 - x1), sign(y2 - y1)

dx, dy = abs(x2 - x1), abs(y2 - y1)

if dx == 0 and dy == 0: set\_pixel(img, round(x1), round(y1), color); return

if dy > dx: dx, dy, flag = dy, dx, 1

else: flag = 0

m = dy / dx

e = m - 0.5

x, y = x1, y1

for \_ in range(int(dx) + 1):

set\_pixel(img, int(x), int(y), color)

if e >= 0:

if flag: x += sx

else: y += sy

e -= 1

if flag: y += sy

else: x += sx

e += m

def brez\_int\_line(img, x1, y1, x2, y2, color):

x1, y1, x2, y2 = map(int, [round(x1), round(y1), round(x2), round(y2)])

dx, dy = abs(x2 - x1), abs(y2 - y1)

sx, sy = sign(x2 - x1), sign(y2 - y1)

err = dx - dy

while True:

set\_pixel(img, x1, y1, 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 builtin\_line(img, x1, y1, x2, y2, color):

draw = ImageDraw.Draw(img)

draw.line((x1, y1, x2, y2), fill=color)

def foot\_of\_perpendicular(px, py, x1, y1, x2, y2):

dx, dy = x2 - x1, y2 - y1

denom = dx\*dx + dy\*dy

if denom == 0: return (x1, y1)

t = ((px-x1)\*dx + (py-y1)\*dy) / denom

return (x1 + t\*dx, y1 + t\*dy)

def save\_ppm\_ascii(image, path):

w, h = image.size

with open(path, "w") as f:

f.write("P3\n")

f.write(f"{w} {h}\n255\n")

for y in range(h):

row = []

for x in range(w):

r, g, b = image.getpixel((x, y))

row.append(f"{r} {g} {b}")

f.write(" ".join(row) + "\n")

class App:

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

self.root = root

root.title("ЛР3")

self.W, self.H = 600, 600

self.img = Image.new("RGB", (self.W, self.H), (255,255,255))

self.photo = ImageTk.PhotoImage(self.img)

ctrl = tk.Frame(root); ctrl.pack(side=tk.LEFT, fill=tk.Y, padx=6, pady=6)

tk.Button(ctrl, text="Открыть SVG", command=self.load\_svg).pack(fill="x", pady=2)

tk.Label(ctrl, text="Алгоритм:").pack(pady=(8,0))

self.alg\_var = tk.StringVar(value="dda")

for t,v in [("ЦДА (DDA)","dda"),("Брезенхем","brez\_f"),

("Брезенхем целочисленный","brez\_i"),("Встроенный","builtin")]:

tk.Radiobutton(ctrl, text=t, variable=self.alg\_var, value=v).pack(anchor="w")

tk.Button(ctrl, text="Нарисовать", command=self.draw\_triangle).pack(fill="x", pady=2)

tk.Button(ctrl, text="Очистить", command=self.clear\_canvas).pack(fill="x", pady=2)

tk.Button(ctrl, text="Сохранить PPM", command=self.save\_ppm).pack(fill="x", pady=2)

self.canvas = tk.Label(root, image=self.photo)

self.canvas.pack(side=tk.RIGHT, padx=6, pady=6)

self.vertices = None

def load\_svg(self):

path = filedialog.askopenfilename(filetypes=[("SVG files","\*.svg")])

if not path: return

with open(path, "r", encoding="utf-8") as f:

text = f.read()

m = re.search(r'points="([\d ,.-]+)"', text)

if not m:

messagebox.showerror("Ошибка", "Не найден <polygon points=...>")

return

nums = list(map(float, re.findall(r'[-]?\d+\.?\d\*', m.group(1))))

if len(nums) < 6:

messagebox.showerror("Ошибка", "Недостаточно координат")

return

self.vertices = nums[:6]

messagebox.showinfo("SVG", f"Загружены вершины: {self.vertices}")

def draw\_triangle(self):

if not self.vertices:

messagebox.showerror("Ошибка", "Сначала загрузите SVG")

return

self.img = Image.new("RGB", (self.W, self.H), (255,255,255))

alg = self.alg\_var.get()

if alg=="dda": func = dda\_line

elif alg=="brez\_f": func = brez\_float\_line

elif alg=="brez\_i": func = brez\_int\_line

else: func = builtin\_line

x1,y1,x2,y2,x3,y3 = self.vertices

blue, red, green = (0,0,200), (200,0,0), (0,150,0)

func(self.img,x1,y1,x2,y2,blue)

func(self.img,x2,y2,x3,y3,blue)

func(self.img,x3,y3,x1,y1,blue)

for (vx,vy),(ax,ay),(bx,by) in [((x1,y1),(x2,y2),(x3,y3)),

((x2,y2),(x3,y3),(x1,y1)),

((x3,y3),(x1,y1),(x2,y2))]:

fx, fy = foot\_of\_perpendicular(vx,vy,ax,ay,bx,by)

func(self.img,vx,vy,fx,fy,red)

set\_pixel(self.img, int(fx), int(fy), green)

self.update\_canvas()

def clear\_canvas(self):

self.img = Image.new("RGB", (self.W, self.H), (255,255,255))

self.update\_canvas()

def update\_canvas(self):

self.photo = ImageTk.PhotoImage(self.img)

self.canvas.configure(image=self.photo)

self.canvas.image = self.photo

def save\_ppm(self):

path = filedialog.asksaveasfilename(defaultextension=".ppm",

filetypes=[("PPM ASCII","\*.ppm")])

if not path: return

save\_ppm\_ascii(self.img, path)

messagebox.showinfo("Сохранено", f"PPM файл: {path}")

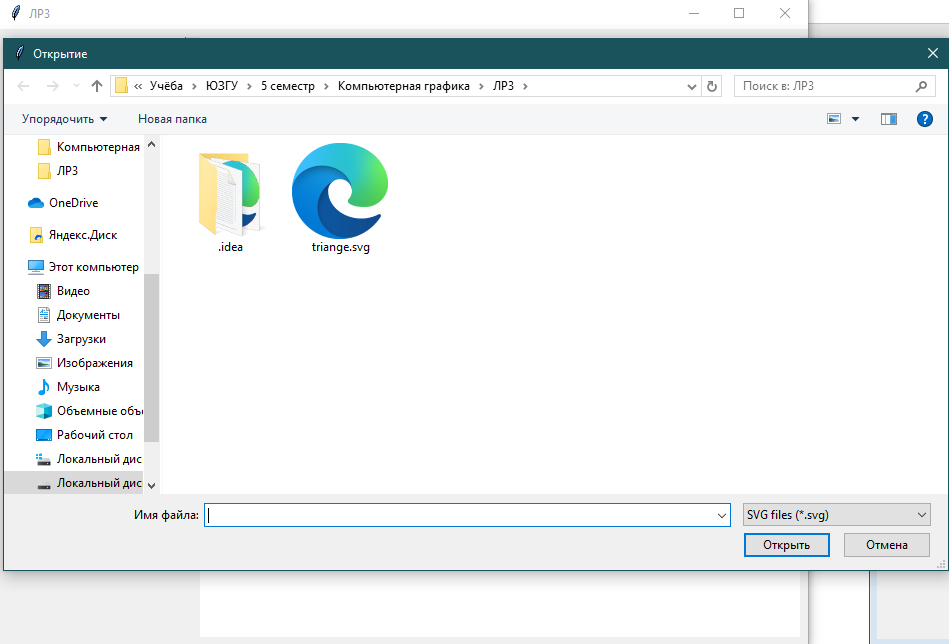
if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

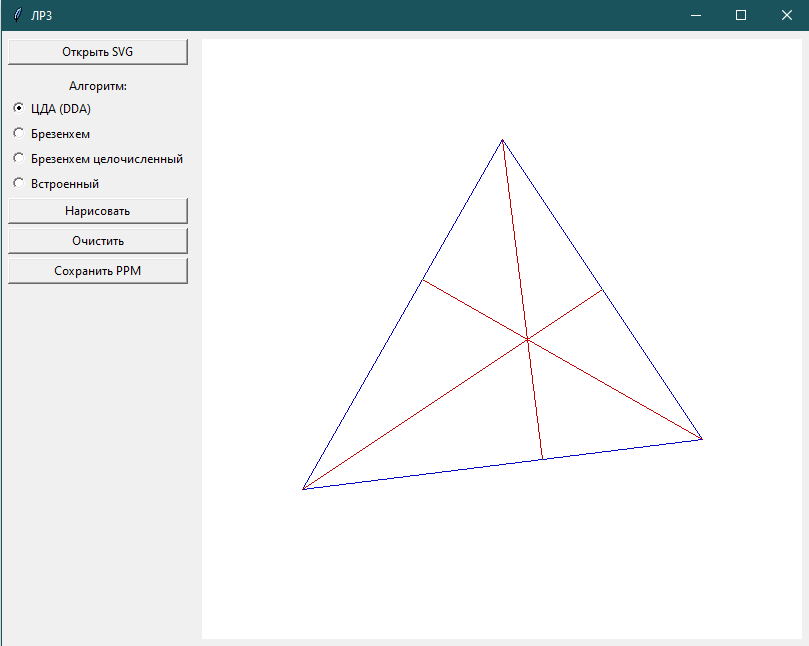
app = App(root)

root.mainloop()

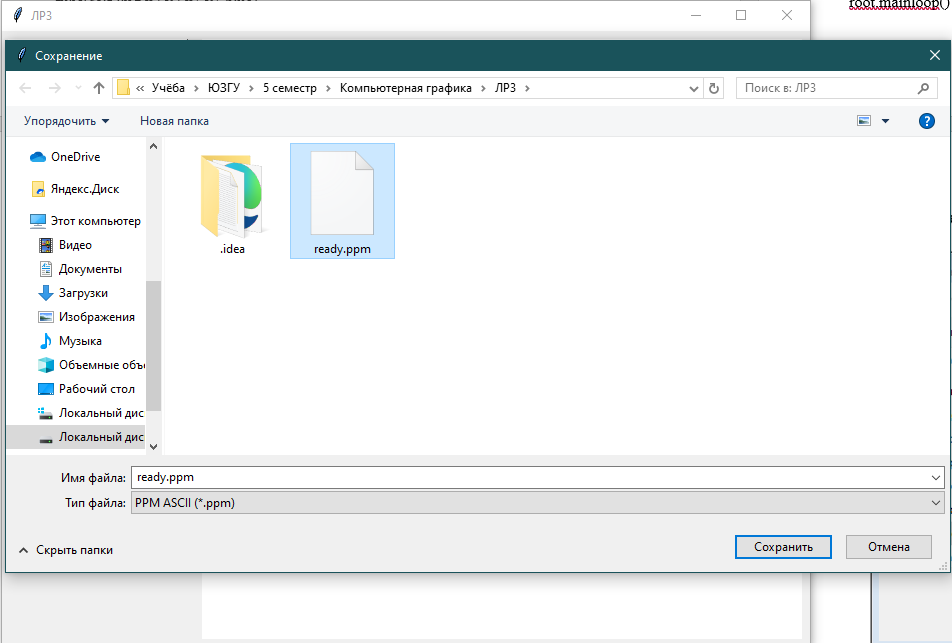
3. Открываем svg файл:



4. Строим треугольник и вершины разными алгоритмами



5. Сохраняем результат в формате ppm:



**Контрольные вопросы**

1. Что такое пиксель?

Пиксель (от англ. "picture element" - элемент изображения) - это наименьший логический элемент растрового изображения. В контексте компьютерной графики:

* Представляет собой отдельную точку на экране или в изображении
* Имеет определенные координаты (x, y) и цвет
* В цветных изображениях обычно кодируется комбинацией красного, зеленого и синего компонентов (RGB)
* В программе представлен как точка с координатами, которую мы закрашиваем определенным цветом

2. Что такое растровое изображение?

Растровое изображение - это изображение, состоящее из сетки пикселей (растра), где каждый пиксель имеет свой цвет и положение. Характеристики:

* Состоит из матрицы пикселей фиксированного размера
* Размер определяется шириной и высотой в пикселях
* Качество зависит от разрешения (количества пикселей на единицу длины)
* Примеры форматов: PNG, JPEG, BMP, PPM (которые используются в программе)
* В отличие от векторной графики, при увеличении теряет качество

3. Для чего нужны алгоритмы растеризации отрезков?

Алгоритмы растеризации отрезков используются для отображения линий (отрезков) на растровых экранах. Они определяют, какие пиксели нужно закрасить, чтобы линия выглядела максимально приближённой к идеальной прямой.

4. Какие ещё алгоритмы растеризации отрезков существуют?

Помимо реализованных в программе алгоритмов, существуют:

Алгоритм средней точки (Midpoint Circle Algorithm) - модификация алгоритма Брезенхема

Алгоритм с использованием целочисленной арифметики - избегает операций с плавающей точкой

Алгоритм Xiaolin Wu - для сглаженного (антиалиасинг) рисования окружностей

Алгоритм на основе цепных дробей

Алгоритм с использованием тригонометрических таблиц - для предвычисленных значений

Алгоритм с применением векторных инструкций - для современных процессоров

**ВЫВОД**

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

Ссылка на гитхаб - https://github.com/Tima2005-rus/ComputerGraphicsLab3