**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра систем автоматизированного проектирования (САПР)**

**отчет**

**по лабораторной работе № 1**

**по дисциплине «Компьютерная графика»**

Тема: **«Представление и комбинированные преобразования плоских и объемных изображений»**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 3311 | Аршин А. Д  Баймухамедов Р. Р.  Пасечный Л. В. |  |
| Преподаватель | Колев Г. Ю. |  |

Санкт-Петербург

2025

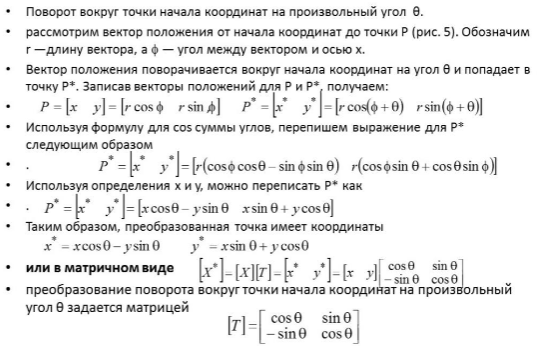
**Цель работы**

Исследование математических методов представления и преобразования графических объектов на плоскости и в пространстве

**Задание (вариант 3):**

Поворот плоского объекта относительно произвольной точки плоскости на заданный угол. Необходимо предусмотреть возможность редактирования положения точки.

Теоретическая основа



**Выполнение лабораторной работы**

В качестве языка программирования для выполнения данной лабораторной работы выберем Python. Для быстрого запуска достаточно иметь установленный Python 3 (модули tkinter и math, которые входят в стандартную библиотеку). В качестве фигуры для поворота выберем треугольник.

**Интерфейс программы**

|  |
| --- |
|  |
| Демонстрация выполненной лабораторной работы №1 |

**Вывод**

В ходе выполнения работы нами были исследованы методы представления и преобразования графических объектов на плоскости и в пространстве. Также получены навыки реализации метода поворота объектов на плоскости с помощью матрицы поворота, а также создания графических приложений.

**Код программы**

import tkinter as tk

import math

class RotationApp:

    # инициализируем окно

    # cохраняем ссылку на главное окно (root) и задаём ему заголовок

    def \_\_init\_\_(self, root):

        self.root = root

        self.root.title("Поворот объекта вокруг произвольной точки")

        self.canvas\_width = 600

        self.canvas\_height = 400

        # создаем сanvas и область для отрисовки

        self.canvas = tk.Canvas(root, width=self.canvas\_width, height=self.canvas\_height, bg="white")

        self.canvas.pack()

        # параметры объекта

        self.original\_points = [(100, 100), (150, 50), (200, 100)]  # исходные координаты треугольник

        self.rotated\_points = self.original\_points.copy() # копия точек, здесь будут записываться измененые координаты

        # точка вокруг которой происходит вращения

        self.rotation\_point = [300, 200]

        # угол поворота в градусах

        self.angle\_deg = 0

        # созадем рамку frame для размещения элементов программы

        control\_frame = tk.Frame(root)

        control\_frame.pack(pady=10)

        # слайдер для поворота фигуры в градусах вокруг заданной точки

        # при любом движении слайдера вызывается метод update\_rotation

        tk.Label(control\_frame, text="Угол поворота фигуры:").grid(row=0, column=0, padx=5)

        self.angle\_slider = tk.Scale(control\_frame, from\_=0, to=360, orient=tk.HORIZONTAL, length=300,

                                     command=self.update\_rotation)

        self.angle\_slider.set(self.angle\_deg)

        self.angle\_slider.grid(row=0, column=1, padx=5)

        # поле ввода для координаты x точки вращения

        # заполняем его начальным значением 300

        tk.Label(control\_frame, text="Точка вращения X:").grid(row=1, column=0, padx=5)

        self.point\_x\_entry = tk.Entry(control\_frame, width=10)

        self.point\_x\_entry.insert(0, str(self.rotation\_point[0]))

        self.point\_x\_entry.grid(row=1, column=1, sticky="w", padx=5)

        # поле ввода для координаты y

        # расположено справа в той же ячейке (с помощью sticky="e" — прижато к правому краю)

        tk.Label(control\_frame, text="Y:").grid(row=1, column=1)

        self.point\_y\_entry = tk.Entry(control\_frame, width=10)

        self.point\_y\_entry.insert(0, str(self.rotation\_point[1]))

        self.point\_y\_entry.grid(row=1, column=1, sticky="e", padx=5)

        # кнопка, которая считывает значения из полей ввода и обновляет точку вращения

        tk.Button(control\_frame, text="Обновить точку", command=self.update\_point\_from\_entry).grid(row=1, column=2, padx=10)

        # подсказка

        tk.Label(root, text="Кликните на холсте, чтобы задать новую точку вращения").pack()

        # привязка клика мышки

        self.canvas.bind("<Button-1>", self.set\_rotation\_point\_by\_click)

        # рисуем первый кадр

        self.draw\_all()

    # поворачивает точку (x,y) вокруг center=(cx,cy) на угол angle\_rad

    # сдвигаем систему координат так, чтобы центр вращения стал началом (вычитаем cx, cy)

    # применяем стандартную матрицу поворота

    def rotate\_point(self, point, center, angle\_rad):

        x, y = point

        cx, cy = center

        # перенос в начало координат

        x -= cx

        y -= cy

        # поворот

        x\_new = x \* math.cos(angle\_rad) - y \* math.sin(angle\_rad)

        y\_new = x \* math.sin(angle\_rad) + y \* math.cos(angle\_rad)

        # обратный перенос

        x\_new += cx

        y\_new += cy

        return x\_new, y\_new

    # обновляет повёрнутые координаты объекта и перерисовывает

    # считываем текущий угол со слайдера

    # переводим градусы в радианы

    # для каждой исходной точки вычисляем её новое положение после поворота

    # обновляем список rotated\_points

    # перерисовываем всё

    def update\_rotation(self, val=None):

        self.angle\_deg = self.angle\_slider.get()

        angle\_rad = math.radians(self.angle\_deg)

        self.rotated\_points = [

            self.rotate\_point(p, self.rotation\_point, angle\_rad)

            for p in self.original\_points

        ]

        self.draw\_all()

    # обновляет точку вращения из полей ввода

    # пытаемся прочитать числа из полей

    # если введено не число (например, буквы) ловим ValueError и ничего не делаем

    # иначе обновляем точку и пересчитываем поворот

    def update\_point\_from\_entry(self):

        try:

            x = float(self.point\_x\_entry.get())

            y = float(self.point\_y\_entry.get())

            self.rotation\_point = [x, y]

            self.update\_rotation()

        except ValueError:

            pass  # игнорируем некорректный ввод

    # устанавливает точку вращения по клику мыши

    def set\_rotation\_point\_by\_click(self, event):

        # постоянно обновляем при действии rotation\_point

        self.rotation\_point = [event.x, event.y]

        self.point\_x\_entry.delete(0, tk.END)

        self.point\_y\_entry.delete(0, tk.END)

        self.point\_x\_entry.insert(0, str(event.x))

        self.point\_y\_entry.insert(0, str(event.y))

        self.update\_rotation()

    # перерисовывает всё на холсте

    def draw\_all(self):

        self.canvas.delete("all")

        # рисуем оси

        self.canvas.create\_line(0, self.canvas\_height//2, self.canvas\_width, self.canvas\_height//2, fill="gray", dash=(2,2))

        self.canvas.create\_line(self.canvas\_width//2, 0, self.canvas\_width//2, self.canvas\_height, fill="gray", dash=(2,2))

        # рисуем исходный объект, который не будет менять положение

        if len(self.original\_points) >= 2:

            self.canvas.create\_polygon(self.original\_points, outline="gray", fill="", dash=(4,4), width=2)

        # рисуем повёрнутый объект который будет менять свое положение вокруг точки

        if len(self.rotated\_points) >= 2:

            self.canvas.create\_polygon(self.rotated\_points, outline="blue", fill="", width=3)

        # рисуем точку вращения

        x, y = self.rotation\_point

        r = 5

        self.canvas.create\_oval(x - r, y - r, x + r, y + r, fill="red")

        self.canvas.create\_text(x + 10, y - 10, text="O", fill="red", font=("Arial", 10))

        # подпись угла в данный момент

        self.canvas.create\_text(50, 20, text=f"Угол: {self.angle\_deg}°", fill="black", font=("Arial", 12))

# Запуск приложения

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

    root = tk.Tk()

    app = RotationApp(root)

    root.mainloop()