Міністерство освіти і науки України
Національний технічний університет України «КПІ» імені Ігоря Сікорського
Кафедра обчислювальної техніки ФІОТ

ЗВІТ
з лабораторної роботи №1
з навчальної дисципліни «Computer Vision»

Тема:

ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ПОБУДОВИ ТА ПЕРЕТВОРЕННЯ КООРДИНАТ ПЛОЩИННИХ (2D) ОБ’ЄКТІВ

Виконав:
Студент 3 курсу кафедри ІПІ ФІОТ,
Навчальної групи ІП-11
Панченко С.В.

Перевірив:
Професор кафедри ОТ ФІОТ
Писарчук О.О.

Київ 2024

І. Мета:

Виявити дослідити та узагальнити особливості побудови та перетворення координат площинних (2D) об’єктів.

ІІ. Завдання:

Лабораторія провідної ІТ-компанії реалізує масштабний проект розробки універсальної платформи з цифрової обробки зображень для задач Computer Vision. Платформа передбачає розташування back-end компоненти на власному хмарному сервері з наданням повноважень користувачам заздалегідь адаптованого front-end функціоналу універсальної платформи. Цим формується унікальна для потреб замовника ERP систем з технологіями Computer Vision. Змовниками ресурсів платформи є: державні та комерційні компанії, що розробляють медичне обладнання з діагностування захворювань за візуальною інформацією; автоматизації аграрного бізнесу в аспекті обліку посівних територій за даними з БПЛА; візуального контролю безпекових заходів на об’єктах критичної інфраструктури: аеропорти, торгівельно-розважальні центри, житлові комплекси тощо.

Вам, як Computer Vision поставлено завдання.

Варіант (Номер дня народження (06.03.2004)	**Технічні умови**	**Графічна фігура**
6	Реалізувати операції: обертання – масштабування – переміщення. 3. операцію реалізувати циклічно, траєкторію зміни положення цієї операції відобразити. Обрати самостійно: бібліотеку, розмір графічного вікна, розмір фігури, параметри реалізації операцій, кольорову гамму усіх графічних об’єктів. Всі операції перетворень мають здійснюватись у межах графічного вікна.	П’ятикутник

Завдання І рівня складності – максимально 7 балів.

Здійснити синтез математичних моделей та розробити програмний скрипт, що реалізує базові операції 2D перетворень над геометричними примітивами. Для розробки використовувати матричні операції та технології композиційних перетворень. Вхідна матриця координат кутів геометричної фігури має бути розширеною.

Функціонал скрипта, що розробляється має реалізувати технічних вимог табл.1 Додатку 1.

ІІІ. Результати виконання лабораторної роботи.

3.1. Синтезована математична модель перетворень графічних об’єктів відповідно до індивідуального завдання.

Відповідно до умов задачі було створено математичну модель для виконання операцій над структурою вхідного графічного об'єкту. Ця модель дозволяє виконувати послідовні дії переміщення, масштабування і обертання у будь-якому порядку.

Обертання вершин геометричного примітиву навколо центроїда та точки, що знаходиться за межами самого примітиву, представляють два паралельні рухи, які за своїми характеристиками аналогічні обертанню планет навколо зірок. Перший рух - це обертання геометричного примітиву навколо центроїда, а другий - обертання навколо зовнішньої точки. Наприклад, у випадку планет обертання навколо центроїда відображає обертання планети навколо власної осі, тоді як обертання навколо зовнішньої точки відповідає обертанню Землі навколо Сонця. Таким чином, для обох рухів використовується одна і та ж формула.

$$x_i^{'} = C_x + (x_i - C_x) * \cos(\theta) - (y_i - C_y) * \sin(\theta)$$
$$y_i^{'} = C_y + (x_i - C_x) * \cos(\theta) + (y_i - C_y) * \sin(\theta)$$

Переміщення вершин на точку $(x_i^{'}, y_i^{'})$ з точки (x_i, y_i) за допомогою трансляційного вектора (dx, dy).

$$x_i^{'} = x_i + dx$$
$$y_i^{'} = y_i + d\,y$$

Масштабування геометричного примітива включає зміну його розмірів навколо деякої точки, якою зазвичай є центроїд. Якщо S – це фактор масштабування, а координати центроїда - (C_x, C_y), то формула масштабування вершин геометричного примітиву являє собою:

$$x_i^{'} = C_x + (x_i - C_x) * S$$
$$y_i^{'} = C_y + (x_i - C_y) * S$$

3.2. Блок схема алгоритму та її опис.

Було прийнято рішення розробити десктопний застосунок з незалежним порядком виконання просторових операцій замість послідовного як сказано в завданні. Це дає змогу переконатися у правильность виконання операцій. Єдиний порядок операцій, що варто розглянути – це при обертанні. Порядок дій наведено на Рисунку 1:

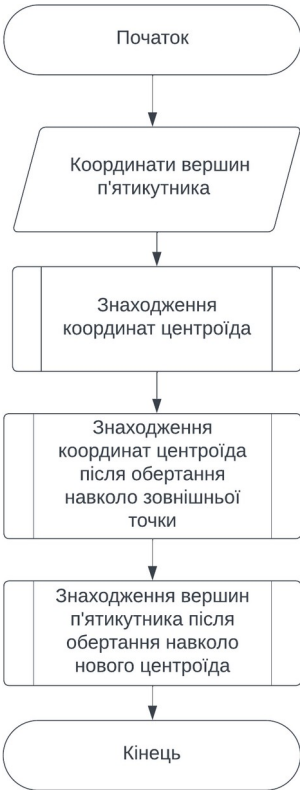

Рис.1. Блок-схема алгоритму обертання.

Робота програми відбувається у вікні десктопного інтерфейсу, де функціонал програми представлений у вигляді кнопок з наступними можливостями:

1. Переміщення шестикутника до кінця лівої грані вікна.
2. Переміщення шестикутника до кінця правої грані вікна.
3. Переміщення шестикутника до кінця верхньої грані вікна.
4. Переміщення шестикутника до кінця нижньої грані вікна.
5. Масштабування, а саме збільшення площі шестикутника.
6. Масштабування, а саме зменшення площі шестикутника.
7. Обертання навколо центру вікна.
8. Видалення всіх "слідів" переміщення та повернення шестикутника у центр вікна.

3.3. Опис структури проекту програми в середовищі PyCharm.

Для реалізації розробленого алгоритму мовою програмування Python з використанням можливостей інтегрованого середовища PyCharm сформовано проект.

Проект базується на бізнес-логіці ООП та має таку структуру.

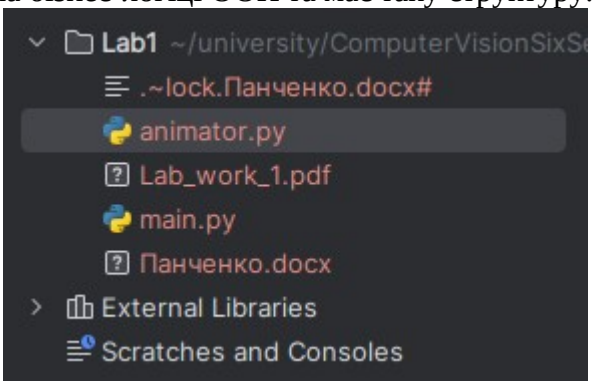

Рис.2. Структура проекту.

Lab_work_1.pdf – завдання лабораторної роботи
Панченко.docx – звіт лабораторної роботи
animator.py – клас анімованих перетворень над п’ятикутником
main.py – основна логіка десктопного інтерфейсу, реалізовного з Tkinter

3.4. Результати роботи програми відповідно до завдання.

Програма видає десктопний додаток з можливістю виконання функцій у будь-якому порядку та їх комбінування. Нижче наведено порядок виконання просторових перетворень згідно з поставленим завданням.

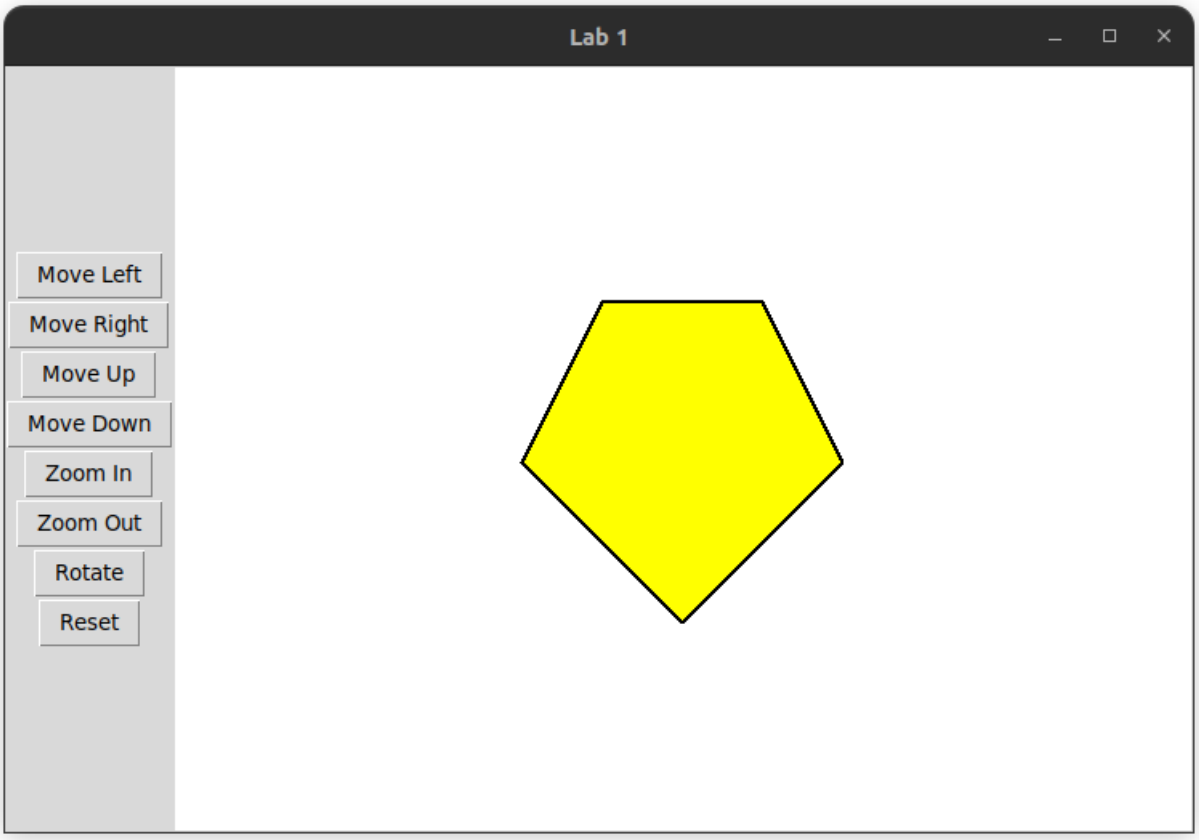

Рис.3. Первинне положення п’ятикутника.

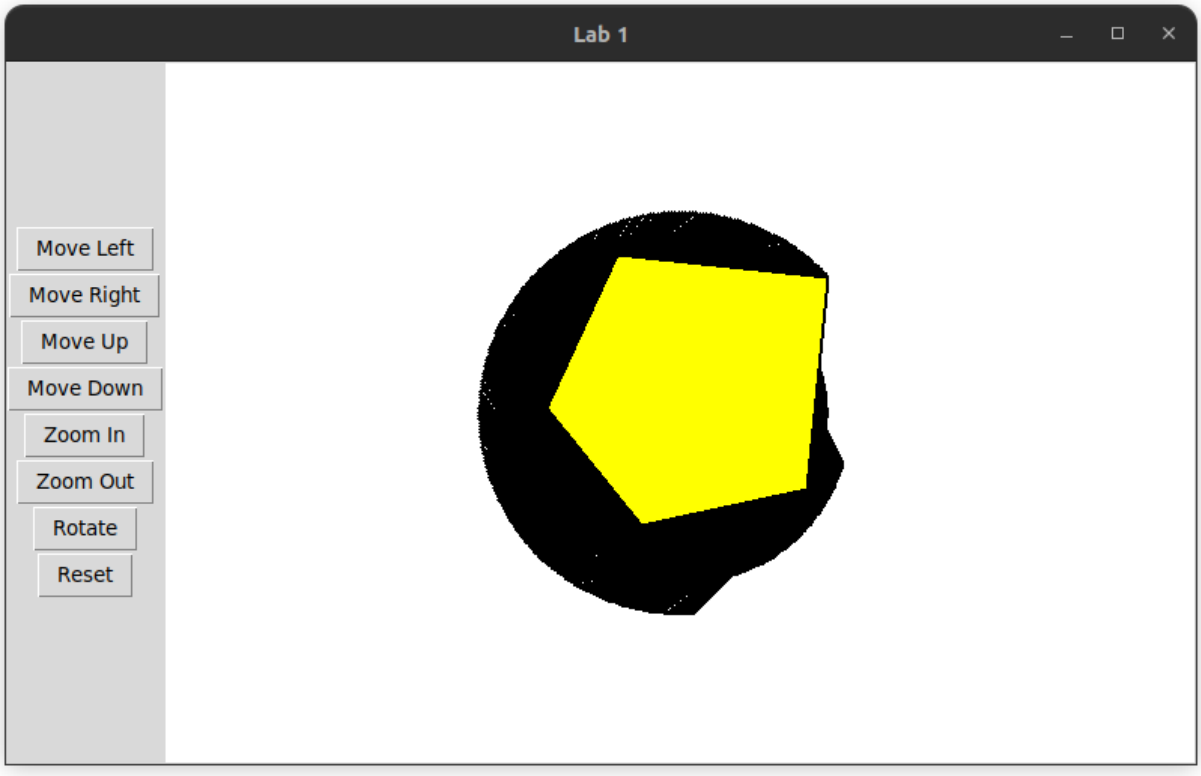

Рис.4. Обертання.

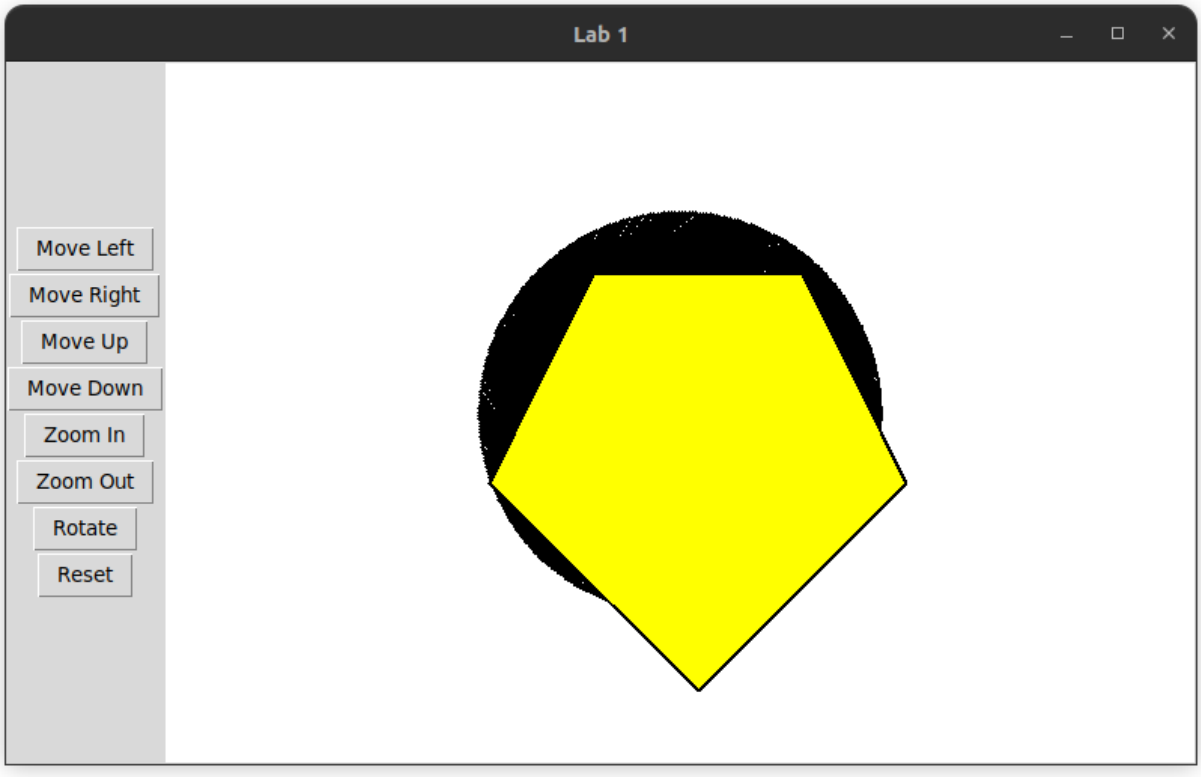

Рис. 5. Збільшення

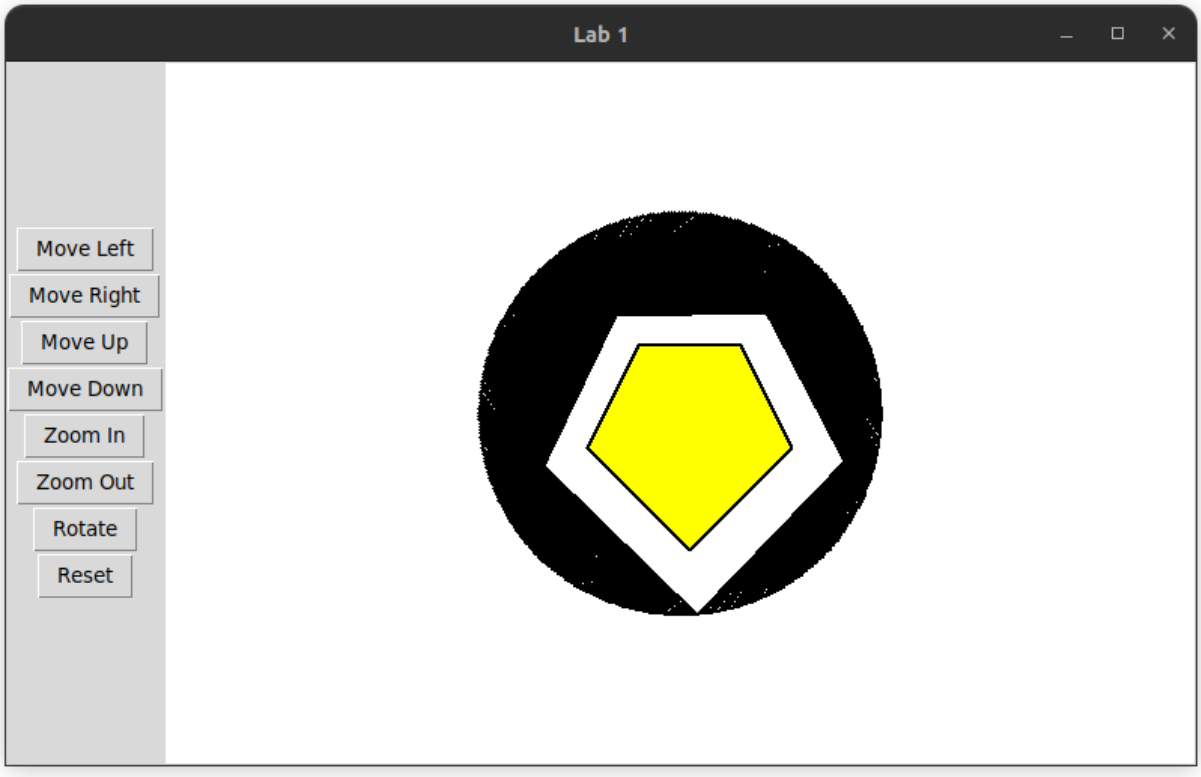

Рис. 6. Зменшення

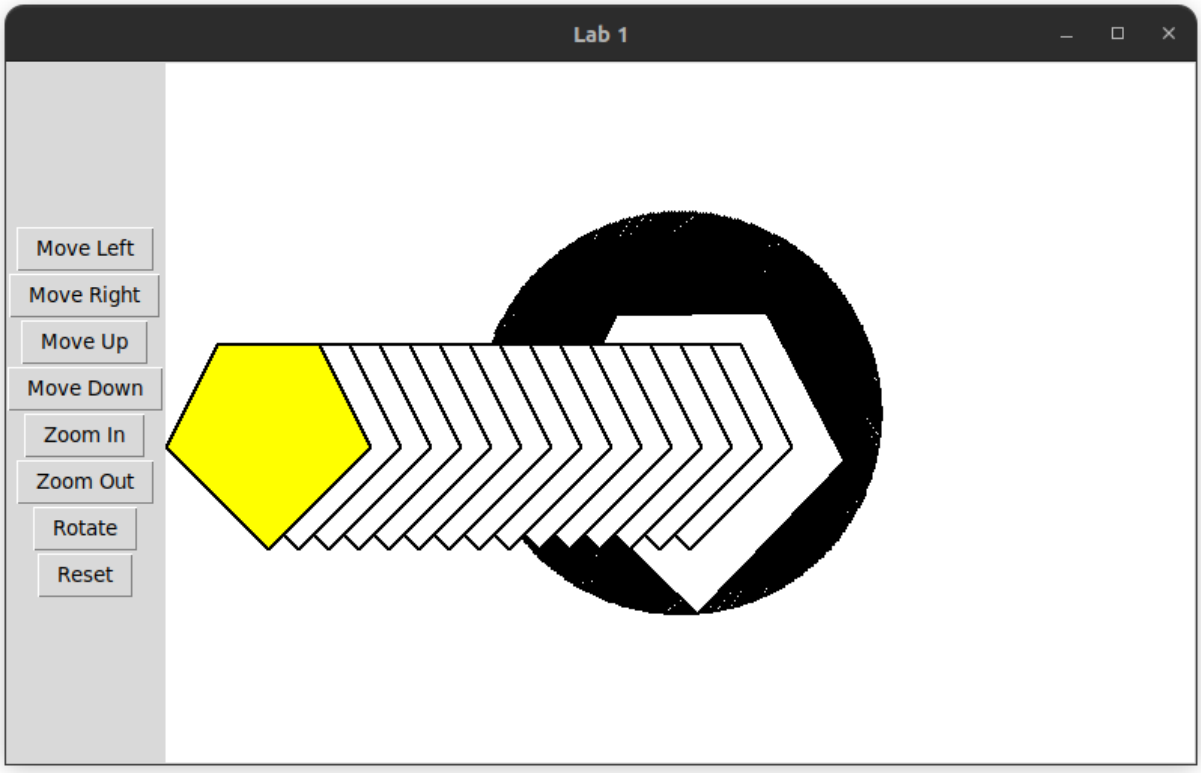

Рис. 7. Переміщення вліво

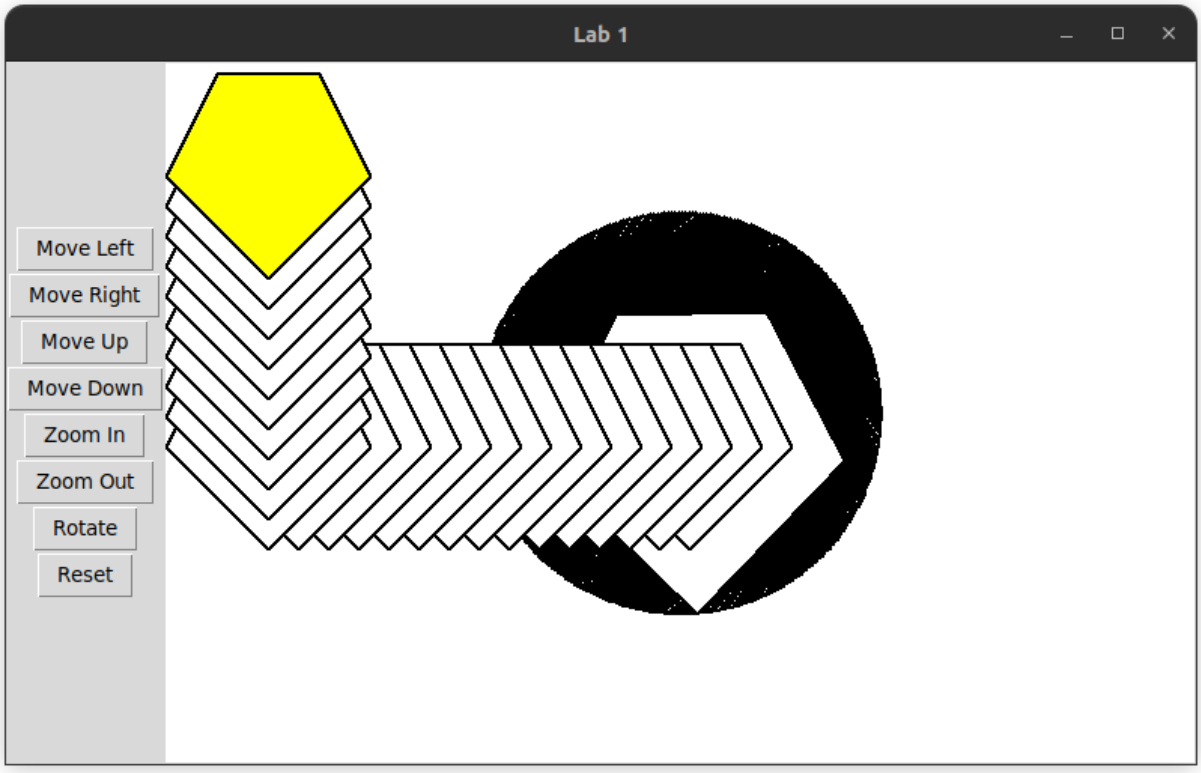

Рис.8. Переміщення вгору

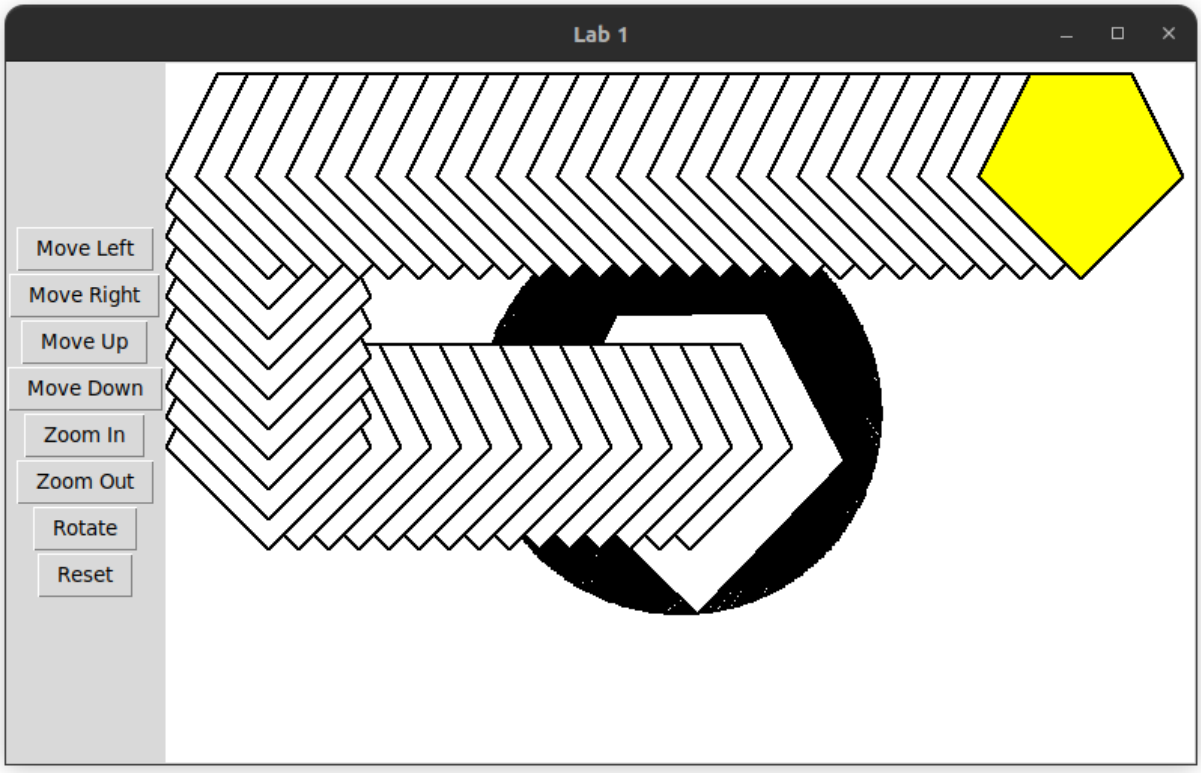

Рис. 9. Переміщення вправо

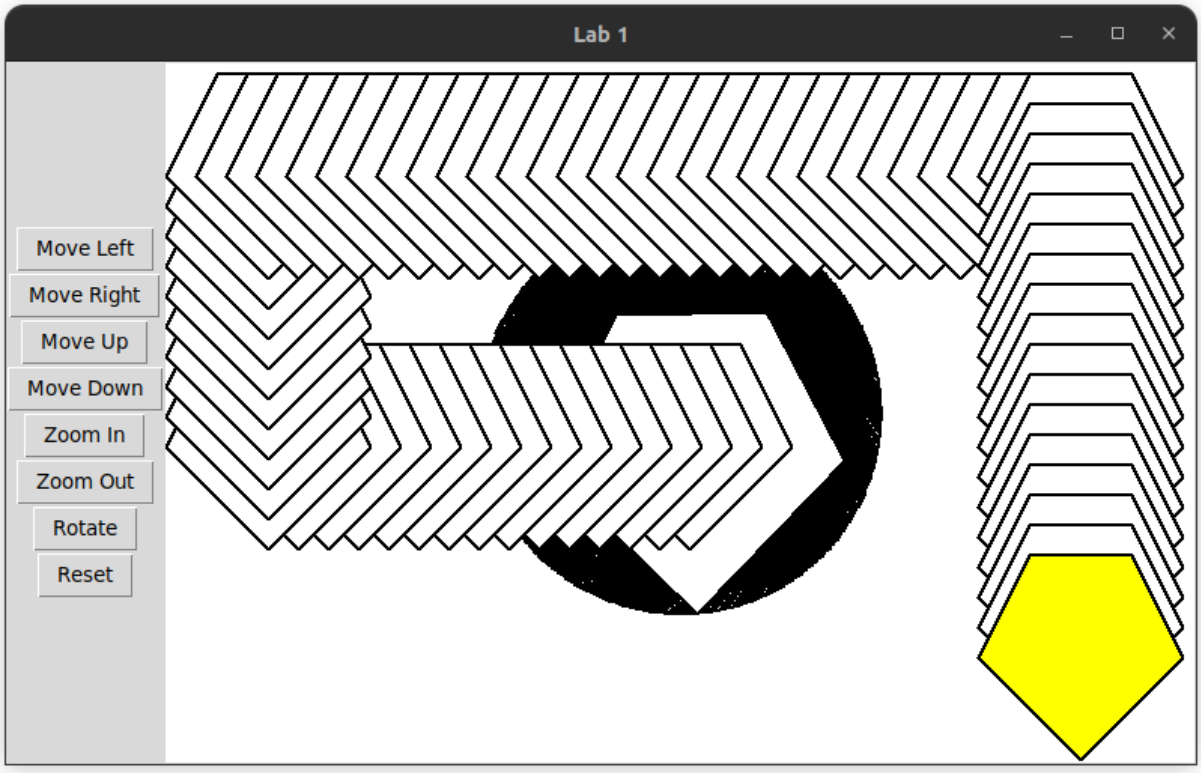

Рис.10. Переміщення вниз

Представлені результати у повному обсязі відповідають завданню лабораторної роботи.

3.5. Програмний код.

Програмний код послідовно втілює алгоритм, зображений на рис.1, і спрямований на досягнення результатів, які представлені на рис.2. Для розробки коду була використана об'єктно-орієнтована парадигма з метою інкапсуляції змінних та методів, надаючи користувачу лише публічний інтерфейс. Обчислення виконувались за допомогою розширеної матриці координат паралелепіпеду. Для цього використовувалися можливості бібліотек Python: math та tkinter.

Файл animator.py

```
import math



class Animator:
    STEP_SIZE = 20

    def __init__(self, canvas):
        self._canvas = canvas
        pentagon = [
            677, 283,
            777, 283,
            827, 383,
            727, 483,
            627, 383,
            677, 283
        ]
        self._pentagon_id = self._canvas.create_polygon(pentagon, outline='black',
fill='yellow', width=2)
        self.traced = []

    @staticmethod
    def _rotate_point_around_point(point, center, angle):
        angle_rad = math.radians(angle)
        ox, oy = center
        px, py = point
        qx = ox + math.cos(angle_rad) * (px - ox) - math.sin(angle_rad) * (py - oy)
        qy = oy + math.sin(angle_rad) * (px - ox) + math.cos(angle_rad) * (py - oy)
        return qx, qy

    def move_left(self):
        coords = self._canvas.coords(self._pentagon_id)
        if self._boundary_check(coords):
            new_coords = [coords[i] - (Animator.STEP_SIZE if not i % 2 else 0) for i in
range(len(coords))]
            if self._boundary_check(new_coords):
                self.traced.append(self._canvas.create_polygon(coords, outline='black',
fill='white', width=2))
                self._canvas.coords(self._pentagon_id, *new_coords)
                self._canvas.tag_raise(self._pentagon_id)
                self._canvas.after(50, lambda: self.move_left())

    def _boundary_check(self, coords) -> bool:
        return (min(coords[::2]) >= 0 and max(coords[::2]) <= self._canvas.winfo_width()
                and min(coords[1::2]) >= 0 and max(coords[1::2]) <=
self._canvas.winfo_height())

    def empty_traces(self):
        for self._pentagon_id in self.traced:
            self._canvas.delete(self._pentagon_id)
```

```
        self.traced.clear()

    def reset(self):
        canvas_width = self._canvas.winfo_width()
        canvas_height = self._canvas.winfo_height()
        center_x, center_y = canvas_width / 2, canvas_height / 2

        coords = self._canvas.coords(self._pentagon_id)
        pentagon_center_x = sum(coords[::2]) / (len(coords) / 2)
        pentagon_center_y = sum(coords[1::2]) / (len(coords) / 2)

        translation_x = center_x - pentagon_center_x
        translation_y = center_y - pentagon_center_y

        new_coords = [coords[i] + (translation_x if i % 2 == 0 else translation_y) for i in
range(len(coords))]
        self._canvas.coords(self._pentagon_id, *new_coords)
        self.empty_traces()

    def rotate(self, spin_angle, orbit_angle, number_of_spins):
        if number_of_spins >= 360:
            return

        coords = self._canvas.coords(self._pentagon_id)

        centroid_x, centroid_y = Animator._get_centroid(coords)
        center_x, center_y = self.get_center()

        spun_coords = []
        for i in range(0, len(coords), 2):
            x, y = Animator._rotate_point_around_point((coords[i], coords[i + 1]), (centroid_x,
centroid_y), spin_angle)
            spun_coords.extend([x, y])

        new_centroid_x, new_centroid_y = Animator._get_centroid(spun_coords)
        orbit_x, orbit_y = Animator._rotate_point_around_point(
            (new_centroid_x, new_centroid_y), (center_x, center_y), orbit_angle)

        translation_x = orbit_x - new_centroid_x
        translation_y = orbit_y - new_centroid_y

        final_coords = [spun_coords[i] + (translation_x if i % 2 == 0 else translation_y) for i in
                        range(len(spun_coords))]

        self.traced.append(self._canvas.create_polygon(coords, outline='black', fill='white',
width=2))
        self._canvas.coords(self._pentagon_id, *final_coords)
        self._canvas.tag_raise(self._pentagon_id)
        self._canvas.after(25, lambda: self.rotate(spin_angle, orbit_angle, number_of_spins +
1))

    def move_up(self):
        coords = self._canvas.coords(self._pentagon_id)
        if self._boundary_check(coords):
            new_coords = [coords[i] - (Animator.STEP_SIZE if i % 2 else 0) for i in
range(len(coords))]
            if self._boundary_check(new_coords):
                self.traced.append(self._canvas.create_polygon(coords, outline='black',
fill='white', width=2))
                self._canvas.coords(self._pentagon_id, *new_coords)
```

```
            self._canvas.tag_raise(self._pentagon_id)
            self._canvas.after(50, lambda: self.move_up())

    def move_down(self):
        coords = self._canvas.coords(self._pentagon_id)
        if self._boundary_check(coords):
            new_coords = [coords[i] + (Animator.STEP_SIZE if i % 2 else 0) for i in
range(len(coords))]
            if self._boundary_check(new_coords):
                self.traced.append(self._canvas.create_polygon(coords, outline='black',
fill='white', width=2))
                self._canvas.coords(self._pentagon_id, *new_coords)
                self._canvas.tag_raise(self._pentagon_id)
                self._canvas.after(50, lambda: self.move_down())

    def move_right(self):
        coords = self._canvas.coords(self._pentagon_id)
        if self._boundary_check(coords):
            new_coords = [coords[i] + (Animator.STEP_SIZE if not i % 2 else 0) for i in
range(len(coords))]
            if self._boundary_check(new_coords):
                self.traced.append(self._canvas.create_polygon(coords, outline='black',
fill='white', width=2))
                self._canvas.coords(self._pentagon_id, *new_coords)
                self._canvas.tag_raise(self._pentagon_id)
                self._canvas.after(50, lambda: self.move_right())

    def zoom(self, scale_factor):
        coords = self._canvas.coords(self._pentagon_id)
        center_x = sum(coords[::2]) / (len(coords) / 2)
        center_y = sum(coords[1::2]) / (len(coords) / 2)
        new_coords = []
        for i in range(len(coords)):
            if i % 2 == 0:
                new_coord = center_x + (coords[i] - center_x) * scale_factor
            else:
                new_coord = center_y + (coords[i] - center_y) * scale_factor
            new_coords.append(new_coord)
        if self._boundary_check(new_coords):
            self._canvas.coords(self._pentagon_id, *new_coords)

    @staticmethod
    def _get_centroid(coords):
        centroid_x = sum(coords[::2]) / (len(coords) / 2)
        centroid_y = sum(coords[1::2]) / (len(coords) / 2)
        return centroid_x, centroid_y

    def get_center(self):
        center_x = self._canvas.winfo_width() / 2
        center_y = self._canvas.winfo_height() / 2
        return center_x, center_y
```

Файл main.py

```
import tkinter as tk
from animator import Animator



root = tk.Tk()
root.title("Lab 1")
```

```
button_frame = tk.Frame(root)
button_frame.pack(side=tk.LEFT, anchor='w')

canvas = tk.Canvas(root, bg='white')
canvas.pack(fill=tk.BOTH, expand=True)



animator = Animator(canvas)

move_left_button = tk.Button(button_frame, text="Move Left",
                     command=lambda: animator.move_left())
move_left_button.pack(side=tk.TOP, expand=True)

move_right_button = tk.Button(button_frame, text="Move Right",
                     command=lambda: animator.move_right())
move_right_button.pack(side=tk.TOP, expand=True)

move_up_button = tk.Button(button_frame, text="Move Up",
                     command=lambda: animator.move_up())
move_up_button.pack(side=tk.TOP, expand=True)

move_down_button = tk.Button(button_frame, text="Move Down",
                     command=lambda: animator.move_down())
move_down_button.pack(side=tk.TOP, expand=True)

zoom_in_button = tk.Button(button_frame, text="Zoom In",
                     command=lambda: animator.zoom(1.2))
zoom_in_button.pack(side=tk.TOP, expand=True)

zoom_out_button = tk.Button(button_frame, text="Zoom Out",
                     command=lambda: animator.zoom(0.8))
zoom_out_button.pack(side=tk.TOP, expand=True)

rotate_button = tk.Button(button_frame, text="Rotate",
                    command=lambda: animator.rotate(1, 4, 0))
rotate_button.pack(side=tk.TOP, expand=True)

reset_button = tk.Button(button_frame, text="Reset",
                   command=lambda: animator.reset())
reset_button.pack(side=tk.TOP, expand=True)
root.mainloop()
```

3.6. Аналіз результатів відлагодження та верифікації результатів роботи програми.

Лагодження та тестування підтвердили, що створений код працює ефективно. Підтвердженням функціональності програмного коду була верифікація, а також порівняння отриманих результатів із технічними вимогами, визначеними у завданні для лабораторної роботи. Це підтверджує, що всі вказівки завдання були виконані в повному обсязі.

IV. Висновки.

Під час виконання лабораторної роботи проведено дослідження особливостей створення плоских (2D) об'єктів, використовуючи можливості алгоритмічної мови високого рівня - Python. Дослідження підтвердило, що моделювання 2D об'єктів та

опрацювання плоских форм відбуваються в реальних розмірах без змін у геометричній структурі.

Виконав: студент Панченко С.В.