Національний технічний університет України «КПІ ім. Ігоря Сікорського»
Факультет Інформатики та Обчислювальної Техніки

Кафедра інформаційних систем та технологій

Лабораторна робота №1

з дисципліни «Технології Computer Vision»

на тему

«ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ПОБУДОВИ ТА ПЕРЕТВОРЕННЯ КООРДИНАТ ПЛОЩИННИХ (2D) ТА ПРОСТОРОВИХ (3D) ОБ’ЄКТІВ»

Виконала:
студентка групи ІС-12
Павлова Софія

Перевірив:
Баран Д. Р.

Київ – 2023

1. Постановка задачі

Мета роботи:

Виявити дослідити та узагальнити особливості формування та перетворення координат площинних (2d) та просторових (3d) об'єктів.

Завдання ІІІ рівня:

1. 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.

4	Реалізувати операції: **обертання – переміщення – масштабування**. **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.	Ромб

Рисунок 1 – Варіант завдання І рівня складності

2. Здійснити синтез математичних моделей та розробити програмний скрипт, що реалізує базові операції 3D перетворень над геометричними примітивами: аксонометрична проекція будь-якого типу та з циклічне обертання (анімація) 3D графічного об'єкту навколо будь-якої обраної внутрішньої віссю. Траєкторію обертання не відображати. Для розробки використовувати матричні операції. Вхідна матриця координат кутів геометричної фігури має бути розширеною.

4	**Динаміка фігури**: графічна фігура з'являється та гасне, змінює колір заливки. **Обрати самостійно:** бібліотеку, розмір графічного вікна, розмір фігури, параметри зміни положення фігури, кольорову гамму усіх графічних об'єктів. Всі операції перетворень мають здійснюватись у межах графічного вікна.	Паралелепіпед

Рисунок 2 – Варіант завдання II рівня складності

2. Виконання

2.1. Робота з 2D геометричними фігурами

Математична модель

Розглянемо геометричні перетворення для операцій ***обертання***, ***переміщення*** та ***масштабування*** в однорідних координатах 2D простору.

Обертання

Обертання реалізується з використанням напрямних косинусів та синусів за моделлю:

$$[x', y', 1] = [x, y, 1] \cdot \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$R(\theta) = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$P' = P \cdot R(\theta)$ – **математична модель обертання**.

Переміщення

Переміщення на задану відтань виконується за моделлю:

$$[x', y', 1] = [x, y, 1] \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Dx & Dy & 1 \end{bmatrix},$$

$$T(Dx, Dy) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Dx & Dy & 1 \end{bmatrix},$$

$P' = P \cdot T(Dx, Dy)$ – **математична модель переміщення**.

<u>Масштабування</u>

Реалізується за моделлю:

$$[x', y', 1] = [x, y, 1] \cdot \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$S(Sx, Sy) = \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$P' = P \cdot S(Sx, Sy)$ – **математична модель масштабування**.

Архітектурне рішення

Реалізуємо синтезовані матричні операції над 2D фігурами за допомогою скрипкового рішення лінійної архітектури.

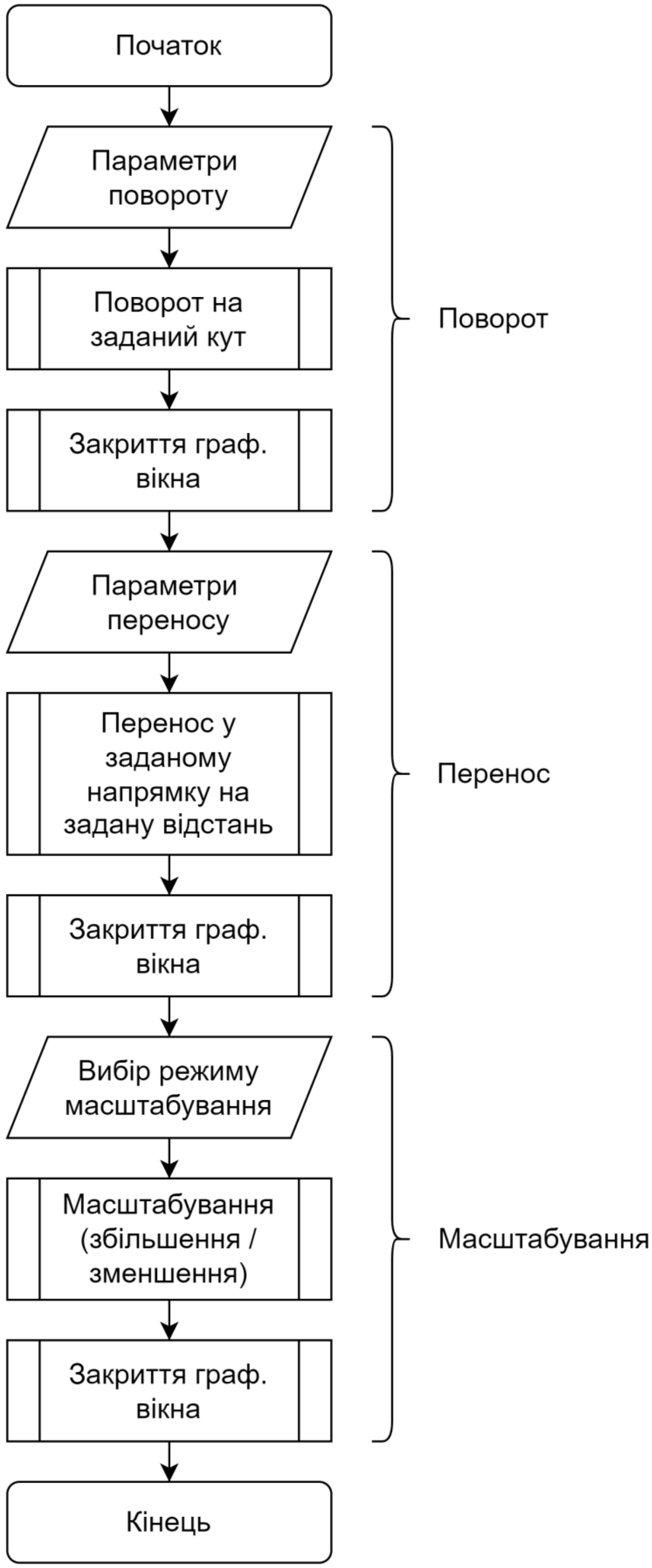

Рисунок 3 – Блок-схема алгоритму програми

Програмна реалізація

Розробимо програму таким чином, щоб користувач міг вільно налаштовувати параметри геометричних операцій.

Задамо функцію, яка ініціалізує вікно й фігуру. У нашому випадку – ромб. Для реалізації взаємодії з графічними вікнами використаємо open source бібліотеку *graphics*.

Лістинг коду:

```
from graphics import *
import math as mt
import time

import imageio
from PIL import ImageGrab

# Ініціалізація вікна і фігури
def init_window(text, xw, yw, size_x, size_y, center_x, center_y):
    # Ініціалізація вікна
    win = GraphWin(text, xw, yw)
    win.setBackground('white')

    # Координати вершин ромба в центрі вікна
    x1 = center_x - size_x / 2
    y1 = center_y
    x2 = center_x
    y2 = center_y + size_y / 2
    x3 = center_x + size_x / 2
    y3 = center_y
    x4 = center_x
    y4 = center_y - size_y / 2

    # Малювання ромба перед обертанням
    diamond = Polygon(Point(x1, y1), Point(x2, y2), Point(x3, y3), Point(x4, y4))
    diamond.setOutline("gray")
    diamond.setFill("gray")
    diamond.draw(win)

    return win, x1, y1, x2, y2, x3, y3, x4, y4

# Блок головних викликів
if __name__ == '__main__':

    # Константи
    xw = 600    # ширина вікна
    yw = 600    # висота вікна

    size_x = 200    # ширина ромба
    size_y = 400    # висота ромба

    center_x = xw / 2   # зміщення висоти ромба
    center_y = yw / 2   # зміщення ширини ромба
```

<u>Результат:</u>

Функція виводить у графічному вікні заданий ромб.

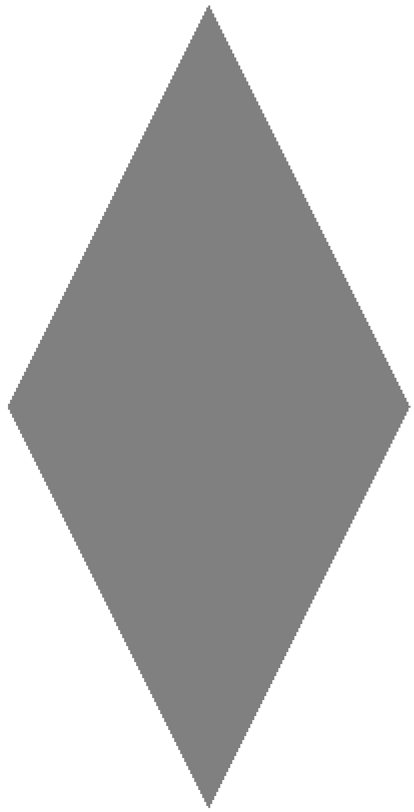

Рисунок 4 – Ініціалізоване вікно і ромб

Обертання

Задамо функцію, яка виконує поворот ромба на заданий кут. Додамо обмеження на існування кута повороту.

<u>Лістинг коду:</u>

```
# Поворот фігури на кут тета
def rotation(win, x1, y1, x2, y2, x3, y3, x4, y4):
    # Вибір кута тета
    print('Оберіть кут обертання фігури (0-360):')
    rotation_angle = int(input('angle:'))
    if rotation_angle in range(0, 361):
        # Перетворення кута обертання в радіани
        theta = mt.radians(rotation_angle)

        # Обчислення центра ромба
        center_x = (x1 + x2 + x3 + x4) / 4
        center_y = (y1 + y2 + y3 + y4) / 4

        # Обчислення нових координат ромба після обертання
        x1_rotated = center_x + (x1 - center_x) * mt.cos(theta) - (y1 - center_y) *
```

```
mt.sin(theta)
        y1_rotated = center_y + (x1 - center_x) * mt.sin(theta) + (y1 - center_y) *
mt.cos(theta)

        x2_rotated = center_x + (x2 - center_x) * mt.cos(theta) - (y2 - center_y) *
mt.sin(theta)
        y2_rotated = center_y + (x2 - center_x) * mt.sin(theta) + (y2 - center_y) *
mt.cos(theta)

        x3_rotated = center_x + (x3 - center_x) * mt.cos(theta) - (y3 - center_y) *
mt.sin(theta)
        y3_rotated = center_y + (x3 - center_x) * mt.sin(theta) + (y3 - center_y) *
mt.cos(theta)

        x4_rotated = center_x + (x4 - center_x) * mt.cos(theta) - (y4 - center_y) *
mt.sin(theta)
        y4_rotated = center_y + (x4 - center_x) * mt.sin(theta) + (y4 - center_y) *
mt.cos(theta)

        # Малювання ромба після обертання
        rotated_diamond = Polygon(Point(x1_rotated, y1_rotated), Point(x2_rotated,
y2_rotated),
                                  Point(x3_rotated, y3_rotated), Point(x4_rotated,
y4_rotated))
        rotated_diamond.setOutline("orange")
        rotated_diamond.setFill("orange")
        rotated_diamond.draw(win)

        win.getMouse()
        win.close()

    return

# Блок головних викликів
if __name__ == '__main__':

    # Константи
    [...]

    # Обертання
    win, x1, y1, x2, y2, x3, y3, x4, y4 = init_window('Обертання ромба', xw, yw, size_x,
size_y, center_x, center_y)
    rotation(win, x1, y1, x2, y2, x3, y3, x4, y4)
```

Результат:

При вводі кута повороту рівному 30 градусів, ромб виконує поворот на заданий кут.

```
Оберіть кут обертання фігури (0-360):
angle:30
```

Рисунок 5 – Поворот ромба

Перенос

Згідно з вищеописаною математичною моделлю реалізуємо матричний перенос ромба. Передбачимо можливість зміщувати ромб у напрямках: *Пн, Пд, Зх, Сх, Пн-зх, Пн-сх, Пд-зх, Пд-сх*.

Переміщення має відбуватись в межах вікна, тому додатково передбачимо обмеження на відстань переміщення.

Лістинг коду:

```
# Перенос фігури в напрямку direction на len одиниць
def move(win, x1, y1, x2, y2, x3, y3, x4, y4):
    # Карта напрямків переносу
    def directions():
        names = ['Пн', 'пн-зх', 'пн-сх', 'Зх', 'Сх', 'пд-зх', 'пд-сх', 'Пд']
        print(f'\n\t\t  {names[0]}\t\t')
        print(f'\t{names[1]}\t {names[2]}\t')
        print(f'{names[3]}\t\t\t\t\t{names[4]}')
        print(f'\t{names[5]}\t {names[6]}\t')
        print(f'\t\t  {names[7]}\t\t\n')

        return names

    # Вибір напрямку та довжини переносу
    print('\nОберіть напрямок переносу:')
    names = directions()
    for i in range(0, 8):
        print(i + 1, '-', names[i])
    direction = int(input('\ndirection:'))
```

```python
    if direction in range(1, 9):
        print('\nОберіть довжину переносу (0-100):')
        len = int(input('len:'))
        if len in range(0, 101):
            # Пн
            if direction == 1:
                dx = 0
                dy = len
            # пн-зх
            elif direction == 2:
                dx = -len
                dy = len
            # пн-сх
            elif direction == 3:
                dx = len
                dy = len
            # Зх
            elif direction == 4:
                dx = -len
                dy = 0
            # Сх
            elif direction == 5:
                dx = len
                dy = 0
            # пд-зх
            elif direction == 6:
                dx = -len
                dy = -len
            # пд-сх
            elif direction == 7:
                dx = len
                dy = -len
            # Пн
            else:
                dx = 0
                dy = -len

            # Обчислення нових координат ромба після переносу
            x1_moved = x1 + dx
            y1_moved = y1 - dy
            x2_moved = x2 + dx
            y2_moved = y2 - dy
            x3_moved = x3 + dx
            y3_moved = y3 - dy
            x4_moved = x4 + dx
            y4_moved = y4 - dy

            # Малювання ромба після переносу
            moved_diamond = Polygon(Point(x1_moved, y1_moved), Point(x2_moved, y2_moved),
                                   Point(x3_moved, y3_moved), Point(x4_moved, y4_moved))
            moved_diamond.setOutline("orange")
            moved_diamond.setFill("orange")
            moved_diamond.draw(win)

            win.getMouse()
            win.close()

    return

# Блок головних викликів
if __name__ == '__main__':

    # Константи
```

```
    [...]
    # Перенос
    win, x1, y1, x2, y2, x3, y3, x4, y4 = init_window('Перенос ромба', xw, yw, size_x,
size_y, center_x, center_y)
    move(win, x1, y1, x2, y2, x3, y3, x4, y4)
```

<u>Результат:</u>

У результаті переносу ромба за Пн-сх напрямком на 100 одиниць, отримуємо наступне зображення.

```
Оберіть напрямок переносу:

          Пн
    пн-зх    пн-сх
Зх                  Сх
    пд-зх    пд-сх
          Пд

1 - Пн
2 - пн-зх
3 - пн-сх
4 - Зх
5 - Сх
6 - пд-зх
7 - пд-сх
8 - Пд

direction:3

Оберіть довжину переносу (0-100):
len:100
```

Рисунок 6 – Переміщення ромба

Циклічне масштабування

Розділимо масштабування на *циклічне збільшення* та *циклічне зменшення*. **Збільшення** буде відбуватись до моменту досягнення краями ромба меж графічного вікна, а **зменшення** буде відбуватись до моменту повного зменшення фігури: зустрічі її вершин у одній точці.

Лістинг коду:

```
# Циклічне масштабування фігури на коефіцієнт k
def scale(win, x1, y1, x2, y2, x3, y3, x4, y4):
    # Обчислення центра ромба
    center_x = (x1 + x2 + x3 + x4) / 4
    center_y = (y1 + y2 + y3 + y4) / 4

    # Обчислення відстаней від центра ромба до його вершин
    x1_dist = x1 - center_x
    y1_dist = y1 - center_y
    x2_dist = x2 - center_x
    y2_dist = y2 - center_y
    x3_dist = x3 - center_x
    y3_dist = y3 - center_y
    x4_dist = x4 - center_x
    y4_dist = y4 - center_y

    # Вибір режиму масштабування
    print('\nОберіть режиму масштабування:')
    print('1 - Збільшення')
    print('2 - Зменшення')
    scale_mode = int(input('mode:'))
    # Якщо режим існує
    if scale_mode in range(1, 3):
        # Збільшення
        if scale_mode == 1:
            scale_koef = 1.05
        else:
            scale_koef = 0.9

    # Змінні для перевірки виходу за межі вікна або повного зменшення фігури
    within_window = True
    fully_scaled = False

    # Створення списку для зберігання зображень
    frames = []

    # Початок циклу масштабування
    while within_window and not fully_scaled:

        # Збереження зображення з вікна
        frame = get_image(win)
        frames.append(frame)

        x1_scaled = x1_dist * scale_koef + center_x
        y1_scaled = y1_dist * scale_koef + center_y

        x2_scaled = x2_dist * scale_koef + center_x
        y2_scaled = y2_dist * scale_koef + center_y
```

```
        x3_scaled = x3_dist * scale_koef + center_x
        y3_scaled = y3_dist * scale_koef + center_y

        x4_scaled = x4_dist * scale_koef + center_x
        y4_scaled = y4_dist * scale_koef + center_y

        # Перевірка, чи ромб залишається в межах вікна
        if all(0 <= coord <= win.getWidth() and 0 <= coord <= win.getHeight() for coord
in
               [x1_scaled, x2_scaled, x3_scaled, x4_scaled]):
            # Затримка перед наступною ітерацією
            time.sleep(0.1)

            # Малювання ромба після масштабування
            scaled_diamond = Polygon(Point(x1_scaled, y1_scaled), Point(x2_scaled,
y2_scaled),
                                     Point(x3_scaled, y3_scaled), Point(x4_scaled,
y4_scaled))
            scaled_diamond.setOutline("orange")
            scaled_diamond.draw(win)

            # Перевірка, чи ромб повністю зменшився
            threshold = 0.1 # Значення, яке ми вважаємо досить малим для зупинки циклу
            if max(abs(x1_scaled - center_x), abs(y1_scaled - center_y)) <= threshold:
                fully_scaled = True

        else:
            within_window = False  # Ромб вийшов за межі вікна, завершення циклу
масштабування

        # Оновлення відстаней до вершин ромба для наступного кроку масштабування
        x1_dist = x1_scaled - center_x
        y1_dist = y1_scaled - center_y
        x2_dist = x2_scaled - center_x
        y2_dist = y2_scaled - center_y
        x3_dist = x3_scaled - center_x
        y3_dist = y3_scaled - center_y
        x4_dist = x4_scaled - center_x
        y4_dist = y4_scaled - center_y

    # Зберігаємо анімацію у форматі GIF
    imageio.mimsave("animation_2d.gif", frames)

    if not within_window:
        print('\nРомб вийшов за межі вікна')
    else:
        print('\nРомб повністю зменшився')

    win.getMouse()
    win.close()

    return

# Блок головних викликів
if __name__ == '__main__':

    # Константи
    [...]
    # Масштабування
    win, x1, y1, x2, y2, x3, y3, x4, y4 = init_window('Масштабування ромба', xw, yw,
size_x, size_y, center_x, center_y)
    scale(win, x1, y1, x2, y2, x3, y3, x4, y4)
```

Результат:

При виборі анімації збільшення, бачимо, що ромб збільшується до виходу за межі вікна.

```
Оберіть режиму масштабування:
1 - Збільшення
2 - Зменшення
mode:1

Ромб вийшов за межі вікна
```

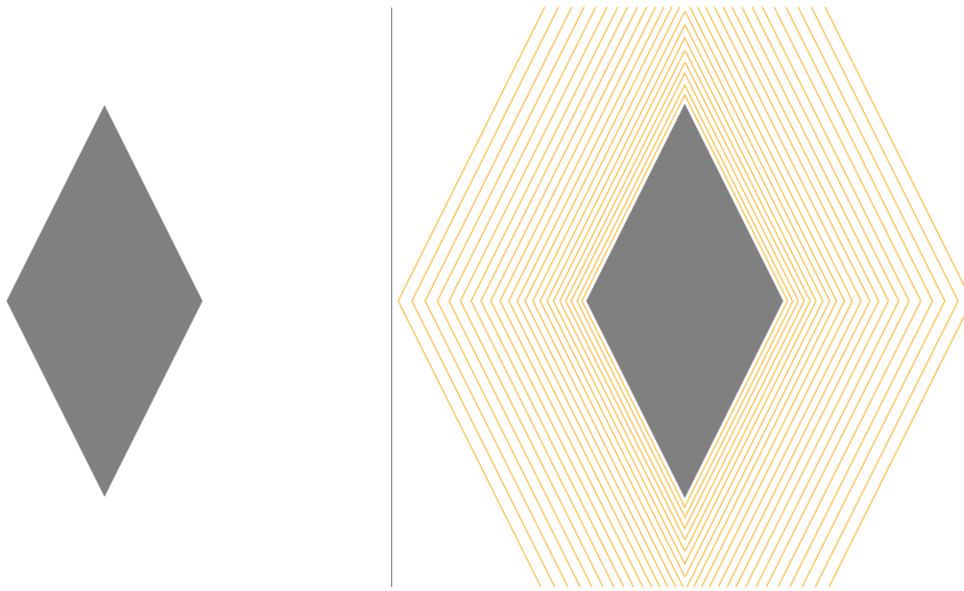

Рисунок 7 – Збільшення ромба

При виборі анімації зменшення, бачимо, що ромб повністю зменшується.

```
Оберіть режиму масштабування:
1 - Збільшення
2 - Зменшення
mode:2

Ромб повністю зменшився
```

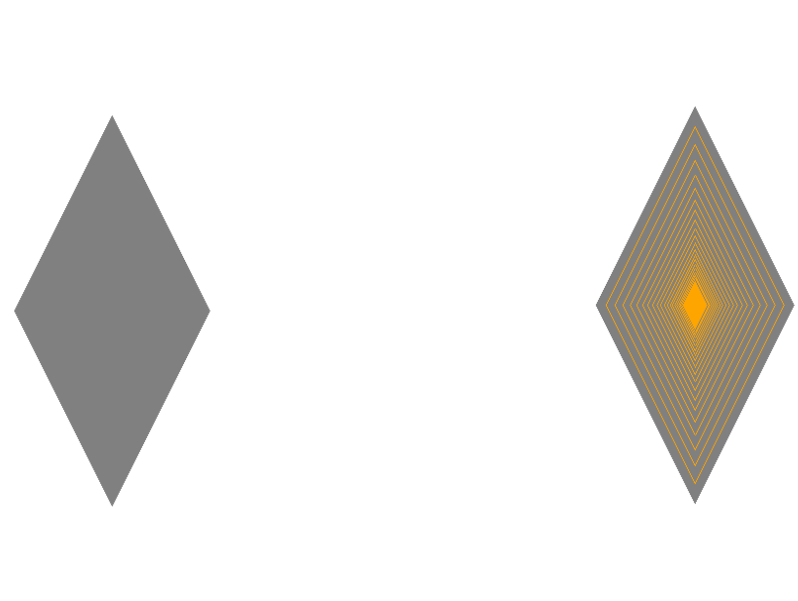

Рисунок 8 – Зменшення ромба

Для наочного відображення циклічної анімації використаємо бібліотеку *imageio* і збережемо результат останньої операції в форматі **.gif**.

У бібліотеці *graphics* не знайшлось необхідної операції для отримання зображення з графічного вікна, тому напишемо власну функцію для цього і додамо її в основний цикл масштабування.

Лістинг коду:

```
# Отримання зображень з вікна
def get_image(win):
    # Отримуємо зображення з вікна
    img = ImageGrab.grab(bbox=(win.master.winfo_x(), win.master.winfo_y(),
                               win.master.winfo_x() + win.master.winfo_width(),
                               win.master.winfo_y() + win.master.winfo_height()))
    img = img.convert('RGB')  # Перетворюємо у RGB формат

    return img
```

Результат:

Результати збережних gif можна переглянути у папці проекту на github.

2.2. Робота з 3D геометричними об’єктами

Математична модель

Розглянемо геометричні перетворення для операцій ***обертання***, ***діметрії, проекції, масштабування*** *та* ***переносу*** в координатах 3D простору.

<u>Обертання</u>

Тривимірне обертання забезпечує сегмент $T_{11}[3 \times 3]$ матриці перетворень T.

У нашому випадку будемо застосовувати обертання навколо осі x на кут θ:

$$T_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & sin\theta & 0 \\ 0 & -sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

<u>Діметрія</u>

Діметрія є частковим випадком аксонометричної проекції, за якої два кути між осями рівні. Узагальнена математична модель аксонометричної проекції:

$$A' = [x \quad y \quad z \quad 1] \cdot \begin{bmatrix} cos\varphi & 0 & -sin\varphi & 0 \\ 0 & 1 & 0 & 0 \\ sin\theta & 0 & cos\varphi & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\varphi & sin\varphi & 0 \\ 0 & -sin\varphi & cos\varphi & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

<u>Проекція</u>

У контексті лабораторної роботи розглянемо ортогональну проекцію на площину xy $(z = 0)$:

$$T_{xy}^{ort} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

<u>Масштабування</u>

Реалізується за допомогою діагональних елементів матриці $T - diagT[4 \times 4]$. Розглянемо варіант з загальною зміною масштабу:

$$A \cdot T = [x \quad y \quad z \quad 1] \cdot \begin{bmatrix} 1/_s & 0 & 0 & 0 \\ 0 & 1/_s & 0 & 0 \\ 0 & 0 & 1/_s & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x' \quad y' \quad z' \quad 1] = \left[\frac{x}{s} \quad \frac{y}{s} \quad \frac{z}{s} \quad 1\right].$$

<u>Перенос</u>

Модель трьохвимірного лінійного переносу в точку $K = [l \quad m \quad n \quad 1]$:

$$A' = A \cdot T = [x \quad y \quad z \quad 1] \cdot \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ l & m & n & 1 \end{bmatrix} = [x + l \quad y + m \quad z + n \quad 1].$$

Архітектурне рішення

Реалізуємо синтезовані матричні операції над 3D об’єктами за допомогою циклічного скрипкового рішення.

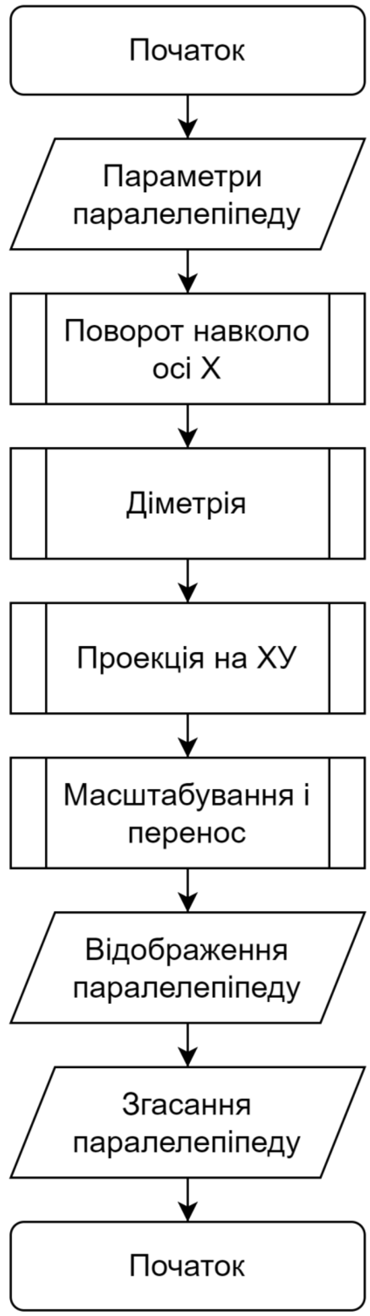

Рисунок 9 – Блок-схема алгоритму програми

Програмна реалізація

Розробимо програму таким чином, щоб головний цикл виконувався допоки користувач не закриє вікно. Реалізуємо **обертання навколо осі X діметричної проекції паралелепіпеда заданого масштабу, зміщеного в центр вікна**.

Для початку реалізуємо функції для ініціалізації вікна та паралелепіпеда.

Лістинг коду:

```
from graphics import *
import numpy as np
import math as mt

import imageio
from PIL import ImageGrab

# Ініціалізація вікна
def init_window(xw, yw):
    win = GraphWin('Паралелепіпед', xw, yw)
    win.setBackground('white')

    return win

# Ініціалізація паралелепіпеда
def init_figure():
    # Вхідна матриця фігури
    prlpd = np.array([[0, 0, 0, 1],
                      [2, 0, 0, 1],
                      [2, 1, 0, 1],
                      [0, 1, 0, 1],
                      [0, 0, 2, 1],
                      [2, 0, 2, 1],
                      [2, 1, 2, 1],
                      [0, 1, 2, 1]])

    print('Вхідна матриця')
    print(prlpd)

    return prlpd

# Блок головних викликів
if __name__ == '__main__':

    # Константи
    xw = 600  # ширина вікна
    yw = 600  # висота вікна
    st = 100  # розмір сторони фігури

    # Головні виклики
    win = init_window(xw, yw)
    prlpd = init_figure()
```

Результат:

Отримано вхідну матрицю паралелепіпеда. Об’єкт ініціалізовано відповідно до початку координат. Тобто в лівому верхньому куту вікна.

```
Вхідна матриця
[[0 0 0 1]
 [2 0 0 1]
 [2 1 0 1]
 [0 1 0 1]
 [0 0 2 1]
 [2 0 2 1]
 [2 1 2 1]
 [0 1 2 1]]
```

Рисунок 10 – Вхідна матриця паралелепіпеду

Обертання

Виконаємо обертання паралелепіпеда відносно осі x. Кут обертання буде одним з параметрів циклу, тому обертання почнуться з 0 градусів і кут зростатиме.

Лістинг коду:

```
# Обертання навколо X
def rotate_x (figure, teta_x):
    teta_rx = (3 / 14 * teta_x) / 180

    f = np.array(
        [[1,    0,                    0,                    0],
         [0,    mt.cos(teta_rx),    mt.sin(teta_rx),    0],
         [0,    -mt.sin(teta_rx),   mt.cos(teta_rx),    0],
         [0,    0,                    0,                    1]])
    ft = f.T
    pr_xy = figure.dot(ft)

    print('\nОбертання навколо X')
    print(pr_xy)

    return pr_xy

# Блок головних викликів
if __name__ == '__main__':

    # Константи
    [...]

    # Головні виклики
    [...]

    angle = 0
    i = 0
    while not win.isClosed():

        # Збереження зображення з вікна
        frame = get_image(win)
        frames.append(frame)

        win.delete("all")  # Очищаємо вікно
```

```
        # Обертання фігури
        prlpd_rotated = rotate_x(prlpd, angle)
```

Результат:

З плином ітерацій циклу бачимо зміну положення паралелепіпеда на заданий циклом кут.

```
Обертання навколо X
[[0. 0. 0. 1.]
 [2. 0. 0. 1.]
 [2. 1. 0. 1.]
 [0. 1. 0. 1.]
 [0. 0. 2. 1.]
 [2. 0. 2. 1.]
 [2. 1. 2. 1.]
 [0. 1. 2. 1.]]
```

```
Обертання навколо X
[[ 0.          0.          0.          1.        ]
 [ 2.          0.          0.          1.        ]
 [ 2.          0.99745006 -0.07136785  1.        ]
 [ 0.          0.99745006 -0.07136785  1.        ]
 [ 0.          0.1427357   1.99490013  1.        ]
 [ 2.          0.1427357   1.99490013  1.        ]
 [ 2.          1.14018576  1.92353228  1.        ]
 [ 0.          1.14018576  1.92353228  1.        ]]
```

Рисунок 11 – Обертання навколо x на 0 та на 1 градус

Діметрія

Для цікавішої візуалізації використаємо діметричну проекцію нашого паралелепіпеда. Для цієї проекції задамо 2 кути повороту відносно осі X та Y.

Лістинг коду:

```
# Аксонометрія
def dimetri(figure, teta_x, teta_y):
    teta_rx = (3 / 14 * teta_x) / 180
    teta_ry = (3 / 14 * teta_y) / 180

    f_1 = np.array(
        [[mt.cos(teta_rx),  0,  -mt.sin(teta_rx),   0],
         [0,                1,  0,                  0],
         [mt.sin(teta_rx),  0,  mt.cos(teta_rx),    1],
         [0,                0,  0,                  0]])
    ft_1 = f_1.T
    pr_xy_1 = figure.dot(ft_1)

    f_2 = np.array(
        [[1,    0,                  0,                  0],
         [0,    mt.cos(teta_ry),    mt.sin(teta_ry),    0],
         [0,    -mt.sin(teta_ry),   mt.cos(teta_ry),    0],
         [0,    0,                  0,                  1]])
    ft_2 = f_2.T
    pr_xy_2 = pr_xy_1.dot(ft_2)

    print('\nДиметрія')
    print(pr_xy_2)
```

```
    return pr_xy_2

# Блок головних викликів
if __name__ == '__main__':

    # Константи
    [...]

    teta_x = 180
    teta_y = -90

    # Головні виклики
    [...]

    angle = 0
    i = 0
    while not win.isClosed():

        [...]

        # Диметрія
        prlpd_2 = dimetri(prlpd_rotated, teta_x, teta_y)
```

Результат:

У результаті отримуємо діметричну проекцію.

```
Обертання навколо X
[[0. 0. 0. 1.]
 [2. 0. 0. 1.]
 [2. 1. 0. 1.]
 [0. 1. 0. 1.]
 [0. 0. 2. 1.]
 [2. 0. 2. 1.]
 [2. 1. 2. 1.]
 [0. 1. 2. 1.]]

Диметрія
[[ 0.         -0.10693798  0.99426569  0.        ]
 [ 1.95425707 -0.15241861  1.41712596  0.        ]
 [ 1.95425707  0.84184709  1.52406395  0.        ]
 [ 0.          0.88732771  1.10120367  0.        ]
 [-0.42529907 -0.31592229  2.93731645  0.        ]
 [ 1.528958   -0.36140291  3.36017673  0.        ]
 [ 1.528958    0.63286278  3.46711471  0.        ]
 [-0.42529907  0.6783434   3.04425444  0.        ]]
```

Рисунок 12 – Діметрична проекція паралелепіпеда

Проекція на XY

За такої проекції нівелюється значення z-ї координати. Важливо розуміти, що подальше відновлення z-ї координати без наявності додаткової інформації неможливе.

Лістинг коду:

```
# Проекція на XY (Z=0)
def project_xy(figure):
   f = np.array(
       [[1, 0, 0, 0],
        [0, 1, 0, 0],
        [0, 0, 0, 0],
        [0, 0, 0, 1]])
   ft = f.T
   pr_xy = figure.dot(ft)

   print('\nПроекція на XY')
   print(pr_xy)

   return pr_xy

# Блок головних викликів
if __name__ == '__main__':

    # Константи
    [...]

    # Головні виклики
    [...]

    angle = 0
    i = 0
    while not win.isClosed():

        [...]

        # Проекція на XY
        pr_xy_3 = project_xy(prlpd_2)
```

Результат:

Бачимо, що порівняно з попереднім кроком (діметрією), -ова координата зникає.

```
Диметрія
[[ 0.         -0.10693798  0.99426569  0.        ]
 [ 1.95425707 -0.15241861  1.41712596  0.        ]
 [ 1.95425707  0.84184709  1.52406395  0.        ]
 [ 0.          0.88732771  1.10120367  0.        ]
 [-0.42529907 -0.31592229  2.93731645  0.        ]
 [ 1.528958   -0.36140291  3.36017673  0.        ]
 [ 1.528958    0.63286278  3.46711471  0.        ]
 [-0.42529907  0.6783434   3.04425444  0.        ]]
```

```
Проекція на XY
[[ 0.         -0.10693798  0.          0.        ]
 [ 1.95425707 -0.15241861  0.          0.        ]
 [ 1.95425707  0.84184709  0.          0.        ]
 [ 0.          0.88732771  0.          0.        ]
 [-0.42529907 -0.31592229  0.          0.        ]
 [ 1.528958   -0.36140291  0.          0.        ]
 [ 1.528958    0.63286278  0.          0.        ]
 [-0.42529907  0.6783434   0.          0.        ]]
```

Рисунок 13 – Проекція паралелепіпеда на XY

Масштабування та зміщення

Масштабування та зміщення важливо робити після обертання об’єкту, оскільки інший порядок дій може призвести до викривлення траєкторії руху.

Виконаємо масштабування та зміщення одночасно. Відмасштабуємо паралелепіпед у 100 разів для кращої візуалізації результатів та змістимо його на центр графічного вікна.

<u>Лістинг коду:</u>

```
# Масштабування та зміщення
def scale_and_shift(prlpd, st, dx, dy, dz):
    # Створення пустої матриці для зберігання результату
    pr_xy = np.zeros_like(prlpd)

    for i in range(len(prlpd)):
        # Масштабування кожної точки
        pr_xy[i][0] = prlpd[i][0] * st + dx
        pr_xy[i][1] = prlpd[i][1] * st + dy
        pr_xy[i][2] = prlpd[i][2] * st + dz
        pr_xy[i][3] = prlpd[i][3]  # Зберігаємо останній елемент без змін

    print('\nМастабування та зміщення')
    print(pr_xy)

    return pr_xy

# Блок головних викликів
if __name__ == '__main__':

    # Константи
    [...]

    # Обчислення зсуву для центру вікна
    dx = (xw - st) / 2
    dy = (yw - st) / 2
    dz = dy
```

```
    # Головні виклики
    [...]

    angle = 0
    i = 0
    while not win.isClosed():

        [...]

        # Масштабування та зміщення фігури
        pr_xy_3_scaled_center = scale_and_shift(pr_xy_3, st, dx, dy, dz)
```

Результат:

Бачимо, що порівняно з попереднім кроком (проекцією), відбулось успішне масштабування та зміщення фігури.

```
Проекція на XY
[[ 0.         -0.10693798  0.          0.        ]
 [ 1.95425707 -0.15241861  0.          0.        ]
 [ 1.95425707  0.84184709  0.          0.        ]
 [ 0.          0.88732771  0.          0.        ]
 [-0.42529907 -0.31592229  0.          0.        ]
 [ 1.528958   -0.36140291  0.          0.        ]
 [ 1.528958    0.63286278  0.          0.        ]
 [-0.42529907  0.6783434   0.          0.        ]]

Мастабування та зміщення
[[250.         239.30620179 250.           0.        ]
 [445.4257072  234.75813939 250.           0.        ]
 [445.4257072  334.18470868 250.           0.        ]
 [250.         338.73277108 250.           0.        ]
 [207.47009327 218.40777102 250.           0.        ]
 [402.89580047 213.85970862 250.           0.        ]
 [402.89580047 313.28627791 250.           0.        ]
 [207.47009327 317.83434031 250.           0.        ]]
```

Рисунок 14 – Масштабування та зміщення паралелепіпеду

Візуалізація об'єкту

Об'єкт задається як список точок з координатами по осях. Для графічної візуалізації нам необхідно лишити лише x, y координати. Це завдання ми вже виконали в попередніх кроках.

Для візуалізації задамо з точок площини, які в подальшому розфарбуємо в різні кольори. Ці площини – об'єкти з якими ми будемо працювати для візуалізації.

Лістинг коду:

```
# Побудова паралелепіпеда
def prlpd_visualisation(win, pr_xy, color):
    Ax = pr_xy[0, 0]
    Ay = pr_xy[0, 1]
    Bx = pr_xy[1, 0]
    By = pr_xy[1, 1]
    Ix = pr_xy[2, 0]
    Iy = pr_xy[2, 1]
    Mx = pr_xy[3, 0]
    My = pr_xy[3, 1]

    Dx = pr_xy[4, 0]
    Dy = pr_xy[4, 1]
    Cx = pr_xy[5, 0]
    Cy = pr_xy[5, 1]
    Fx = pr_xy[6, 0]
    Fy = pr_xy[6, 1]
    Ex = pr_xy[7, 0]
    Ey = pr_xy[7, 1]

    # print(f'A ({Ax}, {Ay})'); print(f'B ({Bx}, {By})'); print(f'I ({Ix}, {Iy})');
print(f'M ({Mx}, {My})');
    # print(f'D ({Dx}, {Dy})'); print(f'C ({Cx}, {Cy})'); print(f'F ({Fx}, {Fy})');
print(f'E ({Ex}, {Ey})');

    obj_6 = Polygon(Point(Bx, By), Point(Cx, Cy), Point(Fx, Fy), Point(Ix, Iy))
    obj_6.setFill(color)
    obj_6.draw(win)

    obj_5 = Polygon(Point(Mx, My), Point(Ax, Ay), Point(Dx, Dy), Point(Ex, Ey))
    obj_5.setFill(color)
    obj_5.draw(win)

    # obj_4 = Polygon(Point(Mx, My), Point(Ix, Iy), Point(Fx, Fy), Point(Ex, Ey))
    # obj_4.setFill(color)
    # obj_4.draw(win)

    obj_3 = Polygon(Point(Ax, Ay), Point(Bx, By), Point(Cx, Cy), Point(Dx, Dy))
    obj_3.setFill(color)
    obj_3.draw(win)

    # obj_2 = Polygon(Point(Dx, Dy), Point(Cx, Cy), Point(Fx, Fy), Point(Ex, Ey))
    # obj_2.setFill(color)
    # obj_2.draw(win)

    obj_1 = Polygon(Point(Ax, Ay), Point(Bx, By), Point(Ix, Iy), Point(Mx, My))
    obj_1.setFill(color)
    obj_1.draw(win)

    return [obj_1, obj_3, obj_5, obj_6]

# Блок головних викликів
if __name__ == '__main__':

    # Константи
    [...]

    # Змінні для зберігання кольорів
    colors = ['gray', 'orange']

    # Обчислення зсуву для центру вікна
```

```
dx = (xw - st) / 2
dy = (yw - st) / 2
dz = dy


# Головні виклики
[...]

angle = 0
i = 0
while not win.isClosed():

    [...]

    # Візуалізація фігури
    objs = prlpd_visualisation(win, pr_xy_3_scaled_center, colors[i])
```

Результат:

У результаті бачимо наш паралелепіпед.

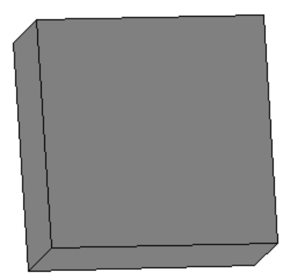

Рисунок 15 – Візуалізація паралелепіпеду

Миготіння об’єкту

Для того, щоб паралелепіпед змінював колір та блимав, необхідно створити цикл у якому всі отримані з функції об’єкти будуть змінювати колір відповідно до списку кольорів, заримуватись у вікні і зникати (очищатись).

Лістинг коду:

```
win.update()  # Оновлюємо вікно

# Збереження зображення з вікна
frame = get_image(win)
frames.append(frame)

i += 1
```

```
    if i >= len(colors):
        i = 0
    angle += 60
    time.sleep(0.1)  # Затримка для анімації

    for obj in objs:  # Закриваємо об'єкти
        obj.undraw()
    time.sleep(0.2)  # Затримка для анімації

    # Збереження зображення з вікна
    frame = get_image(win)
    frames.append(frame)

    if win.checkMouse():
        win.close()

# Зберігаємо анімацію у форматі GIF
imageio.mimsave("animation.gif", frames)
```

Результат:

У результаті бачимо безперервну анімацію обертання діметричної проекції паралелепіпеду. Паралелепіпед змінює колір, зникає і повертається на екран з іншим кольором заливки.

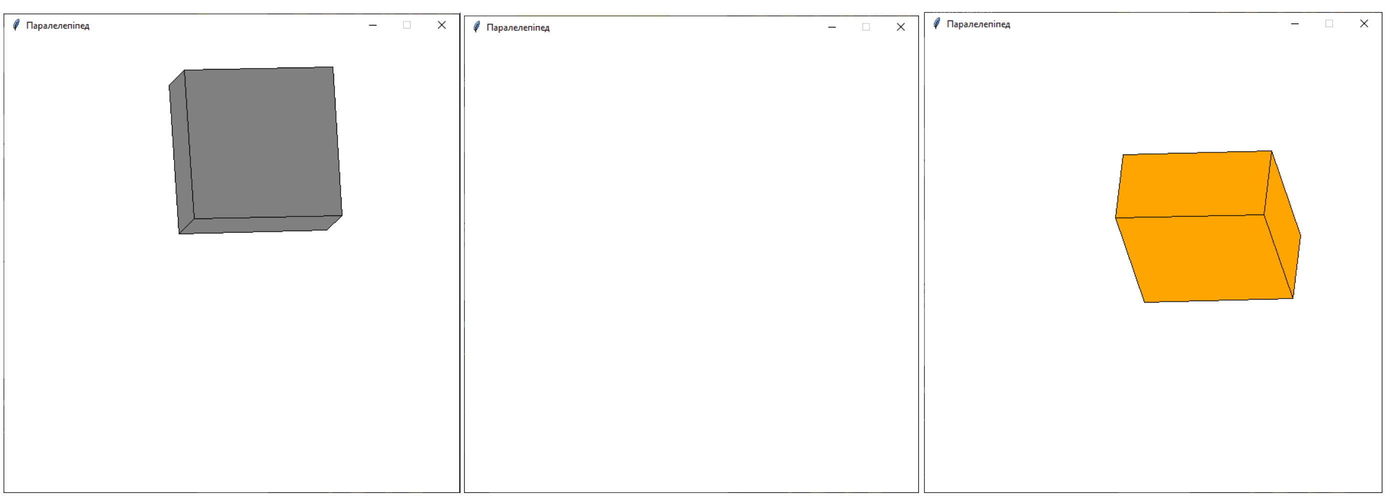

Рисунок 16 – Візуалізація миготіння паралелепіпеду

Для того, щоб наглядніше зобразити результати збережемо результат виконання програми у вікні в форматі **.gif**.

Повні записи анімації можна переглянути у папці проекту на github.

2.3. Аналіз отриманих результатів

Синтезовано математичні моделі для операцій 2D обертання, переміщення та масштабування – підтверджено формулами (1.1-3):

$$R(\theta) = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{1.1}$$

$$T(Dx, Dy) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Dx & Dy & 1 \end{bmatrix}, \tag{1.2}$$

$$S(Sx, Sy) = \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{1.3}$$

Реалізовано скрипкове рішення для синтезованих моделей над 2D фігурою ромба – підтверджено рисунками.

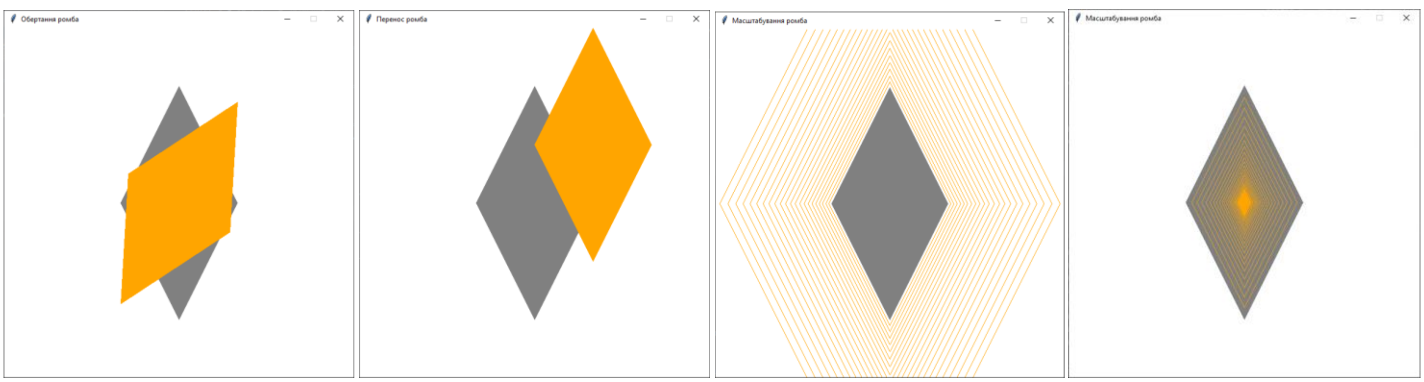

Рисунок 17 – Підтвердження програмної реалізації 2D обертання, переміщення та масштабування

Синтезовано математичні моделі для операцій 3D обертання, діметрії, ортогональної проекції, масштабування та переміщення – підтверджено формулами (1.4-8):

$$T_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & sin\theta & 0 \\ 0 & -sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{1.4}$$

$$A' = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \cdot \begin{bmatrix} cos\varphi & 0 & -sin\varphi & 0 \\ 0 & 1 & 0 & 0 \\ sin\theta & 0 & cos\varphi & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\varphi & sin\varphi & 0 \\ 0 & -sin\varphi & cos\varphi & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \tag{1.5}$$

$$T_{xy}^{ort} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{1.6}$$

$$A \cdot T = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \cdot \begin{bmatrix} 1/s & 0 & 0 & 0 \\ 0 & 1/s & 0 & 0 \\ 0 & 0 & 1/s & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} \frac{x}{s} & \frac{y}{s} & \frac{z}{s} & 1 \end{bmatrix}, \tag{1.7}$$

$$A' = A \cdot T = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ l & m & n & 1 \end{bmatrix} = \begin{bmatrix} x+l & y+m & z+n & 1 \end{bmatrix}. \tag{1.8}$$

Реалізовано скрипкове рішення для синтезованих моделей над 3D фігурою паралелепіпеда, що циклічно обертається, змінює колір і гасне – підтверджено рисунками.

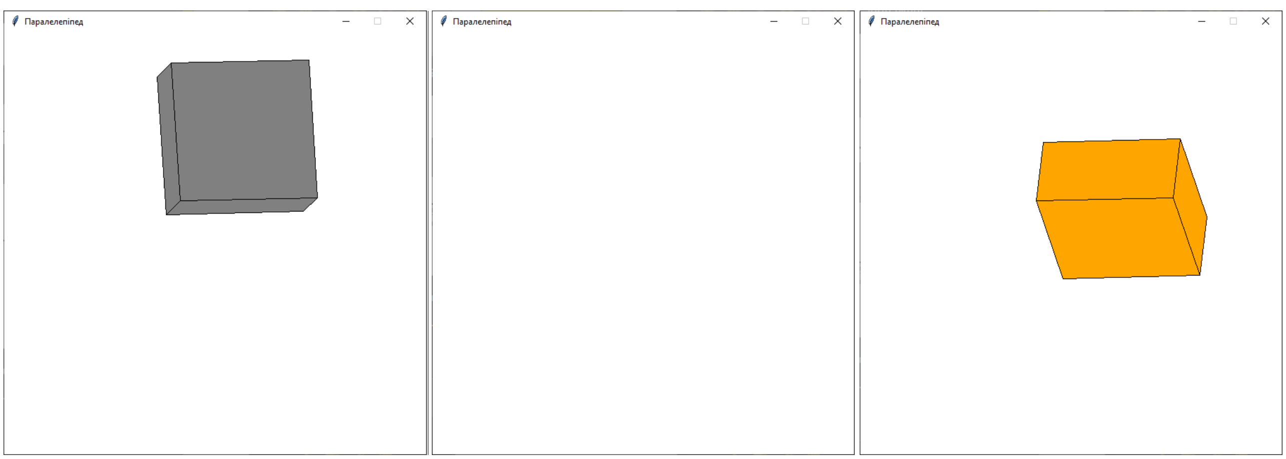

Рисунок 18 – Підтвердження програмної реалізації циклічного 3D обертання, зміни кольору та згасання фігури

Висновок:

У результаті виконання лабораторної роботи отримано практичні та теоретичні навички формування та перетворення координат площинних (2d) та просторових (3d) об’єктів.

Синтезовано та реалізовано програмно математичні моделі базових 2D перетворень у матричному варіанті: обертання, переміщення, масштабування.

Синтезовано та реалізовано програмно математичні моделі 3D перетворень: обертання, проекції, масштабування, переміщення.

Реалізовано 2 програми, відповідно до індивідуальних завдань, які виконують анімаційні перетворення 2D та 3D об’єктів.