Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

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 ІМ-21
Кривохата Марія Юріївна
Номер у списку групи: 12

Перевірив: Баран Д. Р.

Київ 2024

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

Завдання ІІІ рівня: (максимум 9 балів)

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.

20	Реалізувати операції: **масштабування - (переміщення+обертання)**. **2.** операцію реалізувати циклічно, траєкторію зміни положення цієї операції **скрити.** 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.	Пентагон (п'ятикутник)

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.

20	**Динаміка фігури**: графічна фігура з'являється та гасне, змінює колір контуру. **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.	Піраміда з трикутною основою

Хід роботи

2d-фігура:

Синтезована математична модель:

<u>Масштабування</u>

Реалізується за моделлю:

$x' = x * Sx$, $y' = y * Sy$, де Sx , Sy - масштабні коефіцієнти (константи).

$$[x', y', 1] = [x, y, 1] * \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$P' = P * S(S_x, S_y)$ - математична модель масштабування.

Обертання

Обертання реалізується з використанням напрямних косинусів та синусів за моделлю:

$x' = x * \cos\theta - y * \sin\theta$,

$y' = y * \sin\theta + y * \cos\theta$,

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

$P' = P * R(\theta)$ - математична модель обертання

R - матриця обертання, або матриця напрямних косинусів та синусів.

Переміщення

$x' = x + Dx$, $y' = y + Dy$

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

$P' = P * T(Dx, Dy)$ - математична модель переміщення.

Результати архітектурного проектування та їх опис:

Масштабування об'єкта та його поворот відносно *свого центра* (не відносно лівого верхнього кута) під час переміщення було реалізовано за таким алгоритмом:

1. **Початок програми**
2. **Визначення параметрів вікна для зображення масштабування та параметрів п'ятикутника**
3. **Ініціалізація вікна**
4. **Обчислення координат точок п'ятикутника**
   Цей крок реалізовується за допомогою базових знань про геометричні співвідношення п'ятикутника. Знаючи, що кожну наступну точку необхідно мадювати, повертаючись на 72 градуси, та знаючи радіус, можемо за допомогою тригонометричних операцій визначити координати точок. Дефолтні точки п'ятикутника визначаються із розрахунком на те, що (0,0) знаходиться у центрі вікна, а не зліва зверху. Зроблено це для подальшої зручності операцій.
5. **Очищення вікна**
6. **Визначення матриці переносу п'ятикутника до (0,0) – зліва зверху**
   Тут ми ставимо центр нашої фігури у лівий верхній кут. Цей крок зроблено, щоб фігура в подальшому розширювалась відносно свого центру.
7. **Визначення матриці масштабування (збільшення/зменшення)**
   В параметри операції передається коефіцієнт і на цей коефіцієнт масштабується п'ятикутник (коли фігура починає виходити за рамки – від'ємний коефіцієнт зменшення, коли стає занадто малою – додатній коефіцієнт збільшення).

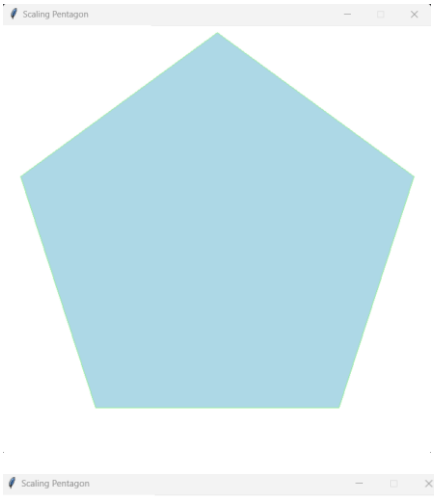
8. **Визначення матриці переносу п’ятикутника назад до центру вікна**
9. **Комбінування перетворень та застосування їх до точок п’ятикутника**
   Тобто тут ми кажемо, що хочемо спочатку перенестись у (0,0), потім збільшитись, а потім повернутись назад. Якщо ми викинули б перенесення, то фігура масштабувалась би “вбік”.
10. **Малювання п’ятикутника**
11. **Закриття вікна із масштабуванням**
12. **Ініціалізація другого вікна із перенесенням+обертом**
13. **Обчислення координат точок п’ятикутника**
14. **Очищення вікна**
15. **Обчислення поточного центру п’ятикутника**
16. **Визначення матриці переносу п’ятикутника до (0,0) – зліва зверху**
17. **Визначення матриці повороту**
    Залежно від траєкторії – вліво або вправо. При цьому п’ятикутник буде обертатись довкола *<u>свого</u>* центру, а не довкола (0,0) (лівий верхній кут).
18. **Визначення матриці переміщення фігури до початкового свого положення**
    Так як для коректного повороту переставляли фігуру у (0,0), то необхідно її перемістити назад.
19. **Визначення матриці переміщення фігури на заданий «крок»**
    Коли закінчили маніпуляції із переміщенням для коректного повороту, то тільки тоді можемо робити переміщення, яке вже є не допоміжним, а дійсно візуально рухатиме нашу фігуру по екрану.
20. **Комбінування перетворень та застосування їх до точок п’ятикутника**
    Переміщення до (0,0) => поворот на заданий кут => переміщення назад до свого початкового положення => переміщення на заданий «крок».
21. **Малювання п’ятикутника**
22. **Закриття вікна із операцією перенесення+оберт**
23. **Кінець програми**

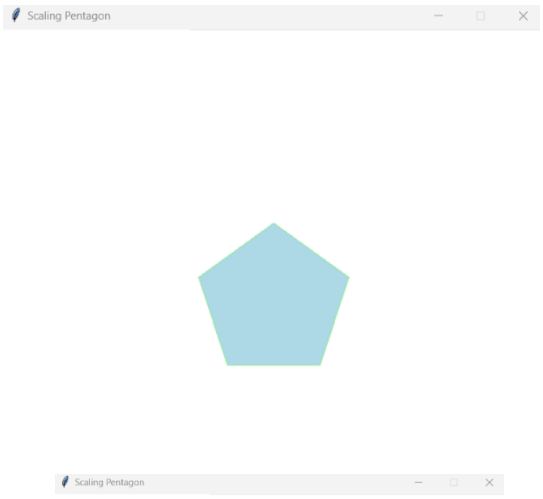
Опис структури проекту програми в середовищі PyCharm:

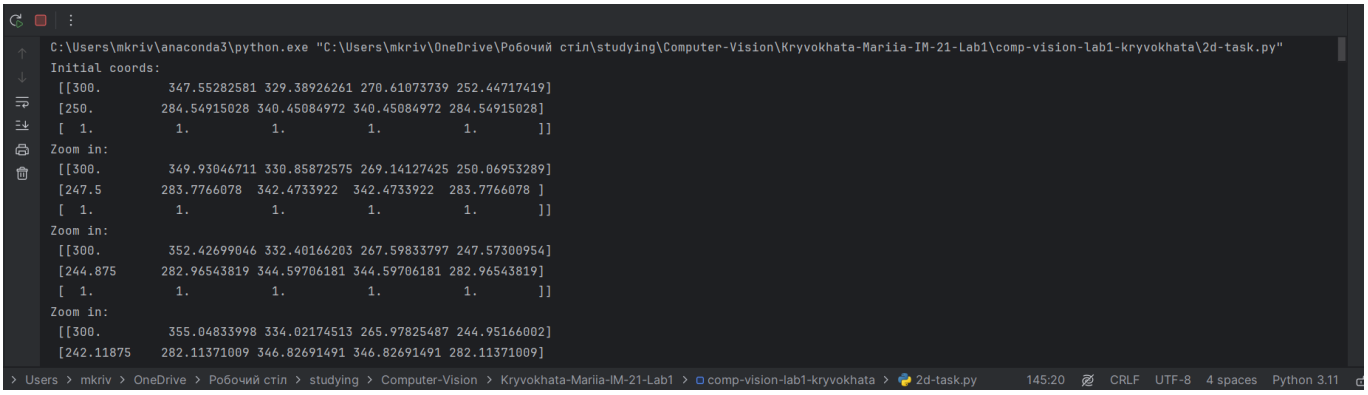
Проєкт має просту структуру – один файл 2d-task.py. Він в свою чергу має залежності від бібліотек, що використовуються у ході роботи: graphics (для того, щоб відмальовувати фігуру та відображати вікно), numpy (для математичних маніпуляцій із матрицями), time (для виставлення очікування перед малюванням кожного наступного фрейму).

Результати роботи програми:

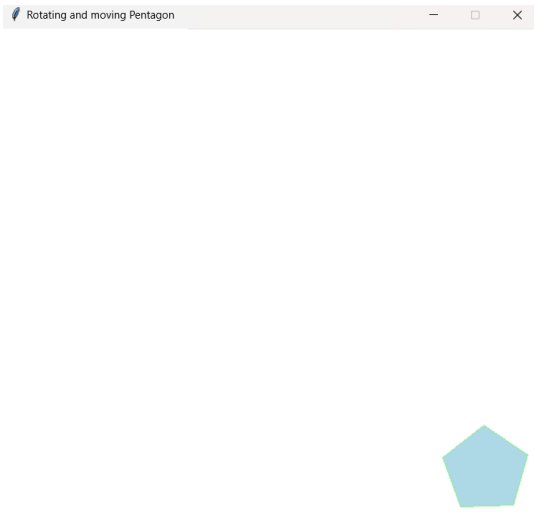
Масштабування:







Поворот+переміщення:




```
Initial coords:
 [[300.         347.55282581 329.38926261 270.61073739 252.44717419]
 [250.         284.54915028 340.45084972 340.45084972 284.54915028]
 [  1.           1.           1.           1.           1.        ]]
Rotated and translated:
 [[277.79894446 256.6941893  302.34642268 351.66580972 336.49463384]
 [345.95357645 291.09462697 254.07046438 286.04722297 342.83410922]
 [  1.           1.           1.           1.           1.        ]]
Rotated and translated:
 [[355.64726254 343.51123507 285.06381974 261.07735798 304.70032467]
 [287.59589691 345.10791188 351.33805388 297.67647843 258.2816589 ]
 [  1.           1.           1.           1.           1.        ]]
Rotated and translated:
 [[265.5984188  307.06916419 359.50005511 350.43338237 292.39897953]
 [304.28742751 262.63298831 289.20208135 347.27712309 356.60037975]
 [  1.           1.           1.           1.           1.        ]]
Rotated and translated:
 [[357.25565802 299.79784199 270.25872168 309.46035736 363.22742094]
 [349.34690308 361.73699609 310.92011505 267.12346235 290.87252343]
 [  1.           1.           1.           1.           1.        ]]
Rotated and translated:
 [[311.88125731 366.83294704 363.9729258  307.25364575 275.0592241 ]
 [271.75169858 292.61378919 351.32269232 366.74469928 317.56712063]
 [  1.           1.           1.           1.           1.        ]]
```

Програмний код, що забезпечує отримання результату:

```
from graphics import *
import numpy as np
import time

def init_window(text, x_wind, y_wind):
    wind = GraphWin(text, x_wind, y_wind)
    wind.setBackground('white')
    return wind


def init_pentagon_points(radius, center_x, center_y):
    # Pentagon coords. We make a turn of 72 to draw a vertice

    # top center
    x1 = center_x
    y1 = center_y - radius

    # top right
    top_angle = 90 - 72
    x2 = center_x + (radius * np.cos(np.radians(top_angle)))
    y2 = center_y - (radius * np.sin(np.radians(top_angle)))

    # bottom right
    bottom_angle = 72 - top_angle
    x3 = center_x + (radius * np.cos(np.radians(bottom_angle)))
    y3 = center_y + (radius * np.sin(np.radians(bottom_angle)))

    # bottom left
    x4 = center_x - (radius * np.cos(np.radians(bottom_angle)))
    y4 = y3

    # top left
    x5 = center_x - (radius * np.cos(np.radians(top_angle)))
    y5 = y2

    points = np.array([[x1, x2, x3, x4, x5],
                       [y1, y2, y3, y4, y5],
                       [1, 1, 1, 1, 1]])
    return points

def draw_pentagon(wind, points, transformation_matrix=None):

    if transformation_matrix is not None:
        points = np.dot(transformation_matrix, points)

    # Drawing default pentagon
    pentagon = Polygon([Point(p[0], p[1]) for p in points.T])
```

```
    pentagon.setOutline("lightgreen")
    pentagon.setFill("lightblue")
    pentagon.draw(wind)

    return pentagon, points

def is_in_frame(wind, points):
    width = wind.getWidth()
    height = wind.getHeight()
    return all(0 <= point[0] <= width and 0 <= point[1] <= height for point in points.T)


def scaling_matrix_func(scale_coef):
    return np.array([
        [scale_coef, 0, 0],
        [0, scale_coef, 0],
        [0, 0, 1]
    ])

def translation_matrix_func(dx, dy):
    return np.array([
        [1, 0, dx],
        [0, 1, dy],
        [0, 0, 1]
    ])

def rotation_matrix_func(angle_rad):
    return np.array([
        [np.cos(angle_rad), -np.sin(angle_rad), 0],
        [np.sin(angle_rad), np.cos(angle_rad), 0],
        [0, 0, 1]
    ])


def scale_pentagon(wind, points, center_x, center_y):
    scale_k = 1.05
    min_radius = 7
    scale_text = 'Zoom in:\n'

    while not wind.isClosed():

        wind.delete('all')

        # Translating pentagon to (0, 0)
        translate_to_origin = translation_matrix_func(-center_x, -center_y)

        # Scaling matrix
        scaling = scaling_matrix_func(scale_k)
```

```
        # Translation matrix to translate the pentagon back to its original position
        translate_back = translation_matrix_func(center_x, center_y)

        # Combining transformations: T_back * Scale * T_origin
        transform_matrix = np.dot(translate_back, np.dot(scaling, translate_to_origin))
        pentagon, points = draw_pentagon(wind, points, transform_matrix)
        print(scale_text, points)

        in_frame = is_in_frame(wind, points)
        current_radius = np.linalg.norm(np.array(points.T[0][:2]) - np.array([center_x, center_y]))

        if not in_frame:
            scale_k = 0.9
            scale_text = 'Zoom out\n'
        elif current_radius <= min_radius:
            scale_k = 1.05
            scale_text = 'Zoom in\n'

        time.sleep(0.1)
        wind.update()


def calculate_center(points):
    center_x = np.mean(points[0, :])
    center_y = np.mean(points[1, :])
    return center_x, center_y

def rotate_move_pentagon(wind, points):

    reverse = False

    while not wind.isClosed():

        wind.delete('all')
        pentagon_center_x, pentagon_center_y = calculate_center(points)

        if reverse:
            angle = -10
            dx = -5  # left
            dy = -4  # up
        else:
            angle = 10
            dx = 5  # right
            dy = 4  # down

        # Translating to (0,0)
        translate_to_origin = translation_matrix_func(-pentagon_center_x, -pentagon_center_y)

        #  Rotation
```

```
        rotation_matrix = rotation_matrix_func(angle)

        # Translating pentagon back to its original position
        translate_back = translation_matrix_func(pentagon_center_x, pentagon_center_y)

        # Moving pentagon
        move_matrix = translation_matrix_func(dx, dy)

        # Combining transformations: Move * T_back * Rotate * T_origin
        transform_matrix = np.dot(move_matrix, np.dot(translate_back, np.dot(rotation_matrix,
translate_to_origin)))

        pentagon, points = draw_pentagon(wind, points, transform_matrix)
        print('Rotated and translated:\n', points)

        if not is_in_frame(wind, points):
            reverse = not reverse

        time.sleep(0.01)
        wind.update()


if __name__ == '__main__':

    xw = 600
    yw = 600

    default_radius = 50

    center_x = xw / 2
    center_y = yw / 2

    # Scaling
    wind = init_window('Scaling Pentagon', xw, yw)
    points = init_pentagon_points(default_radius, center_x, center_y)
    print('Initial coords:\n', points)
    scale_pentagon(wind, points, center_x, center_y)

    # Rotating + moving
    wind2 = init_window('Rotating and moving Pentagon', xw, yw)
    points2 = init_pentagon_points(default_radius, center_x, center_y)
    print('Initial coords:\n', points2)
    rotate_move_pentagon(wind2, points2)
```

3d-програма:

Синтезована математична модель:

Обертання

Обертання забезпечує сегмент T_{11}[3 x 3] матриці перетворень T .

Довкола осі x:

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

Довкола осі y:

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

Аксонометрична проєкція

Узагальнена математична модель аксонометричної проєкції:

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

Проєкція на xy

Для зображення 3д об’єкта на площині використовуватиметься проєкція на площину xy. Відповідно маємо таку математичну модель ортогональної проєкції:

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

Результати архітектурного проектування та їх опис:

1. **Початок програми**
2. **Визначення параметрів вікна для зображення аксонометричної проєкції та параметрів піраміди**
3. **Ініціалізація першого вікна**
4. **Обчислення координат точок піраміди**
5. **Визначення матриці повороту довкола осі x та застосування її до точок піраміди**
6. **Визначення матриці повороту довкола осі y та застосування її до точок піраміди**
   Цей поворот «накладається» на минулий, тому ми по суті повертаємо і відносно x, і відносно y. Так як повертаємо на однаковий кут і там, і там, то виходить діметрична проєкція, що є одним із різновидів аксонометричної проекції.
7. **Переміщення піраміди у центр вікна**
8. **Проєкція піраміди на площину xy**
   Проєкція необхідня для того, щоб ми мали змогу коректно відобразити об’ємну фігуру на площині.

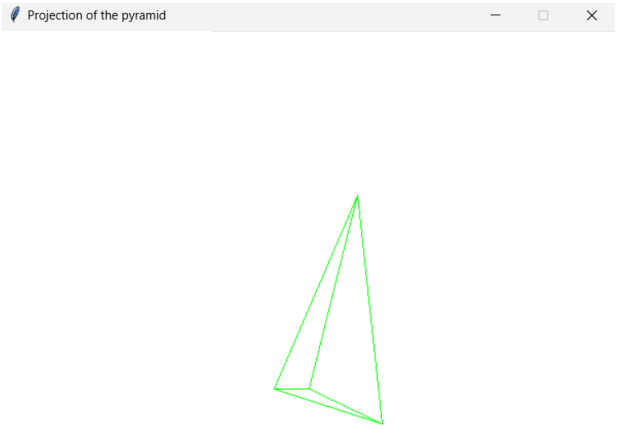
9. **Зображення піраміди**
10. **Ініціалізація другого вікна**
11. **Визначення параметрів піраміди**
12. **Переміщення піраміди до центру вікна**
13. **Очищення вікна**
14. **Визначення, чи фігура на даний момент має бути видимою та її колір**
    Кожні 30 кадрів фігура зникає або з’являється, змінюючи колір.
15. **Визначення поточних трьох координат центру піраміди**
16. **Переміщення піраміди до (0,0)**
    Це переміщення робиться із метою коректного подальшого відображення повороту піраміди, коли вона стоятиме у центрі вікна (не довкола лівого боку вікна, а довкола лінії, що паралельна осі y та проходить безпосередньо «через піраміду»).
17. **Поворот піраміди на заданий кут навколо осі y**
18. **Переміщення піраміди назад до центру вікна**
19. **Проєкція піраміди на площину xy**
    Проєкція необхідня для того, щоб ми мали змогу коректно відобразити об’ємну фігуру на площині.
20. **Зображення піраміди**
21. **Зміна параметрів повороту та рахунку кадрів**
22. **Закриття обох вікон**
23. **Кінець програми**

Опис структури проекту програми в середовищі PyCharm:

Аналогічно до минулого завдання, проєкт має просту структуру – один файл 3d-task.py. Він в свою чергу має залежності від бібліотек, що використовуються у ході роботи: graphics (для того, щоб відмальовувати фігуру та відображати вікно), random (для генерації рандомного кольору), numpy (для математичних маніпуляцій із матрицями), time (для виставлення очікування перед малюванням кожного наступного фрейму).

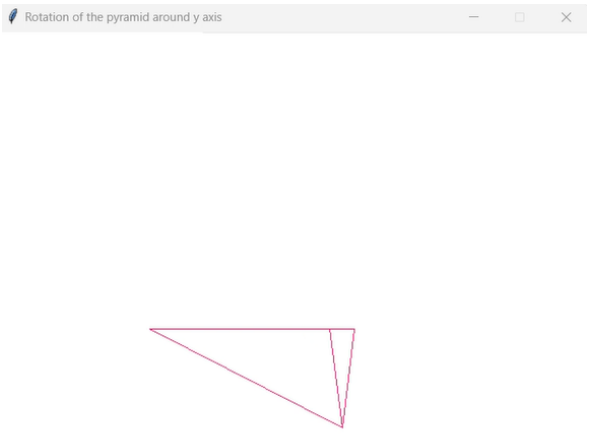
Результати роботи програми:

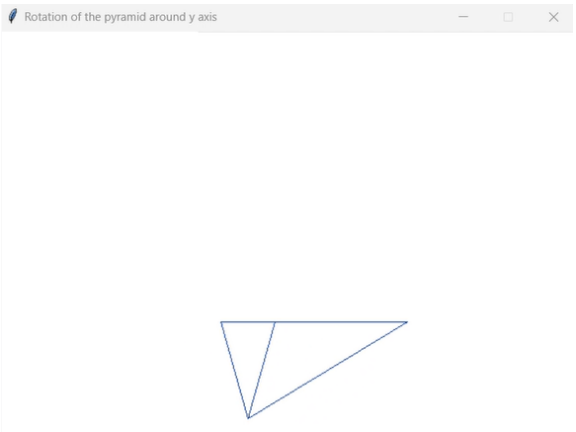
1. **Аксонометрична проєкція**

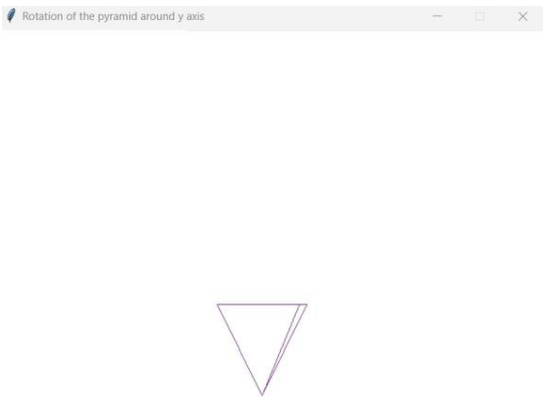


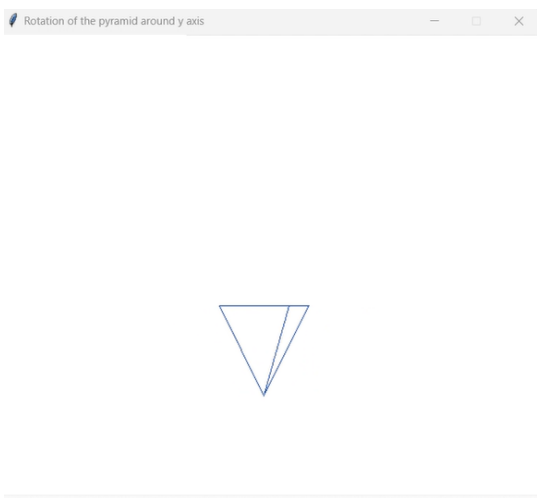

```
Initial pyramid:
 [[  0.   0. 150.   1.]
 [ 50.   0. -50.   1.]
 [-50.   0. -50.   1.]
 [  0. 100. -50.   1.]]
Axonometric projection:
 [[  48.20907073 -140.95389312   17.54666677    1.        ]
 [   1.03131692   46.98463104  -52.83351996    1.        ]
 [ -33.17069741   46.98463104   41.13574212    1.        ]
 [  72.23253191   81.18664537   26.29049156    1.        ]]
Centered:
 [[348.20907073 159.04610688  17.54666677   1.        ]
 [301.03131692 346.98463104 -52.83351996   1.        ]
 [266.82930259 346.98463104  41.13574212   1.        ]
 [372.23253191 381.18664537  26.29049156   1.        ]]
Projection on xy:
 [[348.20907073 159.04610688   0.           1.        ]
 [301.03131692 346.98463104   0.           1.        ]
 [266.82930259 346.98463104   0.           1.        ]
 [372.23253191 381.18664537   0.           1.        ]]
```

2. **Циклічне обертання фігури, що згасає та з’являється, змінюючи колір**










```
Initial pyramid centered:
 [[  0.   0. 150.   1.]
 [ 50.   0. -50.   1.]
 [-50.   0. -50.   1.]
 [  0. 100. -50.   1.]]
Rotated and centered:
 [[170.09618943 300.          75.           1.        ]
 [368.30127019 300.          18.30127019   1.        ]
 [318.30127019 300.         -68.30127019   1.        ]
 [343.30127019 400.         -25.           1.        ]]
Rotated, centered and projected:
 [[170.09618943 300.           0.           1.        ]
 [368.30127019 300.           0.           1.        ]
 [318.30127019 300.           0.           1.        ]
 [343.30127019 400.           0.           1.        ]]
Rotated and centered:
 [[167.55786107 300.          70.42073442   1.        ]
 [367.62095778 300.          20.6738015    1.        ]
 [320.6738015  300.         -67.62095778   1.        ]
 [344.14737964 400.         -23.47357814   1.        ]]
Rotated, centered and projected:
 [[167.55786107 300.           0.           1.        ]
 [367.62095778 300.           0.           1.        ]
 [320.6738015  300.           0.           1.        ]
 [344.14737964 400.           0.           1.        ]]
Rotated and centered:
 [[165.18089306 300.          65.75567202   1.        ]
 [366.85825965 300.          23.02114498   1.        ]
```

Програмний код, що забезпечує отримання результату:

```
from graphics import *
import random
import numpy as np
import time
```

```
def init_window(text, x_wind, y_wind):
    wind = GraphWin(text, x_wind, y_wind)
    wind.setBackground('white')
    return wind

def init_pyramid_points(base_size, height):
    half_base = base_size / 2
    return np.array([
        [0, 0, height, 1],  # Apex
        [half_base, 0, -half_base, 1],  # Base vertices
        [-half_base, 0, -half_base, 1],
        [0, base_size, -half_base, 1]
    ])


def project_xy(pyramid):
    projection_matrix = np.array([
        [1, 0, 0, 0],
        [0, 1, 0, 0],
        [0, 0, 0, 0],
        [0, 0, 0, 1]
    ])
    return pyramid.dot(projection_matrix.T)


def shift_xyz(pyramid, dx, dy, dz):
    shift_matrix = np.array([
        [1, 0, 0, dx],
        [0, 1, 0, dy],
        [0, 0, 1, dz],
        [0, 0, 0, 1]
    ])
    return pyramid.dot(shift_matrix.T)


def rotate_x(pyramid, angle_degree):
    angle_rad = np.radians(angle_degree)
    rotation_matrix = np.array([
        [1, 0, 0, 0],
        [0, np.cos(angle_rad), np.sin(angle_rad), 0],
        [0, -np.sin(angle_rad), np.cos(angle_rad), 0],
        [0, 0, 0, 1]
    ])
    return pyramid.dot(rotation_matrix.T)


def rotate_y(pyramid, angle_degree):
    angle_rad = np.radians(angle_degree)
    rotation_matrix = np.array([
```

```
        [np.cos(angle_rad), 0, -np.sin(angle_rad), 0],
        [0, 1, 0, 0],
        [np.sin(angle_rad), 0, np.cos(angle_rad), 0],
        [0, 0, 0, 1]
    ])
    return pyramid.dot(rotation_matrix.T)


def draw_pyramid(projection_xy, color, wind):
    x1, y1 = projection_xy[0, 0], projection_xy[0, 1]
    x2, y2 = projection_xy[1, 0], projection_xy[1, 1]
    x3, y3 = projection_xy[2, 0], projection_xy[2, 1]
    x4, y4 = projection_xy[3, 0], projection_xy[3, 1]

    sides = [
        Polygon(Point(x1, y1), Point(x2, y2), Point(x3, y3)),
        Polygon(Point(x1, y1), Point(x2, y2), Point(x4, y4)),
        Polygon(Point(x1, y1), Point(x3, y3), Point(x4, y4)),
        Polygon(Point(x2, y2), Point(x3, y3), Point(x4, y4))
    ]

    for side in sides:
        side.draw(wind)
        side.setOutline(color)


def calculate_center(pyramid_centered):
    center_x = np.mean(pyramid_centered[:, 0])
    center_y = np.mean(pyramid_centered[:, 1])
    center_z = np.mean(pyramid_centered[:, 2])
    return center_x, center_y, center_z


def random_color():
    r = lambda: random.randint(0, 255)
    return '#{:02x}{:02x}{:02x}'.format(r(), r(), r())


def animate_pyramid(pyramid_centered, wind):
    angle = 0
    frame_count = 0
    visible = True

    color = random_color()
    center_x, center_y, center_z = calculate_center(pyramid_centered)

    while not wind.isClosed():

        wind.delete('all')
```

```
        if frame_count % 30 == 0:
            visible = not visible
            if visible:
                color = random_color()

        if visible:
            # Shifting pyramid to (0,0)
            pyramid_shifted_to_origin = shift_xyz(pyramid_centered, -center_x, -center_y, -center_z)

            # Rotating
            rotated_pyramid = rotate_y(pyramid_shifted_to_origin, angle)

            # Shifting back
            pyramid_shifted_back = shift_xyz(rotated_pyramid, center_x, center_y, center_z)
            print('Rotated and centered:\n', pyramid_shifted_back)

            # Projecting and drawing the pyramid
            projected_pyramid = project_xy(pyramid_shifted_back)
            draw_pyramid(projected_pyramid, color, wind)
            print('Rotated, centered and projected:\n', projected_pyramid)

        angle += 2
        frame_count += 1
        time.sleep(0.01)
        wind.update()


if __name__ == '__main__':
    xw = 600
    yw = 600

    center_x = xw / 2
    center_y = yw / 2

    base_size = 100
    height = 150

    wind = init_window('Projection of the pyramid', xw, yw)
    pyramid = init_pyramid_points(base_size, height)
    print('Initial pyramid:\n', pyramid)

    rotated_pyramid_x = rotate_x(pyramid, -70)
    rotated_pyramid_xy = rotate_y(rotated_pyramid_x, -70)
    print('Axonometric projection:\n', rotated_pyramid_xy)

    # shifting pyramid to center
    pyramid_centered = shift_xyz(rotated_pyramid_xy, center_x, center_y, 0)
    print('Centered:\n', pyramid_centered)
```

```
projected_pyramid = project_xy(pyramid_centered)

draw_pyramid(projected_pyramid, '#00FF00', wind)
print('Projection on xy:\n', projected_pyramid)

# Animation

wind2 = init_window('Rotation of the pyramid around y axis', xw, yw)
pyramid2 = init_pyramid_points(base_size, height)
pyramid_centered2 = shift_xyz(pyramid2, center_x, center_y, 0)
print('Initial pyramid centered:\n', pyramid)
animate_pyramid(pyramid_centered2, wind2)
```

Висновки: Виконавши лабораторну роботу №1, я виявила та дослідила особливості формування та перетворення координат площинних (2d) та просторових (3d) об’єктів.

В рамках лабораторної роботи було виконано операцію циклічного масштабування п’ятикутника та циклічного повороту+переміщення. У другій частині роботи було реалізовано аксонометричну проекцію піраміди та її циклічне повернення довкола осі y.

Також було розписано математичні моделі та алгоритм виконання всіх програм. Внаслідок проробленої роботи я згадала, як проводити операції перетворення над різниими видами об’єктів, ознайомилась із релевантними інструментами середовища розробки та зрозуміла за яким принципом працює переміщення об’єкту до (0,0) перед його трансформацією та чому воно необхідне.

Особисто я прийшла до висновку, що такі геометричні операції над об’єктами – потужний інструмент і може бути застосований у багатьох сферах, не тільки при невеликій дослідницькій лабораторній роботі.