# «Технології Computer Vision»

## Лабораторна робота №1.

***Виконав: студент групи ІП-11, Лошак Віктор Іванович***

***Перевірив: Писарчук О.О.***

17.02.2024

**Тема роботи: ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ПОБУДОВИ ТА ПЕРЕТВОРЕННЯ КООРДИНАТ ПЛОЩИННИХ (2D) ТА ПРОСТОРОВИХ (3D) ОБ'ЄКТІВ**

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

**Варіант 15**

### Завдання III рівня – максимально 9 балів.

**Реалізувати розробку програмного скрипта із функціоналом I та II рівнів складності.**:

1. Реалізуйте операції: переміщення - обертання - обертання в іншому напрямку Розробіть програмний скрипт, який реалізує основні операції 2D-перетворень на геометричному примітиві - трикутнику. Для розробки використовуйте матричні операції та композиційні перетворення. Матриця координат кутів геометричної фігури повинна бути розширена. Реалізуйте операцію циклічно, приховайте траєкторію зміни його положення. Виберіть: бібліотеку, розмір графічного вікна, розмір фігур, параметри реалізації операцій, колірну схему всіх графічних об'єктів. Усі операції перетворень повинні виконуватися в межах графічного вікна.

2. Розробити програмний скрипт, який реалізує основні операції тривимірних перетворень квадратної піраміди: аксонометричний

проекція будь-якого типу та циклічне обертання (анімація) тривимірного графічного об'єкта навколо вертикальної осі. Для розвитку використовувати матричні операції. Вхідна матриця координат кутів геометричної фігури має бути однорідною. Фігурка повинна з'являтися і зникати, вона повинна з'являтися в різних частинах вікна. Він має випадковим чином змінювати колір свого контуру між появами. Використовуйте tkinter, щоб відобразити фігуру на екрані. Вибирайте самі: розмір вікна, розмір фігури, положення фігури. Усі операції перетворення повинні виконуватись у графічному вікні.

## Перший рівень складності

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import tkinter as tk
```

```
import time
import random
```

In [2]:
```
# triangle_vertices = np.array([[100, 300], [200, 100], [300, 300]])
triangle_vertices = np.array([[120, 320], [100, 90], [310, 280]])
```

In [3]:
```
# def draw_triangle(ax, vertices, color='blue'):
#     polygon = patches.Polygon(vertices, closed=True, color=color, fill=False)
#     ax.add_patch(polygon)
```

In [4]:
```
# fig, ax = plt.subplots()
# draw_triangle(ax, triangle_vertices)
# ax.set_xlim(0, 5)
# ax.set_ylim(0, 5)
# ax.set_aspect('equal')
# plt.show()
```

In [5]:
```
def translate(vertices, tx, ty):
    translation_matrix = np.array([[1, 0, tx],
                                   [0, 1, ty],
                                   [0, 0, 1]])
    # Convert vertices to homogeneous coordinates for translation
    homogeneous_vertices = np.hstack(
        [vertices, np.ones((vertices.shape[0], 1))])
    translated_vertices = np.dot(
        homogeneous_vertices, translation_matrix.T)[:, :2]
    return translated_vertices
```

In [6]:
```
def rotate(vertices, angle_deg, pivot_point):
    angle_rad = np.radians(angle_deg)
    rotation_matrix = np.array([[np.cos(angle_rad), -np.sin(angle_rad), 0],
                                [np.sin(angle_rad),  np.cos(angle_rad), 0],
                                [0,                  0,                 1]])
    # Translate pivot to origin, rotate, then translate back
    pivot_translation = np.array([[1, 0, -pivot_point[0]],
                                  [0, 1, -pivot_point[1]],
                                  [0, 0, 1]])
    pivot_back_translation = np.array([[1, 0, pivot_point[0]],
                                       [0, 1, pivot_point[1]],
                                       [0, 0, 1]])
    transformation_matrix = np.dot(
        np.dot(pivot_back_translation, rotation_matrix), pivot_translation)
    homogeneous_vertices = np.hstack(
        [vertices, np.ones((vertices.shape[0], 1))])
    rotated_vertices = np.dot(homogeneous_vertices,
                              transformation_matrix.T)[:, :2]
    return rotated_vertices
```

In [7]:
```
def draw_triangle(canvas, vertices):
    # Flatten the numpy array vertices and convert to list for Tkinter
    flat_vertices = vertices.flatten().tolist()
    canvas.delete("all")  # Clear previous drawing
    canvas.create_polygon(flat_vertices, outline='black', fill='', width=2)
```

In [8]:
```
def update_position(root, canvas, vertices, tx, ty, canvas_width, canvas_height):
    new_vertices = translate(vertices, tx, ty)

    # Check if any vertex reaches the border of the canvas
```

```
    border_reached = any(x <= 0 or x >= canvas_width or y <= 0 or y >= canvas_he

    if not border_reached:
        draw_triangle(canvas, new_vertices)
        # Schedule the next update
        root.after(100, update_position, root, canvas, new_vertices, tx, ty, can
    else:
        print("Triangle has reached the border.")
```

Рух примітива з використанням трансляції

In [9]:
```
# Create the main window
root = tk.Tk()
root.title("Geometric Primitive with Tkinter")

canvas_width, canvas_height = 400,400

canvas = tk.Canvas(root, width=canvas_width, height=canvas_height, bg='white')
canvas.pack()

tx, ty = 5, -5

update_position(root, canvas, triangle_vertices, tx, ty, canvas_width, canvas_he

root.mainloop()
```

```
Triangle has reached the border.
```

Обертання примітива

In [10]:
```
root = tk.Tk()
root.title("Geometric Primitive with Tkinter")
canvas_width, canvas_height = 400,400
canvas = tk.Canvas(root, width=canvas_width, height=canvas_height, bg='white')
canvas.pack()

pivot_point =[200, 200]

draw_triangle(canvas, triangle_vertices)
canvas.create_oval(pivot_point[0]-3, pivot_point[1]-3, pivot_point[0]+3, pivot_p

rotated_vertices = rotate(triangle_vertices, 45, pivot_point)
root.after(2000, draw_triangle, canvas, rotated_vertices)
root.after(2010, canvas.create_oval, pivot_point[0]-3, pivot_point[1]-3, pivot_p
root.mainloop()
```

Зворотнє обертання примітива

In [11]:
```
root = tk.Tk()
root.title("Geometric Primitive with Tkinter")
canvas_width, canvas_height = 400,400
canvas = tk.Canvas(root, width=canvas_width, height=canvas_height, bg='white')
canvas.pack()

pivot_point =[200, 200]

draw_triangle(canvas, triangle_vertices)
canvas.create_oval(pivot_point[0]-3, pivot_point[1]-3, pivot_point[0]+3, pivot_p
```

```
rotated_vertices = rotate(triangle_vertices, -115, pivot_point)
root.after(2000, draw_triangle, canvas, rotated_vertices)
root.after(2010, canvas.create_oval, pivot_point[0]-3, pivot_point[1]-3, pivot_p

root.mainloop()
```

## Другий рівень складності

In [12]:
```
def translate(vertices, tx, ty, tz):
    translation_matrix = np.array([[1, 0, 0, tx],
                                   [0, 1, 0, ty],
                                   [0, 0, 1, tz],
                                   [0, 0, 0, 1]])

    return np.dot(
        vertices, translation_matrix.T)
```

In [13]:
```
# Rotation matrix around the X-axis
def rotation_matrix_x(angle):
    rad = np.radians(angle)
    return np.array([
        [1, 0, 0, 0],
        [0, np.cos(rad), -np.sin(rad), 0],
        [0, np.sin(rad), np.cos(rad), 0],
        [0, 0, 0, 1]
    ])
```

In [14]:
```
# Rotation matrix around the Y-axis
def rotation_matrix_y(angle):
    rad = np.radians(angle)
    return np.array([
        [np.cos(rad), -np.sin(rad), 0, 0],
        [np.sin(rad), np.cos(rad), 0, 0],
        [0, 0, 1, 0],
        [0, 0, 0, 1]
    ]
    )
```

In [15]:
```
# Rotation matrix around the Y-axis
def rotation_matrix_z(angle):
    rad = np.radians(angle)
    return np.array([
        [np.cos(rad), 0, -np.sin(rad), 0],
        [0, 1, 0, 0],
        [np.sin(rad), 0, np.cos(rad), 0],
        [0, 0, 0, 1]
    ]
    )
```

In [16]:
```
# Axonometric projection matrix
def get_projection_matrix():
    return np.array([
        [1, 0, 0, 0],
        [0, 1, 0, 0],
        [0, 0, 0, 0],
        [0, 0, 0, 1]
    ])
```

In [17]:
```
# Convert 3D vertices to 2D vertices
def project_vertices(vertices, projection_matrix):
    projected = vertices @ projection_matrix.T
    return projected[:, :2]  # Return only x and y for 2D drawing
```

In [18]:
```
# Draw the pyramid on a Tkinter canvas
def draw_pyramid(canvas, vertices_2d, color):
    canvas.delete("all")
    canvas.create_polygon(vertices_2d[:4].flatten().tolist(), outline=color, fil
    # Draw sides
    for i in range(4):
        side = [vertices_2d[i][0], vertices_2d[i][1], vertices_2d[4][0], vertice
        canvas.create_line(side, fill=color)
```

In [19]:
```
# Main animation function
def animate_pyramid(canvas, angle=0):
    global pyramid_vertices
    # Rotate the pyramid
    rotated_vertices = pyramid_vertices @ rotation_matrix_z(angle)
    rotated_vertices = rotated_vertices @ rotation_matrix_x(20)
    # Project the rotated vertices to 2D
    projection_matrix = get_projection_matrix()
    vertices_2d = project_vertices(rotated_vertices, projection_matrix)
    # Randomize position and color
    offset_x, offset_y = random.randint(50, 350), random.randint(50, 350)
    color = random.choice(['red', 'green', 'blue', 'orange', 'purple'])
    # Draw the pyramid
    draw_pyramid(canvas, vertices_2d + np.array([offset_x, offset_y]), color)
    # Schedule next animation frame
    canvas.after(1000, lambda: animate_pyramid(canvas, angle + 5))
```

In [20]:
```
# Define the pyramid vertices in homogenous coordinates (x, y, z, 1)
pyramid_vertices = np.array([
    [0, 0, 0, 1],  # Base vertices
    [100, 0, 0, 1],
    [100, 0, 100, 1],
    [0, 0, 100, 1],
    [50, -100, 50, 1]  # Top vertex
])
```

In [21]:
```
# Setup Tkinter window and canvas
root = tk.Tk()
root.title("3D Pyramid Animation with Tkinter")
canvas = tk.Canvas(root, width=400, height=400, bg='white')
canvas.pack()

# Start the animation
animate_pyramid(canvas)
draw_pyramid(canvas, pyramid_vertices, 'red')

root.mainloop()
```

## Висновок

У цій лабораторії ми успішно виконали лінійні та афінні перетворення (трансляцію) на різних двовимірних і тривимірних фігурах. Під час виконання ми працювали з

розширеними координатами, щоб представити дії над об’єктами у вигляді матриць. Під час роботи ми також поглибили знання про Tkinter і знайшли способи представлення крапок як векторів у багатовимірному просторі. Було поглиблено знання про типи проекцій, зокрема вивчено різницю між ортографічною, косою та проекцією в перспективі.