Lab 1: Geometric Primitives and Transformations

CPV301

Nguyen Hoang Hai[1]

[1]FPT University

September 2023

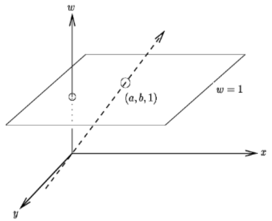
Table of Contents

Homogeneous coordinates

Using homogeneous coordinate for representing the coordinates of 2D point:

$$\begin{bmatrix} x \\ y \end{bmatrix} \longrightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

This way, we will easily represent transformations through matrices. From there, the composition of transformations can be calculated by performing matrix multiplication.

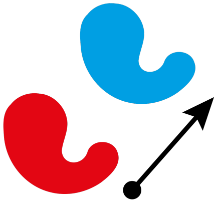


After applying transformations, we divide the first and second coordinates by the third coordinate to get the actual coordinates in 2D plane:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \longrightarrow \begin{bmatrix} x/w \\ y/w \end{bmatrix}$$

Translation

We need to perform the translation with the given vector $v = (v_x, v_y)$.



Assume p is the origin position of an object, then the translation function T_v will work as:

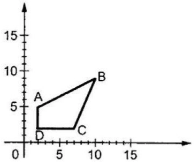
$$T_v(p) = p + v$$

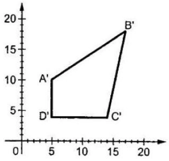
Using homogeneous coordinates, we can represent the translation through the following linear transformation:

$$T_v(p) = \begin{bmatrix} 1 & 0 & v_x \\ 0 & 1 & v_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Scaling

We change the size of the object, either expand or compress, by scaling with the given vector $v = (v_x, v_y)$.





With homogeneous coordinates, the scaling can be expressed by the following linear transformation:

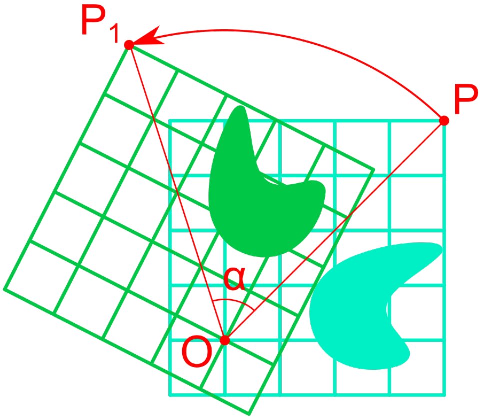
$$S_v(p) = \begin{bmatrix} v_x & 0 & 0 \\ 0 & v_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotation

We rotate the object at particular angle θ (theta) from its origin.

Using homogeneous coordinates, we can formulate the linear transformation for rotation as follows:

$$R_\theta(p) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



Perspective transformation matrix

Given the coordinates of the corners of the original quadrangle and the target quadrangle, find the perspective transformation matrix.

This task can be done by using the function getPerspectiveTransform() of OpenCV. The source code is available on GitHub with detailed description.

They solved this problem by setting up a linear system to transform the corresponding points. Note, passing the list of coordinates of two quadrangles into the function must ensure that the points are arranged in order: top-left, top-right, bottom-right, bottom-left.

We can easily handle this by 2 steps:

- Divide 4 points into two sets: bottom_points and top_points based on their y-coordinate.
- For each set (has 2 points), we continue arrange based on their x-coordinate.

```
1st-------2nd
  |         |
  |         |
  |         |
3rd-------4th
```

Table of Contents

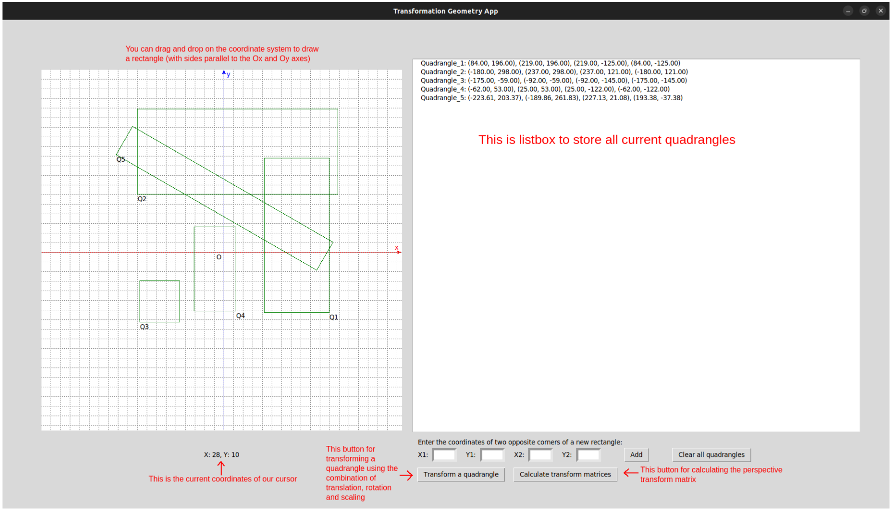
Dependencies

Tkinter is the library be chosen to design GUI for this app. It is the standard GUI library for Python. Tkinter provides a fast and easy way to create applications.

Dependencies

Two libraries NumPy and OpenCV are imported for mathematical calculations, matrices operations and transformations.

First view



You can drag and drop on the coordinate system to draw a rectangle (with sides parallel to the Ox and Oy axes)

```
Quadrangle_1: (84.00, 196.00), (219.00, 196.00), (219.00, -125.00), (84.00, -125.00)
Quadrangle_2: (-180.00, 298.00), (237.00, 298.00), (237.00, 121.00), (-180.00, 121.00)
Quadrangle_3: (-175.00, -59.00), (-92.00, -59.00), (-92.00, -145.00), (-175.00, -145.00)
Quadrangle_4: (-62.00, 53.00), (25.00, 53.00), (25.00, -122.00), (-62.00, -122.00)
Quadrangle_5: (-223.61, 203.37), (-189.86, 261.83), (227.13, 21.08), (193.38, -37.38)
```

This is listbox to store all current quadrangles

Enter the coordinates of two opposite corners of a new rectangle:

X1: Y1: X2: Y2:

X: 28, Y: 10

This is the current coordinates of our cursor

This button for transforming a quadrangle using the combination of translation, rotation and scaling

This button for calculating the perspective transform matrix

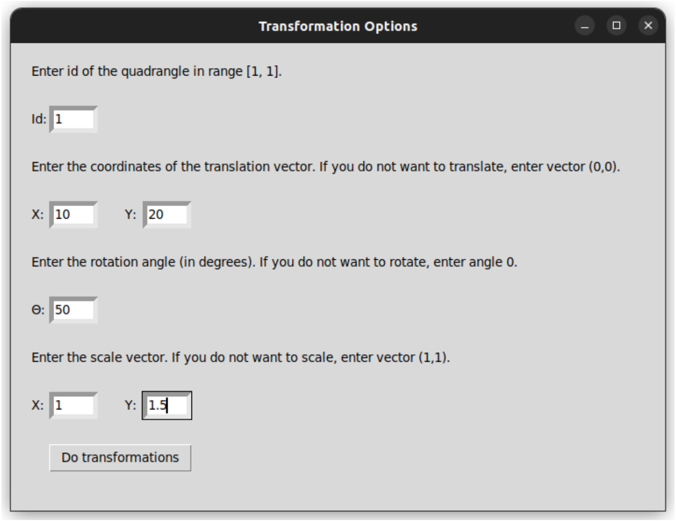
GUI Overview

For simplicity, the application only allows adding new rectangles (with sides parallel to the Ox and Oy axes) by specifying the coordinates of two opposite corners (fill out the form in the bottom using the keyboard, or drag and drop in the coordinate system with the mouse). Other types of quadrangle can be created by performing a transform from the original quadrangles.

The listbox will show the coordinates of all current quadrangles with the corresponding name tag.

By default, the origin is in the upper left corner, and the part of the coordinate system we see represents the first quadrant (where $x, y > 0$). To standardize the coordinate system, we need to translate the origin and invert the y-coordinate.

Transformation options



Enter id of the quadrangle in range [1, 1].

Id: 1

Enter the coordinates of the translation vector. If you do not want to translate, enter vector (0,0).

X: 10 Y: 20

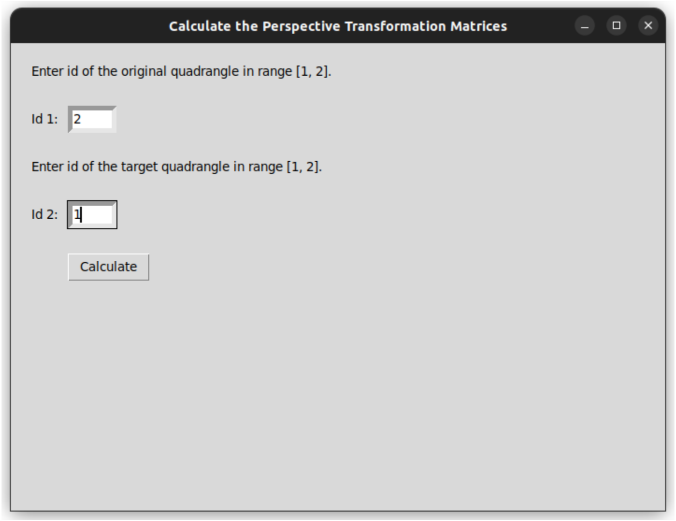
Enter the rotation angle (in degrees). If you do not want to rotate, enter angle 0.

Θ: 50

Enter the scale vector. If you do not want to scale, enter vector (1,1).

X: 1 Y: 1.5

Calculating perspective transformation



Calculate the Perspective Transformation Matrices

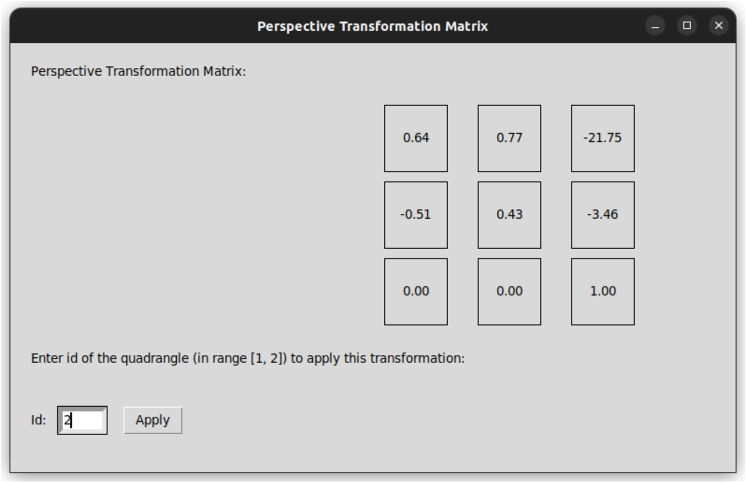
Enter id of the original quadrangle in range [1, 2].

Id 1: 2

Enter id of the target quadrangle in range [1, 2].

Id 2: 1

Applying the perspective transformation



Perspective Transformation Matrix:

0.64	0.77	-21.75
-0.51	0.43	-3.46
0.00	0.00	1.00

Enter id of the quadrangle (in range [1, 2]) to apply this transformation:

Id: 2

Features Explanation

In the **Transform Options** window, we choose the quadrangle to transform by specifying its ID. Then, enter all coefficients for the transformation, this transformation is the combination of translation, rotation and scaling.

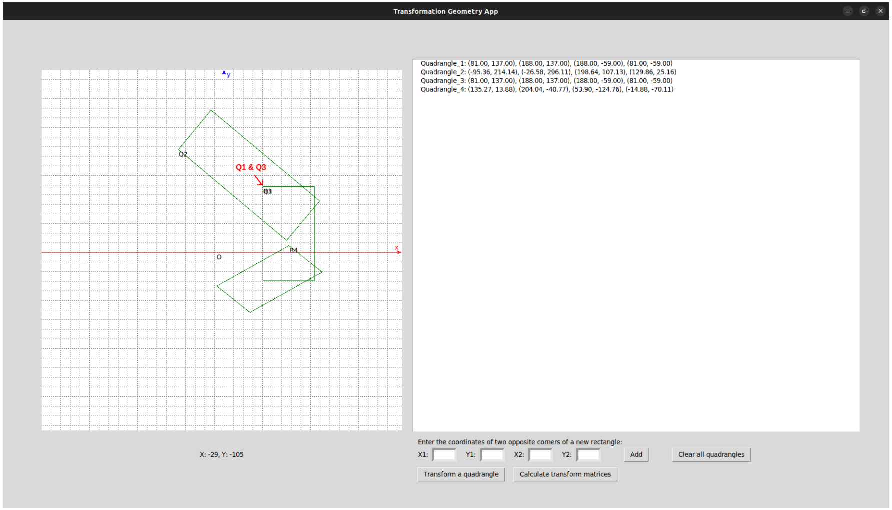
In the **Calculate the Perspective Transformation Matrix** window, we need to specify the IDs of the original (id1) and target (id2) quadrangles.

In the **Perspective Transformation Matrix** window (result window), we can see the perspective matrix for this transformation. Then, we can apply it to our quadrangles.

All fields are handled error input: entering non-number, entering id out of range, blank fields.

Table of Contents

Features Testing



Transformation Geometry App
y
x
O
Q2
Q1 & Q3
Q1
Q4
X: -29, Y: -105
Enter the coordinates of two opposite corners of a new rectangle:
X1:
Y1:
X2:
Y2:
Add
Clear all quadrangles
Transform a quadrangle
Calculate transform matrices

```
Quadrangle_1: (81.00, 137.00), (188.00, 137.00), (188.00, -59.00), (81.00, -59.00)
Quadrangle_2: (-95.36, 214.14), (-26.58, 296.11), (198.64, 107.13), (129.86, 25.16)
Quadrangle_3: (81.00, 137.00), (188.00, 137.00), (188.00, -59.00), (81.00, -59.00)
Quadrangle_4: (135.27, 13.88), (204.04, -40.77), (53.90, -124.76), (-14.88, -70.11)
```

Results

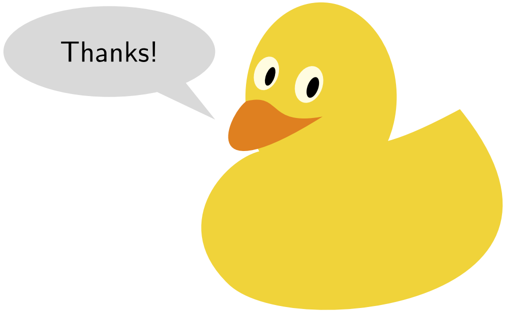
In the previous slide, there are 4 quadrangles Q1, Q2, Q3 and Q4 in which Q1 and Q3 overlap each other.

Quadrangle Q1 is the initial quadrangle created, then we use the transformation (slide 13) to create Q2. Next, we calculate the perspective matrix of the reverse transformation from Q2 to Q1 (slide 14) and then apply it again to Q2 to create Q3 (slide 15), therefore Q3 overlap with Q1. To create Q4, we apply this perspective matrix on Q1.

Note, the angles of Q4 are not right angles due to observational error during the calculation process.

Source code

https://github.com/hoanghai1803/CPV301/blob/main/lab1/lab1.py


Thanks!