HOMOGENOUS CO-ORDINATES

Homogenous co-ordinates are a mathematical representation used in projective geometry & Computer Graphics

They provide a way to represent points, vectors & transformations in a uniform manner

We extend the Cartesian system by using an additional coordinate usually denoted as "w"

A point in Homogenous system is represented as (x, y, z, w) where (x, y, z) are the cartesian coordinates
(w) is called the "Scaling factor"

GEOMETRIC TRANSFORMATIONS

There are several types of geometric transformations:

- > Translation
- > Rotation
- > Scaling
- > Shearing
- > Reflection
- > Projection

These tranformations can be applied individually or in combinations in order to achieve better effects

They are widely used in fields like Computer Graphics, Compiter Vision, Designing & modelling, etc.

We will be particularly discussing about translation, rotation and scaling

TRANSLATION

Translation refers to moving of an object in a certain direction without changing its size/shape

It is done by adding constant values to the coordinates of the object

Lets say working with a 2D system, we have an object's coordinates as (x, y)

Thus, in Homogenous coordinates, we can represent it as (x, y, 1) = V

Further, lets say we want to translate this position vector by 2 units in +x and 1 unit in -y

Then we can make a respective "translation matrix" using the homogenous coordinates called T such that:

V*T = V'
(where V' represents the final homogenous coordinates after translation)

Thus, we can represent T as:

```
[1   0   0]
|0   1   0|
[tx  ty  1]        here, (tx = 2) & (ty = -1)
```

This makes the final equation for a single translation as:

```
                [1   0   0]
     [x   y   1]|0   1   0| = [x'  y'  1]
                [tx  ty  1]
```

Similarly, if we have to perform 3 translations, namely T1 T2 and T3 then we can write the final homo-coords as V':

V*Tf = V'

Here, Tf = (T1*T2*T3)
And each Ti represents the respective translation matrix for ith translation

ROTATION

Rotation as a transformation involves rotating an object around a fixed point or axis

Size/shape of the object is preserved

Very much similar to translation, a rotation can also be represented in homogenous coordinates as a matrix

```
    [cosT   sinT   0]
R = |-sinT  cosT   0|          (T represents theta ie. the angle by which object is rotated)
    [0      0      1]
```

Thus, if an object with homogenous coordinates [x   y   1] is to be rotated by theta angle:

V*R = V'   (V' represents final homogenous

coordinates)

```
                [cosT   sinT   0]
    [x   y   1]|-sinT  cosT   0| = [x'  y'  1]
                [0      0      1]
```

And also similar to translation, multiple rotations can be handled by finding a FINAL rotation matrix which is the multiplied with the homogenous coordinates

SCALING

Scaling leads to change in the "size" if the object by either enlarging or shrinking the object

This achieved by multiplying the coordinates of the vector by a scalar value

Scaling is also done in the same way. We can denote the scaling factors in a matrix form as:

```
            [Sx  0   0]
            |0   Sy  0|            (Sx = scaling along x axis)
            [0   0   1]            (Sy = scaling along y axis)
```

Thus, say a position coordinate [x   y   1] undergoing scaling
It is enlarged to twice its size along x axis while halved along y axis
So Sx = 2 while Sy = 0.5

```
                [Sx  0   0]
    [x   y   1]|0   Sy  0| = [x'  y'  1]
                [0   0   1]
```

The above calculation provides us with the final homogenoues coordinates of the object after scaling

EXAMPLE

Question

Say there is a square ABCD such that A(1,1) B(3,1) C(3,3) D(1,3) and its centre is at E(2,2)

We need to perform a set of geometrical transformations on the square

> The final object should be a square with thrice its original

side length
    > It sholuld also be rotated 90 degrees clockwise
    > Its centre should stay at same E(2,2)

    Find the resultant "transformation matrix" for the above changes and also find all the final coordinates

Solution

    We first need to plan it step-wise:
            1) translating the sqaure such that the centre is at Origin
            2) scaling the square to 3 times its size in x as well as y axis
            3) rotating the whole square by -90 deg (since clockwise rotation)
            4) translating it back such that new centre at original (2,2) coordinates

    Thus, there will be T1 matirix for step1 translation, S for step2 scaling, R for step3 rotation and then finally T2 for step4 translation

    Putting down values of the matrices mentioned above:

$$T1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -2 & -2 & 1 \end{bmatrix}$$

$$S = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 2 & 1 \end{bmatrix}$$

    Now, we can also represent the 4 coordinates altogether in one single matrix to reduce the number of steps

$$M = \begin{bmatrix} A \\ B \end{bmatrix}$$ or in homogenous coordinates, it is basically $$M = \begin{bmatrix} 1 & 1 & 1 \\ 3 & 1 & 1 \end{bmatrix}$$

```
        |C|
|3   3   1|
        [D]
[1   3   1]
```

Further, we can find the resultant transformation matrix:

result = T1*S*R*T2

```
                [0   -3  0]
result = |3   0   0|
                [-4  8   1]
```

Last step in the method is to find the final coordinate matrix M'

M*result = M'

```
[1   1   1] * [0   -3  0]   [-1  5   1]
|3   1   1|   |3   0   0| = |-1  -1  1| = M'
|3   3   1|   [-4  8   1]   |5   -1  1|
[1   3   1]                 [5   5   1]
```

Hence, from the final coordinate matrix M' that is formed, we can get the new cartesian coordinates of the vertices

A'(-1, 5)
B'(-1, -1)
C'(5, -1)
D'(5, 5)

While, if we find the centre if the new coordinates, it comes out to still be E(2, 2)

So this also proves that our method has worked.

Plus, intial edge length was 2
Whereas now, it is 6 so the whole square is scaled to thrice its size

CODE (animation)

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

def animate(i):
    ax.clear()
```

```
    ax.set_xlim(-5, 5)
    ax.set_ylim(-5, 5)

    # Original square coordinates
    coords = np.array([[1, 1], [3, 1], [3, 3], [1, 3]])

    # Translate to origin
    translated_coords = coords - np.mean(coords, axis=0)

    # Scale the square
    scaled_coords = translated_coords * 3

    # Rotate 90 degrees clockwise
    theta = np.radians(90)
    rotation_matrix = np.array([[np.cos(theta), -np.sin(theta)],
[np.sin(theta), np.cos(theta)]])
    rotated_coords = scaled_coords.dot(rotation_matrix)

    # Translate back to (2, 2)
    final_coords = rotated_coords + np.array([2, 2])

    # Plot the square
    ax.plot(final_coords[:, 0], final_coords[:, 1], color='blue')
    ax.fill(final_coords[:, 0], final_coords[:, 1], color='blue',
alpha=0.3)

fig, ax = plt.subplots()
ani = animation.FuncAnimation(fig, animate, frames=100, interval=50)

plt.show()
```

```
The kernel failed to start as '_psutil_linux' could not be imported
from 'most likely due to a circular import'.

Click <a
href='https://aka.ms/kernelFailuresModuleImportErrFromFile'>here</a>
for more info.
```