**Batch: C2** **Roll No.: 16010121060**

**Experiment No. 05**

**TITLE**: Write a program to perform 2D and 3D transformation

## AIM:

Write a program to perform 2D and 3D transformation

a. Translation

b. Scaling

c. Rotation

d. Shear

e. Reflection

## Expected OUTCOME of Experiment:

*Students should write appropriate CO*

## Books/ Journals/ Websites referred:

https://cse18-iiith.vlabs.ac.in/exp/transformations-rotation/

https://cse18-iiith.vlabs.ac.in/exp/transformations-scaling/

https://cse18-iiith.vlabs.ac.in/exp/transformations-translation/

https://cse18-iiith.vlabs.ac.in/exp/2d-demo/

https://cse18-iiith.vlabs.ac.in/exp/3d-articulated-arm/

## Algorithm/ Pseudo code for each process:

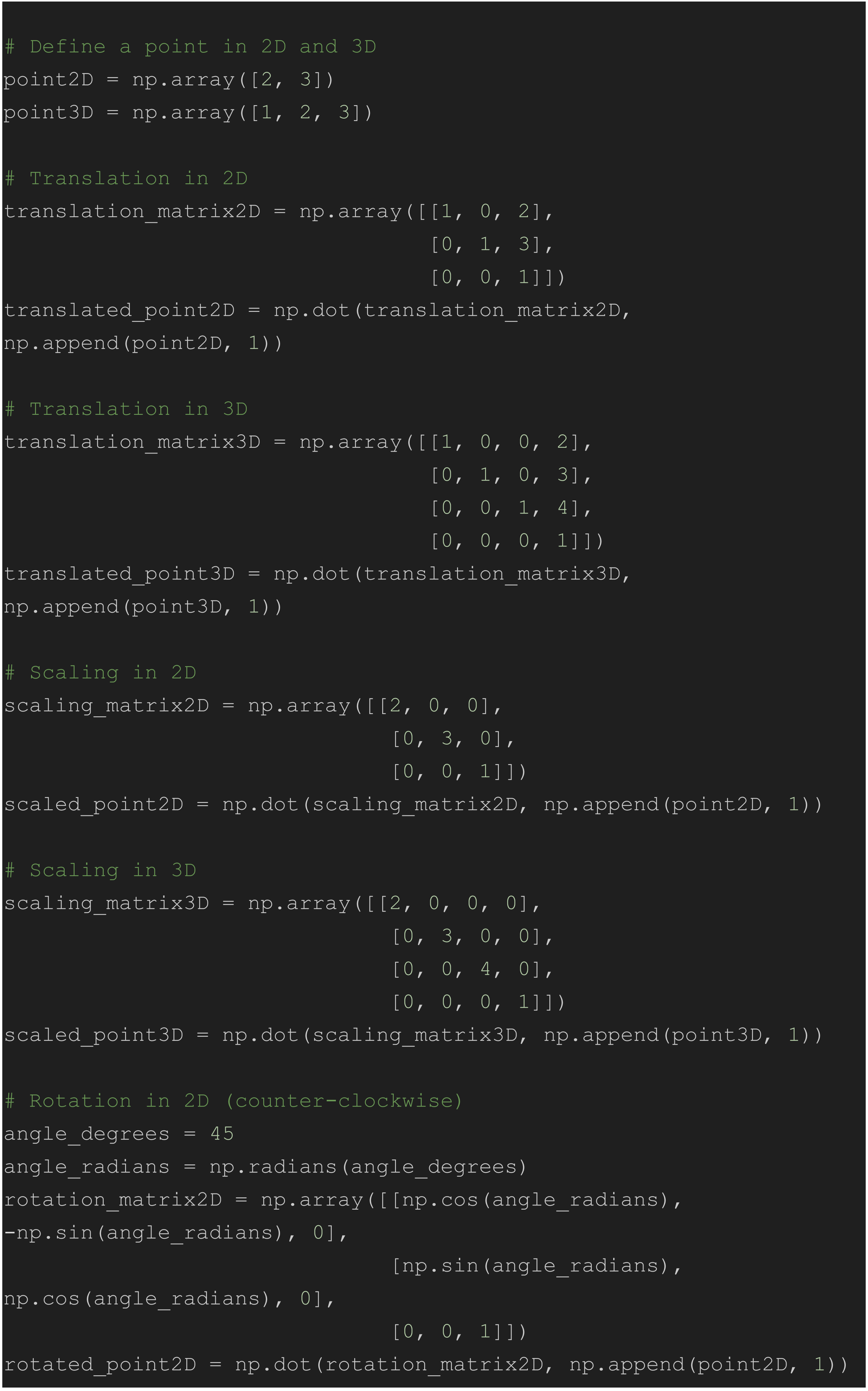
1) Get input points
2) Multiply matrix

These operations are only matrix multiplications.

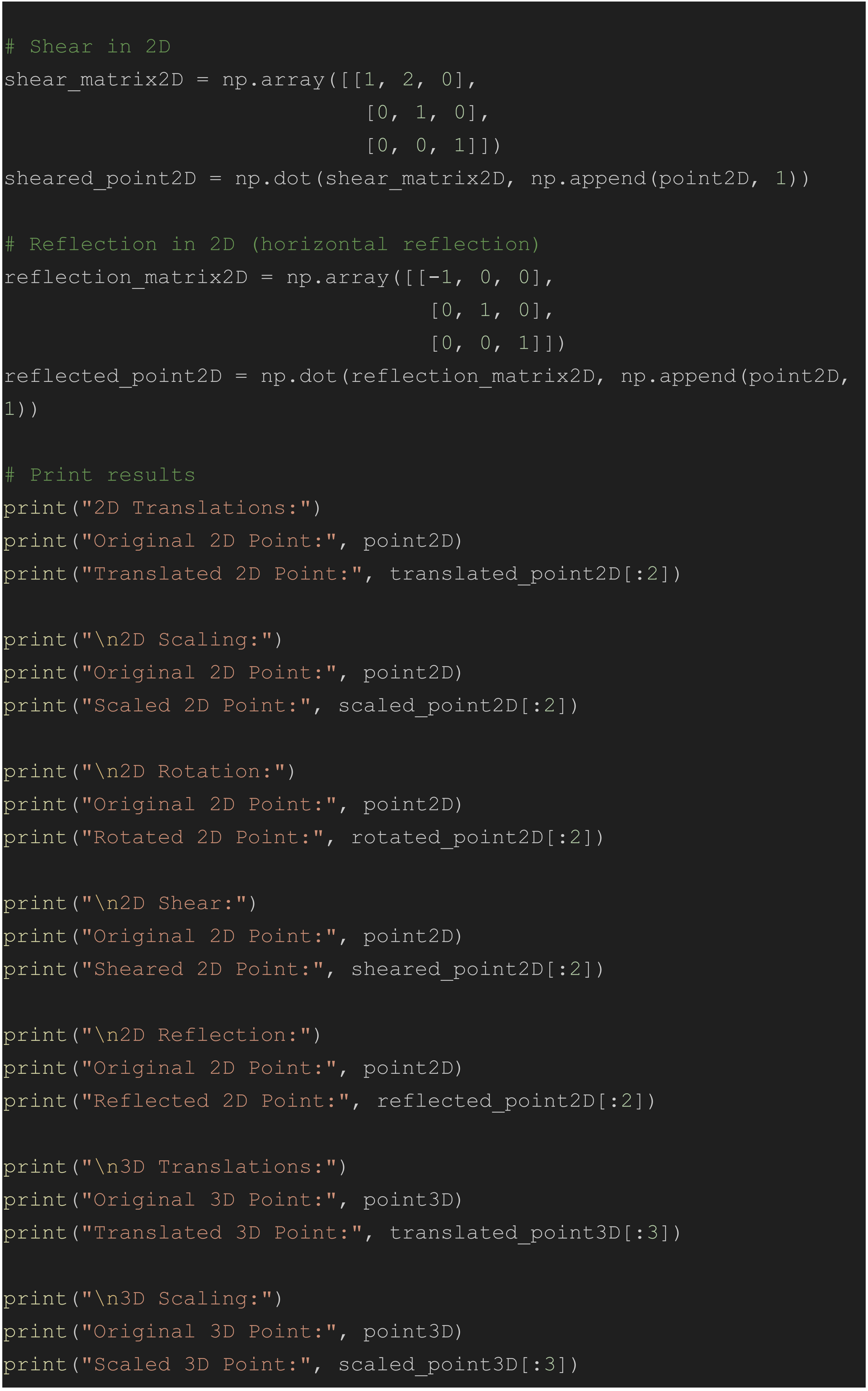
## Implementation details:

Here numpy is used for making the matrix multiplications.

```
import numpy as np
```



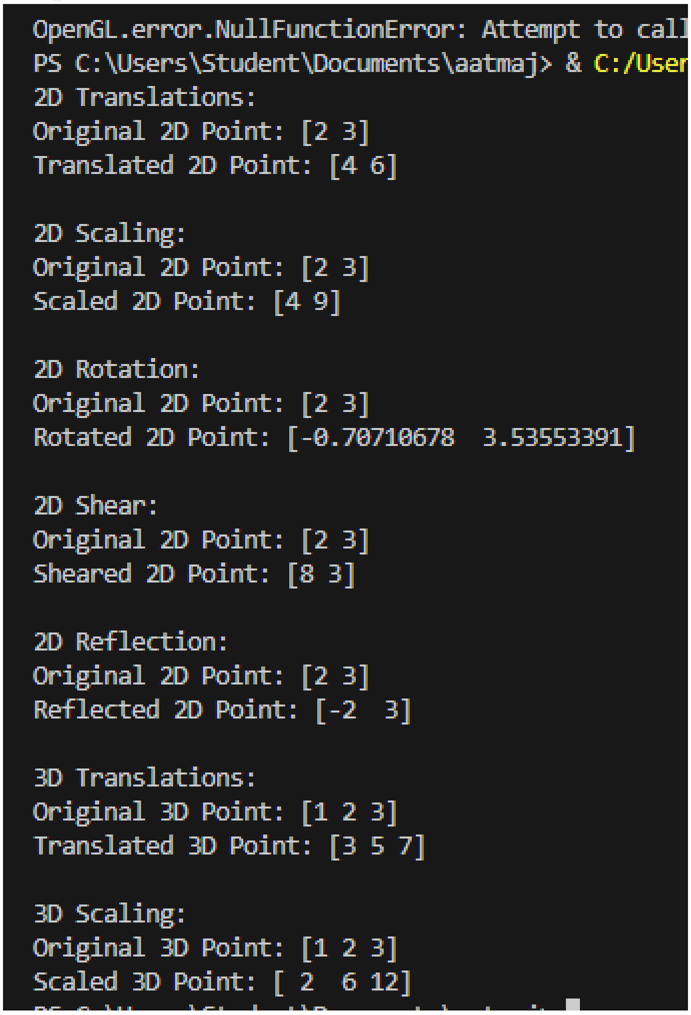
```
# Define a point in 2D and 3D
point2D = np.array([2, 3])
point3D = np.array([1, 2, 3])

# Translation in 2D
translation_matrix2D = np.array([[1, 0, 2],
                                 [0, 1, 3],
                                 [0, 0, 1]])
translated_point2D = np.dot(translation_matrix2D,
np.append(point2D, 1))

# Translation in 3D
translation_matrix3D = np.array([[1, 0, 0, 2],
                                 [0, 1, 0, 3],
                                 [0, 0, 1, 4],
                                 [0, 0, 0, 1]])
translated_point3D = np.dot(translation_matrix3D,
np.append(point3D, 1))

# Scaling in 2D
scaling_matrix2D = np.array([[2, 0, 0],
                              [0, 3, 0],
                              [0, 0, 1]])
scaled_point2D = np.dot(scaling_matrix2D, np.append(point2D, 1))

# Scaling in 3D
scaling_matrix3D = np.array([[2, 0, 0, 0],
                              [0, 3, 0, 0],
                              [0, 0, 4, 0],
                              [0, 0, 0, 1]])
scaled_point3D = np.dot(scaling_matrix3D, np.append(point3D, 1))

# Rotation in 2D (counter-clockwise)
angle_degrees = 45
angle_radians = np.radians(angle_degrees)
rotation_matrix2D = np.array([[np.cos(angle_radians),
-np.sin(angle_radians), 0],
                              [np.sin(angle_radians),
np.cos(angle_radians), 0],
                              [0, 0, 1]])
rotated_point2D = np.dot(rotation_matrix2D, np.append(point2D, 1))
```



```
# Shear in 2D
shear_matrix2D = np.array([[1, 2, 0],
                            [0, 1, 0],
                            [0, 0, 1]])
sheared_point2D = np.dot(shear_matrix2D, np.append(point2D, 1))

# Reflection in 2D (horizontal reflection)
reflection_matrix2D = np.array([[-1, 0, 0],
                                 [0, 1, 0],
                                 [0, 0, 1]])
reflected_point2D = np.dot(reflection_matrix2D, np.append(point2D,
1))

# Print results
print("2D Translations:")
print("Original 2D Point:", point2D)
print("Translated 2D Point:", translated_point2D[:2])

print("\n2D Scaling:")
print("Original 2D Point:", point2D)
print("Scaled 2D Point:", scaled_point2D[:2])

print("\n2D Rotation:")
print("Original 2D Point:", point2D)
print("Rotated 2D Point:", rotated_point2D[:2])

print("\n2D Shear:")
print("Original 2D Point:", point2D)
print("Sheared 2D Point:", sheared_point2D[:2])

print("\n2D Reflection:")
print("Original 2D Point:", point2D)
print("Reflected 2D Point:", reflected_point2D[:2])

print("\n3D Translations:")
print("Original 3D Point:", point3D)
print("Translated 3D Point:", translated_point3D[:3])

print("\n3D Scaling:")
print("Original 3D Point:", point3D)
print("Scaled 3D Point:", scaled_point3D[:3])
```

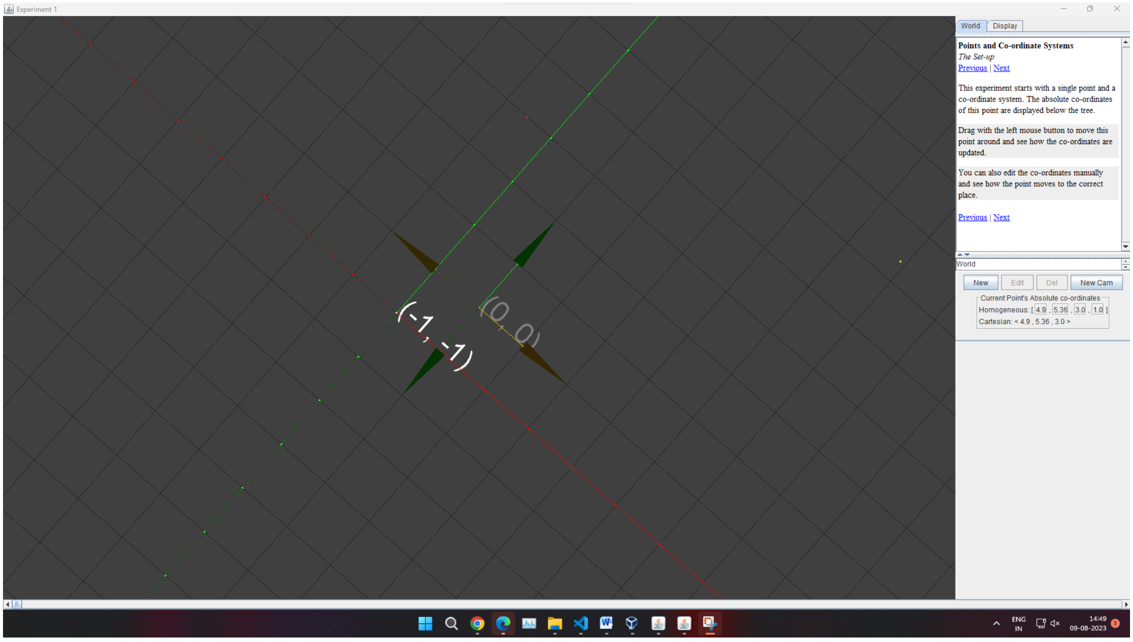
Credits - ChatGPT was used as code was very easy to implement. (only matrix multiplication)

**Output(s) (Screen Shot):**



```
OpenGL.error.NullFunctionError: Attempt to call
PS C:\Users\Student\Documents\aatmaj> & C:/User
2D Translations:
Original 2D Point: [2 3]
Translated 2D Point: [4 6]

2D Scaling:
Original 2D Point: [2 3]
Scaled 2D Point: [4 9]

2D Rotation:
Original 2D Point: [2 3]
Rotated 2D Point: [-0.70710678  3.53553391]

2D Shear:
Original 2D Point: [2 3]
Sheared 2D Point: [8 3]

2D Reflection:
Original 2D Point: [2 3]
Reflected 2D Point: [-2  3]

3D Translations:
Original 3D Point: [1 2 3]
Translated 3D Point: [3 5 7]

3D Scaling:
Original 3D Point: [1 2 3]
Scaled 3D Point: [ 2  6 12]
```

**Screenshots from VLab(if any):**



**Points and Co-ordinate Systems**

*The Set-up*

Previous | Next

This experiment starts with a single point and a co-ordinate system. The absolute co-ordinates of this point are displayed below the tree.

Drag with the left mouse button to move this point around and see how the co-ordinates are updated.

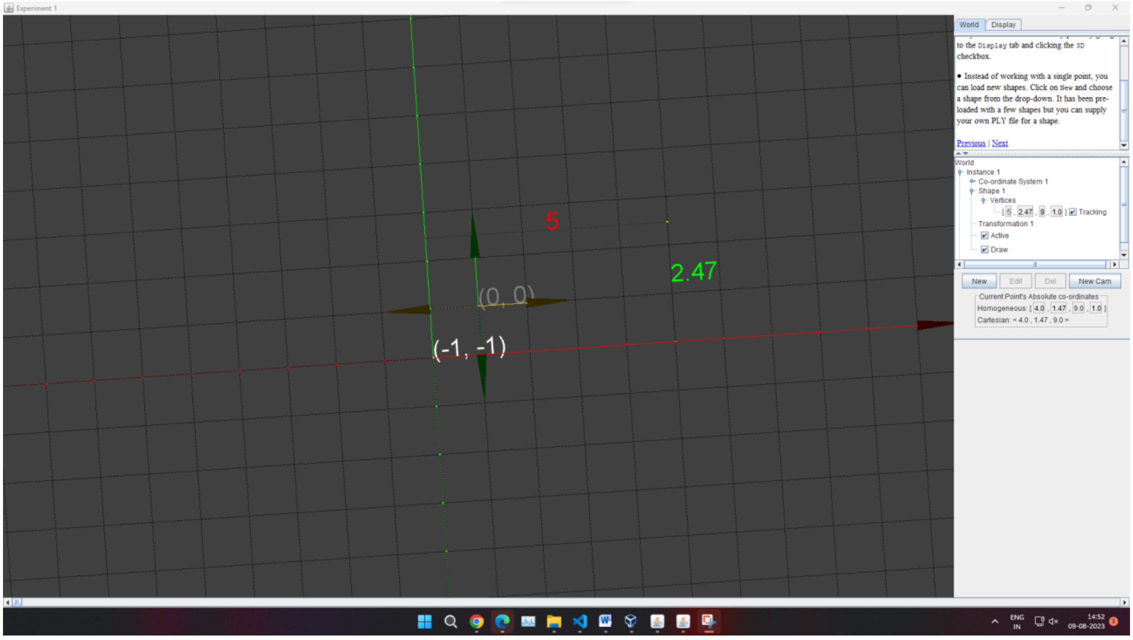
You can also edit the co-ordinates manually and see how the point moves to the correct place.

Previous | Next

Current Point's Absolute co-ordinates

Homogeneous: [ 4.9 , 5.36 , 3.0 , 1.0 ]

Cartesian: < 4.9 , 5.36 , 3.0 >



to the Display tab and clicking the 3D checkbox.

- Instead of working with a single point, you can load new shapes. Click on New and choose a shape from the drop-down. It has been pre-loaded with a few shapes but you can supply your own PLY file for a shape.

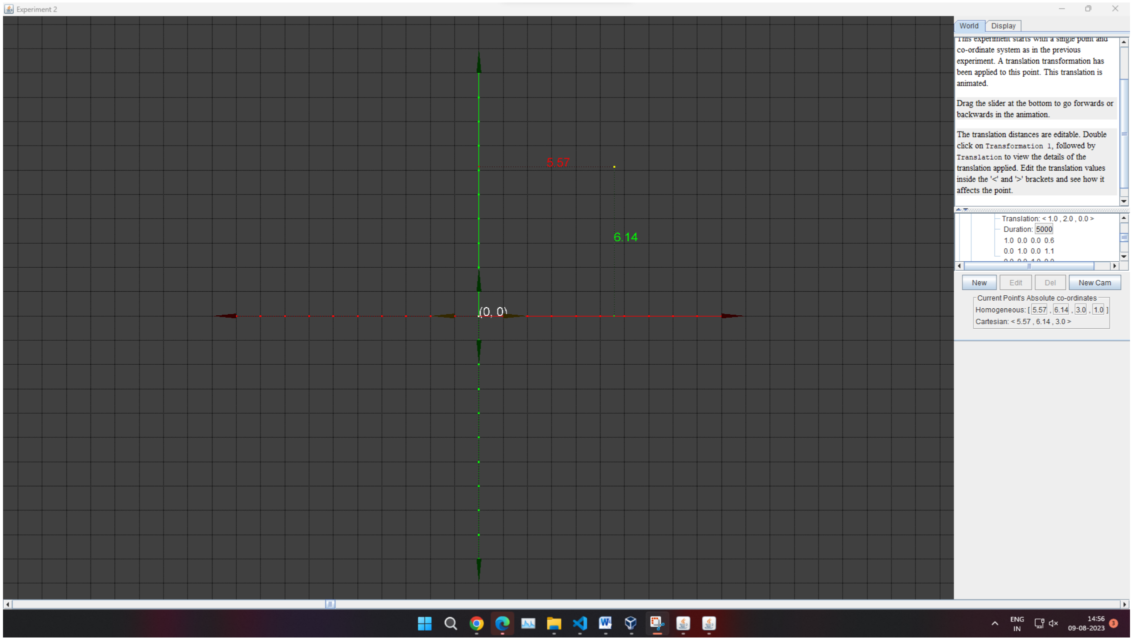
Previous | Next

World
Instance 1
Co-ordinate System 1
Shape 1
Vertices
[ 5 , 2.47 , 9 , 1.0 ] ☑ Tracking
Transformation 1
☑ Active
☑ Draw

Current Point's Absolute co-ordinates

Homogeneous: [ 4.0 , 1.47 , 9.0 , 1.0 ]

Cartesian: < 4.0 , 1.47 , 9.0 >



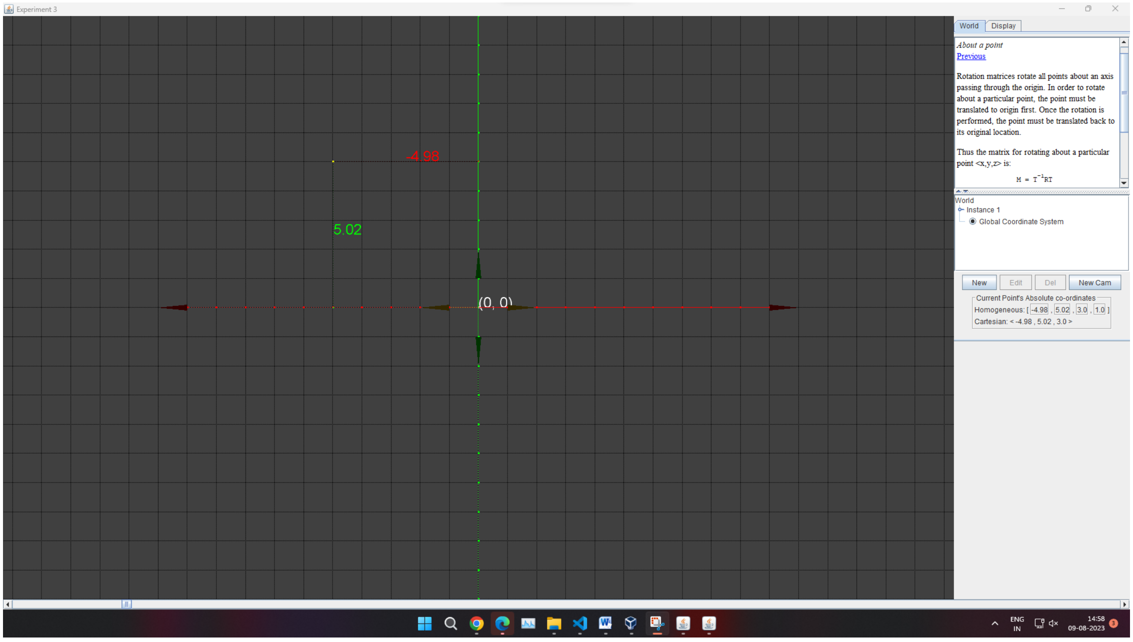
This experiment starts with a single point and co-ordinate system as in the previous experiment. A translation transformation has been applied to this point. This translation is animated.

Drag the slider at the bottom to go forwards or backwards in the animation.

The translation distances are editable. Double click on `Transformation 1`, followed by `Translation` to view the details of the translation applied. Edit the translation values inside the '<' and '>' brackets and see how it affects the point.

```
Translation: < 1.0 , 2.0 , 0.0 >
Duration: 5000
1.0  0.0  0.0  0.6
0.0  1.0  0.0  1.1
```

Current Point's Absolute co-ordinates
Homogeneous: [ 5.57 , 6.14 , 3.0 , 1.0 ]
Cartesian: < 5.57 , 6.14 , 3.0 >



*About a point*
Previous

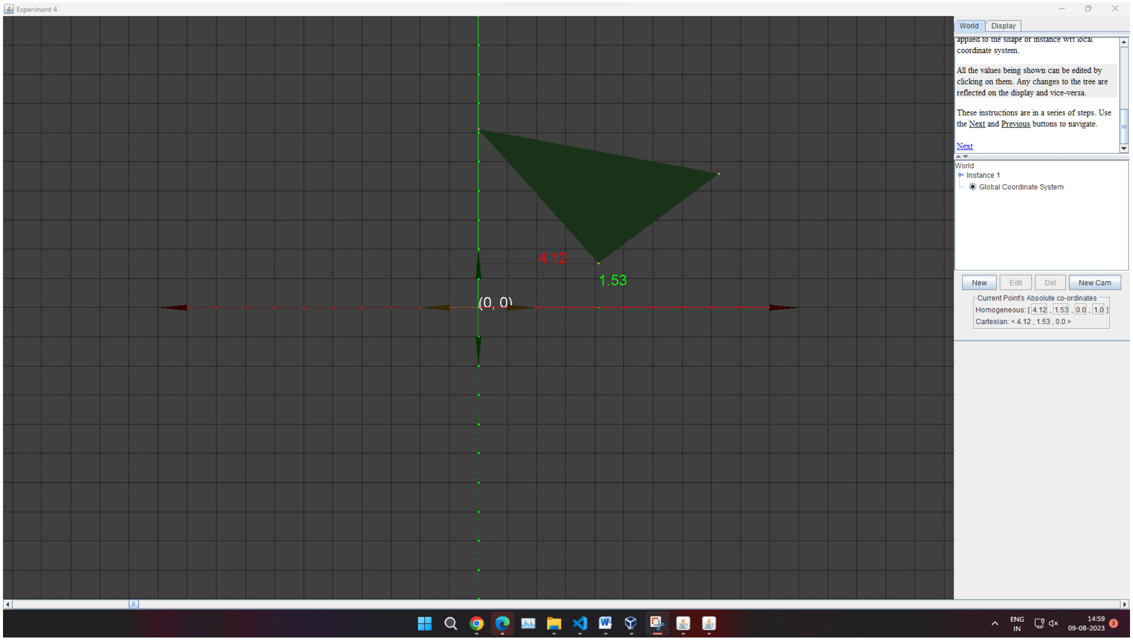
Rotation matrices rotate all points about an axis passing through the origin. In order to rotate about a particular point, the point must be translated to origin first. Once the rotation is performed, the point must be translated back to its original location.

Thus the matrix for rotating about a particular point <x,y,z> is:

$$M = T^{-1}RT$$

World
Instance 1
● Global Coordinate System

Current Point's Absolute co-ordinates
Homogeneous: [ -4.98 , 5.02 , 3.0 , 1.0 ]
Cartesian: < -4.98 , 5.02 , 3.0 >



applied to the shape or instance wrt local coordinate system.

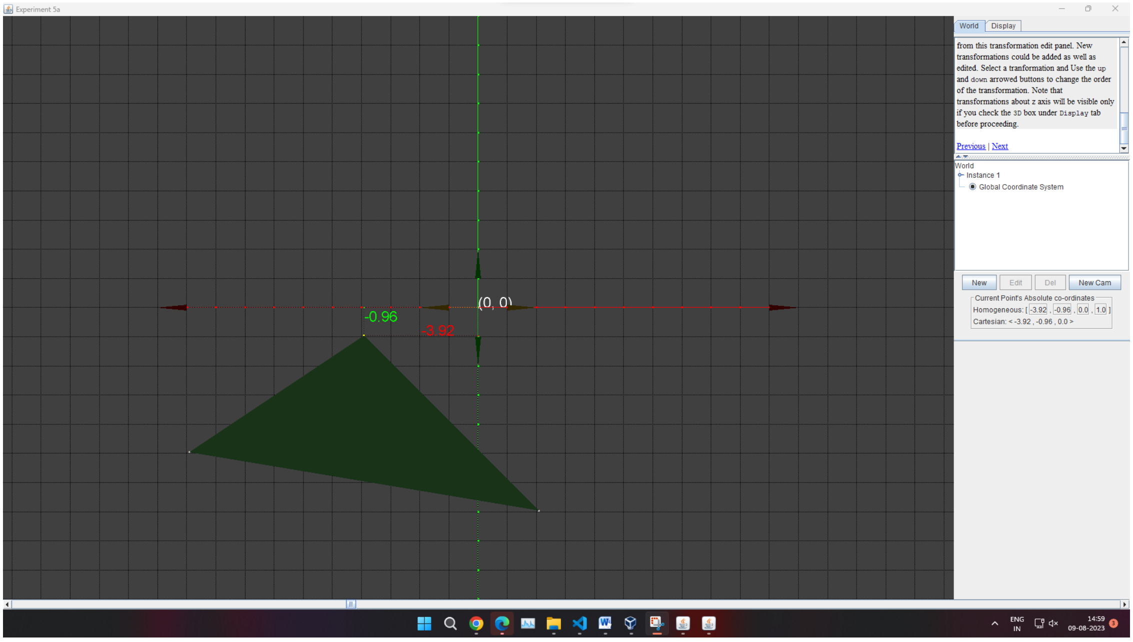
All the values being shown can be edited by clicking on them. Any changes to the tree are reflected on the display and vice-versa.

These instructions are in a series of steps. Use the Next and Previous buttons to navigate.

Next

World
Instance 1
Global Coordinate System

Current Point's Absolute co-ordinates
Homogeneous: [ 4.12 , 1.53 , 0.0 , 1.0 ]
Cartesian: < 4.12 , 1.53 , 0.0 >



from this transformation edit panel. New transformations could be added as well as edited. Select a tranformation and Use the up and down arrowed buttons to change the order of the transformation. Note that transformations about z axis will be visible only if you check the 3D box under Display tab before proceeding.

Previous | Next

World
Instance 1
Global Coordinate System

Current Point's Absolute co-ordinates
Homogeneous: [ -3.92 , -0.96 , 0.0 , 1.0 ]
Cartesian: < -3.92 , -0.96 , 0.0 >

## Conclusion and discussion:

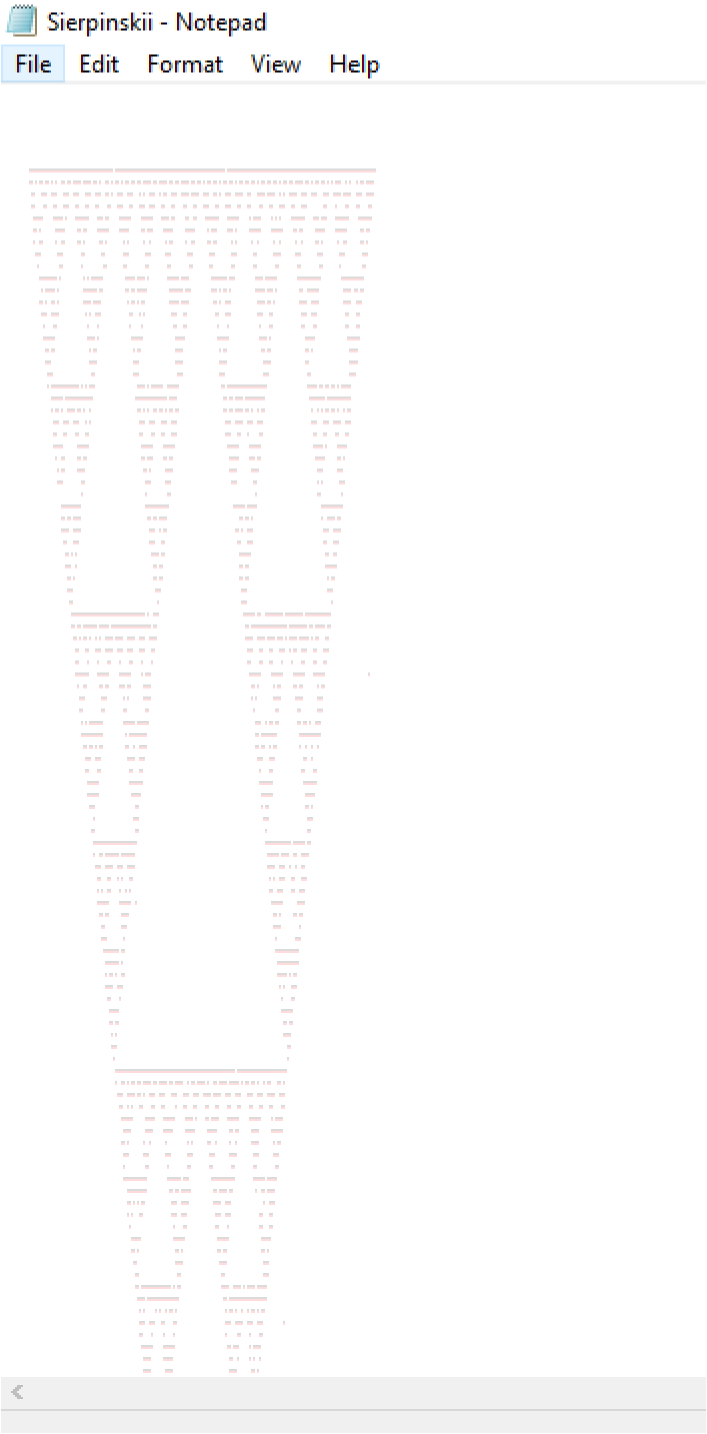
Vlabs have been performed successfully, along with the 2D & 3D Transformation program. We understood how matrix multiplications work

**Date: 27 sept 2023**

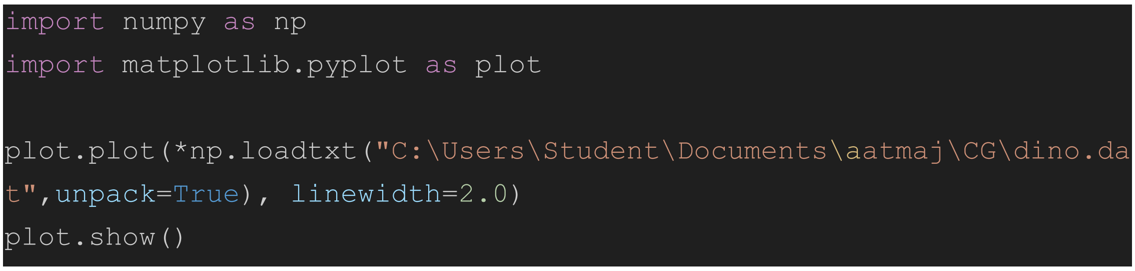
Post lab

Implement the code to **draw the Sierpinski Gasket**

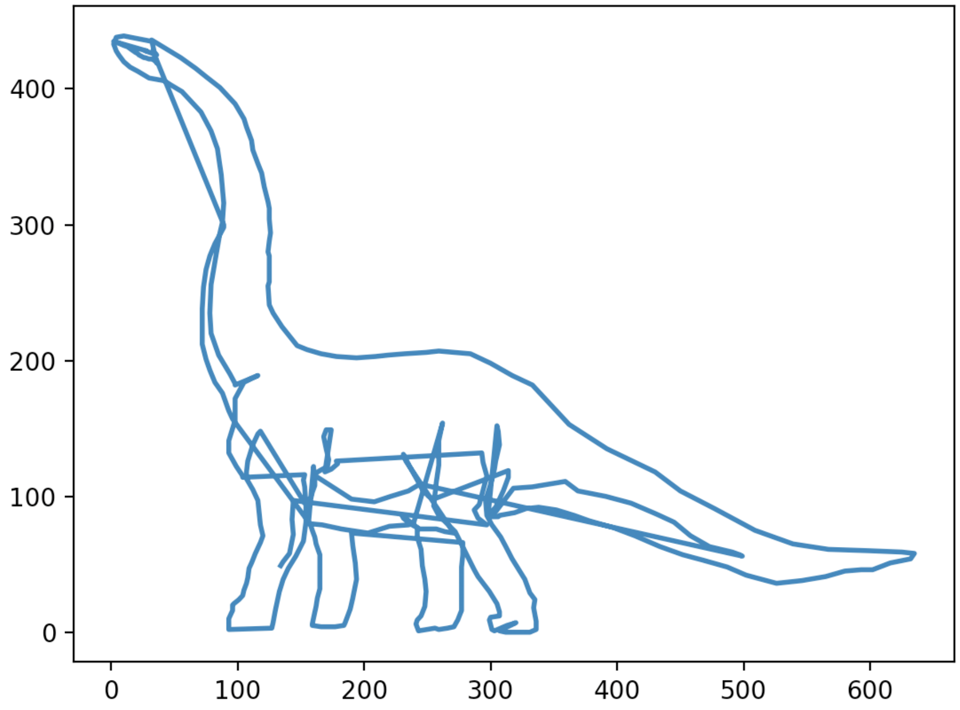
```
public class Sierpinski_Triangle extends Fractal {

  Sierpinski_Triangle() {
    SetNumberOfVertices();
    SetVertices();
    Time = 100000;
    Range = 100;
    Dot = '*';
  }

  public void SetNumberOfVertices() {
    NumberOfVertices = 3;
  }

  public void go() {
    Complex seed, Vertex;

    Hashtable = new hashtable(100);
    // Length of the hashtable is to be set according to the time
taken for the execution.

    seed = SetInitialSeed();
    // Set the initial seed

    for (int i = 0; i <= Time; i++) {
      Vertex = vertices[RandomVertexGenerator()]; // Set the random
Vertex
      seed = Complex.ScalarDivision(Complex.Addition(seed, Vertex),
2.0);
      Hashtable.Add(normalized(seed));
    }
  }
}
```



Draw dinosaur using Dino.dat Assignment



```
import numpy as np
import matplotlib.pyplot as plot

plot.plot(*np.loadtxt("C:\Users\Student\Documents\aatmaj\CG\dino.dat",unpack=True), linewidth=2.0)
plot.show()
```


400
300
200
100
0
0
100
200
300
400
500
600