Ex. No. : 5 **Date:** 25/09/2025

Register No.: 231701059 **Name: Surya R**

3D Transformations on Basic Objects (Cube, Pyramid)

AIM:

To write a program that allows the user to perform 3D transformations (translation, scaling, rotation) on basic 3D objects like a cube **or** pyramid**, and visualize the results.**

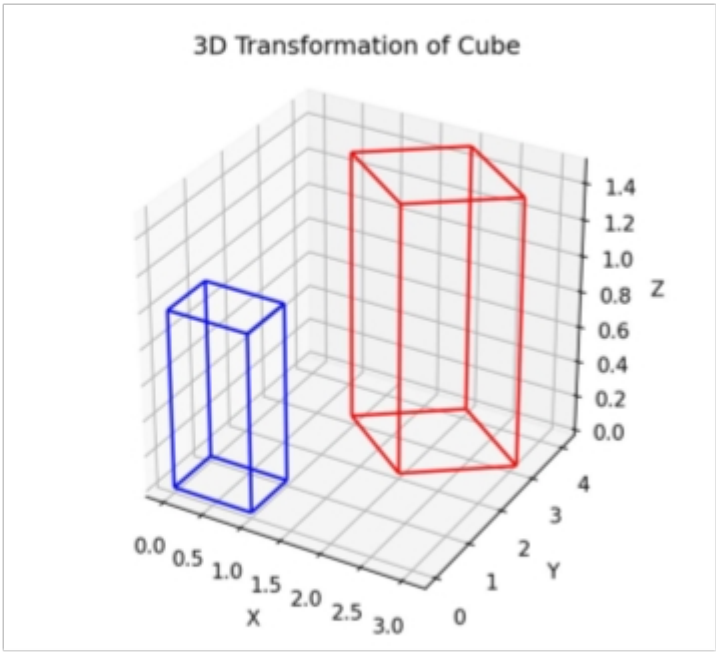
Procedure:

1. Define a 3D object using vertices and edges (cube or pyramid).
2. Use 4×4 homogeneous transformation matrices for:
   - Translation
   - Scaling
   - Rotation (around x, y, z axes)
3. Multiply the object's coordinates with the transformation matrix.
4. Project 3D points to 2D for visualization.
5. Display both original and transformed objects.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.art3d import Line3DCollection

def draw_edges(ax, vertices, edges, color='b'):
    lines = [(vertices[start], vertices[end]) for start, end in edges]
    ax.add_collection3d(Line3DCollection(lines, colors=color))

def translation_matrix(tx, ty, tz):
    return np.array([[1, 0, 0, tx],
                     [0, 1, 0, ty],
                     [0, 0, 1, tz],
                     [0, 0, 0, 1]])

def scaling_matrix(sx, sy, sz):
    return np.array([[sx, 0, 0, 0],
```

```
                [0, sy, 0, 0],
                [0, 0, sz, 0],
                [0, 0, 0, 1]])

def rotation_matrix_z(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]])

def apply_transform(vertices, matrix):
    transformed = []
    for v in vertices:
        vec = np.array([*v, 1])
        result = matrix @ vec
        transformed.append(result[:3])
    return transformed
# Define Cube
vertices = [(0,0,0), (1,0,0), (1,1,0), (0,1,0),
          (0,0,1), (1,0,1), (1,1,1), (0,1,1)]

edges = [(0,1),(1,2),(2,3),(3,0),
        (4,5),(5,6),(6,7),(7,4),
        (0,4),(1,5),(2,6),(3,7)]

# Apply transformations
T = translation_matrix(2, 2, 0)
S = scaling_matrix(1.5, 1.5, 1.5)
R = rotation_matrix_z(45)

transformed_vertices = apply_transform(vertices, T @ S @ R)
# Plotting
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
draw_edges(ax, vertices, edges, 'blue') # Original cube
draw_edges(ax, transformed_vertices, edges, 'red') # Transformed cube

ax.set_title("3D Transformation of Cube")
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_box_aspect([1,1,1])
plt.show()
```



Result:

The user was able to perform translation, scaling, and rotation on a 3D cube. The transformed cube was successfully rendered and visualized.