BANGLADESH UNIVERSITY OF BUSINESS AND TECHNOLOGY

Committed to Academic Excellence

Lab Report

Title	Marks
Course Code: CSE-342 **Course Title**: Computer Graphics **Lab no**: 06	

Submitted By	Submitted To
Name: Junaeid Hasan Bijoy **ID**: 21225103299 **Intake**: 49 **Section**: 05 Department of CSE Bangladesh University of Business & Technology	**Name**: Md. Khairul Islam **Designation**: Lecturer Department of CSE Bangladesh University of Business & Technology

Submission Date

Teacher's Signature

1. **Title**: Geometric Composite Transformation of Triangle and Rectangle.

2. **Objective**:

The main objectives of this lab are learn how to understand and apply geometric transformations on simple shapes like triangles and rectangles. We will perform composite transformations by combining translation, rotation, scaling, and reflection. The goal is to see how these
transformations change the shape's position, size, and orientation on a 2D plane.

3. **Environment**:

- Operating System: Windows
- Programming Language: Python
- Editor: Jupyter Notebook
- Graphics Library: matplotlib
- Hardware: Standard computer system with basic graphics capabilities

4. **Introduction**:

Geometric transformations are operations that move or change a shape while preserving some of its properties. In computer graphics, transformations are fundamental to manipulating objects
for modeling, animation, and rendering. A composite transformation is a sequence of multiple transformations applied to a shape to achieve complex effects. This experiment focuses on applying
composite transformations such as translation, rotation, scaling, and reflection on two basic shapes — a
triangle and a rectangle — and observing how their positions and orientations change.

5. 1.**Algorithm For Ractangle**:

Step 1: Input

- Take initial coordinates of the object (e.g., list of x and y points).

Step 2: Scaling

- For each point (x, y):
  - Multiply x by scaling factor sx.
  - Multiply y by scaling factor sy.
- Store new points as (x1, y1).

Step 3: Translation

- For each scaled point (x1, y1):
  - Add translation factor tx to x1.

        - Add translation factor ty to y1.
    - Store new points as (x2, y2).

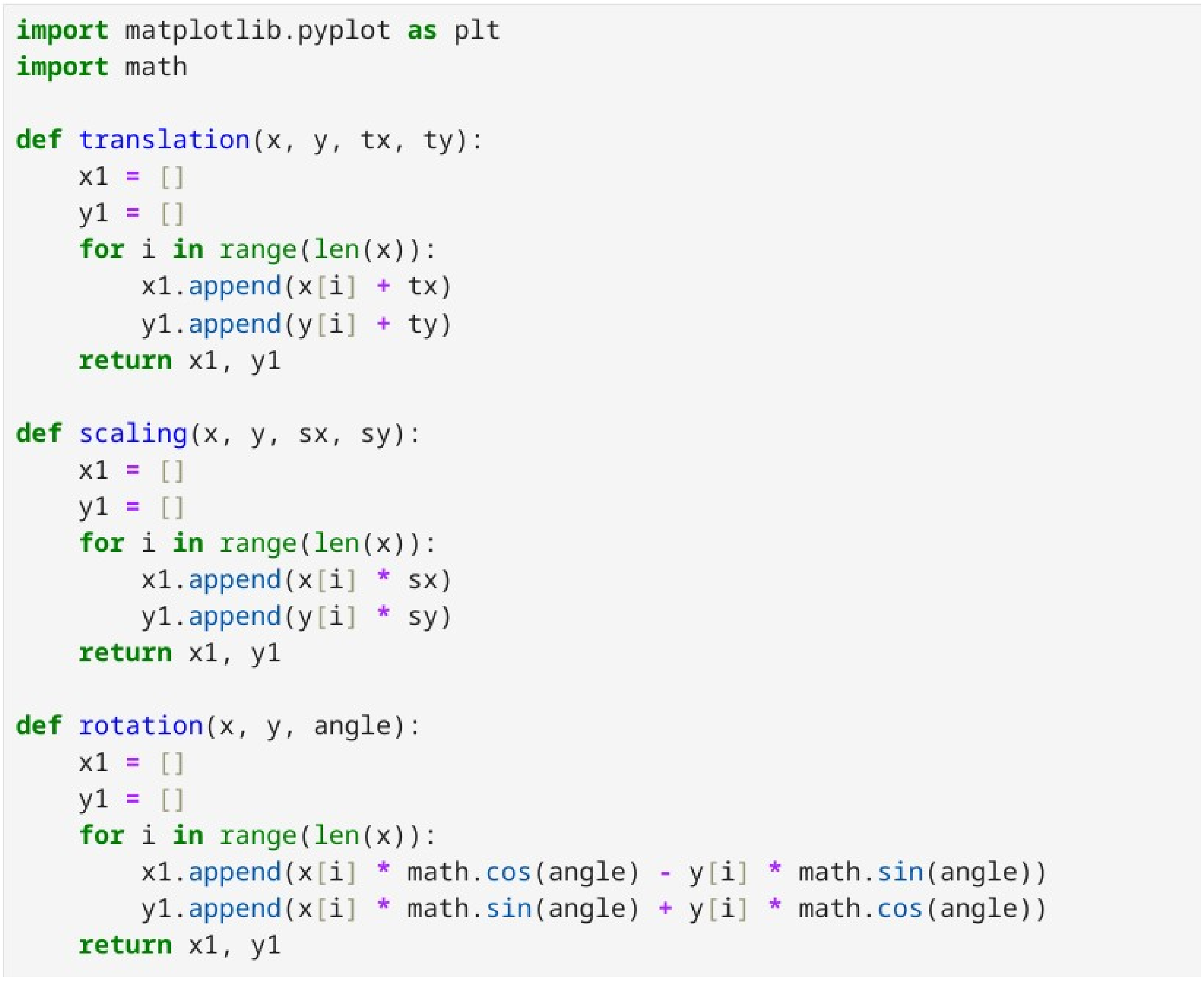
Step 4: Rotation

- For each translated point (x2, y2):
    - Rotate by a given angle θ (in radians).
    - Calculate:
        - New x-coordinate: $x3 = x2 * \cos(\theta) - y2 * \sin(\theta)$
        - New y-coordinate: $y3 = x2 * \sin(\theta) + y2 * \cos(\theta)$
- Store rotated points as (x3, y3).

Step 5: Output

- Plot or return the final transformed points (x3, y3).

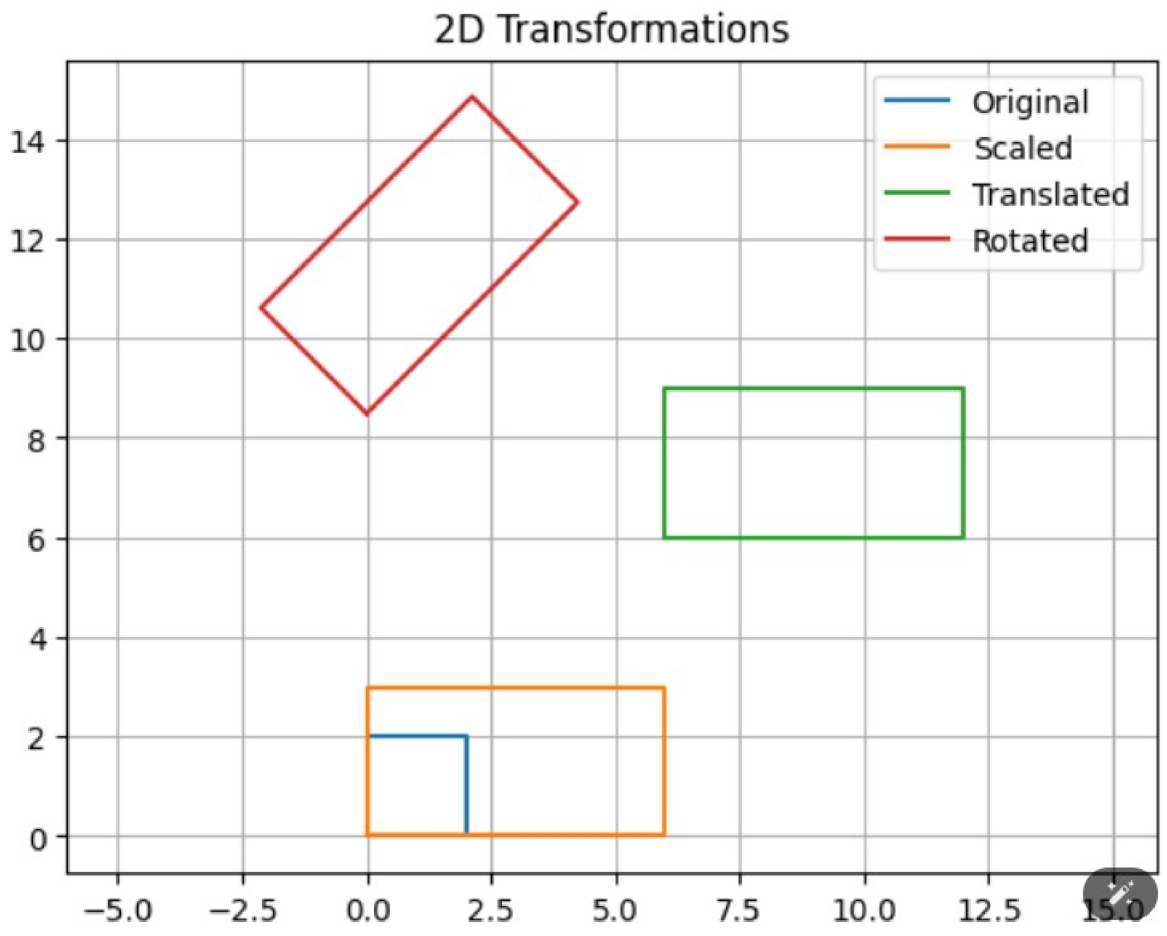
6. 1.**Code**:



```python
import matplotlib.pyplot as plt
import math

def translation(x, y, tx, ty):
    x1 = []
    y1 = []
    for i in range(len(x)):
        x1.append(x[i] + tx)
        y1.append(y[i] + ty)
    return x1, y1

def scaling(x, y, sx, sy):
    x1 = []
    y1 = []
    for i in range(len(x)):
        x1.append(x[i] * sx)
        y1.append(y[i] * sy)
    return x1, y1

def rotation(x, y, angle):
    x1 = []
    y1 = []
    for i in range(len(x)):
        x1.append(x[i] * math.cos(angle) - y[i] * math.sin(angle))
        y1.append(x[i] * math.sin(angle) + y[i] * math.cos(angle))
    return x1, y1
```

```
# Define original square coordinates
x = [0, 2, 2, 0, 0]
y = [0, 0, 2, 2, 0]

# Plot original shape
plt.plot(x, y, label="Original")

# Scale by 3 (x) and 1.5 (y)
x1, y1 = scaling(x, y, 3, 1.5)
plt.plot(x1, y1, label="Scaled")

# Translate by (-4, +3)
x2, y2 = translation(x1, y1, 6, 6)
plt.plot(x2, y2, label="Translated")

# Rotate by 45 degrees
x3, y3 = rotation(x2, y2, math.pi/4)
plt.plot(x3, y3, label="Rotated")

# Final plot settings
plt.legend()
plt.axis('equal')  # Keep aspect ratio
plt.grid(True)
plt.title("2D Transformations")
plt.show()
```

7.1.**Output**:


2D Transformations
Original
Scaled
Translated
Rotated
14
12
10
8
6
4
2
0
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0

5.2 Algorithm For Triangle:

Step 1: Input

- Read the coordinates of the original shape: lists x and y.
- Read scaling factors sx and sy.
- Read translation values tx and ty.
- Read rotation angle theta (in radians).

Step 2: Scaling

- For each point (x[i], y[i]):
    - Multiply x[i] by sx → scaled x-coordinate.
    - Multiply y[i] by sy → scaled y-coordinate.
- Save the new points in lists x1, y1.

Step 3: Translation

- For each scaled point (x1[i], y1[i]):
    - Add tx to x1[i] → translated x-coordinate.
    - Add ty to y1[i] → translated y-coordinate.
- Save the new points in lists x2, y2.

Step 4: Rotation

- For each translated point (x2[i], y2[i]):
    - New x-coordinate:
      x3[i] = x2[i] * cos(theta) - y2[i] * sin(theta)
    - New y-coordinate:
      y3[i] = x2[i] * sin(theta) + y2[i] * cos(theta)
- Save the new points in lists x3, y3.

Step 5: Output

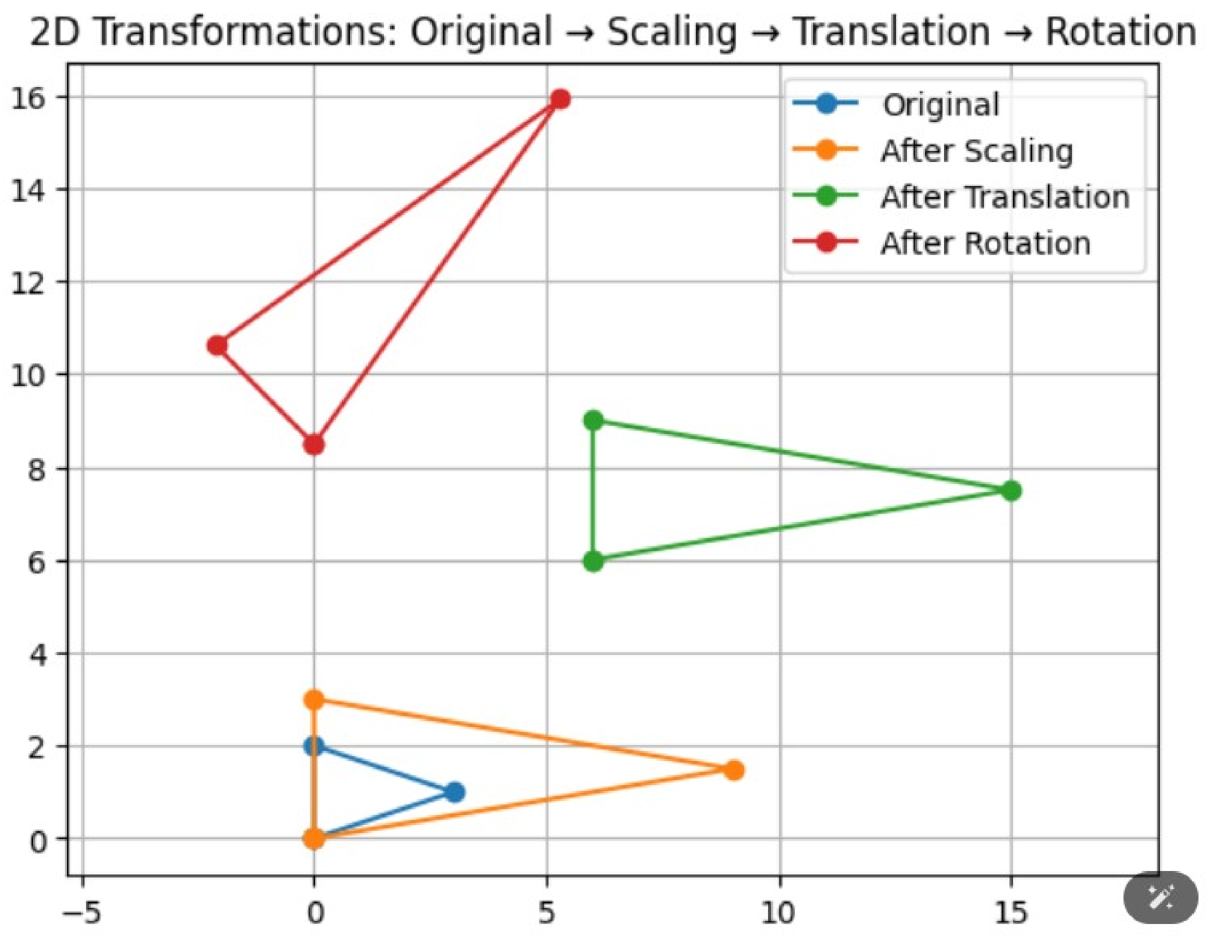
- Plot the original and the transformed shape using matplotlib.

6.2.**Code For Triangle**

```
import matplotlib.pyplot as plt
import math

def translation(x, y, tx, ty):
    x1 = []
    y1 = []
    for i in range(len(x)):
        x1.append(x[i] + tx)
        y1.append(y[i] + ty)
    return x1, y1

def scaling(x, y, sx, sy):
    x1 = []
    y1 = []
    for i in range(len(x)):
        x1.append(x[i] * sx)
        y1.append(y[i] * sy)
    return x1, y1

def rotation(x, y, angle):
    x1 = []
    y1 = []
    for i in range(len(x)):
        x1.append(x[i] * math.cos(angle) - y[i] * math.sin(angle))
    for i in range(len(x)):
        y1.append(x[i] * math.sin(angle) + y[i] * math.cos(angle))
    return x1, y1
```

```python
# Original coordinates
x = [0, 3, 0, 0]
y = [0, 1, 2, 0]

# Plot original shape
plt.plot(x, y, label="Original", marker='o')

# Scaling
x1, y1 = scaling(x, y, 3, 1.5)
plt.plot(x1, y1, label="After Scaling", marker='o')

# Translation
x2, y2 = translation(x1, y1, 6, 6)
plt.plot(x2, y2, label="After Translation", marker='o')

# Rotation
x3, y3 = rotation(x2, y2, math.pi/4)  # Rotate by 45 degrees
plt.plot(x3, y3, label="After Rotation", marker='o')

# Final plot settings
plt.legend()
plt.grid(True)
plt.axis('equal')
plt.title("2D Transformations: Original → Scaling → Translation → Rotation")
plt.show()
```

7.2.**Output**:



8. **Discussion & conclusion**:

Through this lab, we learned how to perform and combine multiple geometric transformations on basic shapes. We observed that composite transformations could significantly alter a shape's appearance and position. This experiment enhanced our understanding of transformation matrices, the importance of the order of transformations, and their practical applications in computer graphics, animation, and simulations.