Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Mini Report

on

"Lab 4"

[Course Code: COMP 342]

(For partial fulfillment of III Year/ I Semester in Computer Science)

Submitted By

Nisham Ghimire (17)

Submitted To

Mr. Dhiraj Shrestha

Department of Computer Science and Engineering

Submission Date

14th January, 2024

LAB 4

1. *Write a Program to implement:*
    a. *2D Translation*
    b. *2D Rotation*
    c. *2D Scaling*
    d. *2D Reflection*
    e. *2D Shearing*

(For doing these Transformations consider any 2D shapes (Line, Triangle, Rectangle etc), and use Homogeneous coordinate Systems)

1. 2D Translation:

Algorithm:

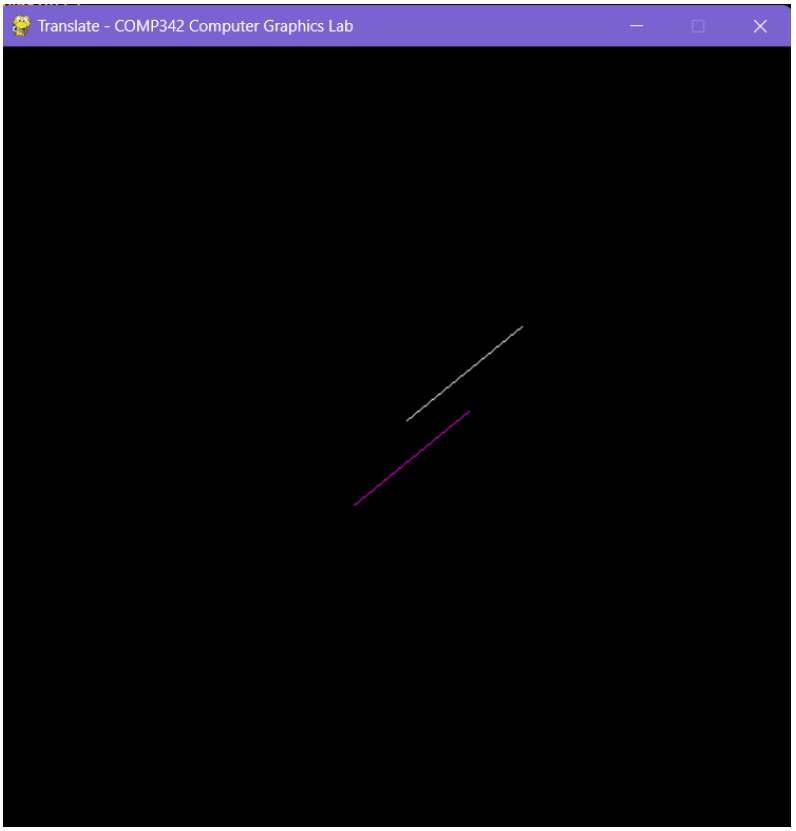
i. Create an empty set for the translated points *'P'*.
ii. For each point in the original set:
a. Take a Point *P*(*x*,*y*):
Calculate the translated x-coordinate *x'=x+dx*.Calculate the translated y- coordinate *y'=y+dy*.
b. Add the Translated Point to *'P'*:
Add the point *P'*(*x'*,*y'*) to the set of translated points *'P'*.
iii. Display the set of translated points *'P'*.

Source Code:

```
import numpy as np
from typing import Tuple
import pygame as pg
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLU import *
from math import *

Coordinate = Tuple[float, float]

def translate(point: Coordinate, translateX_by: int, translateY_by: int) -> Coordinate:
    x, y = point
    m = np.array([[x], [y], [1]])
    tx_m = np.array([[1, 0, translateX_by], [0, 1, translateY_by], [0, 0, 1]])
    result = np.dot(tx_m, m)
    return tuple(result[:2, 0])

def displayPaint():
    st_point: Coordinate = (-4, -6)
    end_point: Coordinate = (7, 3)
    tsl_By = (5, 8)
    st_tps = translate(st_point, tsl_By[0], tsl_By[1])
    end_tps = translate(end_point, tsl_By[0], tsl_By[1])

    glBegin(GL_LINES)
    glColor3f(1.0, 0.0, 1.0)
    glVertex2f(st_point[0], st_point[1])
    glVertex2f(end_point[0], end_point[1])
    glEnd()

    glBegin(GL_LINES)
    glColor3f(1.0, 1.0, 1.0)
    glVertex2f(st_tps[0], st_tps[1])
```

```
    glVertex2f(end_tps[0], end_tps[1])
    glEnd()

def main():
    pg.init()
    pg.display.set_mode((600, 600), DOUBLEBUF | OPENGL | GL_RGB)
    pg.display.set_caption("Translate - COMP342 Computer Graphics Lab")

    gluPerspective(150, 1, 1, 10)
    glTranslatef(0.0, 0.0, -10)

    while True:
        for ev in pg.event.get():
            if ev.type == pg.QUIT:
                pg.quit()
                quit()

        displayPaint()
        pg.display.flip()

if __name__ == "__main__":
    main()
```

Output:



2. 2D Scaling:

Algorithm:

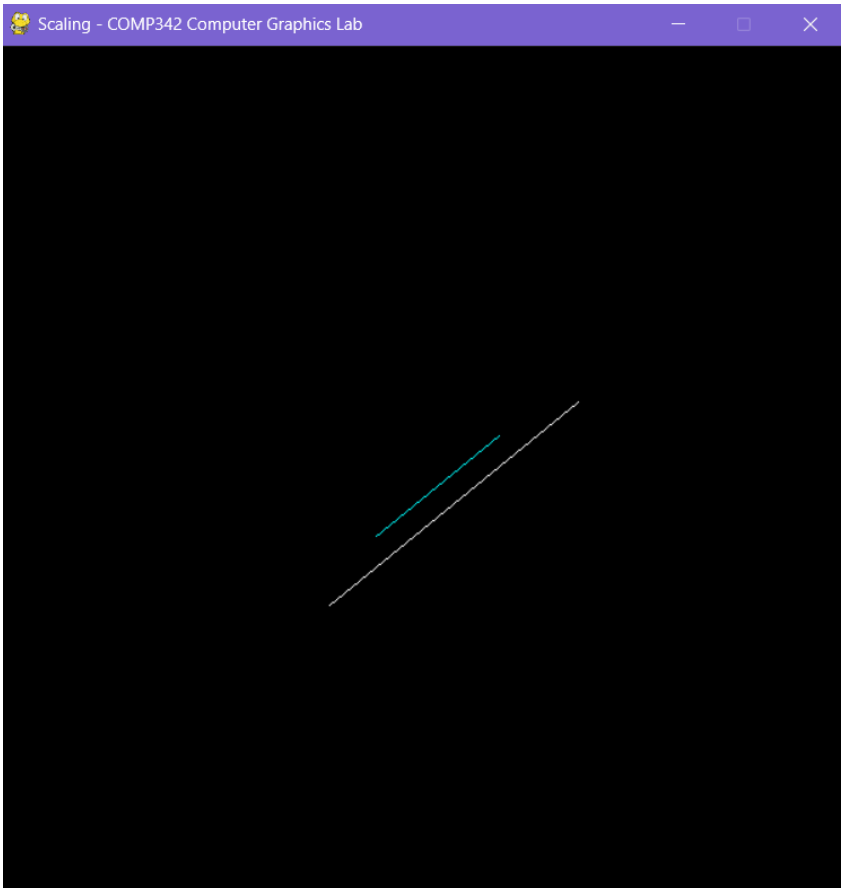
1. Input the coordinates of the 2D point: (x, y).
2. Input the scaling factors for the x-axis (Sx) and y-axis (Sy).
3. Compute the scaled coordinates using the following formulas:
    - x' = x * Sx
    - y' = y * Sy
4. The new coordinates (x', y') represent the scaled point.

Source Code:

```
import numpy as np
from typing import Tuple
import pygame as pg
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLU import *
from math import *

Coordinate = Tuple[float, float]

def scale(point: Coordinate, scaleX_by: int, scaleY_by: int) -> Coordinate:
    x, y = point
    m = np.array([[x], [y], [1]])
    tx_m = np.array([[scaleX_by, 0, 0], [0, scaleY_by, 0], [0, 0, 1]])
    result = np.dot(tx_m, m)
    return tuple(result[:2, 0])

def displayPaint():
    st_point: Coordinate = (-4, -6)
    end_point: Coordinate = (7, 3)
    scale_By = (2, 2)
    st_tps = scale(st_point, scale_By[0], scale_By[1])
    end_tps = scale(end_point, scale_By[0], scale_By[1])

    glBegin(GL_LINES)
    glColor3f(0.0, 1.0, 1.0)
    glVertex2f(st_point[0], st_point[1])
    glVertex2f(end_point[0], end_point[1])
    glEnd()

    glBegin(GL_LINES)
    glColor3f(1.0, 1.0, 1.0)
    glVertex2f(st_tps[0], st_tps[1])
    glVertex2f(end_tps[0], end_tps[1])
    glEnd()

def main():
    pg.init()
    pg.display.set_mode((600, 600), DOUBLEBUF | OPENGL | GL_RGB)
    pg.display.set_caption("Scaling - COMP342 Computer Graphics Lab")

    gluPerspective(150, 1, 1, 10)
    glTranslatef(0.0, 0.0, -10)

    while True:
        for ev in pg.event.get():
            if ev.type == pg.QUIT:
                pg.quit()
                quit()
```

```
        displayPaint()
        pg.display.flip()

if __name__ == "__main__":
    main()
```

Output:



3. 2D Rotation:

Algorithm:

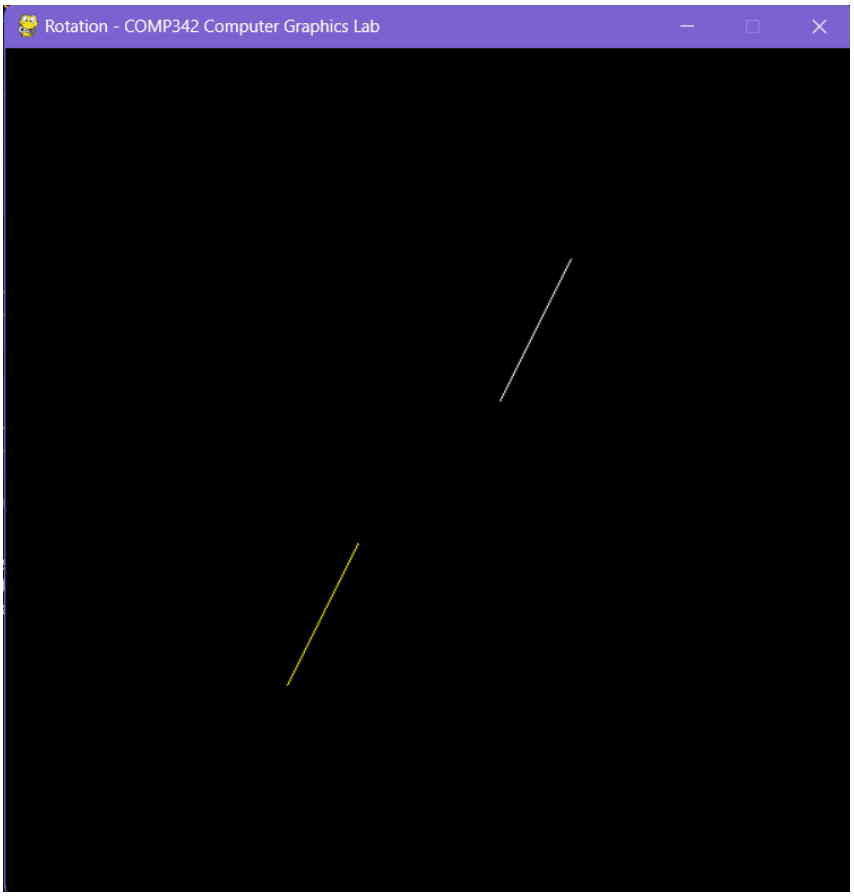
1. Input:
   - Coordinates of the point to be rotated: (x,y)
   - Coordinates of the center of rotation: (h,k)
   - Rotation angle: θ (in degrees or radians)
2. Translate to Origin:
   - Translate the system so that the center of rotation becomes the origin: $x'=x-h,\ y'=y-k$
3. Rotate Around Origin:
   - Use the following formulas to calculate the new coordinates (x'',y'') after rotation aroundthe translated origin: $x''=x'\cdot\cos(\theta)-y'\cdot\sin(\theta)$, $y''=x'\cdot\sin(\theta)+y'\cdot\cos(\theta)$
4. Translate Back:
   - Translate the system back to its original position: $x'''=x''+h,\ y'''=y''+k$
5. Output:
   - The new coordinates (x''',y''') represent the result of the rotation.

Source Code:

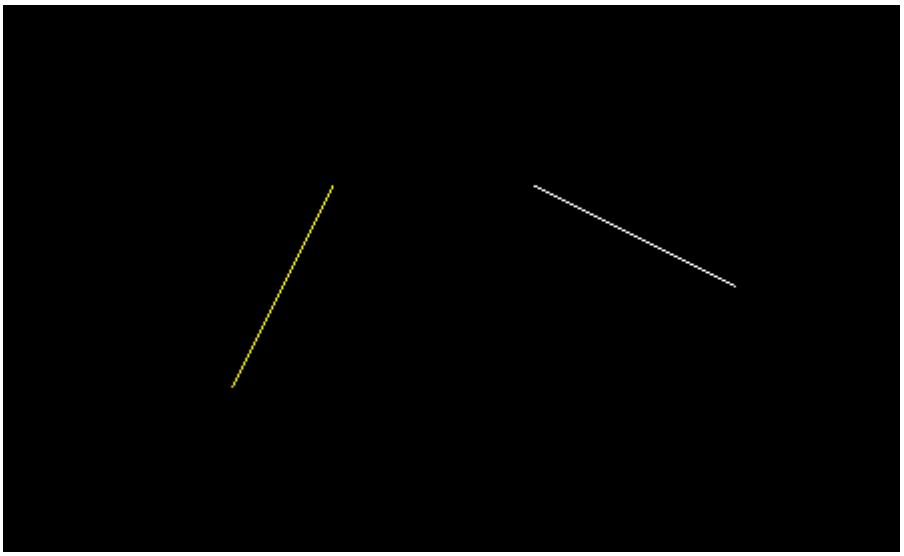
```python
import numpy as np
from typing import Tuple
import pygame as pg
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLU import *
from math import *

Coordinate = Tuple[float, float]

def rotate(point: Coordinate, rotateBy: float) -> Coordinate:
    x, y = point
    m = np.array([[x], [y], [1]])
    a = np.deg2rad(rotateBy)
    tx_m = np.array([[np.cos(a), -np.sin(a), 0], [np.sin(a), np.cos(a), 0], [0, 0, 1]])
    result = np.dot(tx_m, m)
    return tuple(result[:2, 0])

def displayPaint():
    st_point: Coordinate = (2, 2)
    end_point: Coordinate = (4, 6)
    rotateByAngle = 180
    st_tps = rotate(st_point, rotateByAngle)
    end_tps = rotate(end_point, rotateByAngle)

    glBegin(GL_LINES)
    glColor3f(1.0, 1.0, 0.0)
    glVertex2f(st_point[0], st_point[1])
    glVertex2f(end_point[0], end_point[1])
    glEnd()

    glBegin(GL_LINES)
    glColor3f(1.0, 1.0, 1.0)
    glVertex2f(st_tps[0], st_tps[1])
    glVertex2f(end_tps[0], end_tps[1])
    glEnd()
```

```python
def main():
    pg.init()
    pg.display.set_mode((600, 600), DOUBLEBUF | OPENGL | GL_RGB)
    pg.display.set_caption("Rotation - COMP342 Computer Graphics Lab")

    gluPerspective(260, 1, 1, 10)
    glTranslatef(0.0, 0.0, -10)

    while True:
        for ev in pg.event.get():
            if ev.type == pg.QUIT:
                pg.quit()
                quit()

        displayPaint()
        pg.display.flip()

if __name__ == "__main__":
    main()
```

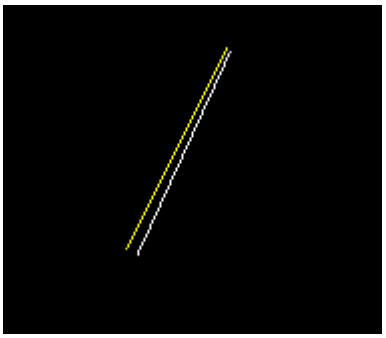
Output:(180 degree)



90 Degree



2 Degree



4. Reflection

Algorithm:

1. Input:
    - Read the set of 2D points.
2. Initialize:
    - Set up an empty list to store the transformed points.
3. Reflection Transformation:
    - For each point (x, y) in the original set:
        - Compute the reflected point (x', y') using the formula:
            - $x' = x$
            - $y' = -y$
4. Store Transformed Points:
    - Add the reflected point (x', y') to the list of transformed points.
5. Output:
    - The list of transformed points represents the 2D reflection along the x-axis.

Source Code:

```
import pygame
from pygame.locals import *
from OpenGL.GL import *
from math import cos, sin, radians

# Initial coordinates of the line in homogeneous coordinates
line = [[-0.5, -0.5, 1], [0.5, -0.5, 1]]
line1 = [[], []]
reflection_axis = 'x'


def draw_line(line, color):
    glBegin(GL_LINES)
    glColor3f(color[0], color[1], color[2])  # line color
    glVertex2f(line[0][0] / line[0][2], line[0][1] / line[0][2])
    glVertex2f(line[1][0] / line[1][2], line[1][1] / line[1][2])
    glEnd()


def reflect(line):
    if reflection_axis == 'x':
        reflection_matrix = [
            [1, 0, 0],
            [0, -1, 0],
            [0, 0, 1]
        ]
    elif reflection_axis == 'y':
        reflection_matrix = [
            [-1, 0, 0],
            [0, 1, 0],
            [0, 0, 1]
        ]
    else:
        raise ValueError("Invalid reflection axis")

    new_line = []
    for i in range(len(line)):
        x, y, w = line[i]
        new_coords = [
            reflection_matrix[0][0] * x + reflection_matrix[0][1] * y +
            reflection_matrix[0][2] * w,
            reflection_matrix[1][0] * x + reflection_matrix[1][1] * y +
            reflection_matrix[1][2] * w,
            reflection_matrix[2][0] * x + reflection_matrix[2][1] * y +
            reflection_matrix[2][2] * w
        ]
        new_line.append(new_coords)
    return new_line


def main():
    pygame.init()
    display = (800, 600)
    pygame.display.set_mode(display, DOUBLEBUF | OPENGL)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    glOrtho(-1, 1, -1, 1, -1, 1)
    glMatrixMode(GL_MODELVIEW)
```

```
    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                quit()

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        draw_line(line, [1.0, 1.0, 1.0])  # Original line in white
        line1 = reflect(line)
        draw_line(line1, [1.0, 0.0, 0.0])  # Reflected line in red
        pygame.display.flip()
        pygame.time.wait(10)


if __name__ == "__main__":
    main()
```

Output:

5. Shearing

Algorithm:

Horizontal Shear:

1. Input:
   - x,y coordinates of the point.
   - Shear factor k.
2. Calculate the new x coordinate in the sheared system: $x'=x+k\cdot y$
3. Keep the y coordinate unchanged: $y'=y$
4. Output:
   - The new coordinates (x',y') represent the sheared point.

Vertical Shear:

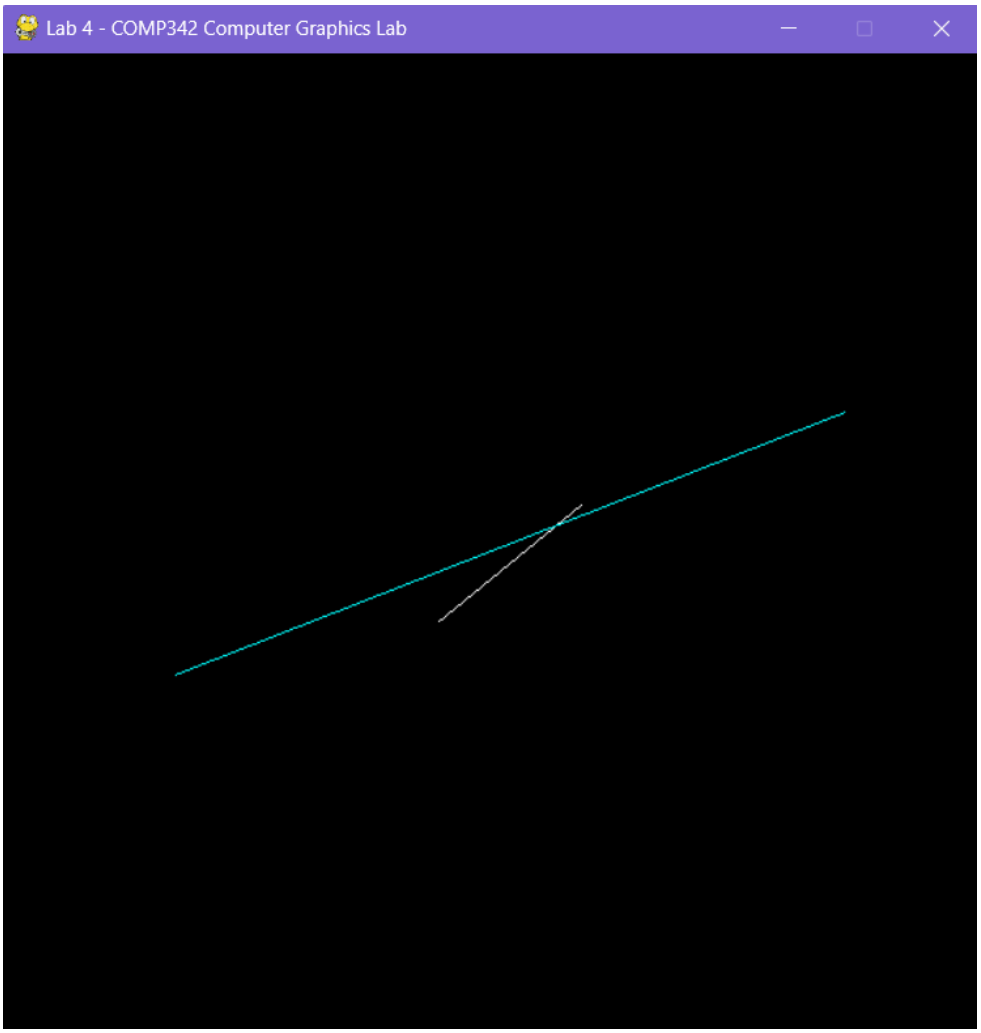
1. Input:
    - x,y coordinates of the point.
    - Shear factor k.
2. Keep the x coordinate unchanged: $x'=x$
3. Calculate the new y coordinate in the sheared system: $y'=y+k.x$
4. Output:
    - The new coordinates (x',y') represent the sheared point.

Source Code:

```
import numpy as np
from typing import Tuple
import pygame as pg
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLU import *
from math import *

Coordinate = Tuple[float, float]

def shear(point: Coordinate, shearX_by: int, shearY_by: int) -> Coordinate:
    x, y = point
    m = np.array([[x], [y], [1]])
    shearX_m = np.array([[1, shearX_by, 0], [0, 1, 0], [0, 0, 1]])
    shearY_m = np.array([[1, 0, 0], [shearY_by, 1, 0], [0, 0, 1]])

    composite_m = np.dot(shearX_m, shearY_m)
    sheared_m = np.dot(composite_m, m)

    xT, yT, _ = sheared_m
    return (xT[0], yT[0])

def displayPaint():
    st_point: Coordinate = (-4, -6)
    end_point: Coordinate = (7, 3)
    shear_By = (2, 1)
    st_tps = shear(st_point, shear_By[0], shear_By[1])
    end_tps = shear(end_point, shear_By[0], shear_By[1])

    glBegin(GL_LINES)
    glColor3f(1.0, 1.0, 1.0)
    glVertex2f(st_point[0], st_point[1])
    glVertex2f(end_point[0], end_point[1])
    glEnd()

    glBegin(GL_LINES)
    glColor3f(0.0, 1.0, 1.0)
    glVertex2f(st_tps[0], st_tps[1])
    glVertex2f(end_tps[0], end_tps[1])
    glEnd()

def main():
    pg.init()
    pg.display.set_mode((600, 600), DOUBLEBUF | OPENGL | GL_RGB)
    pg.display.set_caption("Shearing - COMP342 Computer Graphics Lab")

    gluPerspective(150, 1, 1, 10)
    glTranslatef(0.0, 0.0, -10)
```

```
    while True:
        for ev in pg.event.get():
            if ev.type == pg.QUIT:
                pg.quit()
                quit()

        displayPaint()
        pg.display.flip()

if __name__ == "__main__":
    main()
```

Output:



Conclusion:

In this lab, I delved into the world of 2D transformations using Python and OpenGL. I played around with translation, reflection, shearing, rotation, and scaling of geometric shapes in a 2D space. The hands-on work with transformation matrices and OpenGL's rendering tools emphasized the importance of grasping 2D transformation matrices in computer graphics. Connecting theory to application, I saw firsthand how these mathematical operations directly influenced graphical elements, showcasing their crucial role in crafting engaging images, particularly in the realm of digital art.