Experiment V:2D transformations

Write a menu driven program to Implement 2D transformations on an equilateral triangle

A: Translation

B: Rotation

C: Scaling

D: Reflection

ALGORITHM

1. START
2. Display Menu
3. Input Choice
4. Call the functions according to choice
    a. If choice = 1
       Do translation by inputting the amount to be translated and add it to coordinate
    b. If choice = 2
       Take choice again to get rotation by origin or arbitrary point and perform rotation and update each point with as x = x.cos(theta) - y.sin(theta) and y = y.cos(theta) + x.sin(theta)
    c. If choice = 3
       Take choice again to get scale by origin or arbitrary point and perform scaling by multiplying with the coordinate
    d. If choice = 4
       Display another menu to show 6 modes of reflection and get choice and perform reflection by the specified method
5. STOP

PROGRAM

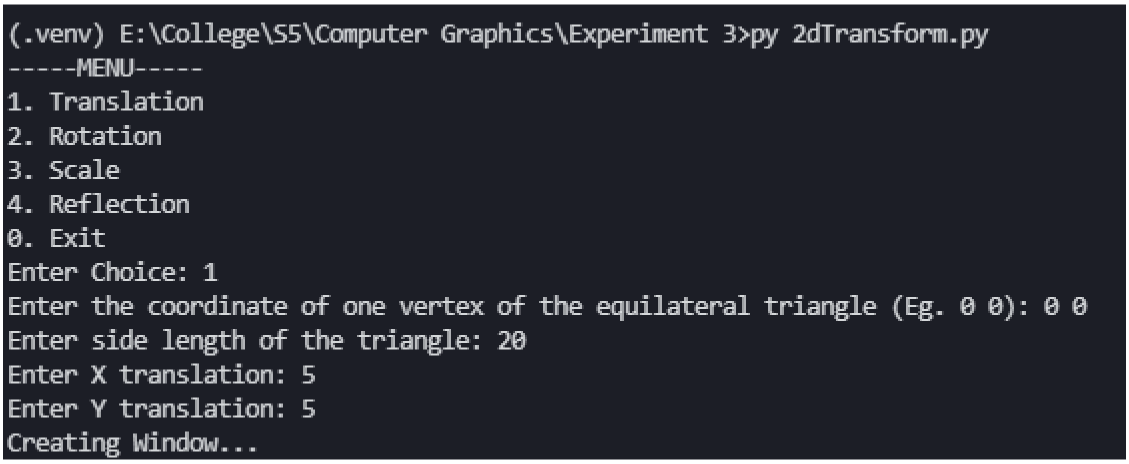
```
# Importing dependencies

import OpenGL

from OpenGL.GL import *

from OpenGL.GLU import *

from OpenGL.GLUT import *


from itertools import permutations


import sys

from math import cos, sin, pi, sqrt, atan
```

```
# Constants to set window size and size of points
WINDOW_POSITION = 100


# Clear screen and set origin
def init() -> None:
    glClearColor(0.0, 0.0, 0.0, 1.0)
    gluOrtho2D(-WINDOW_POSITION, WINDOW_POSITION, -WINDOW_POSITION, 
WINDOW_POSITION)


# Function to display menu
def display_menu() -> int:
    print("-----MENU-----")
    print(f"1. Translation")
    print(f"2. Rotation")
    print(f"3. Scale")
    print(f"4. Reflection")
    print(f"0. Exit")
    return int(input("Enter Choice: "))


# Function to display window
def display_window(vertices: list, choice: int, title: str) -> None:
    # Get the vertices of transformed object
    if choice == 1:
        points = get_translate_points(vertices)
    elif choice == 2:
        while True:
            print("1. Rotated about origin")
            print("2. Rotated about an arbitary point")
            ch = int(input("Enter you choice: "))
            if ch == 1:
                points = get_rotated_points(vertices)
                break
            elif ch == 2:
                points = get_rotated_arbitary(vertices)
                break
            print("Enter a valid choice!")
    elif choice == 3:
        while True:
            print("1. Scale about origin")
            print("2. Scale about an arbitary point")
            ch = int(input("Enter you choice: "))
            if ch == 1:
                points = get_scaled_points(vertices)
                break
            elif ch == 2:
                points = get_scaled_arbitary(vertices)
                break
            print("Enter a valid choice!")

    elif choice == 4:
        points = get_reflected_points(vertices)
    else:
        points = []
    print("Creating Window...")
    glutInit(sys.argv)
    glutInitDisplayMode(GLUT_RGB)
    glutInitWindowSize(500,500)
    glutInitWindowPosition(50, 50)
    glutCreateWindow(f"{title} | Abhinav Rajesh")
    glutDisplayFunc(lambda: plot_transformation(vertices, points))
    init()
    glutMainLoop()

# Function to calculate the translated points
def get_translate_points(vertices: list) -> list:
    tx = int(input("Enter X translation: "))
    ty = int(input("Enter Y translation: "))
    points = []
    for x, y in vertices:
        points.append((x + tx, y + ty))
    return points

# Function to calculate the rotated points
def get_rotated_points(vertices: list) -> list:
    theta = (pi / 180) * int(input("Enter degrees to be rotated: "))
    points = []
    for x, y in vertices:
        points.append((round(x * cos(theta) - y * sin(theta)), round(x * sin(theta) + y * cos(theta))))
    return points

# Function to calculate the rotated points about an arbitary point
def get_rotated_arbitary(vertices: list) -> list:
    x, y = map(int, input("Enter arbitary coordinate x, y: ").split(" "))
    translated_vertices = []

    for vertice in vertices:
        translated_vertices.append((vertice[0] - x, vertice[1] - y))

    points = get_rotated_points(translated_vertices)
    translated_points = []
    for point in points:
        translated_points.append((point[0] + x, point[1] + y))

    return translated_points



# Function to calculate the scaled points
```

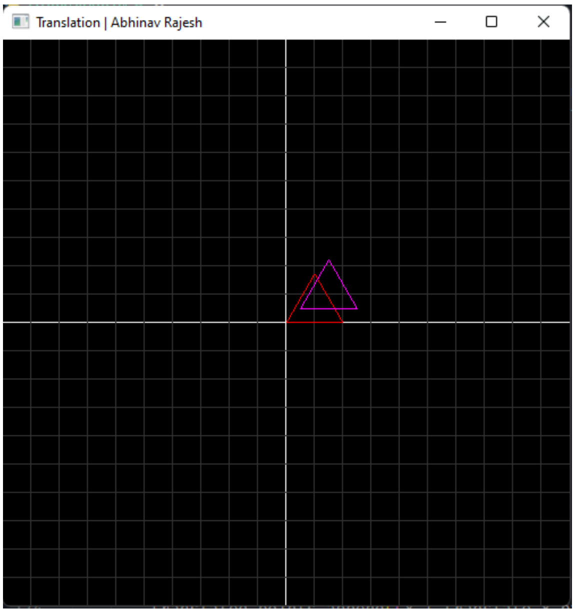
```
def get_scaled_points(vertices: list) -> list:
    tx = int(input("Enter scale along X: "))
    ty = int(input("Enter scale along Y: "))
    points = []
    for x, y in vertices:
        points.append((x * tx, y * ty))
    return points

def get_scaled_arbitary(vertices: list) -> list:
    x, y = map(int, input("Enter arbitary coordinate x, y: ").split(" "))
    translated_vertices = []

    for vertice in vertices:
        translated_vertices.append((vertice[0] - x, vertice[1] - y))

    points = get_scaled_points(translated_vertices)
    translated_points = []
    for point in points:
        translated_points.append((point[0] + x, point[1] + y))

    return translated_points

# Function to calculate the reflected points
def get_reflected_points(vertices: list) -> list:
    menu_options = {
        1: "X-axis",
        2: "Y-axis",
        3: "About origin",
        4: "x = y",
        5: "x = -y",
        6: "Arbitary line"
    }
    while True:
        try:
            print("----REFLECTION ALONG----")
            for key in menu_options.keys():
                print(f"{key}. {menu_options[key]}")
            choice = int(input("Enter your choice(1/2/3/4/5): "))
            if choice not in menu_options.keys():
                raise Exception("Invalid choice!")
            points = []
            if choice == 6:
                a, b, c = map(int, input("Enter values of a, b and c in the equation: ax + by + c = 0 : ").split("
"))
                points = get_reflected_arbitary(vertices, a, b, c)
            else:
                for x, y in vertices:
                    if choice == 1:
                        points.append((x, -y))
                    elif choice == 2:
                        points.append((-x, y))
                    elif choice == 3:
                        points.append((-x, -y))
                    elif choice == 4:
                        points.append((y, x))
                    elif choice == 5:
                        points.append((-y, -x))
            break
        except Exception as e:
            print(e)
    return points


def get_reflected_arbitary(vertices: list, a: int, b: int, c: int):
    translate_x_by = - c / a
    theta = atan(-a/b) if b != 0 else pi/2
    translated_points = []
    for x, y in vertices:
        translated_points.append((x + translate_x_by, y))
    rotated_points = []
    for x, y in translated_points:
        rotated_points.append((round(x * cos(-theta) - y * sin(-theta)), round(x * sin(-theta) + y * cos(-theta))))
    reflected_points = []
    for x, y in rotated_points:
        reflected_points.append((x, -y))
    rerotated_points = []
    for x, y in reflected_points:
        rerotated_points.append((round(x * cos(theta) - y * sin(theta)), round(x * sin(theta) + y * cos(theta))))
    retranslated_points = []
    for x, y in rerotated_points:
        retranslated_points.append((x - translate_x_by, y))
    return retranslated_points


# Function to draw perpendicular lines at a point "variable"
def draw_lines(variable: int):
    glBegin(GL_LINES)
    glVertex2f(variable,WINDOW_POSITION)
    glVertex2f(variable,-WINDOW_POSITION)
    glEnd()
    glBegin(GL_LINES)
    glVertex2f(WINDOW_POSITION,variable)
    glVertex2f(-WINDOW_POSITION,variable)
    glEnd()


# Function to plot transformation
def plot_transformation(vertices: list, points: list):
    # Plot axes and grid
    plot_axes()
    plot_grid()
    # Plot the actual object
    glColor3f(1,0,0)
    createObject(vertices)
    # Plot the transformed object
    glColor3f(1,0,1)
    createObject(points)
    glFlush()

# Create x and y axes
def plot_axes():
    glClear(GL_COLOR_BUFFER_BIT)
    glColor3f(1,1,1)
    draw_lines(0)

# Create grid for reference each unit is (WINDOW_POSITION / 10) units apart
def plot_grid():
    glColor3f(0.2, 0.2, 0.2)
    for i in range(-WINDOW_POSITION, WINDOW_POSITION, int(WINDOW_POSITION/10)):
        # Condition to not override the axes
        if i != 0:
            draw_lines(i)

#  Get the required input
def get_input() -> list:
    while True:
        try:
            x1, y1 = map(int, input("Enter the coordinate of one vertex of the equilateral triangle (Eg. 0 0): 
").split(" "))
            side = int(input("Enter side length of the triangle: "))
            break
        except ValueError as e:
            print("Please enter the values in correct format!")
    return [(x1, y1), (x1 + side, y1), (x1 + side / 2, y1 + sqrt(3) * side / 2)]


# Function to create the object
def createObject(points: list) -> None:
    final_points = permutations(points, 2)
    for point in list(final_points):
        glBegin(GL_LINES)
        glVertex2f(point[0][0],point[0][1])
        glVertex2f(point[1][0],point[1][1])
        glEnd()


def main():
    choice = 1


    titleList = {
        1: "Translation",
        2: "Rotation",
        3: "Scale",
        4: "Reflection"
    }


    while choice != 0:
        choice = display_menu()
        if choice in titleList.keys():
            # Checks if it's a valid input
            vertices = get_input()
            display_window(vertices, choice, titleList[choice])
        elif choice == 0:
            # To handle exit from program
            print("Exiting Program...")
        else:
```

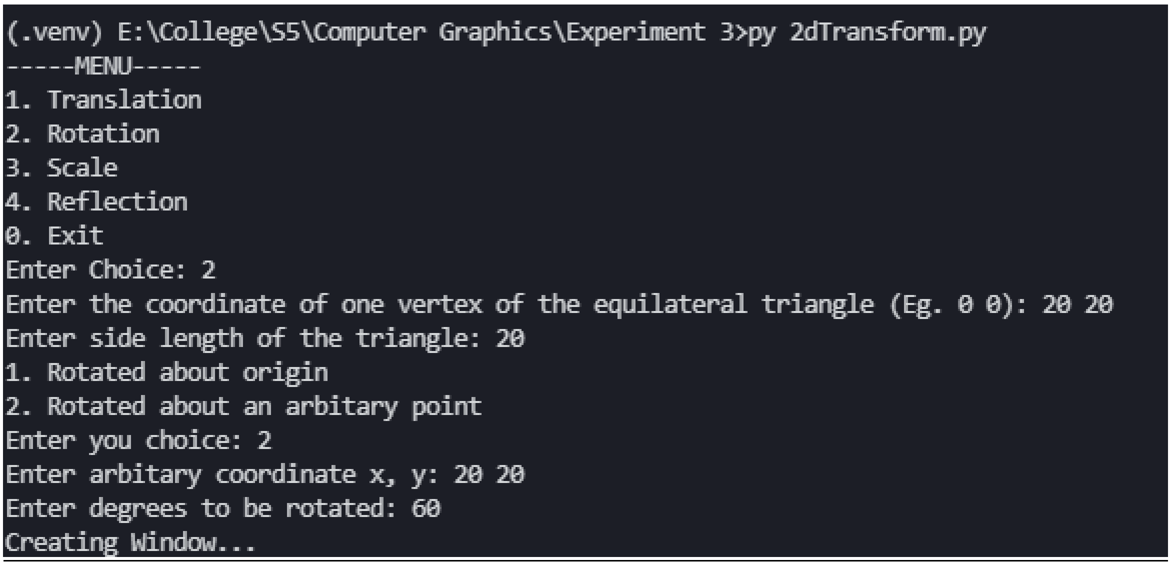
```
        # To handle invalid choice
        print("Invalid Choice! Try again.")

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

INPUT

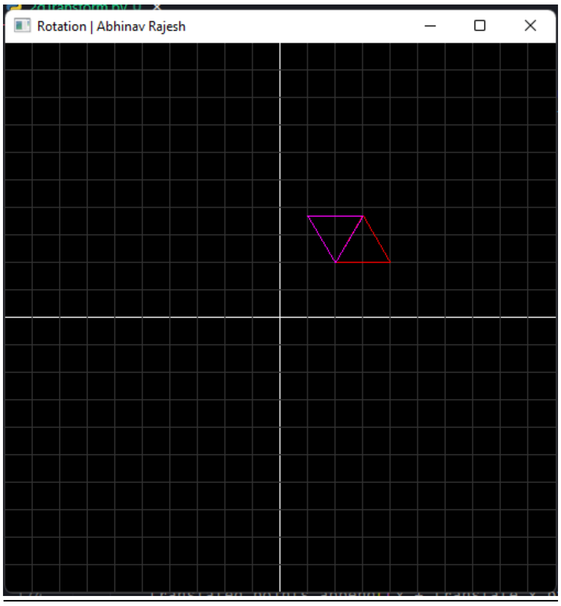


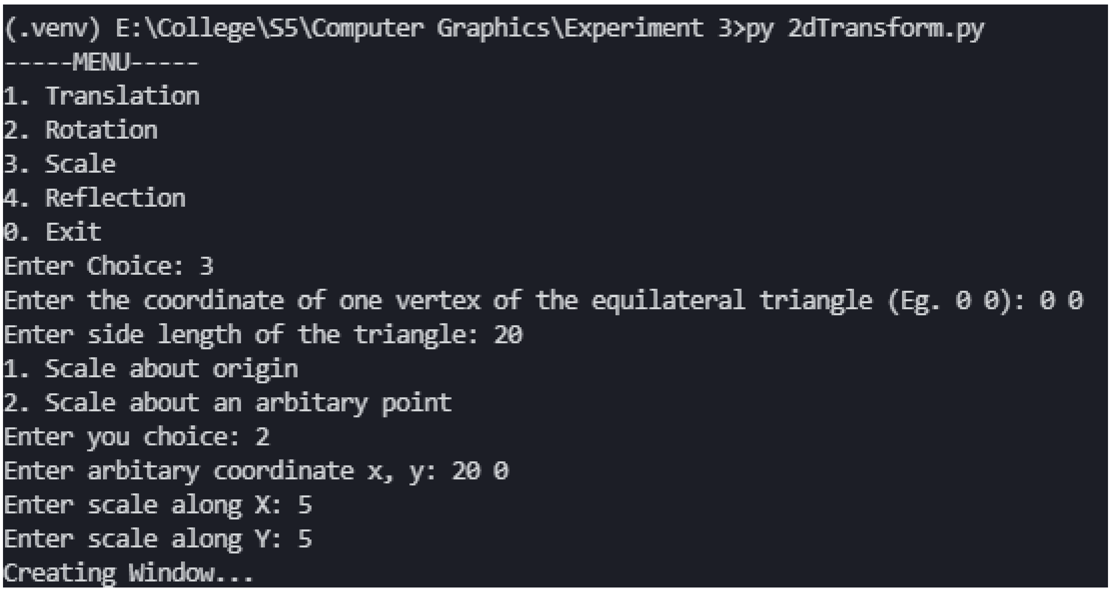
```
(.venv) E:\College\S5\Computer Graphics\Experiment 3>py 2dTransform.py
-----MENU-----
1. Translation
2. Rotation
3. Scale
4. Reflection
0. Exit
Enter Choice: 1
Enter the coordinate of one vertex of the equilateral triangle (Eg. 0 0): 0 0
Enter side length of the triangle: 20
Enter X translation: 5
Enter Y translation: 5
Creating Window...
```



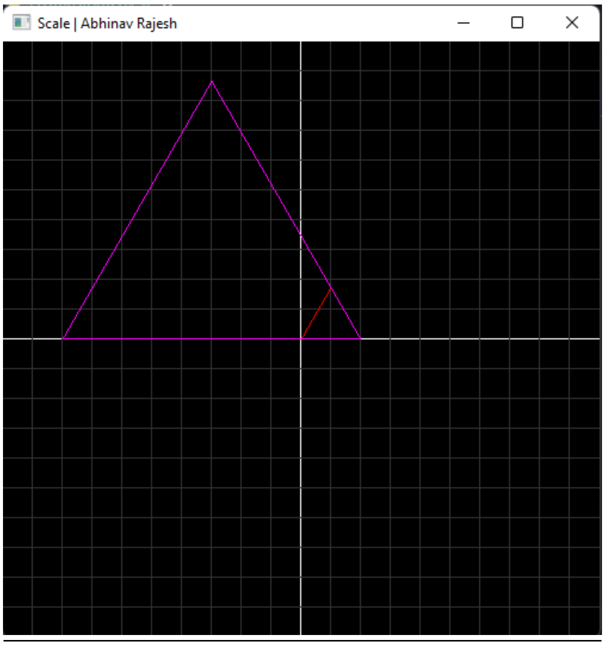


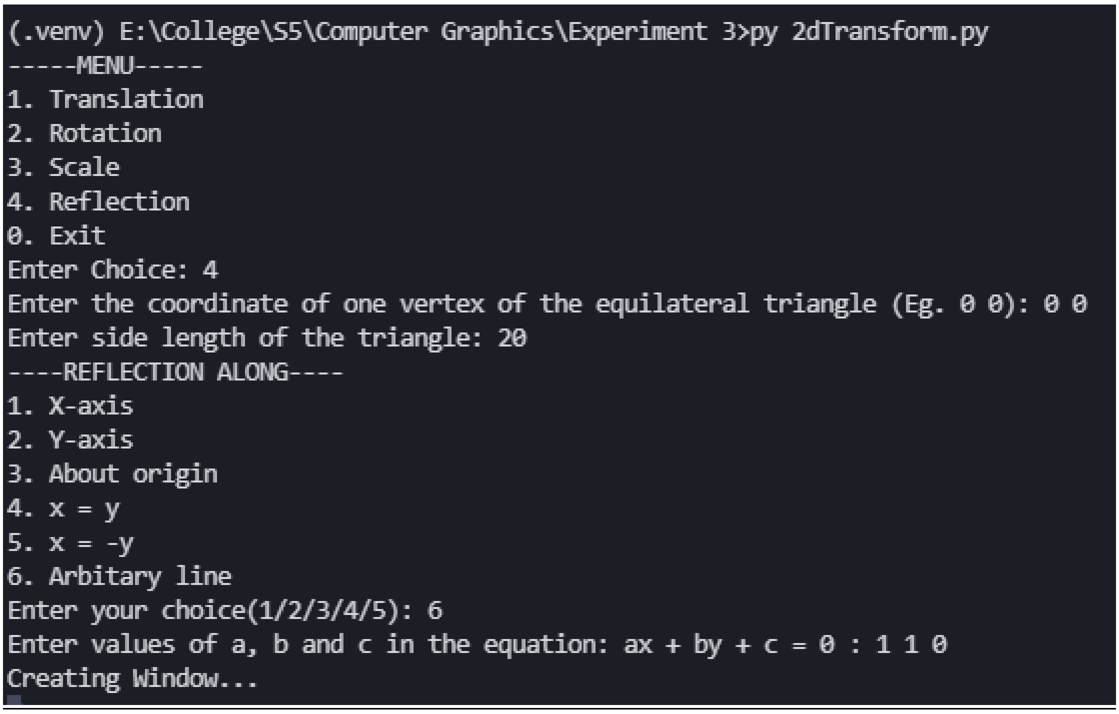
```
(.venv) E:\College\S5\Computer Graphics\Experiment 3>py 2dTransform.py
-----MENU-----
1. Translation
2. Rotation
3. Scale
4. Reflection
0. Exit
Enter Choice: 2
Enter the coordinate of one vertex of the equilateral triangle (Eg. 0 0): 20 20
Enter side length of the triangle: 20
1. Rotated about origin
2. Rotated about an arbitary point
Enter you choice: 2
Enter arbitary coordinate x, y: 20 20
Enter degrees to be rotated: 60
Creating Window...
```



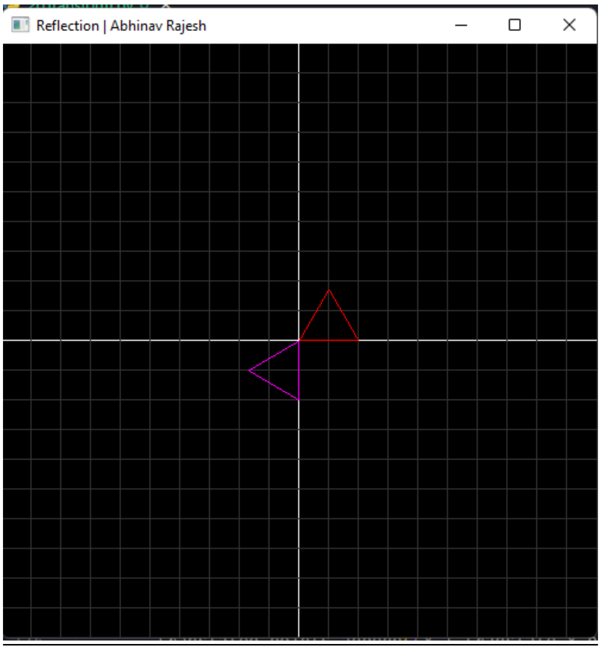


```
(.venv) E:\College\S5\Computer Graphics\Experiment 3>py 2dTransform.py
-----MENU-----
1. Translation
2. Rotation
3. Scale
4. Reflection
0. Exit
Enter Choice: 3
Enter the coordinate of one vertex of the equilateral triangle (Eg. 0 0): 0 0
Enter side length of the triangle: 20
1. Scale about origin
2. Scale about an arbitary point
Enter you choice: 2
Enter arbitary coordinate x, y: 20 0
Enter scale along X: 5
Enter scale along Y: 5
Creating Window...
```





```
(.venv) E:\College\S5\Computer Graphics\Experiment 3>py 2dTransform.py
-----MENU-----
1. Translation
2. Rotation
3. Scale
4. Reflection
0. Exit
Enter Choice: 4
Enter the coordinate of one vertex of the equilateral triangle (Eg. 0 0): 0 0
Enter side length of the triangle: 20
----REFLECTION ALONG----
1. X-axis
2. Y-axis
3. About origin
4. x = y
5. x = -y
6. Arbitary line
Enter your choice(1/2/3/4/5): 6
Enter values of a, b and c in the equation: ax + by + c = 0 : 1 1 0
Creating Window...
```


Reflection | Abhinav Rajesh