KATHMANDU UNIVERSITY

Department of Computer Science and Engineering

Dhulikhel, Kavre

COMP-342

Lab Report: III

Submitted by:

Aakash Thakur

Roll no:20

CS-30 (III/I)

Submitted to:

Mr. Dhiraj Shrestha

Department of Computer Science and Engineering

Implement Midpoint Ellipse Drawing Algorithm

Algorithm:

1. Input r_x, r_y and the ellipse center (x_c,y_c) and obtain the first point on an ellipse centered on the origin as $(x_0,y_0) = (0, r_y)$
2. Calculate the initial value of the decision parameter in region 1 as $p1_0 = r_y^2 - r_x^2 r_y + ¼ r_x^2$
3. At each x_k position in region 1, starting at k=0 perform the following test:
   If $p1_k < 0$: $x_{k+1} = x_k+1$, $y_{k+1}= y_k$
   $$p1_{k+1} = p1_k + 2r_y^2 x_{k+1} + r_y^2$$
   Else $x_{k+1} = x_k+1$, $y_{k+1}= y_k-1$
   $$p1_{k+1}=p1_k + 2r_y^2 x_{k+1} – 2r_x^2 y_{k+1} + r_y^2$$
   with
   $$2r_y^2 x_{k+1} = 2r_y^2 x_k + 2r_y^2 \text{ , } 2r_x^2 y_{k+1} = 2r_x^2 y_k - 2r_x^2$$
   And continue until $2r_y^2 x >= 2r_x^2 y$
4. Calculate the initial values of decision parameter in region 2 using the last point (x_0,y_0) calculated in region 1 as
   $$p2_0 = r_y^2 (x_0 + ½)^2 + r_x^2(y_0-1)^2 – r_x^2 r_y^2$$
5. At each y_k position in region 2, starting at k=0, perform the following test:
   If $p2_k <=0$: $x_{k+1} = x_k+1$, $y_{k+1}= y_k-1$
   $$p2_{k+1}=p2_k + 2r_y^2 x_{k+1} – 2r_x^2 y_{k+1} + r_x^2$$
   else : $x_{k+1} = x_k$, $y_{k+1}= y_k-1$
   $$p2_{k+1}=p2_k – 2r_x^2 y_{k+1} + r_x^2$$
6. Using the same incremental calculations for x and y as in region 1.
7. Determine the symmetry points in the other three quadrants.
8. Move each calculated pixel position (x,y) onto the elliptical path centered on (x_c,y_c) and plot the co-ordinate values:
   $$x = x + x_c \text{ , } y = y + y_c$$

Source Code:

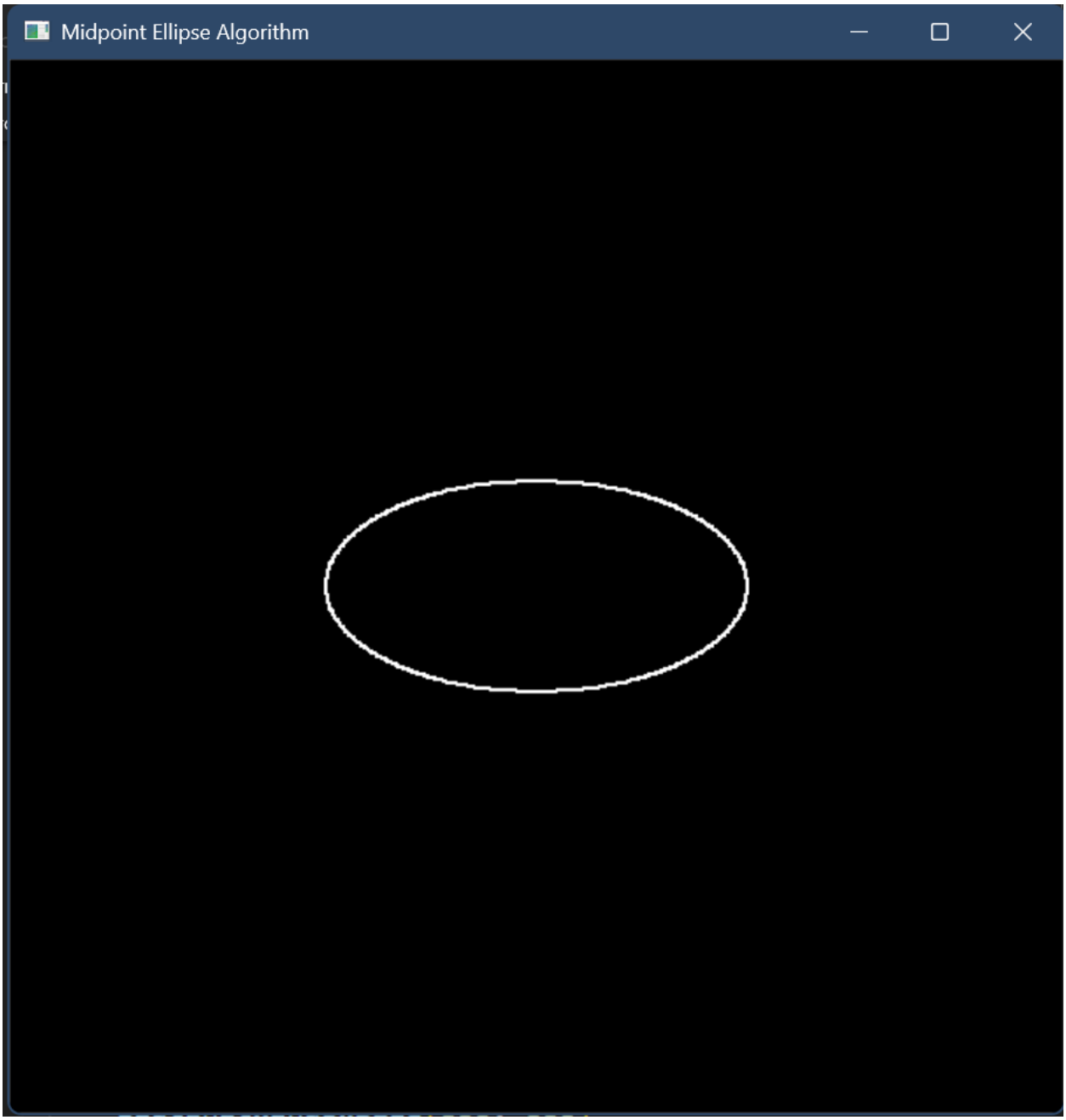
```python
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *

ellipse_points = []

xc = int(input("Enter center x (xc): "))
yc = int(input("Enter center y (yc): "))
rx = int(input("Enter x-radius (rx): "))
ry = int(input("Enter y-radius (ry): "))


def plot_symmetric_points(x, y):
    ellipse_points.extend([
        ( x + xc,  y + yc),
        (-x + xc,  y + yc),
        ( x + xc, -y + yc),
        (-x + xc, -y + yc)
    ])


def midpoint_ellipse():
    global ellipse_points
    ellipse_points = []

    rx2 = rx * rx
    ry2 = ry * ry

    x = 0
    y = ry

    p1 = ry2 - (rx2 * ry) + (0.25 * rx2)

    dx = 2 * ry2 * x
    dy = 2 * rx2 * y
```

```
while dx < dy:
    plot_symmetric_points(x, y)

    if p1 < 0:
        x += 1
        dx += 2 * ry2
        p1 += dx + ry2
    else:
        x += 1
        y -= 1
        dx += 2 * ry2
        dy -= 2 * rx2
        p1 += dx - dy + ry2

p2 = (ry2 * (x + 0.5) ** 2) + (rx2 * (y - 1) ** 2) - (rx2 * ry2)

while y >= 0:
    plot_symmetric_points(x, y)

    if p2 <= 0:
        x += 1
        y -= 1
        dx += 2 * ry2
        dy -= 2 * rx2
        p2 += dx - dy + rx2
    else:
        y -= 1
        dy -= 2 * rx2
        p2 += rx2 - dy
```

```
def display():
    glClear(GL_COLOR_BUFFER_BIT)
    glColor3f(1, 1, 1)
    glPointSize(2)

    glBegin(GL_POINTS)
    for x, y in ellipse_points:
        glVertex2i(int(x), int(y))
    glEnd()

    glFlush()


def init():
    glClearColor(0, 0, 0, 1)
    gluOrtho2D(-500, 500, -500, 500)


def main():
    glutInit()
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB)
    glutInitWindowSize(600, 600)
    glutInitWindowPosition(100, 100)
    glutCreateWindow(b"Midpoint Ellipse Algorithm")

    init()
    midpoint_ellipse()
    glutDisplayFunc(display)
    glutMainLoop()

main()
```

Output:

```
PS D:\codes\cg> py .\lab3\ellipse.py
Enter center x (xc): 0
Enter center y (yc): 0
Enter x-radius (rx): 200
Enter y-radius (ry): 100
```



Write a Program to implement:

Shapes.py

```python
import numpy as np
import math

def create_circle(cx, cy, r, segments=100):
    points = []
    for i in range(segments):
        theta = 2 * math.pi * i / segments
        x = cx + r * math.cos(theta)
        y = cy + r * math.sin(theta)
        points.append([x, y, 1])
    return np.array(points)

shapes = {
    "Triangle": np.array([
        [50, 50, 1],
        [100, 50, 1],
        [75, 100, 1]
    ]),

    "Rectangle": np.array([
        [150, 50, 1],
        [250, 50, 1],
        [250, 120, 1],
        [150, 120, 1]
    ]),

    "Line": np.array([
        [50, 200, 1],
        [200, 300, 1]
    ]),

    # "Circle": create_circle(cx=350, cy=200, r=50, segments=100)
}

def apply_transform(shape, matrix):
    return shape @ matrix.T
```

Utils.py

```python
from OpenGL.GL import *

def draw_shape(shape, color):
    glColor3f(*color)
    glBegin(GL_LINE_LOOP)
    for x, y, _ in shape:
        glVertex2f(x, y)
    glEnd()
```

2D Translation

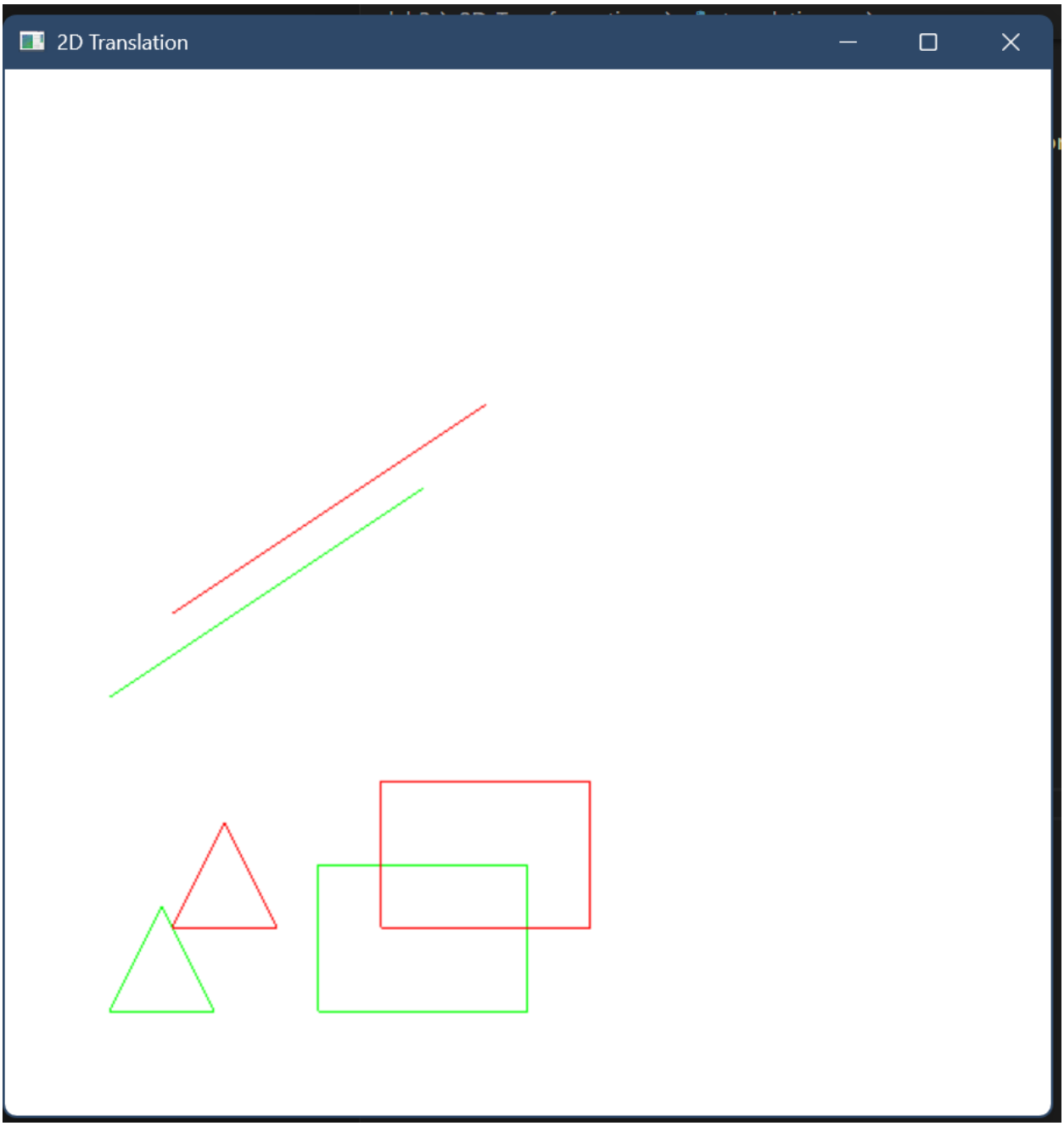
Code:

```python
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *
import numpy as np

from shapes import shapes, apply_transform
from utils import draw_shape

tx = float(input("Enter translation in X direction: "))
ty = float(input("Enter translation in Y direction: "))

T = np.array([
    [1, 0, tx],
    [0, 1, ty],
    [0, 0, 1]
])

translated_shapes = {}

print("\nTranslated Coordinates:")
for name, shape in shapes.items():
    translated_shapes[name] = apply_transform(shape, T)
    print(f"\n{name}:")
    print(translated_shapes[name])

def display():
    glClear(GL_COLOR_BUFFER_BIT)

    for name in shapes:
        draw_shape(shapes[name], (0, 1, 0))
        draw_shape(translated_shapes[name], (1, 0, 0))

    glFlush()

def init():
    glClearColor(1, 1, 1, 1)
    gluOrtho2D(0, 500, 0, 500)
```

```
glutInit()
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB)
glutInitWindowSize(600, 600)
glutCreateWindow(b"2D Translation")
init()
glutDisplayFunc(display)
glutMainLoop()
```

Output:

```
PS D:\codes\cg> py .\lab3\2D-Transformations\translation.py
Enter translation in X direction: 30
Enter translation in Y direction: 40

Translated Coordinates:

Triangle:
[[ 80.  90.   1.]
 [130.  90.   1.]
 [105. 140.   1.]]

Rectangle:
[[180.  90.   1.]
 [280.  90.   1.]
 [280. 160.   1.]
 [180. 160.   1.]]

Line:
[[ 80. 240.   1.]
 [230. 340.   1.]]
```



2D Rotation

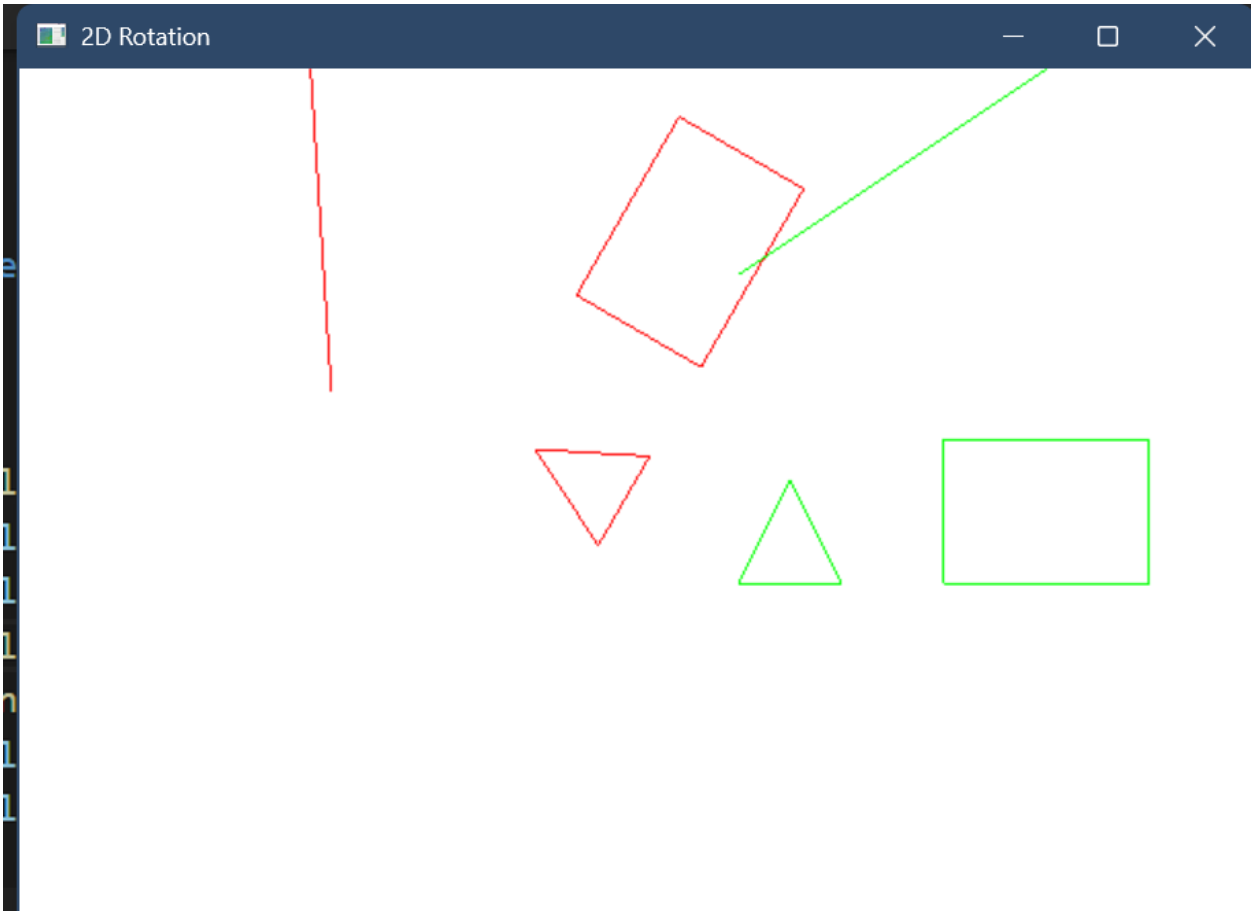
Code:

```python
import numpy as np
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *

from shapes import shapes, apply_transform
from utils import draw_shape

angle = float(input("Enter rotation angle (degrees): "))
rad = np.radians(angle)

R = np.array([
    [np.cos(rad), -np.sin(rad), 0],
    [np.sin(rad),  np.cos(rad), 0],
    [0, 0, 1]
])

rotated_shapes = {}

print("\nRotated Coordinates:")
for name, shape in shapes.items():
    rotated_shapes[name] = apply_transform(shape, R)
    print(f"\n{name}:")
    print(rotated_shapes[name])

def display():
    glClear(GL_COLOR_BUFFER_BIT)
```

```
    for name in shapes:
        draw_shape(shapes[name], (0, 1, 0))
        draw_shape(rotated_shapes[name], (1, 0, 0))

    glFlush()

def init():
    glClearColor(1, 1, 1, 1)
    gluOrtho2D(-300, 300, -300, 300)

glutInit()
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB)
glutInitWindowSize(600, 600)
glutCreateWindow(b"2D Rotation")
init()
glutDisplayFunc(display)
glutMainLoop()
```

Output:

```
PS D:\codes\cg> py .\lab3\2D-Transformations\rotation.py
Enter rotation angle (degrees): 60

Rotated Coordinates:

Triangle:
[[-18.30127019  68.30127019   1.        ]
 [  6.69872981 111.60254038   1.        ]
 [-49.10254038 114.95190528   1.        ]]

Rectangle:
[[ 31.69872981 154.90381057   1.        ]
 [ 81.69872981 241.50635095   1.        ]
 [ 21.07695155 276.50635095   1.        ]
 [-28.92304845 189.90381057   1.        ]]

Line:
[[-148.20508076  143.30127019    1.        ]
 [-159.80762114  323.20508076    1.        ]]
```



2D Scaling

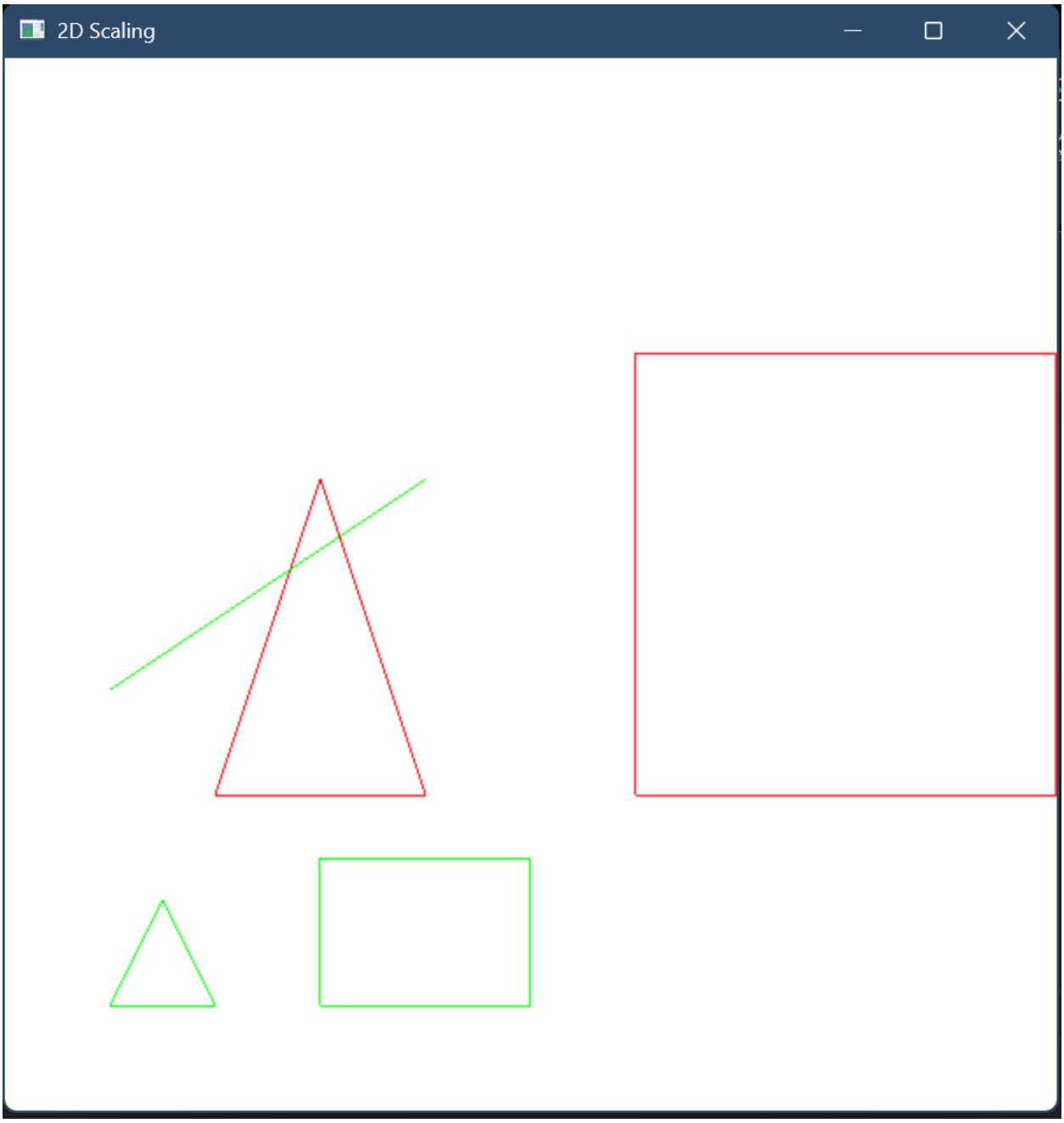
Code:

```python
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *
import numpy as np

from shapes import shapes, apply_transform
from utils import draw_shape

sx = float(input("Enter scaling factor in X direction: "))
sy = float(input("Enter scaling factor in Y direction: "))

S = np.array([
    [sx, 0, 0],
    [0, sy, 0],
    [0, 0, 1]
])

scaled_shapes = {}

print("\nScaled Coordinates (About Origin):")
for name, shape in shapes.items():
    scaled_shapes[name] = apply_transform(shape, S)
    print(f"\n{name}:")
    print(scaled_shapes[name])

def display():
    glClear(GL_COLOR_BUFFER_BIT)

    for name in shapes:
        draw_shape(shapes[name], (0, 1, 0))
        draw_shape(scaled_shapes[name], (1, 0, 0))
```

```
    glFlush()

def init():
    glClearColor(1, 1, 1, 1)
    gluOrtho2D(0, 500, 0, 500)

glutInit()
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB)
glutInitWindowSize(600, 600)
glutCreateWindow(b"2D Scaling")
init()
glutDisplayFunc(display)
glutMainLoop()
```

Output:

```
PS D:\codes\cg> py .\lab3\2D-Transformations\scaling.py
Enter scaling factor in X direction: 2
Enter scaling factor in Y direction: 3

Scaled Coordinates (About Origin):

Triangle:
[[100. 150.   1.]
 [200. 150.   1.]
 [150. 300.   1.]]

Rectangle:
[[300. 150.   1.]
 [500. 150.   1.]
 [500. 360.   1.]
 [300. 360.   1.]]

Line:
[[100. 600.   1.]
 [400. 900.   1.]]
```



2D Reflection

Code:

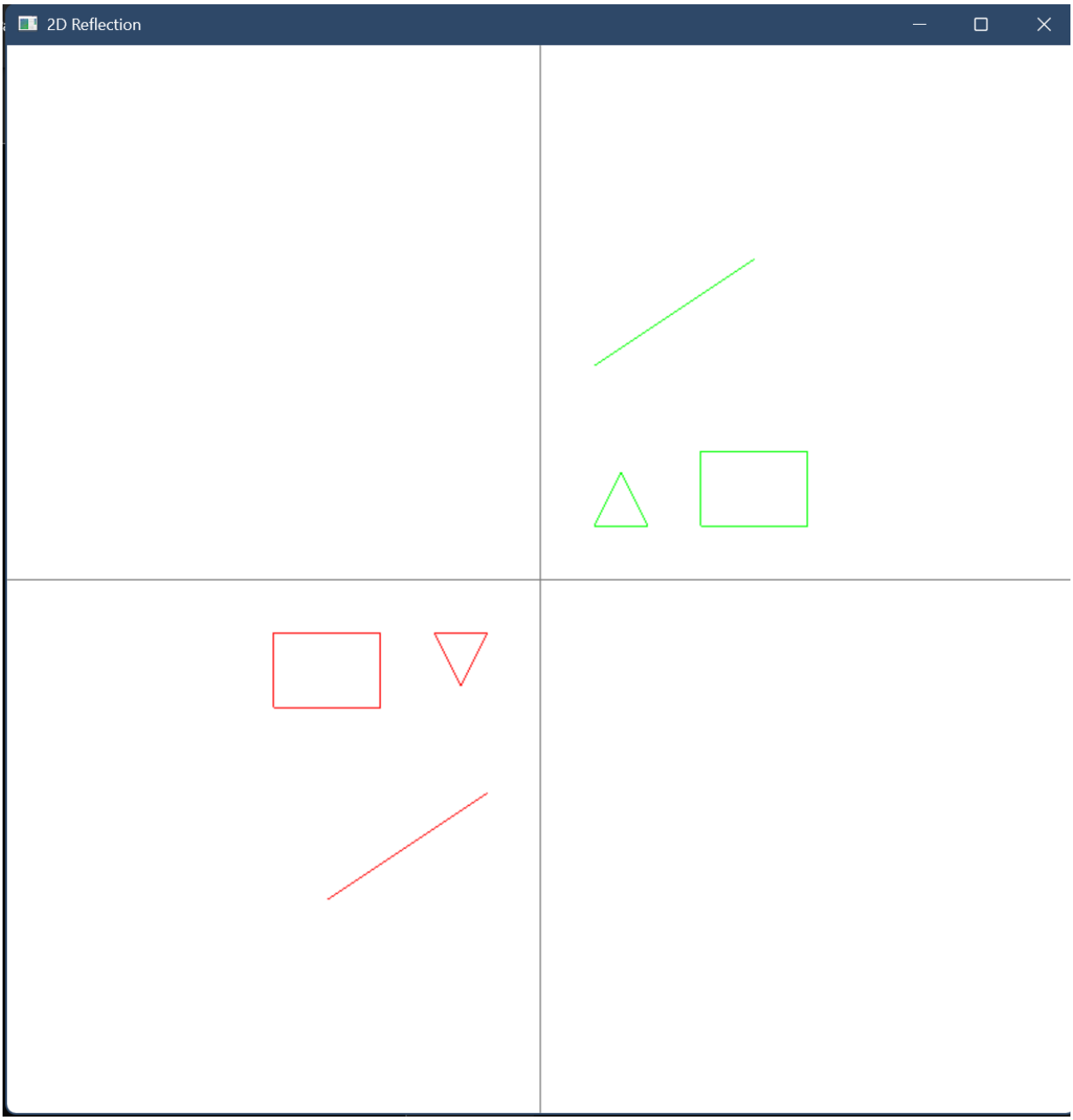
```python
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *
import numpy as np

from shapes import shapes, apply_transform
from utils import draw_shape

print("Choose axis of reflection:")
print("1: X-axis")
print("2: Y-axis")
print("3: Origin (both axes)")
choice = int(input("Enter your choice (1/2/3): "))

if choice == 1:
    R = np.array([
        [1, 0, 0],
        [0, -1, 0],
        [0, 0, 1]
    ])
elif choice == 2:
    R = np.array([
        [-1, 0, 0],
        [0, 1, 0],
        [0, 0, 1]
    ])
elif choice == 3:
    R = np.array([
        [-1, 0, 0],
        [0, -1, 0],
        [0, 0, 1]
    ])
```

```
else:
    print("Invalid choice! Defaulting to X-axis.")
    R = np.array([
        [1, 0, 0],
        [0, -1, 0],
        [0, 0, 1]
    ])

reflected_shapes = {}

print("\nReflected Coordinates:")
for name, shape in shapes.items():
    reflected_shapes[name] = apply_transform(shape, R)
    print(f"\n{name}:")
    print(reflected_shapes[name])

def draw_axes():
    """Draw X and Y axes."""
    glColor3f(0.5, 0.5, 0.5)
    glLineWidth(1)
    glBegin(GL_LINES)
    glVertex2f(-500, 0)
    glVertex2f(500, 0)
    glVertex2f(0, -500)
    glVertex2f(0, 500)
    glEnd()

def display():
    glClear(GL_COLOR_BUFFER_BIT)

    draw_axes()

    for name in shapes:
        draw_shape(shapes[name], (0, 1, 0))
        draw_shape(reflected_shapes[name], (1, 0, 0))

    glFlush()
```

```
def init():
    glClearColor(1, 1, 1, 1)
    gluOrtho2D(-500, 500, -500, 500)

glutInit()
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB)
glutInitWindowSize(800, 800)
glutCreateWindow(b"2D Reflection")
init()
glutDisplayFunc(display)
glutMainLoop()
```

Output:

```
Choose axis of reflection:
1: X-axis
2: Y-axis
3: Origin (both axes)
Enter your choice (1/2/3): 3

Reflected Coordinates:

Triangle:
[[ -50  -50    1]
 [-100  -50    1]
 [ -75 -100    1]]

Rectangle:
[[-150  -50    1]
 [-250  -50    1]
 [-250 -120    1]
 [-150 -120    1]]

Line:
[[ -50 -200    1]
 [-200 -300    1]]
```



2D Shearing

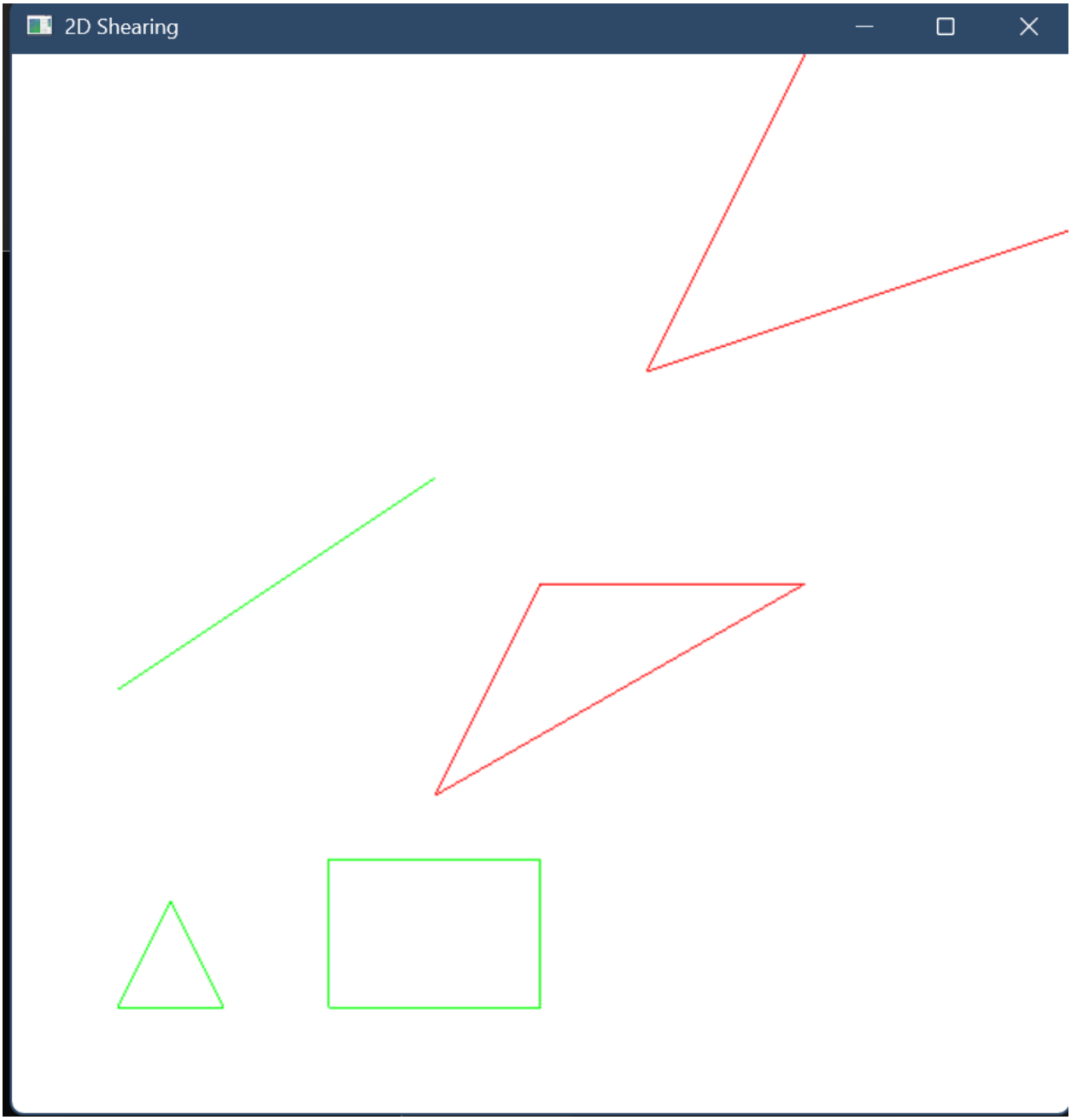
Code:

```python
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *
import numpy as np

from shapes import shapes, apply_transform
from utils import draw_shape

shx = float(input("Enter shear factor in X direction: "))
shy = float(input("Enter shear factor in Y direction: "))

Sh = np.array([
    [1, shx, 0],
    [shy, 1, 0],
    [0, 0, 1]
])

sheared_shapes = {}

print("\nSheared Coordinates (About Origin):")
for name, shape in shapes.items():
    sheared_shapes[name] = apply_transform(shape, Sh)
    print(f"\n{name}:")
    print(sheared_shapes[name])

def display():
    glClear(GL_COLOR_BUFFER_BIT)

    for name in shapes:
        draw_shape(shapes[name], (0, 1, 0))
        draw_shape(sheared_shapes[name], (1, 0, 0))

    glFlush()
```

```
def init():
    glClearColor(1, 1, 1, 1)
    gluOrtho2D(0, 500, 0, 500)

glutInit()
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB)
glutInitWindowSize(600, 600)
glutCreateWindow(b"2D Shearing")
init()
glutDisplayFunc(display)
glutMainLoop()
```

Output:

```
PS D:\codes\cg> py .\lab3\2D-Transformations\shearing.py
Enter shear factor in X direction: 3
Enter shear factor in Y direction: 2

Sheared Coordinates (About Origin):

Triangle:
[[200. 150.   1.]
 [250. 250.   1.]
 [375. 250.   1.]]

Rectangle:
[[300. 350.   1.]
 [400. 550.   1.]
 [610. 620.   1.]
 [510. 420.   1.]]

Line:
[[6.5e+02 3.0e+02 1.0e+00]
 [1.1e+03 7.0e+02 1.0e+00]]
```



Composite Transformations in a 2D shape

Source Code:

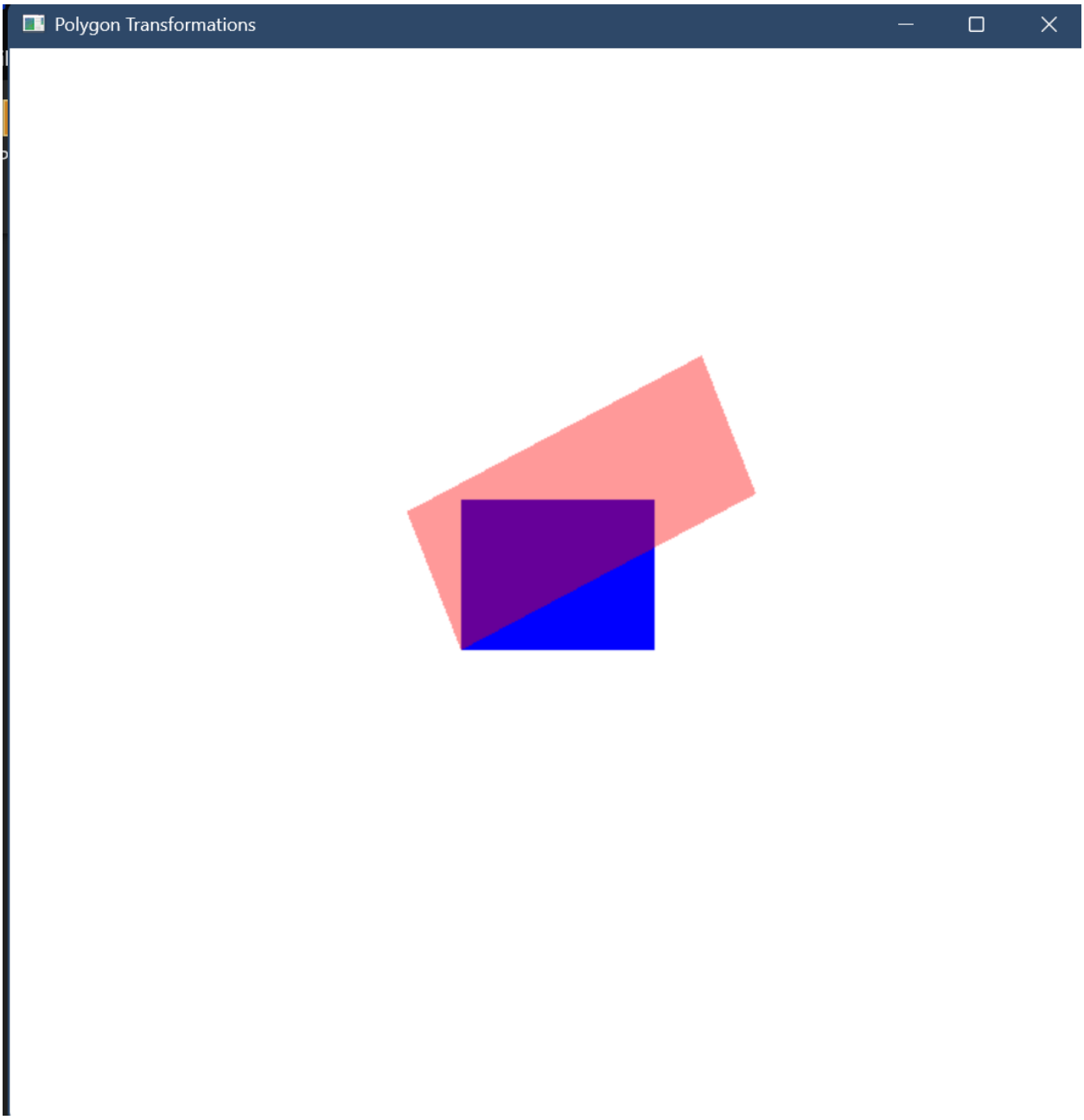
```python
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *
import math
import sys

original_shape = [
    (10, 20),
    (10, 90),
    (100, 90),
    (100, 20)
]

final_shape = []

def mat_mult(A, B):
    return [[sum(A[i][k] * B[k][j] for k in range(3))
             for j in range(3)] for i in range(3)]

def apply_matrix(M, pts):
    res = []
    for x, y in pts:
        nx = M[0][0]*x + M[0][1]*y + M[0][2]
        ny = M[1][0]*x + M[1][1]*y + M[1][2]
        res.append((round(nx, 2), round(ny, 2)))
    return res

def translate(tx, ty):
    return [[1,0,tx],[0,1,ty],[0,0,1]]
```

```python
def rotate(theta):
    r = math.radians(theta)
    return [
        [math.cos(r), -math.sin(r), 0],
        [math.sin(r),  math.cos(r), 0],
        [0, 0, 1]
    ]

def shear(shx, shy):
    return [[1, shx, 0],[shy, 1, 0],[0,0,1]]

def scale(sx, sy):
    return [[sx,0,0],[0,sy,0],[0,0,1]]

def closest_to_origin(pts):
    return min(pts, key=lambda p: p[0]**2 + p[1]**2)

def perform_transformations():
    global final_shape

    theta = 30
    shx, shy = 0.3, 0.2
    sx, sy = 1.5, 1.2

    xc, yc = closest_to_origin(original_shape)
    print("Closest point to origin:", (xc, yc))


    #translation
    M = translate(-xc, -yc)
    print("After translation to origin:", apply_matrix(M, original_shape))

    #rotation
    M = mat_mult(rotate(theta), M)
    print("After rotation:", apply_matrix(M, original_shape))

    #shearing
    M = mat_mult(shear(shx, shy), M)
    print("After shearing:", apply_matrix(M, original_shape))

    #scaling
    M = mat_mult(scale(sx, sy), M)
    print("After scaling:", apply_matrix(M, original_shape))
```

```
    #translation
    M = mat_mult(translate(xc, yc), M)
    final_shape = apply_matrix(M, original_shape)

    print("\nFinal transformed coordinates:")
    for p in final_shape:
        print(p)

def draw_shape(pts, color, alpha=1.0):
    glColor4f(color[0], color[1], color[2], alpha)
    glBegin(GL_POLYGON)
    for x, y in pts:
        glVertex2f(x, y)
    glEnd()

def display():
    glClear(GL_COLOR_BUFFER_BIT)

    draw_shape(original_shape, (0, 0, 1), 1.0)
    draw_shape(final_shape, (1, 0, 0), 0.4)

    glFlush()

def init():
    glClearColor(1, 1, 1, 1)

    glEnable(GL_BLEND)
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)

    gluOrtho2D(-200, 300, -200, 300)

def main():
    perform_transformations()

    glutInit(sys.argv)
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA)
    glutInitWindowSize(700, 700)
    glutCreateWindow(b"Polygon Transformations")
    init()
    glutDisplayFunc(display)
    glutMainLoop()

main()
```

Output:

```
PS D:\codes\cg> py .\lab3\composite.py
 Closest point to origin: (10, 20)
 After translation to origin: [(0, 0), (0, 70), (90, 70), (90, 0)]
 After rotation: [(0.0, 0.0), (-35.0, 60.62), (42.94, 105.62), (77.94, 45.0)]
 After shearing: [(0.0, 0.0), (-16.81, 53.62), (74.63, 114.21), (91.44, 60.59)]
 After scaling: [(-0.0, 0.0), (-25.22, 64.35), (111.94, 137.05), (137.16, 72.71)]

 Final transformed coordinates:
 (10.0, 20.0)
 (-15.22, 84.35)
 (121.94, 157.05)
 (147.16, 92.71)
```


Polygon Transformations