

Kathmandu University
Department of Computer Science and
Engineering
Dhulikhel, Kavrepalanchowk



A Lab Report
On
"COMP 342"

Submitted by: Aaditya K.C
Roll No:05
Level:UNG CS(III/I)

Submitted to: Dhiraj Shrestha
Department of Computer Science and Engineering
Submission Date: 01/04/2026

1. Implement midpoint Ellipse drawing Algorithm

MIDPOINT ELLIPSE DRAWING ALGORITHM

1. Input the x radius r_x , y radius r_y , and the center of the ellipse (x_c, y_c) .
2. Initialize the starting point of the ellipse as
 $x = 0$ and $y = r_y$.
3. Calculate the squared values $rx2 = r_x * r_x$ and $ry2 = r_y * r_y$.
4. Initialize the decision parameter for Region 1 as
 $p1 = ry2 - (rx2 * ry) + (rx2 / 4)$.
5. **REGION 1**
Repeat the following steps while
 $2 * ry2 * x < 2 * rx2 * y$
 - a. Plot the four symmetric points
 $(x_c + x, y_c + y)$, $(x_c - x, y_c + y)$,
 $(x_c + x, y_c - y)$, $(x_c - x, y_c - y)$
 - b. If $p1$ is less than 0
 $x = x + 1$
 $p1 = p1 + (2 * ry2 * x) + ry2$
 - c. Else
 $x = x + 1$
 $y = y - 1$
 $p1 = p1 + (2 * ry2 * x) - (2 * rx2 * y) + ry2$
6. Initialize the decision parameter for Region 2 as
 $p2 = (ry2 * (x + 0.5) * (x + 0.5))$
 $+ (rx2 * (y - 1) * (y - 1))$
 $- (rx2 * ry2)$
7. **REGION 2**
Repeat the following steps while y is greater than or equal to 0
 - a. Plot the four symmetric points
 $(x_c + x, y_c + y)$, $(x_c - x, y_c + y)$,
 $(x_c + x, y_c - y)$, $(x_c - x, y_c - y)$

b. If p_2 is greater than 0

$y = y - 1$

$p_2 = p_2 - (2 * rx_2 * y) + rx_2^2$

c. Else

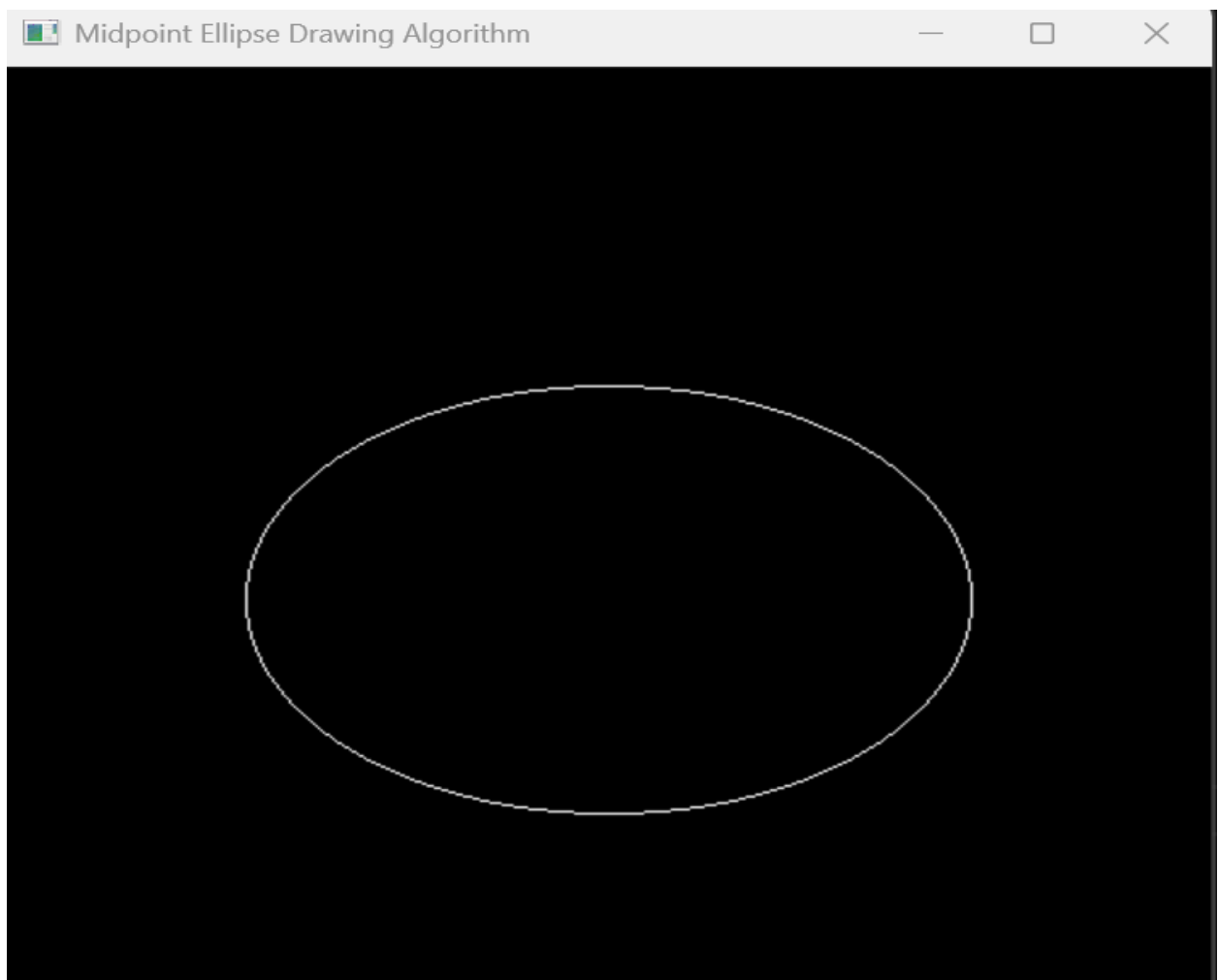
$x = x + 1$

$y = y - 1$

$p_2 = p_2 + (2 * ry_2 * x) - (2 * rx_2 * y) + rx_2^2$

8. **Stop the algorithm when y becomes less than 0.**

The ellipse is completely drawn.



ellipse.py X

lab3 > ellipse.py > ...

```
1  from OpenGL.GL import *
2  from OpenGL.GLUT import *
3  from OpenGL.GLU import *
4
5  def plot_ellipse_points(xc, yc, x, y):
6      glVertex2f(xc + x, yc + y)
7      glVertex2f(xc - x, yc + y)
8      glVertex2f(xc + x, yc - y)
9      glVertex2f(xc - x, yc - y)
10
11 def midpoint_ellipse(xc, yc, rx, ry):
12     x = 0
13     y = ry
14
15     rx2 = rx * rx
16     ry2 = ry * ry
17
18     dx = 2 * ry2 * x
19     dy = 2 * rx2 * y
20
21     # Region 1
22     p1 = ry2 - (rx2 * ry) + (0.25 * rx2)
23
24     glBegin(GL_POINTS)
25     while dx < dy:
26         plot_ellipse_points(xc, yc, x, y)
27         if p1 < 0:
28             x += 1
29             dx = dx + 2 * ry2
30             p1 = p1 + dx + ry2
31         else:
32             x += 1
33             y -= 1
34             dx = dx + 2 * ry2
35             dy = dy - 2 * rx2
```

ellipse.py X

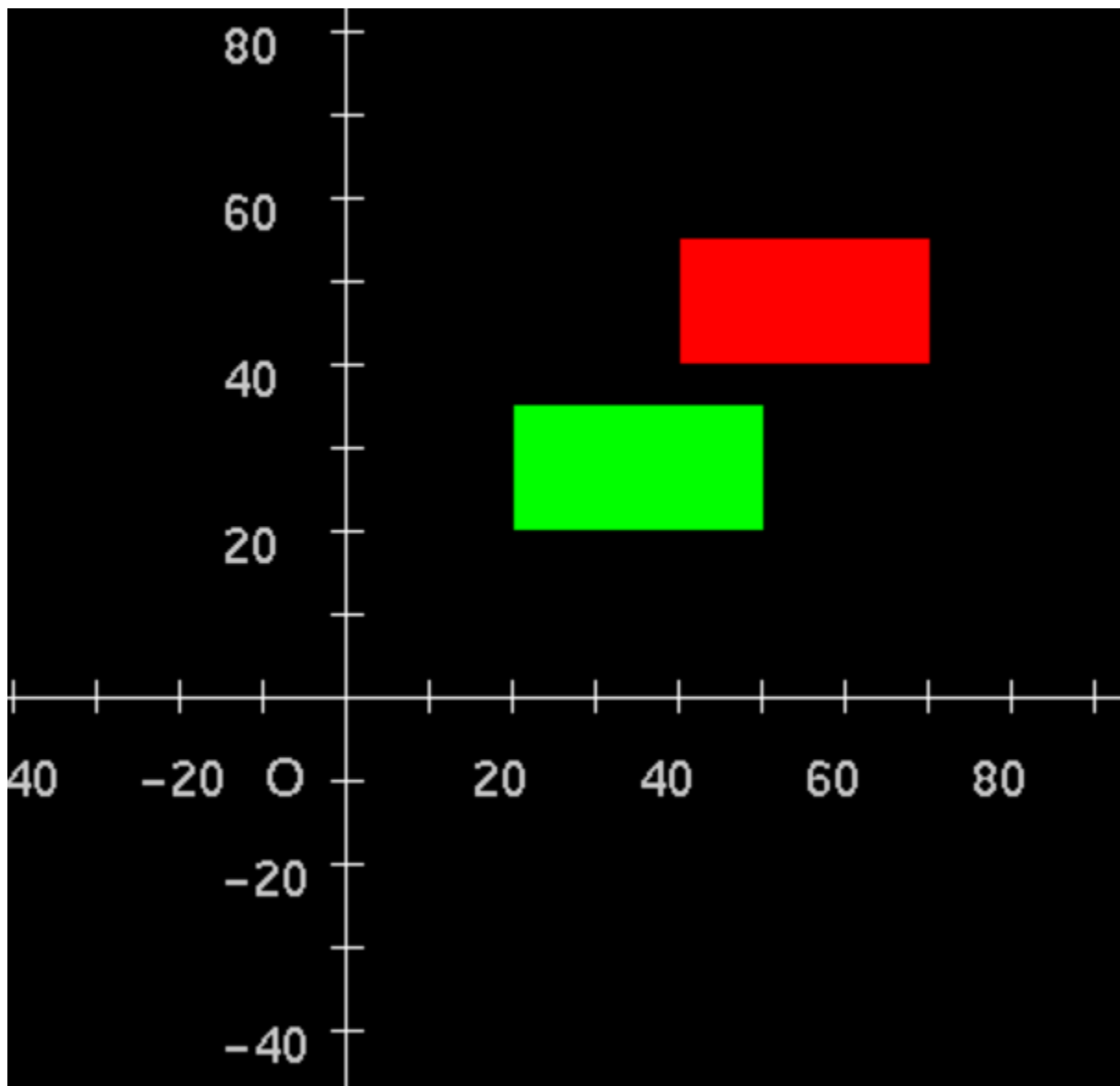
lab3 > ellipse.py > plot_ellipse_points

```
11 def midpoint_ellipse(xc, yc, rx, ry):
33     y -= 1
34     dx = dx + 2 * ry2
35     dy = dy - 2 * rx2
36     p1 = p1 + dx - dy + ry2
37
38     # Region 2
39     p2 = (ry2 * (x + 0.5) * (x + 0.5)) + \
40         (rx2 * (y - 1) * (y - 1)) - (rx2 * ry2)
41
42     while y >= 0:
43         plot_ellipse_points(xc, yc, x, y)
44         if p2 > 0:
45             y -= 1
46             dy = dy - 2 * rx2
47             p2 = p2 + rx2 - dy
48         else:
49             x += 1
50             y -= 1
51             dx = dx + 2 * ry2
52             dy = dy - 2 * rx2
53             p2 = p2 + dx - dy + rx2
54     glEnd()
55
56 def display():
57     glClear(GL_COLOR_BUFFER_BIT)
58     glColor3f(1, 1, 1)
59     midpoint_ellipse(250, 250, 150, 100)
60     glFlush()
61
62     glutInit()
63     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB)
64     glutInitWindowSize(500, 500)
65     glutCreateWindow(b"Midpoint Ellipse Drawing Algorithm")
66     glClearColor(0, 0, 0, 0)
67     gluOrtho2D(0, 500, 0, 500)
68     glutDisplayFunc(display)
69     glutMainLoop()
```

main* 140t (X) 0/0

2. Write a Program to implement:
a. 2D Translation

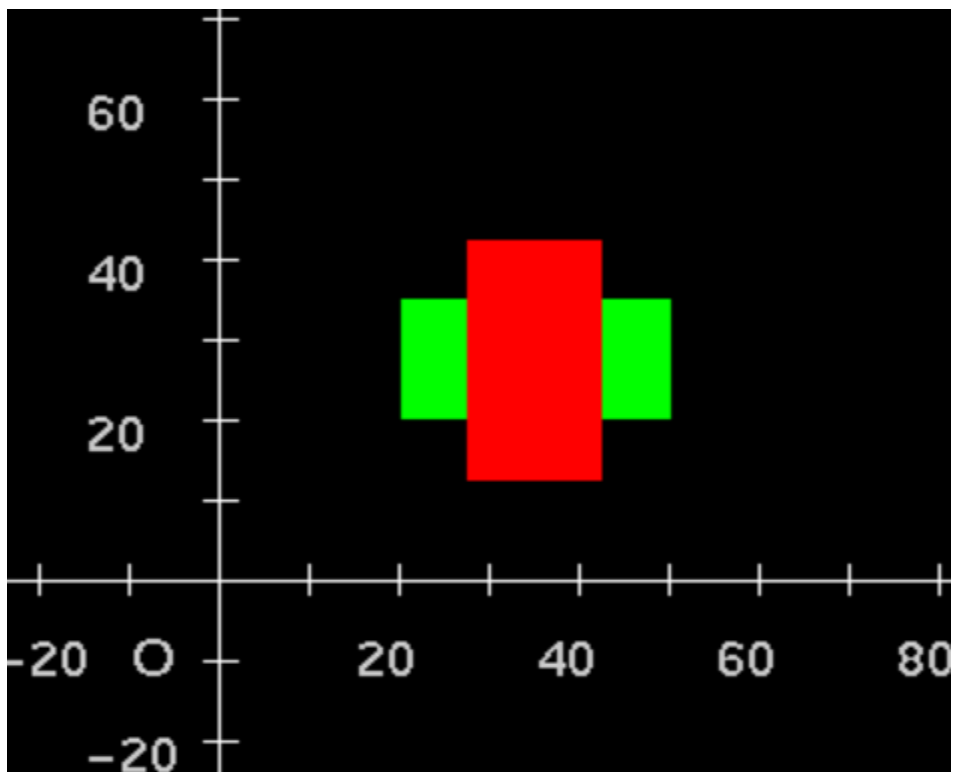
```
# Transformation Matrix
if choice == 1:  # Translation
    tx, ty = params["tx"], params["ty"]
    T = [[1, 0, tx], [0, 1, ty], [0, 0, 1]]
```



b. 2D Rotation

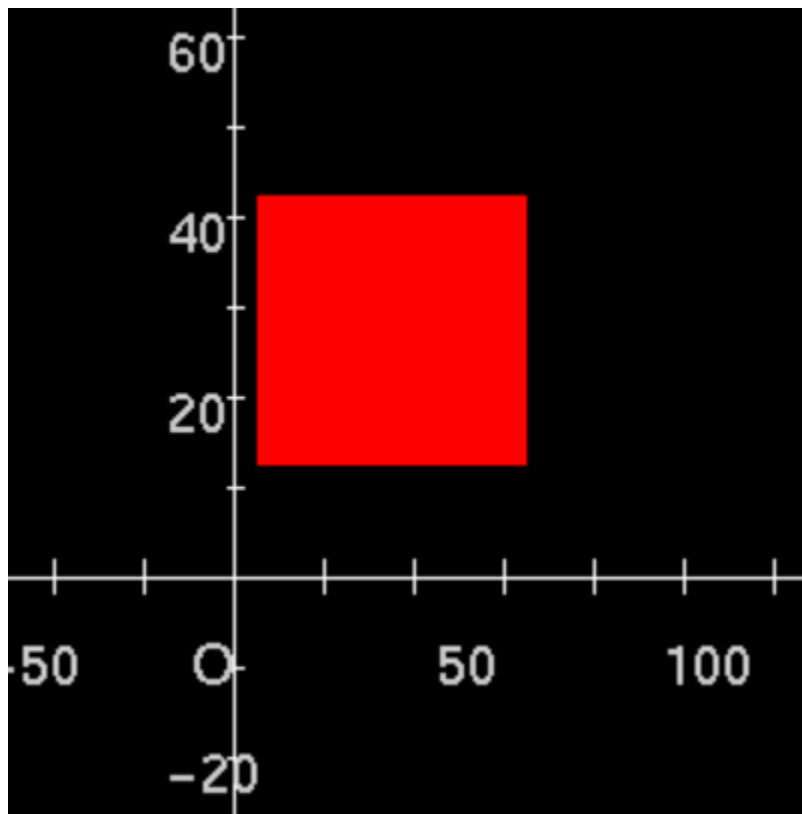
```
elif choice == 2: # Rotation (around object center)
    a = math.radians(params["angle"])
    cx, cy = rect_center
    # Translate to origin, rotate, translate back
    T1 = [[1, 0, -cx], [0, 1, -cy], [0, 0, 1]]
    T2 = [[math.cos(a), -math.sin(a), 0],
          [math.sin(a),  math.cos(a), 0],
          [0, 0, 1]]
    T3 = [[1, 0, cx], [0, 1, cy], [0, 0, 1]]

    temp = multiply_matrix(T1, rect)
    temp = multiply_matrix(T2, temp)
    transformed = multiply_matrix(T3, temp)
    draw_shape(transformed, (1, 0, 0))
    glFlush()
    return
```



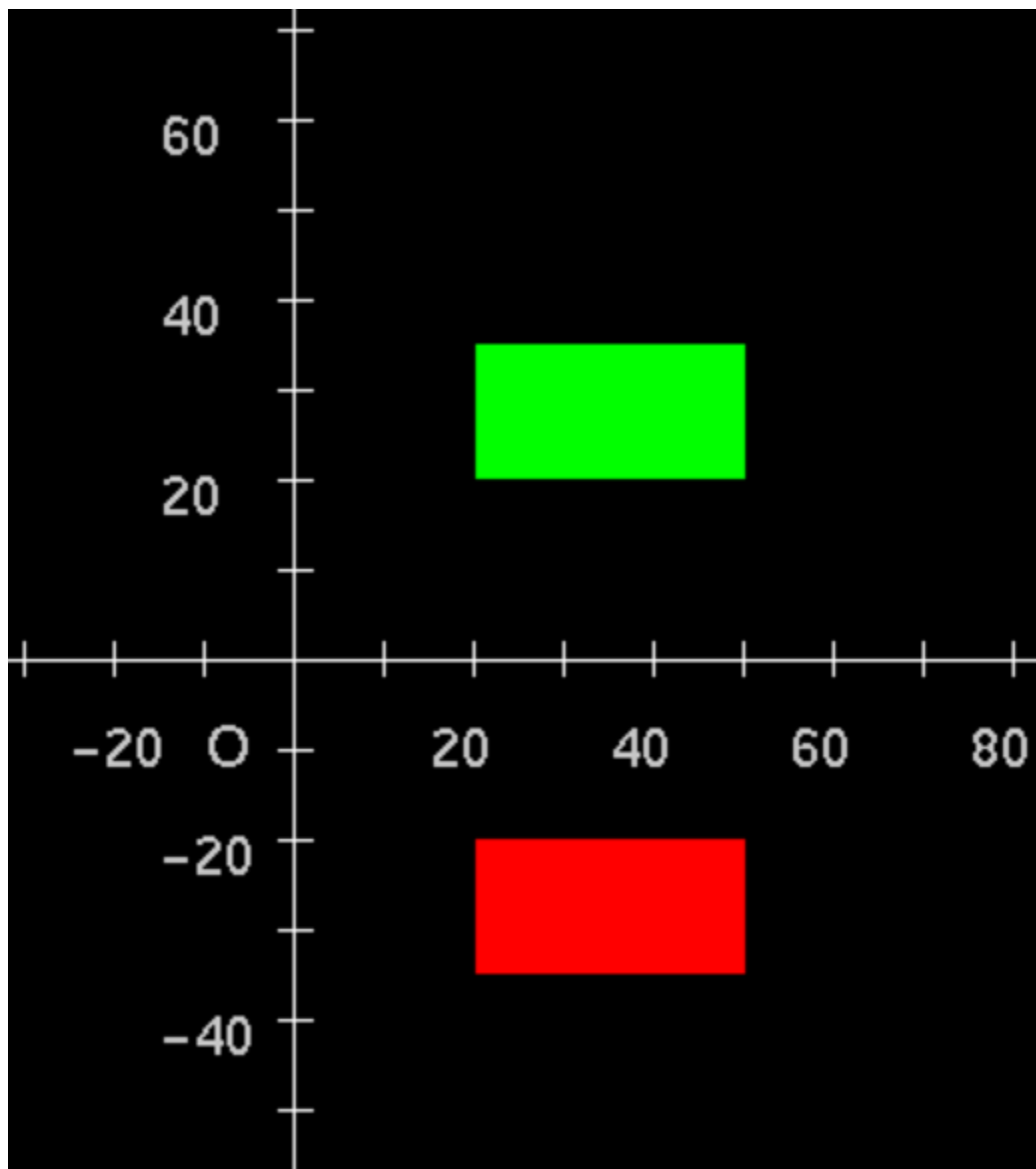
c. 2D Scaling

```
elif choice == 3: # Scaling
    sx, sy = params["sx"], params["sy"]
    if sx == 0 or sy == 0:
        print("Warning: Scaling factor cannot be zero. Using 1.0")
        sx = sx if sx != 0 else 1.0
        sy = sy if sy != 0 else 1.0
    T = [[sx, 0, 0], [0, sy, 0], [0, 0, 1]]
```



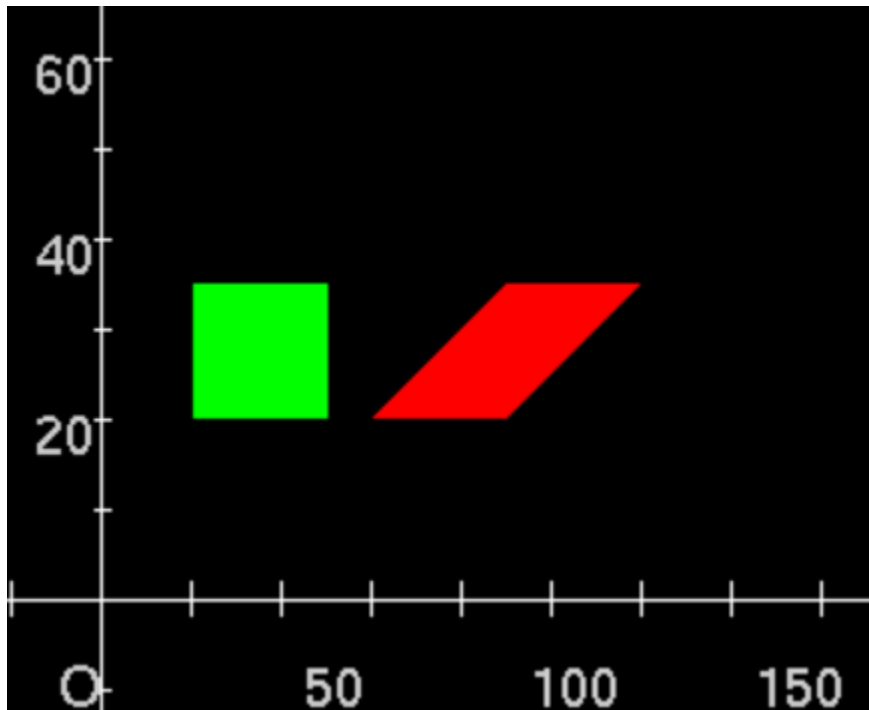
d. 2D Reflection

```
elif choice == 4: # Reflection (X-axis or Y-axis)
    axis = input("Reflect on X-axis (x) or Y-axis (y)? ").lower()
    if axis == 'x':
        T = [[1, 0, 0], [0, -1, 0], [0, 0, 1]]
    elif axis == 'y':
        T = [[-1, 0, 0], [0, 1, 0], [0, 0, 1]]
    else:
        print("Invalid choice. Using X-axis reflection.")
        T = [[1, 0, 0], [0, -1, 0], [0, 0, 1]]
```



e. 2D Shearing

```
elif choice == 5: # Shearing
    shx = params["shx"]
    T = [[1, shx, 0], [0, 1, 0], [0, 0, 1]]
```



f. Composite Transformations (Atleast 4 Transformations)

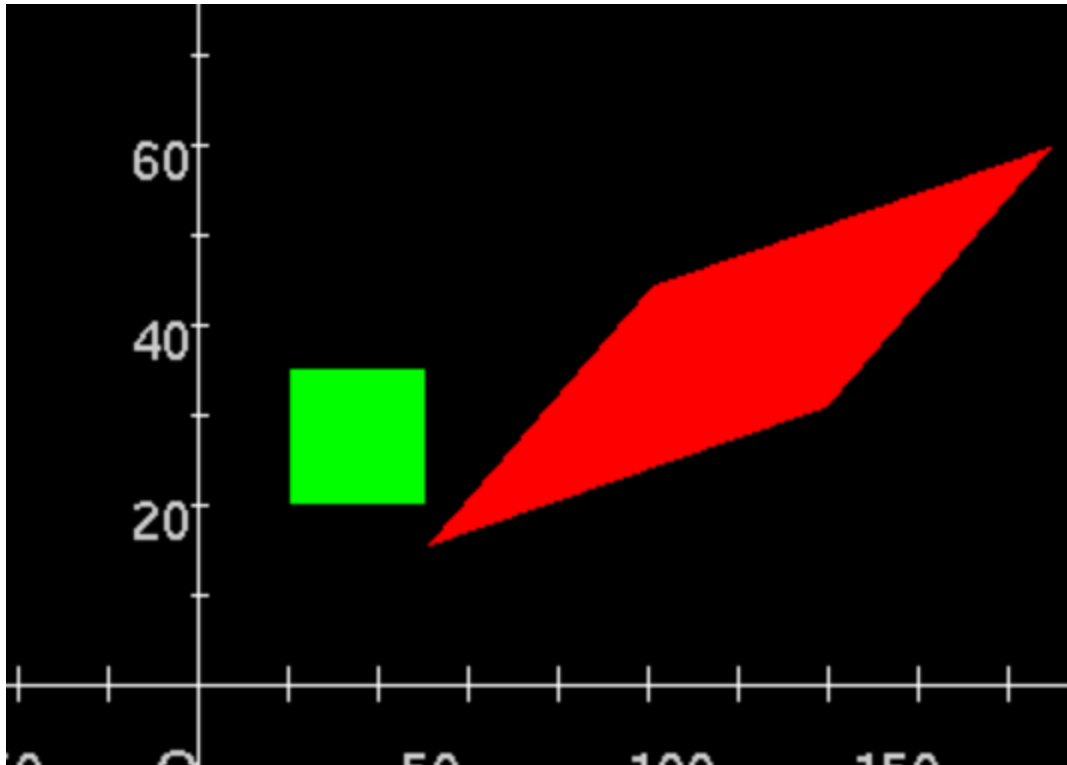
```
elif choice == 6: # Composite (Scale -> Rotate -> Translate -> Shear)
    tx, ty = params["tx"], params["ty"]
    sx, sy = params["sx"], params["sy"]
    shx = params["shx"]
    a = math.radians(params["angle"])

    if sx == 0 or sy == 0:
        print("Warning: Scaling factor cannot be zero. Using 1.0")
        sx = sx if sx != 0 else 1.0
        sy = sy if sy != 0 else 1.0

    # Scale first
    T_scale = [[sx, 0, 0], [0, sy, 0], [0, 0, 1]]
    # Then rotate around center
    cx, cy = rect_center
    T_trans_to_origin = [[1, 0, -cx], [0, 1, -cy], [0, 0, 1]]
    T_rotate = [[math.cos(a), -math.sin(a), 0],
                [math.sin(a),  math.cos(a), 0],
                [0, 0, 1]]
    T_trans_back = [[1, 0, cx], [0, 1, cy], [0, 0, 1]]
    # Then translate
    T_translate = [[1, 0, tx], [0, 1, ty], [0, 0, 1]]
    # Then shear
    T_shear = [[1, shx, 0], [0, 1, 0], [0, 0, 1]]

    temp = multiply_matrix(T_scale, rect)
    temp = multiply_matrix(T_trans_to_origin, temp)
    temp = multiply_matrix(T_rotate, temp)
    temp = multiply_matrix(T_trans_back, temp)
    temp = multiply_matrix(T_translate, temp)
    transformed = multiply_matrix(T_shear, temp)

    draw_shape(transformed, (1, 0, 0))
    glFlush()
    return
```



1. Translation Matrix

Translation by (t_x, t_y) :

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

2. Rotation Matrix

Rotation by angle θ (about origin):

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3. Scaling Matrix

Scaling by factors (sx, sy):

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

4. Reflection Matrix

Reflection about X-axis:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Reflection about Y-axis:

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Reflection about Origin:

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

5. Shearing Matrix

X-direction Shearing:

$$\begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Y-direction Shearing:

$$\begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

6. Composite Transformation Matrix

If we apply:

Translation \rightarrow Rotation \rightarrow Scaling \rightarrow Shearing

Then the composite matrix is:

$$[T] \times [R] \times [S] \times [Sh]$$

Final point is computed as:

$$P' = \text{Composite Matrix} \times P$$