Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Lab Report 4: 2D & 3D Transformations

COMP 342

(For partial fulfillment of 3rd Year/ 2nd Semester in Computer Engineering)

Submitted to:

Mr. Dhiraj Shrestha

Department of Computer Science and Engineering

Submitted by:

Neha Malla

C.E.

Roll No. 27

1. **Title:** Implementing 2D Transformations: Translation, Rotation, Scaling, Reflection and Shearing of any 2D shape, using homogeneous coordinate system.

Formulae:

Translate(tx, ty):

$$\text{Translation Matrix} = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix}$$

Rotate(angle):

$$\text{Rotation Matrix} = \begin{bmatrix} \cos(angle) & -\sin(angle) & 0 \\ \sin(angle) & \cos(angle) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Scale(sx, sy):

$$\text{Scaling Matrix} = \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Reflect(y = x):

$$\text{Reflection Matrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

ShearX(shx):

$$\text{XShearing Matrix} = \begin{bmatrix} 1 & shx & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

ShearY(shy):

$$\text{YShearing Matrix} = \begin{bmatrix} 1 & 0 & 0 \\ shy & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

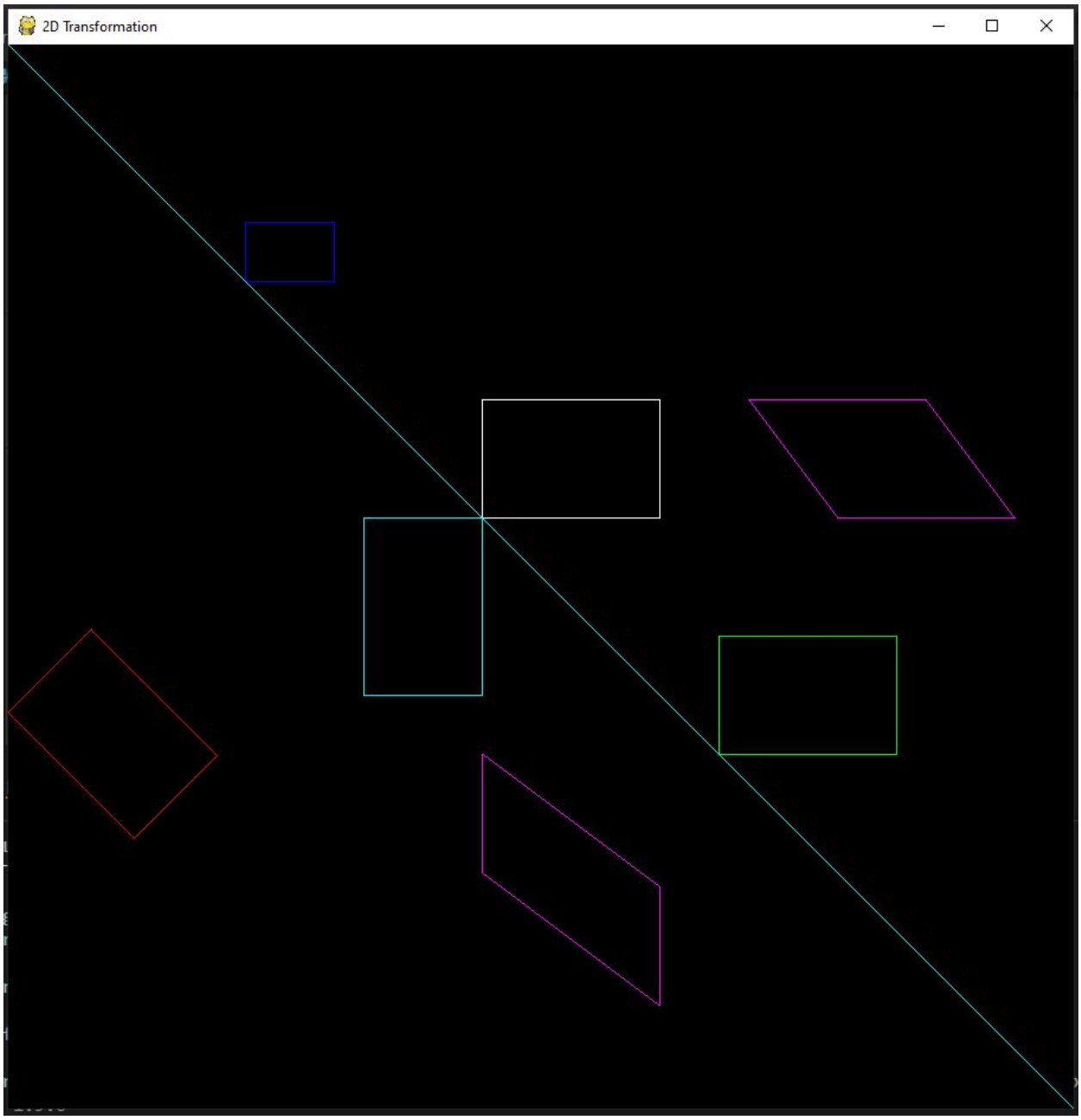
Source code:

```
import pygame
import pygame.gfxdraw
import numpy as np
from math import sin, cos, pi


# Initializing the game engine
pygame.init()

# Colors in RGB format
white = (255, 255, 255)
black = (0, 0, 0)
red = (255, 0, 0)
green = (0, 255, 0)
blue = (0, 0, 255)
cyan = (0, 255, 255)
magenta = (255, 0, 255)

# height and width of the screen
width = 900
height = 900
screen = pygame.display.set_mode([width, height], pygame.RESIZABLE)
pygame.display.set_caption("2D Transformation")

# A 2D shape to be transformed
rectangle = np.array([[400, 550, 550, 400], [300, 300, 400, 400], [1, 1,
1, 1]])


# Function to draw the shape from homogeneous coordinates
def Draw(homogeneousCoordinates, colour):
    coordinates = homogeneousCoordinates.transpose()
    pygame.gfxdraw.polygon(screen, coordinates[0:4, 0:2], colour)


# Function to translate the points
def Translate(tx, ty):
    translationMatrix = np.array([[1, 0, tx], [0, 1, ty], [0, 0, 1]])
```

```
    Draw(np.dot(translationMatrix, rectangle), green)


# Function to rotate the points
def Rotate(angle):
     rotationMatrix = np.array([[cos(angle), -sin(angle), 0], [sin(angle),
cos(angle), 0], [0, 0, 1]])
    Draw(np.dot(rotationMatrix, rectangle), red)


# Function to scale the points
def Scale(sx, sy):
    scalingMatrix = np.array([[sx, 0, 0], [0, sy, 0], [0, 0, 1]])
    Draw(np.dot(scalingMatrix, rectangle), blue)


# Function to reflect the points
def Reflect():
    reflectionMatrix = np.array([[0, 1, 0], [1, 0, 0], [0, 0, 1]])
    pygame.gfxdraw.line(screen, 0, 0, 900, 900, cyan)
    Draw(np.dot(reflectionMatrix, rectangle), cyan)


# Function to shear the points in x axis
def ShearX(shx):
    XshearingMatrix = np.array([[1, shx, 0], [0, 1, 0], [0, 0, 1]])
    Draw(np.dot(XshearingMatrix, rectangle), magenta)


# Function to shear the points in y axis
def ShearY(shy):
    YshearingMatrix = np.array([[1, 0, 0], [shy, 1, 0], [0, 0, 1]])
    Draw(np.dot(YshearingMatrix, rectangle), magenta)


done = False

while not done:
    # If user clicks close
    for event in pygame.event.get():
```

```
        if event.type == pygame.QUIT:
            done = True

    # The screen background as black
    screen.fill(black)

    # Triangle at original position (coloured white)
    Draw(rectangle, white)

    # Translate by 200, 200 (coloured green)
    Translate(200, 200)

    # Rotate by 45 degree (coloured red)
    Rotate(pi/4)

    # Scale to half (coloured blue)
    Scale(0.5, 0.5)

    # Reflect on y = x line (coloured cyan)
    Reflect()

    # Shear on both axis (coloured magenta)
    ShearX(0.75)
    ShearY(0.75)

    # Update the contents of the entire display
    pygame.display.flip()

pygame.quit()
```

Output:



Conclusion:

The rectangle in white is the initial 2D shape drawn. The green one is translated by (200, 200), the red one is rotated by 45 degree, the blue one is scaled to half, the magenta ones are sheared in x axis and y axis, and finally the cyan one is reflected on y = x line (which is drawn as reference for the eyes).

2. **Title:** Implementing Cohen Sutherland Line Clipping Algorithm.

Algorithm:

1. Assign Region codes for each endpoints
2. If both endpoints have Region code 0000, then trivially accept the line
3. Else, perform logical AND operation for both region codes:
    3.1. If the result is NOT 0000, then trivially reject the line
    3.2. Else:
        3.2.1. Choose endpoints lying outside the clipping window
        3.2.2. Find intersection points at widow boundary
        3.2.3. Replace endpoints with intersection point and update region code
        3.2.4. Repeat step 2 until we find a clipped line either trivially accepted or rejected.
4. Repeat step 1 for all other lines

Source code:

```
import pygame
import pygame.gfxdraw
from math import floor


# Initializing the game engine
pygame.init()

# Colors in RGB format
white = (255, 255, 255)
black = (0, 0, 0)
red = (255, 0, 0)
green = (0, 255, 0)
cyan = (0, 255, 255)
magenta = (255, 0, 255)
```

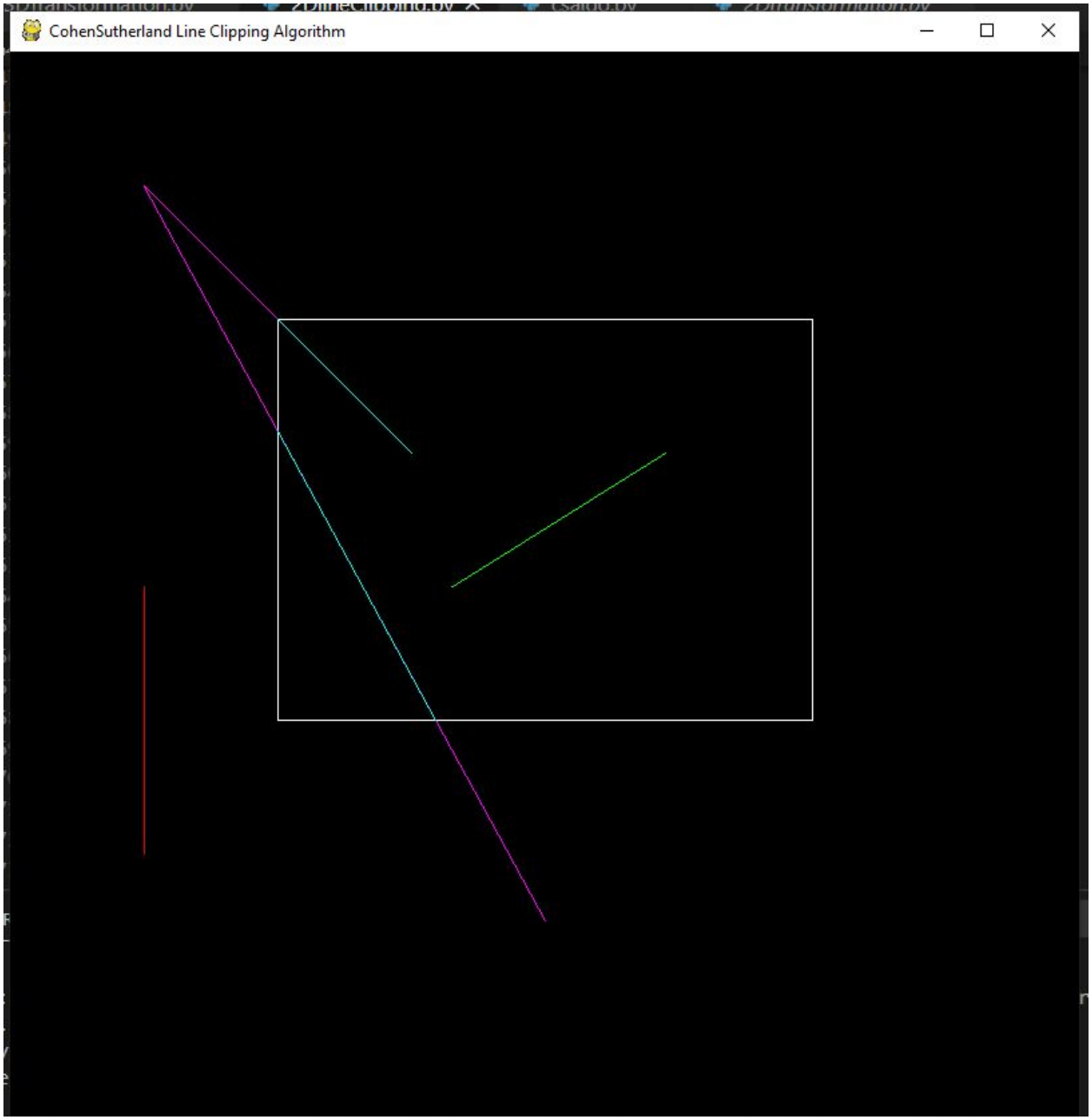
```
# height and width of the screen
width = 800
height = 800
screen = pygame.display.set_mode([width, height], pygame.RESIZABLE)
pygame.display.set_caption("CohenSutherland Line Clipping Algorithm")

# Defining region codes
INSIDE = 0 # 0000
TOP = 8  # 1000
BOTTOM = 4 # 0100
RIGHT = 2 # 0010
LEFT = 1 # 0001
TOPLEFT = 9 # 1001
TOPRIGHT = 10 # 1010
BOTTOMLEFT = 5 # 0101
BOTTOMRIGHT = 6 # 0110

# Defining x_max, y_max and x_min, y_min for rectangle
xMin = 200
yMin = 200
xMax = 600
yMax = 500


# Function to compute region code for a point(x, y)
def RegionCode(x, y):
    if x < xMin:
        if y < yMin:
            return BOTTOMLEFT
        elif y > yMax:
            return TOPLEFT
        return LEFT
    elif x > xMax:
        if y < yMin:
            return BOTTOMRIGHT
        elif y > yMax:
            return TOPRIGHT
        return RIGHT
    else:
        if y < yMin:
```

```
            return BOTTOM
        elif y > yMax:
            return TOP
        return INSIDE


# Function for line clipping
def CohenSutherland(x1, y1, x2, y2):
    code1 = RegionCode(x1, y1)
    code2 = RegionCode(x2, y2)

    if code1 == 0 and code2 == 0:
        # Entire line accepted
        pygame.gfxdraw.line(screen, x1, y1, x2, y2, green)
    else:
        if code1 & code2 != 0:
            # Entire line rejected
            pygame.gfxdraw.line(screen, x1, y1, x2, y2, red)
        else:
            codes = [0, 0]
            # for points outside clipping window
            if code1 != 0:
                codes[0] = code1
            if code2 != 0:
                codes[1] = code2

            index = 0
            m = (y2-y1)/(x2 - x1)
            for code in codes:
                if index == 0:
                    x0 = x1
                    y0 = y1
                else:
                    x0 = x2
                    y0 = y2

                while code != 0:
                    if code & 1 == 1:
                        # Clipping from Left
                        x = xMin
```

```
                        y = y0 + m * (x - x0)
                        code = RegionCode(x, y)
                        codes[index] = code
                    elif code & 2 == 2:
                        # Clipping from Right
                        x = xMax
                        y = y0 + m * (x - x0)
                        code = RegionCode(x, y)
                        codes[index] = code

                    if code == 8:
                        # Clipping from Top
                        y = yMax
                        x = x0 + (y - y0)/m
                        code = RegionCode(x, y)
                        codes[index] = code
                    elif code == 4:
                        # Clipping from Bottom
                        y = yMin
                        x = x0 + (y - y0)/m
                        code = RegionCode(x, y)
                        codes[index] = code

                    if index == 0:
                        x1, y1 = floor(x), floor(y)
                    else:
                        x2, y2 = floor(x), floor(y)

                index += 1
            pygame.gfxdraw.line(screen, x1, y1, x2, y2, cyan)


done = False

while not done:
    # If user clicks close
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            done = True
```

```
    # The screen background as black
    screen.fill(black)

    # Clipping window (coloured white)

pygame.gfxdraw.polygon(screen,[(xMin,yMin),(xMax,yMin),(xMax,yMax),(xMin,y
Max)],white)

    # Trivially rejected line
    CohenSutherland(100, 400, 100, 600)

    # Trivially accepted line
    CohenSutherland(330, 400, 490, 300)

    # Original line
    pygame.gfxdraw.line(screen, 100, 100, 300, 300, magenta)
    # Clipped line
    CohenSutherland(100, 100, 300, 300)

    # Original line
    pygame.gfxdraw.line(screen, 100, 100, 400, 650, magenta)
    # Clipped line
    CohenSutherland(100, 100, 400, 650)

    # Update the contents of the entire display
    pygame.display.flip()

pygame.quit()
```

Output:



Conclusion:

The rectangle in white is the clipping window. The line in green is trivially accepted whereas the line in red is trivially rejected. The line in cyan is the clipped line whose clipped out part (line outside the clipping window) is drawn in magenta (just for reference).

3. **Title:** Implementing 3D Transformations: Translation, Rotation, Scaling on any 3D shape

Formulae:

Translate(tx, ty, tz):

$$\text{Translation Matrix} = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotate(angle):

$$\text{XRotation Matrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(angle) & -\sin(angle) & 0 \\ 0 & \sin(angle) & \cos(angle) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{YRotation Matrix} = \begin{bmatrix} \cos(angle) & 0 & \sin(angle) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(angle) & 0 & \cos(angle) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{ZRotation Matrix} = \begin{bmatrix} \cos(angle) & -\sin(angle) & 0 & 0 \\ \sin(angle) & \cos(angle) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scale(sx, sy, sz):

$$\text{Scaling Matrix} = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Source Code:

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
from itertools import product, combinations
from math import sin, cos, pi
```

```
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.set_aspect("auto")

# cube
r = [-1, 1]
for s, e in combinations(np.array(list(product(r, r, r))), 2):
    if np.sum(np.abs(s-e)) == r[1]-r[0]:
        ax.plot3D(*zip(s, e), color="b")


# Function to translate
def Translate(tx, ty, tz):
    for s, e in combinations(np.array(list(product(r, r, r))), 2):
        if np.sum(np.abs(s-e)) == r[1]-r[0]:
            sTranslated = [s[0] + tx, s[1] + ty, s[2] + tz]
            eTranslated = [e[0] + tx, e[1] + ty, e[2] + tz]
            ax.plot3D(*zip(sTranslated,eTranslated), color="g")


# Function to rotate in x axis
def XRotate(angle):
    for s, e in combinations(np.array(list(product(r, r, r))), 2):
        if np.sum(np.abs(s-e)) == r[1]-r[0]:
             sRotated = [s[0], s[1] * cos(angle) - s[2] * sin(angle), s[1]
* sin(angle) + s[2] * cos(angle)]
             eRotated = [e[0], e[1] * cos(angle) - e[2] * sin(angle), e[1]
* sin(angle) + e[2] * cos(angle)]
            ax.plot3D(*zip(sRotated,eRotated), color="r")


# Function to rotate in y axis
def YRotate(angle):
    for s, e in combinations(np.array(list(product(r, r, r))), 2):
        if np.sum(np.abs(s-e)) == r[1]-r[0]:
            sRotated = [s[0] * cos(angle) + s[2] * sin(angle), s[1], -s[0]
* sin(angle) + s[2] * cos(angle)]
```

```
            eRotated = [e[0] * cos(angle) + e[2] * sin(angle), e[1], -e[0]
* sin(angle) + e[2] * cos(angle)]
            ax.plot3D(*zip(sRotated,eRotated), color="y")


# Function to rotate in z axis
def ZRotate(angle):
    for s, e in combinations(np.array(list(product(r, r, r))), 2):
        if np.sum(np.abs(s-e)) == r[1]-r[0]:
            sRotated = [s[0] * cos(angle) - s[1] * sin(angle), s[0] *
sin(angle) + s[1] * cos(angle), s[2]]
            eRotated = [e[0] * cos(angle) - e[1] * sin(angle), e[0] *
sin(angle) + e[1] * cos(angle), e[2]]
            ax.plot3D(*zip(sRotated,eRotated), color="k")


# Function to scale
def Scale(sx, sy, sz):
    for s, e in combinations(np.array(list(product(r, r, r))), 2):
        if np.sum(np.abs(s-e)) == r[1]-r[0]:
            sScaled = [s[0] * sx, s[1] * sy, s[2] * sz]
            eScaled = [e[0] * sx, e[1] * sy, e[2] * sz]
            ax.plot3D(*zip(sScaled,eScaled), color="m")


# # sphere
# u, v = np.mgrid[0:2*np.pi:20j, 0:np.pi:10j]
# x = np.cos(u)*np.sin(v)
# y = np.sin(u)*np.sin(v)
# z = np.cos(v)
# ax.plot_wireframe(x, y, z, color="r")

Translate(2, 2, 2)
XRotate(pi/4)
YRotate(pi/4)
ZRotate(pi/4)
Scale(2, 2, 2)

plt.show()
```

Output:

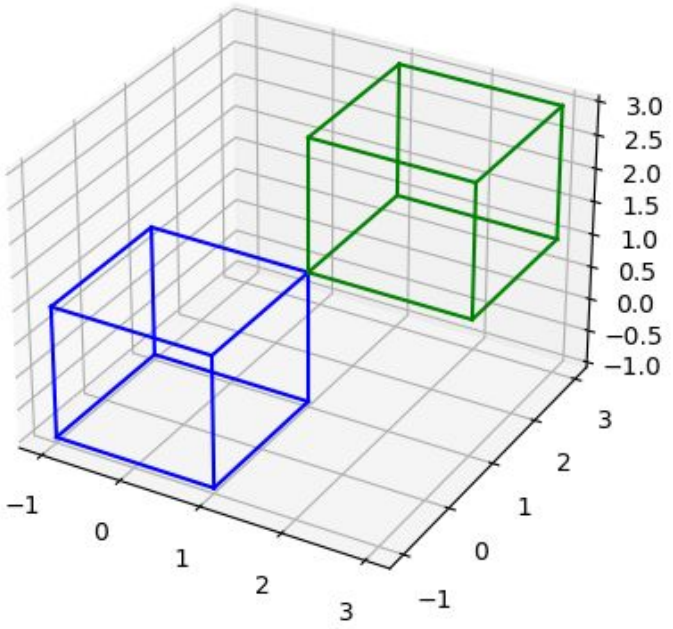


Figure: Translation by (2, 2, 2)

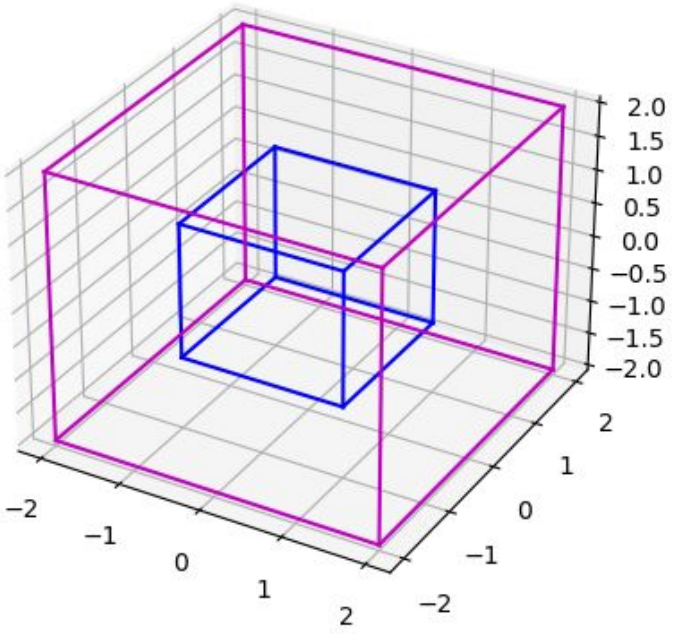


Figure: Scaled to double the size

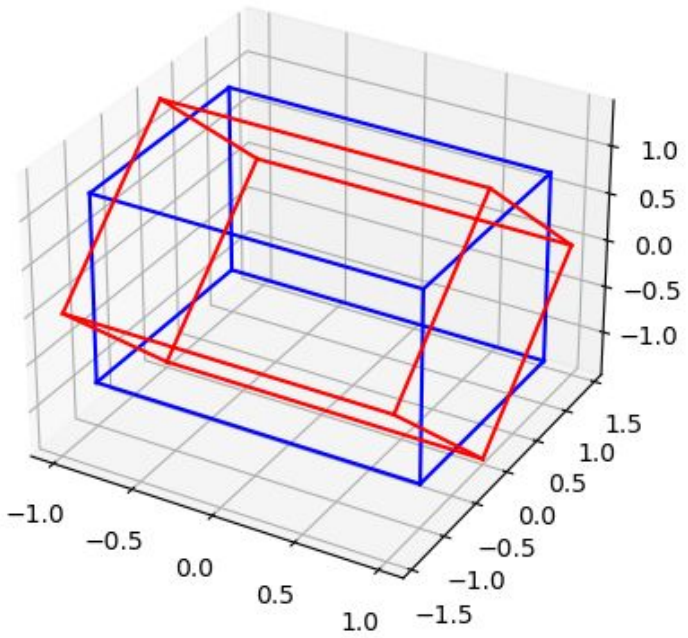


Figure: Rotated by 45 degree along x axis

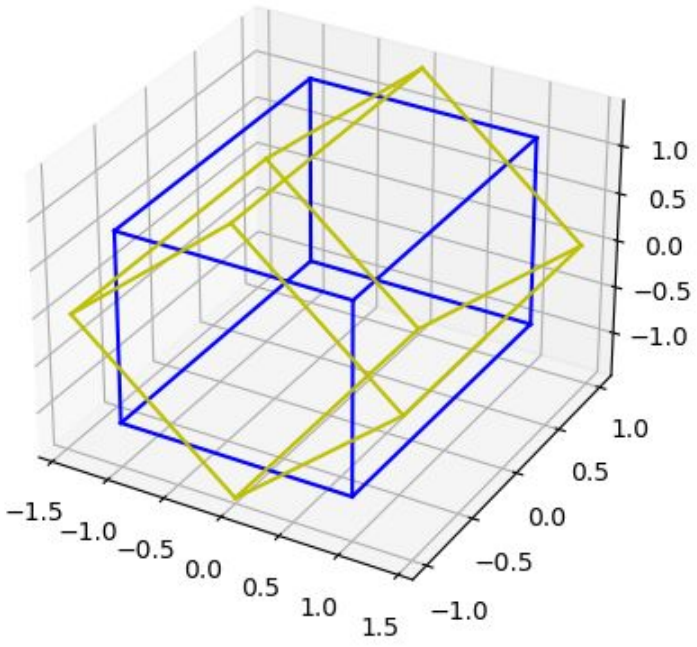


Figure: Rotated by 45 degree along y axis

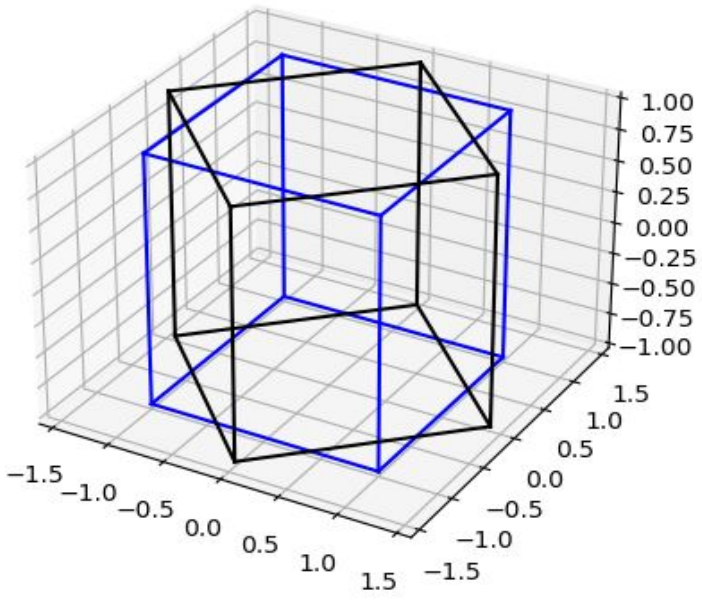


Figure: Rotated by 45 degree along z axis

Conclusion:

Pygame doesn't support 3D graphics, thus Axes3D from matplotlib's 3D toolkit was used. All the Transformations are shown in figures above, where the cube in blue colour is at original position and the other coloured cube is the transformed one.