Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Computer Graphics Lab Report 06

on

‘**3D Transformations and Projection Algorithms - Lab 06 Task**’

Submitted By:

Reewaj Khanal (61)

Submitted to:

Mr. Dhiraj Shrestha

Assistant Professor

Department of Computer Science and Engineering

School of Engineering

Kathmandu University

Dhulikhel, Kavre

Submission Date: Wednesday 12 June 2024

Question No. 1 Implement the following 3D transformations using the 3D shapes provided by Opengl:

- Translation
- Rotation
- Shearing
- Scaling

Answer:

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


# Cube vertices
vertices = [
    [-1, -1, -1],
    [1, -1, -1],
    [1, 1, -1],
    [-1, 1, -1],
    [-1, -1, 1],
    [1, -1, 1],
    [1, 1, 1],
    [-1, 1, 1]
]


# Cube edges
edges = [
    [0, 1],
```

```
    [1, 2],
    [2, 3],
    [3, 0],
    [4, 5],
    [5, 6],
    [6, 7],
    [7, 4],
    [0, 4],
    [1, 5],
    [2, 6],
    [3, 7]
]

# Transformation parameters
translation = [0.0, 0.0, 0.0]
scaling_factor = 1.0
rotation_axis = [1, 0, 0]
rotation_angle = 0
shearing_factors = [0.0, 0.0, 0.0]
current_transformation = None

def draw_cube():
    glBegin(GL_LINES)
    for edge in edges:
        for vertex in edge:
            glVertex3fv([v * scaling_factor for v in vertices[vertex]])  # Scale each
vertex
    glEnd()

def apply_translation():
    glPushMatrix()
```

```
    glTranslatef(translation[0], translation[1], translation[2])
    draw_cube()
    glPopMatrix()

def apply_scaling():
    glPushMatrix()
    glScalef(scaling_factor, scaling_factor, scaling_factor)
    draw_cube()
    glPopMatrix()

def apply_rotation():
    global rotation_axis, rotation_angle
    glPushMatrix()
    glRotatef(rotation_angle, *rotation_axis)
    draw_cube()
    glPopMatrix()

def apply_shearing(axis):
    shearing_matrix = np.identity(4, dtype=np.float32)
    if axis == 'x':
        shearing_matrix[1, 0] = shearing_factors[0]  # Shearing only along the x-axis
    elif axis == 'y':
        shearing_matrix[2, 1] = shearing_factors[0]  # Shearing only along the y-axis
    elif axis == 'z':
        shearing_matrix[3, 2] = shearing_factors[0]  # Shearing only along the z-axis
    glMultMatrixf(shearing_matrix)
    draw_cube()

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

```
    glLoadIdentity()
    gluLookAt(3, 3, 3, 0, 0, 0, 0, 1, 0)  # Camera position

    if current_transformation == 'translation':
        apply_translation()
    elif current_transformation == 'scaling':
        apply_scaling()
    elif current_transformation == 'rotation':
        apply_rotation()
    elif current_transformation == 'shearing':
        apply_shearing('y')  # Change the axis here

    glFlush()

def key_callback(window, key, scancode, action, mods):
        global translation, scaling_factor, rotation_angle, rotation_axis,
shearing_factors, current_transformation
    if key == glfw.KEY_UP and action == glfw.PRESS:
        if current_transformation == 'translation':
            translation[1] += 0.1
    elif key == glfw.KEY_DOWN and action == glfw.PRESS:
        if current_transformation == 'translation':
            translation[1] -= 0.1
    elif key == glfw.KEY_LEFT and action == glfw.PRESS:
        if current_transformation == 'translation':
            translation[0] -= 0.1
        elif current_transformation == 'shearing':
            shearing_factors[0] -= 0.1
    elif key == glfw.KEY_RIGHT and action == glfw.PRESS:
        if current_transformation == 'translation':
            translation[0] += 0.1
```

```
        elif current_transformation == 'shearing':
            shearing_factors[0] += 0.1
    elif key == glfw.KEY_W and action == glfw.PRESS:
        if current_transformation == 'translation':
            translation[2] += 0.1
    elif key == glfw.KEY_S and action == glfw.PRESS:
        if current_transformation == 'translation':
            translation[2] -= 0.1
    elif key == glfw.KEY_Z and action == glfw.PRESS:
        if current_transformation == 'scaling':
            scaling_factor += 0.1
        elif current_transformation == 'rotation':
            rotation_axis = [0, 0, 1]
            rotation_angle += 15
    elif key == glfw.KEY_X and action == glfw.PRESS:
        if current_transformation == 'scaling':
            scaling_factor -= 0.1
        elif current_transformation == 'rotation':
            rotation_axis = [1, 0, 0]
            rotation_angle += 15
    elif key == glfw.KEY_Y and action == glfw.PRESS:
        if current_transformation == 'rotation':
            rotation_axis = [0, 1, 0]
            rotation_angle += 15
    elif key == glfw.KEY_M and action == glfw.PRESS:
        main_menu(window)


def main_menu(window):
    global current_transformation
    print("\nMain Menu:")
```

```
    print("1. Translation")
    print("2. Scaling")
    print("3. Rotation")
    print("4. Shearing")
    print("5. Exit")
    choice = input("Select a transformation (1-5): ")

    if choice == '1':
        current_transformation = 'translation'
         print("Use arrow keys to translate the cube. Press 'W' and 'S' to move along the Z-axis.")
    elif choice == '2':
        current_transformation = 'scaling'
        print("Press 'Z' to increase and 'X' to decrease scaling.")
    elif choice == '3':
        current_transformation = 'rotation'
        print("Press 'X', 'Y', or 'Z' to rotate around the respective axis.")
    elif choice == '4':
        current_transformation = 'shearing'
        print("Use left and right arrow keys to shear along the Y-axis.")
    elif choice == '5':
        glfw.set_window_should_close(window, True)
    else:
        print("Invalid choice. Please try again.")
        main_menu(window)

def main():
    if not glfw.init():
        return
    window = glfw.create_window(800, 600, "3D Transformations", None, None)
    if not window:
```

```
        glfw.terminate()
        return

    glfw.make_context_current(window)
    glfw.set_key_callback(window, key_callback)

    glClearColor(0.0, 0.0, 0.0, 0.0)
    glEnable(GL_DEPTH_TEST)

    glMatrixMode(GL_PROJECTION)
    gluPerspective(45, 800 / 600, 0.1, 100.0)
    glMatrixMode(GL_MODELVIEW)

    main_menu(window)

    while not glfw.window_should_close(window):
        glfw.poll_events()
        display()
        glfw.swap_buffers(window)

    glfw.terminate()

if __name__ == "__main__":
    main()
#use arrow Keys for the translation movements explicitly
#use Z for scale up & X for scale down
#use X, Y, & Z for the required rotation
#use Left & Right arrow keys for shearing
```

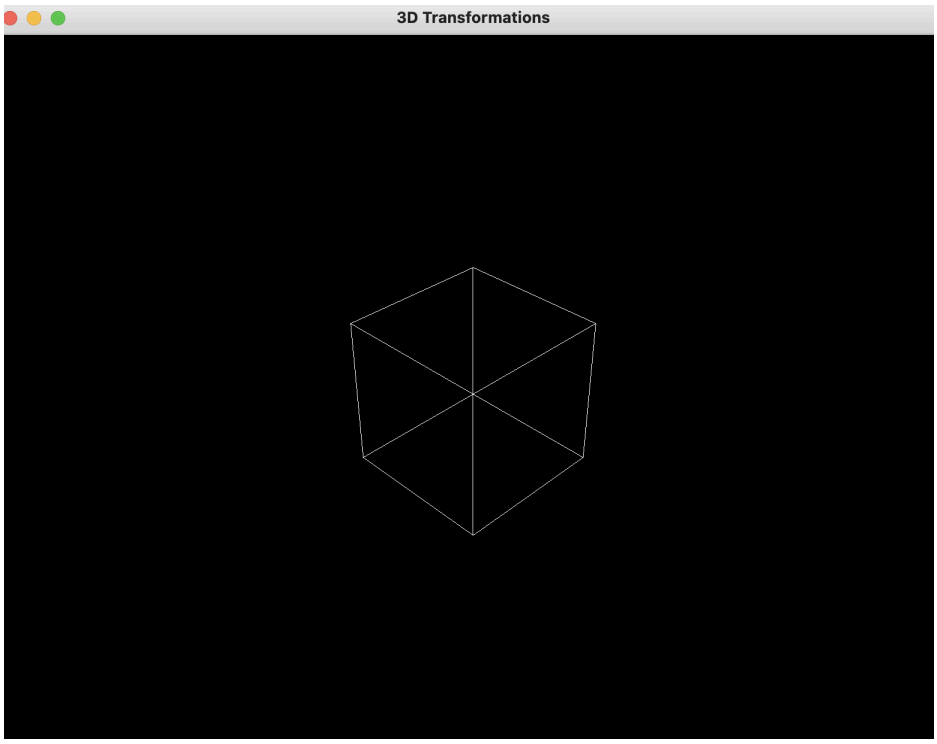
Inputs and Outputs:

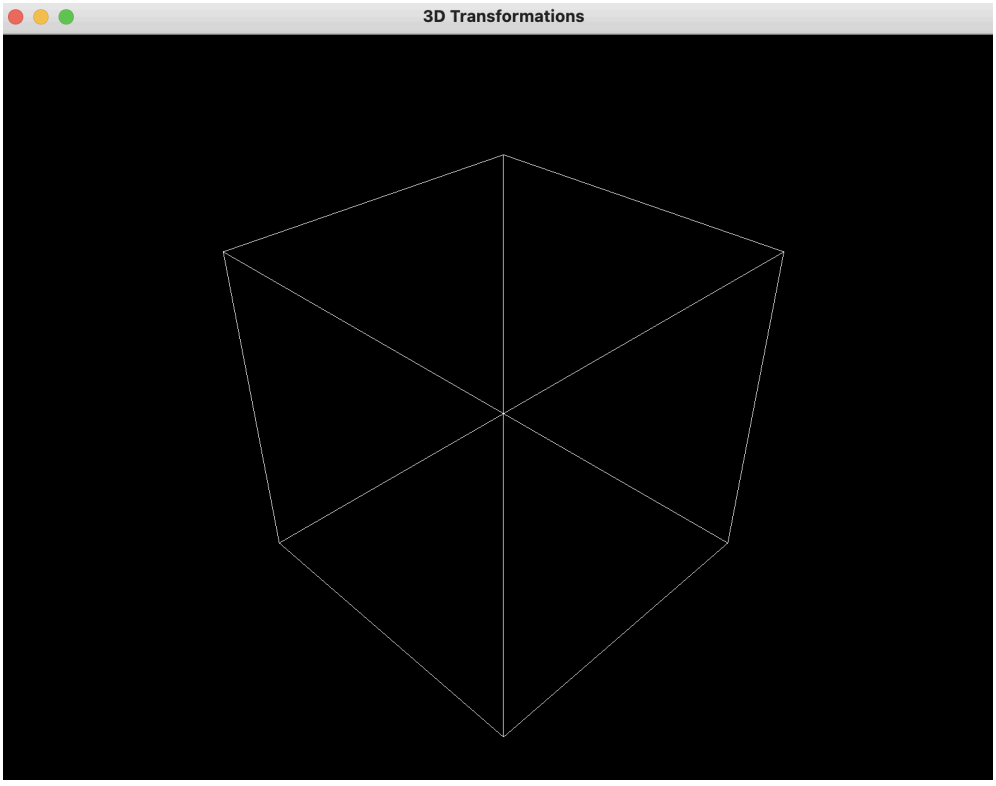
```
(base) reewajkhanal.rk10@RK10 LAB06 % python 3
d.py

Main Menu:
1. Translation
2. Scaling
3. Rotation
4. Shearing
5. Exit
Select a transformation (1-5): 1
Use arrow keys to translate the cube. Press 'W
' and 'S' to move along the Z-axis.
```

```
(base) reewajkhanal.rk10@RK10 LAB06 % python 3
d.py

Main Menu:
1. Translation
2. Scaling
3. Rotation
4. Shearing
5. Exit
Select a transformation (1-5): 2
Press 'Z' to increase and 'X' to decrease scal
ing.
```


3D Transformations


3D Transformations

Question No. 2 Implement the Perspective Projection

Answer:

```
import pygame
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLU import *
import numpy as np


# Define the vertices of the cuboid
vertices = [
    [-1, -1, -1],
    [1, -1, -1],
    [1, 1, -1],
    [-1, 1, -1],
    [-1, -1, 1],
    [1, -1, 1],
    [1, 1, 1],
    [-1, 1, 1]
]


# Define the faces of the cuboid
faces = [
    (0, 1, 2, 3),
```

```
    (4, 5, 6, 7),
    (0, 1, 5, 4),
    (2, 3, 7, 6),
    (0, 3, 7, 4),
    (1, 2, 6, 5)
]

# Define colors for each face
colors = [
    (1, 0, 0),  # Red
    (0, 1, 0),  # Green
    (0, 0, 1),  # Blue
    (1, 1, 0),  # Yellow
    (1, 0, 1),  # Magenta
    (0, 1, 1)   # Cyan
]

def draw_cuboid():
    """Draws a cuboid with colored faces."""
    for i, face in enumerate(faces):
        glColor3fv(colors[i])
        glBegin(GL_QUADS)
        for vertex in face:
            glVertex3fv(vertices[vertex])
        glEnd()

def draw_projection_plane():
    """Draws a projection plane with transparency."""
    glColor4f(1, 1, 1, 0.3)  # White with 30% opacity
    glBegin(GL_QUADS)
```

```
    glVertex3f(-2, -2, -5)
    glVertex3f(2, -2, -5)
    glVertex3f(2, 2, -5)
    glVertex3f(-2, 2, -5)
    glEnd()

def draw_grid():
    """Draws a grid and axes for better visualization."""
    glColor3fv((0.5, 0.5, 0.5))
    glBegin(GL_LINES)
    for i in range(-10, 11):
        glVertex3f(i, -1, -10)
        glVertex3f(i, -1, 10)
        glVertex3f(-10, -1, i)
        glVertex3f(10, -1, i)
    glEnd()

    # Draw axes
    glColor3fv((1, 0, 0))  # X axis in red
    glBegin(GL_LINES)
    glVertex3f(0, 0, 0)
    glVertex3f(2, 0, 0)
    glEnd()

    glColor3fv((0, 1, 0))  # Y axis in green
    glBegin(GL_LINES)
    glVertex3f(0, 0, 0)
    glVertex3f(0, 2, 0)
    glEnd()
```

```
    glColor3fv((0, 0, 1))  # Z axis in blue
    glBegin(GL_LINES)
    glVertex3f(0, 0, 0)
    glVertex3f(0, 0, 2)
    glEnd()

def perspective_projection(fov):
    """Sets up a perspective projection matrix with a given field of view (FOV)."""
    glMatrixMode(GL_PROJECTION)  # Select the projection matrix
    glLoadIdentity()  # Reset the projection matrix
    gluPerspective(fov, (800 / 600), 0.1, 50.0)  # Set up a perspective projection
    glMatrixMode(GL_MODELVIEW)  # Select the modelview matrix
    glLoadIdentity()  # Reset the modelview matrix

def main():
    """Main function to initialize and run the OpenGL/pygame window."""
    pygame.init()  # Initialize pygame
    display = (800, 600)  # Set the display size
     pygame.display.set_mode(display, DOUBLEBUF | OPENGL)   # Set the display mode to double buffer and OpenGL

    # Initialize rotation, scaling, shearing, and field of view variables
    rotation_x, rotation_y = 0, 0
    scale = 1.0
    shear = 0.0
    fov = 45

    perspective_projection(fov)  # Apply the perspective projection
      glTranslatef(0, 0, -8)    # Translate the scene to view both the object and projection plane
```

```
# Enable blending for transparency
glEnable(GL_BLEND)
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)


while True:
    for event in pygame.event.get():  # Event handling
        if event.type == pygame.QUIT:  # Exit condition
            pygame.quit()  # Quit pygame
            quit()  # Exit the program
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:  # Rotate left
                rotation_y -= 5
            if event.key == pygame.K_RIGHT:  # Rotate right
                rotation_y += 5
            if event.key == pygame.K_UP:  # Rotate up
                rotation_x -= 5
            if event.key == pygame.K_DOWN:  # Rotate down
                rotation_x += 5
            if event.key == pygame.K_a:  # Decrease FOV
                fov -= 5
                if fov < 10:
                    fov = 10
                perspective_projection(fov)
            if event.key == pygame.K_d:  # Increase FOV
                fov += 5
                if fov > 120:
                    fov = 120
                perspective_projection(fov)
            if event.key == pygame.K_s:  # Scale down
                scale -= 0.1
```

```
                    if scale < 0.1:
                        scale = 0.1
                if event.key == pygame.K_w:  # Scale up
                    scale += 0.1
                if event.key == pygame.K_q:  # Shear left
                    shear -= 0.1
                if event.key == pygame.K_e:  # Shear right
                    shear += 0.1

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)  # Clear the screen and depth buffer

        glPushMatrix()  # Save the current matrix

        # Draw grid and axes
        draw_grid()

        # Apply scaling
        glScalef(scale, scale, scale)

        # Apply shearing
        shear_matrix = np.array([
            [1, shear, 0, 0],
            [0, 1, 0, 0],
            [0, 0, 1, 0],
            [0, 0, 0, 1]
        ], dtype=np.float32)
        glMultMatrixf(shear_matrix.T)

        # Apply rotation
        glRotatef(rotation_x, 1, 0, 0)  # Apply rotation around the x-axis
```

```
        glRotatef(rotation_y, 0, 1, 0)  # Apply rotation around the y-axis


        # Draw the cuboid
        draw_cuboid()
        glPopMatrix()  # Restore the previous matrix


        # Draw the projection plane
        draw_projection_plane()


        pygame.display.flip()  # Swap the buffers
        pygame.time.wait(10)  # Wait for 10 milliseconds


main()
```

Inputs and Outputs:

```
(base) reewajkhanal.rk10@RK10 LAB06 % python p
respproj\ IV.py
pygame 2.5.2 (SDL 2.28.3, Python 3.10.9)
Hello from the pygame community. https://www.p
ygame.org/contribute.html
```