Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

LAB-6

[Course : COMP 342]

Submitted by

Sudip Bhattarai

Roll no:11

Submitted to

Mr.Dhiraj Shrestha
Department of Computer Science and Engineering

JUN 13 , 2024,

Implement Transformation of 3D Objects

For this lab, we have drawn a cube to represent a 3D operation and performed the following operations:

- Translation

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

- Scaling

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

- Rotation (along X ,Y and Z Axis)

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

- Shear

$$\begin{bmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Code Implementation:

Here we have defined homogeneous transformation for above shown transformation

```python
import numpy as np

def add_homogeneous_coordinate(vertices):
    return np.hstack((vertices, np.ones((vertices.shape[0], 1))))

def remove_homogeneous_coordinate(vertices):
    return vertices[:, :3]

def translate(vertices, x, y, z):
    T = np.array([
        [1, 0, 0, x],
        [0, 1, 0, y],
        [0, 0, 1, z],
        [0, 0, 0, 1]
    ])
    return np.dot(vertices, T.T)

def rotate_x(vertices, angle):
    angle = np.radians(angle)
    cos_a = np.cos(angle)
    sin_a = np.sin(angle)
    R_x = np.array([
        [1, 0, 0, 0],
        [0, cos_a, -sin_a, 0],
        [0, sin_a, cos_a, 0],
        [0, 0, 0, 1]
    ])
    return np.dot(vertices, R_x.T)

def rotate_y(vertices, angle):
    angle = np.radians(angle)
    cos_a = np.cos(angle)
    sin_a = np.sin(angle)
    R_y = np.array([
        [cos_a, 0, sin_a, 0],
        [0, 1, 0, 0],
        [-sin_a, 0, cos_a, 0],
        [0, 0, 0, 1]
    ])
    return np.dot(vertices, R_y.T)

def rotate_z(vertices, angle):
    angle = np.radians(angle)
    cos_a = np.cos(angle)
    sin_a = np.sin(angle)
    R_z = np.array([
        [cos_a, -sin_a, 0, 0],
        [sin_a, cos_a, 0, 0],
        [0, 0, 1, 0],
        [0, 0, 0, 1]
    ])
    return np.dot(vertices, R_z.T)

def scale(vertices, sx, sy, sz):
    S = np.diag([sx, sy, sz, 1])
    return np.dot(vertices, S)

def shear(vertices, sh_xy, sh_xz, sh_yz):
    S = np.array([
        [1, sh_xy, sh_xz, 0],
        [0, 1, sh_yz, 0],
        [0, 0, 1, 0],
        [0, 0, 0, 1]
    ])
    return np.dot(vertices, S.T)


def apply_transformations(vertices, state):
    homogeneous_vertices = add_homogeneous_coordinate(vertices)
    transformed_vertices = translate(homogeneous_vertices,
*state['translate']) transformed_vertices = rotate_x(transformed_vertices, state['rotate_x'])
    transformed_vertices = rotate_y(transformed_vertices, state['rotate_y'])
    transformed_vertices = rotate_z(transformed_vertices, state['rotate_z'])
    transformed_vertices = scale(transformed_vertices, *state['scale'])
    transformed_vertices = shear(transformed_vertices, *state['shear'])
    return remove_homogeneous_coordinate(transformed_vertices)
```

To draw a 3D object, we simply draw a cube

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

vertices = np.array([
    [-1, -1, 1],
    [1, -1, 1],
    [1, 1, 1],
    [-1, 1, 1],
    [-1, -1, -1],
    [1, -1, -1],
    [1, 1, -1],
    [-1, 1, -1]
], dtype=float)

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
]

faces = [
    [0, 1, 2, 3],   # Front face
    [3, 2, 6, 7],   # Top face
    [7, 6, 5, 4],   # Back face
    [4, 5, 1, 0],   # Bottom face
    [5, 6, 2, 1],   # Right face
    [7, 4, 0, 3]    # Left face
]

def draw_cube(vertices):
    glBegin(GL_QUADS)
    for i, face in enumerate(faces):
        glColor3fv(colors[i])
        for vertex in face:

 glVertex3fv(vertices[vertex])
```

Opengl Implementation:

```
import glfw
from OpenGL.GL import *
from cube_3dobject import draw_cube, vertices
from transformation import apply_transformations
from OpenGL.GLUT import *
from OpenGL.GLU import *
projection_mode = 'perspective'

def custom_perspective():
  glMatrixMode(GL_PROJECTION)
  glLoadIdentity()
  gluPerspective(45, (800 / 600), 1, 50.0)
  glMatrixMode(GL_MODELVIEW)

def custom_orthographic():
  glMatrixMode(GL_PROJECTION)
  glLoadIdentity()
  glOrtho(-2, 2, -2, 2, 1, 10)
  glMatrixMode(GL_MODELVIEW)

def set_projection(mode):
  global projection_mode
  projection_mode = mode
  if projection_mode == 'perspective':
    custom_perspective()
  elif projection_mode == 'orthographic':
    custom_orthographic()

transformation_state = {
  'translate': [0.0, 0.0, 0.0],
```

```
    'rotate_x': 0.0,
    'rotate_y': 0.0,
    'rotate_z': 0.0,
    'scale': [1.0, 1.0, 1.0],
    'shear': [0.0, 0.0, 0.0]
}


def key_callback(window, key, scancode, action, mods):
    global transformation_state
    if action == glfw.PRESS:
        if key == glfw.KEY_ESCAPE:
            glfw.set_window_should_close(window, True)
        elif key == glfw.KEY_UP:
            transformation_state['translate'][1] += 0.1
        elif key == glfw.KEY_DOWN:
            transformation_state['translate'][1] -= 0.1
        elif key == glfw.KEY_LEFT:
            transformation_state['translate'][0] -= 0.1
        elif key == glfw.KEY_RIGHT:
            transformation_state['translate'][0] += 0.1
        elif key == glfw.KEY_R:  # Rotate
            transformation_state['rotate_y'] += 10.0
        elif key == glfw.KEY_Q :  # Rotate X+
            transformation_state['rotate_x'] += 10.0
        elif key == glfw.KEY_W:  # Rotate X-
            transformation_state['rotate_x'] -= 10.0
        elif key == glfw.KEY_E:  # Rotate Z+
            transformation_state['rotate_z'] += 10.0
        elif key == glfw.KEY_T:  # Rotate Z-
            transformation_state['rotate_z'] -= 10.0
```

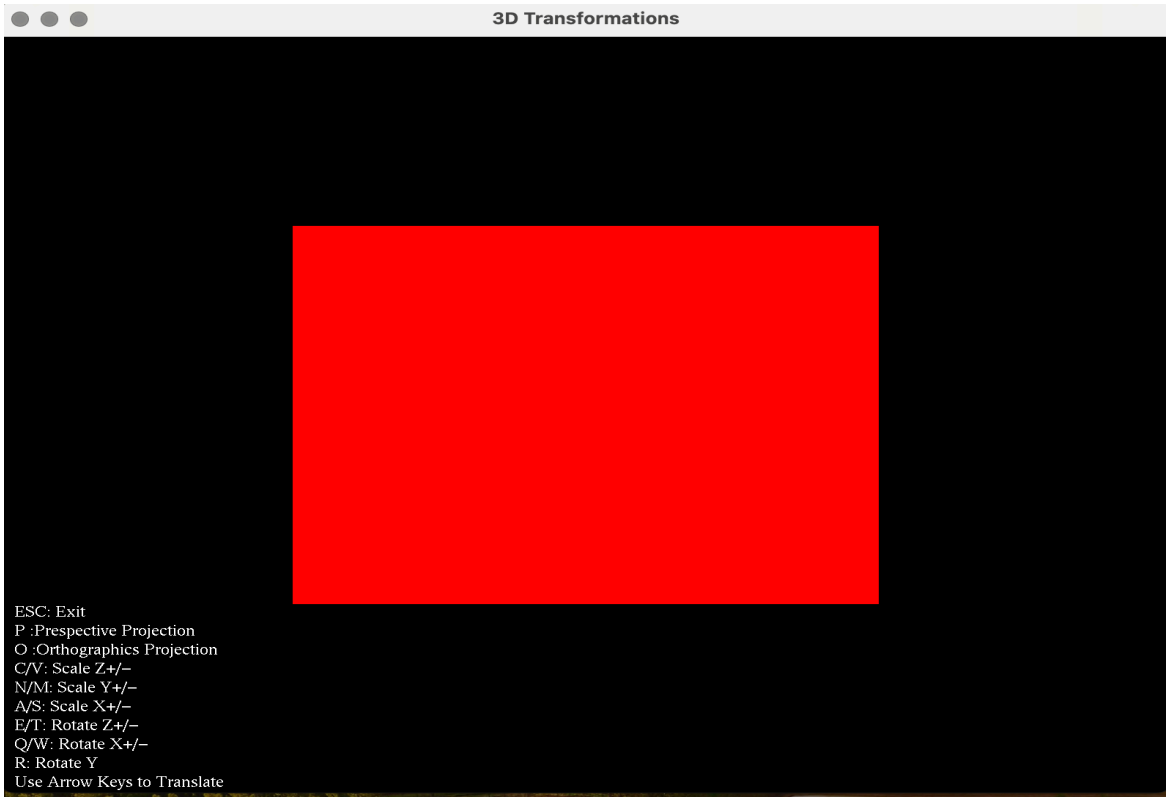
```
    elif key == glfw.KEY_A:  # Scale X+
        transformation_state['scale'][0] += 0.1
    elif key == glfw.KEY_S:  # Scale X-
        transformation_state['scale'][0] -= 0.1
    elif key == glfw.KEY_N:  # Scale Y+
        transformation_state['scale'][1] += 0.1
    elif key == glfw.KEY_M:  # Scale Y-
        transformation_state['scale'][1] -= 0.1
    elif key == glfw.KEY_C:  # Scale Z+
        transformation_state['scale'][2] += 0.1
    elif key == glfw.KEY_V:  # Scale Z-
        transformation_state['scale'][2] -= 0.1
    elif key == glfw.KEY_P:
        set_projection('perspective')
    elif key == glfw.KEY_O:
        set_projection('orthographic')

def draw_text(x, y, text):
  glWindowPos2f(x, y)
  for ch in text:
      glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, ord(ch))

def display():
  global vertices
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
  glLoadIdentity()

  # Apply transformations
  transformed_vertices = apply_transformations(vertices, transformation_state)

  glTranslatef(0.0, 0.0, -10.0)
```

```
    glRotatef(transformation_state['rotate_x'], 1.0, 0.0, 0.0)
    glRotatef(transformation_state['rotate_y'], 0.0, 1.0, 0.0)
    glRotatef(transformation_state['rotate_z'], 0.0, 0.0, 1.0)
    glScalef(*transformation_state['scale'])

    draw_cube(transformed_vertices)
    glColor3f(1, 1, 1)
    draw_text(20, 10, "Use Arrow Keys to Translate")
    draw_text(20, 40, "R: Rotate Y")
    draw_text(20, 70, "Q/W: Rotate X+/-")
    draw_text(20, 100, "E/T: Rotate Z+/-")
    draw_text(20, 130, "A/S: Scale X+/-")
    draw_text(20, 160, "N/M: Scale Y+/-")
    draw_text(20, 190, "C/V: Scale Z+/-")
    draw_text(20, 220, "O :Orthographics Projection")
    draw_text(20, 250, "P :Prespective Projection")
    draw_text(20, 280, "ESC: Exit")

    glfw.swap_buffers(window)


def main():
    global window
    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)
```

```
    glEnable(GL_DEPTH_TEST)
    set_projection('orthographic')
    glfw.set_key_callback(window, key_callback)

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

    glfw.terminate()

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

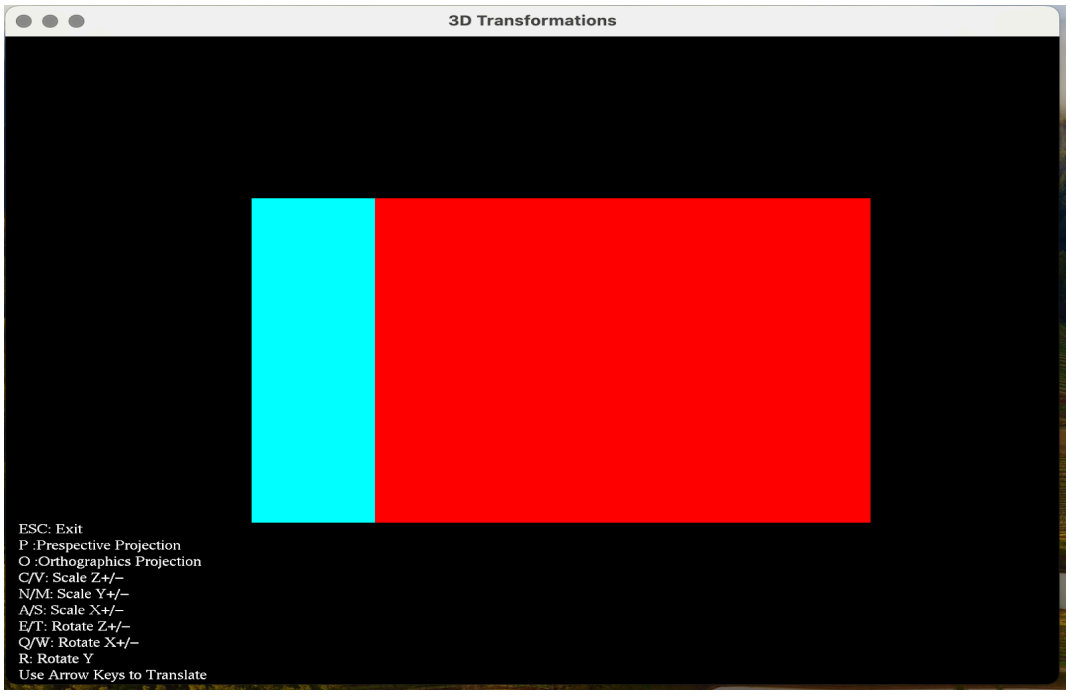
The above code is the implementation of 3D transfer using a transformation matrix .Here, i have also implemented perspective and orthographic projection to display an object .The output are displayed as

- Initial Object

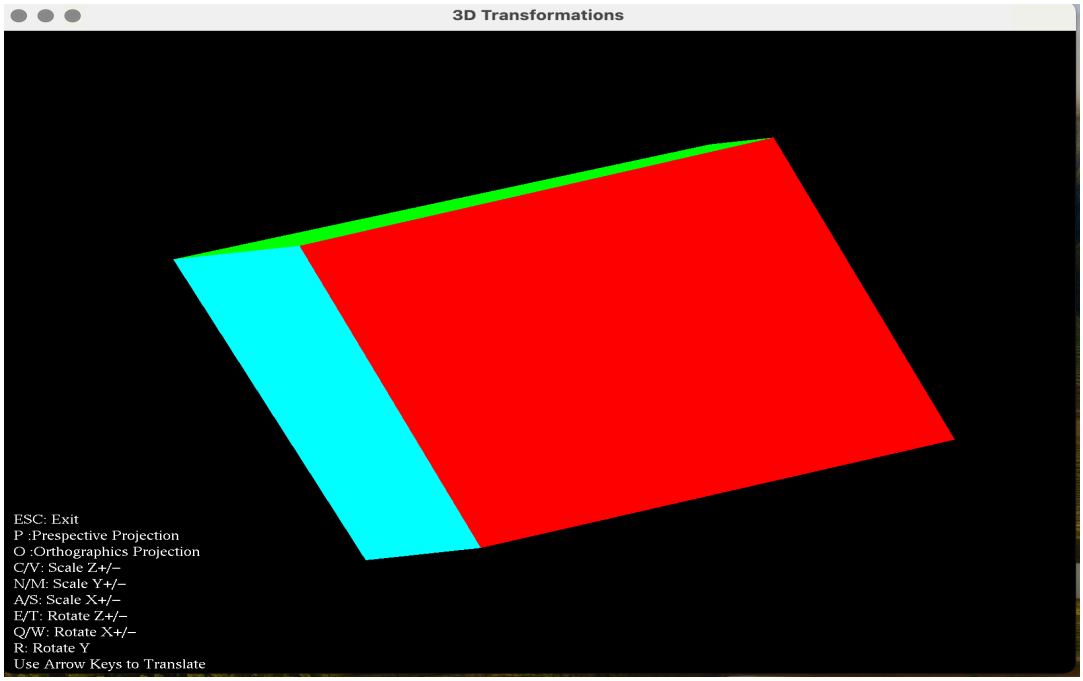


- Rotation

About y-axis



About Z-axis



- Scaling