Computer Graphics Lab 3

OpenGL Basic and Mathematics

---

1. Set Theory and Basic Primitives in OpenGL

1.1) Run the code and observe how basic shapes are drawn

```
import pygame
from OpenGL.GL import *
from OpenGL.GLU import *
from pygame.locals import *

def init_gl():
    glEnable(GL_DEPTH_TEST)
    glClearColor(0.0, 0.0, 0.0, 1.0)

def draw_polygon(vertices, color):
    glBegin(GL_POLYGON)
    glColor3fv(color)
    for vertex in vertices:
        glVertex3fv(vertex)
    glEnd()

def main():
    pygame.init()
    display = (800, 600)
    pygame.display.set_mode(display, DOUBLEBUF | OPENGL)
    pygame.display.set_caption('OpenGL Shapes Workshop')

    init_gl()

    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluOrtho2D(-1, 1, -1, 1)
    glMatrixMode(GL_MODELVIEW)

    current_shape = 'triangle'
    clock = pygame.time.Clock()
    fps = 60

    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
```

```
            return
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_t:
                current_shape = 'triangle'
            elif event.key == pygame.K_s:
                current_shape = 'square'
            elif event.key == pygame.K_ESCAPE:
                pygame.quit()
                return

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

    if current_shape == 'triangle':
        draw_polygon([(-0.6, -0.4, 0.0), (0.6, -0.4, 0.0), (0.0, 0.6, 0.0)], (1.0, 0.0, 0.0))
    elif current_shape == 'square':
        draw_polygon([(-0.4, -0.4, 0.0), (0.4, -0.4, 0.0), (0.4, 0.4, 0.0), (-0.4, 0.4, 0.0)], (1.0, 1.0, 0.0))

    pygame.display.flip()
    clock.tick(fps)

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

1.2) Press 'T' to display a triangle, 'S' to display a square

1.3) Exercvise:

- Add a new shape - pentagon
- Change the colors
- Change the size of shapes (using matrix operation only)
- Add rotation to the shapes (using matrix operation only)
- Controls for the enhanced workshop:
  1. Shape Selection:
     - T': Triangle
     - 'S': Square
     - 'P': Pentagon
  2. Color Modification:
     - R': Increase red component
     - 'G': Increase green component
     - 'B': Increase blue component
  3. Size Control:
     - Up Arrow: Increase size
     - Down Arrow: Decrease size
  4. Rotation:
     - Left Arrow: Rotate counterclockwise
     - Right Arrow: Rotate clockwise

Hint:

Key event:

pygame.K-t : ‘t’, pygame.K-s : ‘s’, pygame.K-p : ‘p’, pygame.K-r : ‘r’, pygame.K-g : ‘g’, pygame.K-b ‘b’, pygame.K-UP : arrow up, pygame.K-DOWN : arrow down, pygame.K-LEFT : arrow left, pygame.K-RIGHT : arrow right

Matric operaton:

Scaling Matrix:

```
| sx 0 0 |
| 0 sy 0 |
| 0  0 1 |
where:
- sx = scaling factor in the x-direction
- sy = scaling factor in the y-direction
Vertex: (x, y) -> Column Vector: | x |
                                 | y |
                                 | 1 |
Scaling Calculation:
| sx  0  0 |   | x |   | sx*x |
| 0  sy  0 | x | y | = | sy*y |
| 0   0  1 |   | 1 |   |  1   |
Result: (sx*x, sy*y)
```

Rotation Matrix (counterclockwise about the origin):

```
| cos(θ) -sin(θ) 0 |
| sin(θ) cos(θ)  0 |
|   0      0     1 |
where:
 - θ = angle of rotation (in radians)
Vertex: (x, y) -> Column Vector: | x |
                                 | y |
                                 | 1 |
Rotation Calculation:
| cos(θ) -sin(θ) 0 |   | x |   | x*cos(θ) - y*sin(θ) |
| sin(θ) cos(θ)  0 | x | y | = | x*sin(θ) + y*cos(θ) |
|    0     0     1 |   | 1 |   |          1          |
Result: (x*cos(θ) - y*sin(θ), x*sin(θ) + y*cos(θ))
```

2. 3D Visualization

2.1) Run the code below:

```
import pygame
from OpenGL.GL import *
from OpenGL.GLU import *
from pygame.locals import *
import math
import numpy

def init_gl():
    glEnable(GL_DEPTH_TEST)
    glClearColor(0.0, 0.0, 0.0, 1.0)
    glEnable(GL_LIGHTING)
    glEnable(GL_LIGHT0)
    glLightfv(GL_LIGHT0, GL_POSITION, (1, 1, 1, 0))
    glLightfv(GL_LIGHT0, GL_AMBIENT, (0.2, 0.2, 0.2, 1))
    glLightfv(GL_LIGHT0, GL_DIFFUSE, (0.8, 0.8, 0.8, 1))
    glEnable(GL_COLOR_MATERIAL)

def draw_cube(vertices):
    glBegin(GL_QUADS)

    glNormal3fv((0, 0, -1))
    for vertex in vertices[0]:
        glVertex3fv(vertex)

    glNormal3fv((0, 0, 1))
    for vertex in vertices[1]:
        glVertex3fv(vertex)

    glNormal3fv((-1, 0, 0))
    for vertex in vertices[2]:
        glVertex3fv(vertex)

    glNormal3fv((1, 0, 0))
    for vertex in vertices[3]:
        glVertex3fv(vertex)
```

```
    glNormal3fv((0, -1, 0))
    for vertex in vertices[4]:
        glVertex3fv(vertex)

    glNormal3fv((0, 1, 0))
    for vertex in vertices[5]:
        glVertex3fv(vertex)

    glEnd()

def scale_matrix(sx, sy, sz):
    return numpy.array([
        [sx, 0, 0, 0],
        [0, sy, 0, 0],
        [0, 0, sz, 0],
        [0, 0, 0, 1]
    ])

def rotate_matrix_z(angle_degrees):
    angle_radians = math.radians(angle_degrees)
    c = math.cos(angle_radians)
    s = math.sin(angle_radians)
    return numpy.array([
        [c, -s, 0, 0],
        [s, c, 0, 0],
        [0, 0, 1, 0],
        [0, 0, 0, 1]
    ])

def transform_vertices(vertices, matrix):
    transformed_vertices = []
    for face in vertices:
        transformed_face = []
        for vertex in face:
            vertex = numpy.array([vertex[0], vertex[1], vertex[2], 1])
            transformed_vertex = matrix.dot(vertex)
```

```
            transformed_face.append((transformed_vertex[0], transformed_vertex[1], transformed_vertex[2]))
        transformed_vertices.append(transformed_face)
    return transformed_vertices

def draw_axes():
    glBegin(GL_LINES)
    glColor3f(1, 0, 0)  # Red for x-axis
    glVertex3f(0, 0, 0)
    glVertex3f(2, 0, 0)
    glVertex3f(1.8, 0.2, 0)  # Letter X
    glVertex3f(2.2, -0.2, 0)
    glVertex3f(1.8, -0.2, 0)  # Letter X
    glVertex3f(2.2, 0.2, 0)

    glColor3f(0, 1, 0)  # Green for y-axis
    glVertex3f(0, 0, 0)
    glVertex3f(0, 2, 0)
    glVertex3f(0.2, 1.8, 0)  # Letter Y
    glVertex3f(0, 2.2, 0)
    glVertex3f(-0.2, 1.8, 0)  # Letter Y
    glVertex3f(0, 2.2, 0)

    glColor3f(0, 0, 1)  # Blue for z-axis
    glVertex3f(0, 0, 0)
    glVertex3f(0, 0, 2)
    glVertex3f(0.2, 0.2, 1.8)  # Letter Z
    glVertex3f(-0.2, 0.2, 1.8)
    glVertex3f(-0.2, 0.2, 1.8)  # Letter Z
    glVertex3f(0.2, -0.2, 1.8)
    glVertex3f(0.2, -0.2, 1.8)  # Letter Z
    glVertex3f(-0.2, -0.2, 1.8)
    glEnd()

def main():
    pygame.init()
    display = (800, 600)
    pygame.display.set_mode(display, DOUBLEBUF | OPENGL)
```

```
pygame.display.set_caption('3D Cube Workshop')

init_gl()

glMatrixMode(GL_PROJECTION)
glLoadIdentity()
gluPerspective(45, (display[0]/display[1]), 0.1, 50.0)
glMatrixMode(GL_MODELVIEW)

# Adjust camera position and orientation
glTranslatef(1.5, 1.5, -7)  # Move and distance the camera
glRotatef(30, 1, 1, 0)     # Rotate for a better viewing angle

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)],
    [(-1, -1, -1), (-1, 1, -1), (-1, 1, 1), (-1, -1, 1)],
    [(1, -1, -1), (1, 1, -1), (1, 1, 1), (1, -1, 1)],
    [(-1, -1, -1), (1, -1, -1), (1, -1, 1), (-1, -1, 1)],
    [(-1, 1, -1), (1, 1, -1), (1, 1, 1), (-1, 1, 1)]
]

scale = 1.0
angle = 0.0
clock = pygame.time.Clock()
fps = 60

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            return
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_UP:
                scale += 0.1
            elif event.key == pygame.K_DOWN:
                scale -= 0.1
```

```
                scale = max(0.1, scale)
            elif event.key == pygame.K_LEFT:
                angle -= 10
            elif event.key == pygame.K_RIGHT:
                angle += 10
            elif event.key == pygame.K_ESCAPE:
                pygame.quit()
                return

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

        scaling = scale_matrix(scale, scale, scale)
        rotation = rotate_matrix_z(angle)
        transformation = rotation.dot(scaling)

        transformed_vertices = transform_vertices(vertices, transformation)
        draw_cube(transformed_vertices)
        draw_axes()  # Draw the axes

        pygame.display.flip()
        clock.tick(fps)

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

2.2) Modify the code to

- Show 3D cube in center of display.
- Rotate along Y-Axis