Міністерство освіти і науки України
Національний технічний університет України «КПІ» імені Ігоря Сікорського
Кафедра обчислювальної техніки ФІОТ

ЗВІТ
з лабораторної роботи № 1
з навчальної дисципліни «Computer Vision»

Тема:

ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ПОБУДОВИ ТА ПЕРЕТВОРЕННЯ КООРДИНАТ ПЛОЩИННИХ (2D) ОБ'ЄКТІВ

Виконав:
Студент 3 курсу кафедри ІПІ ФІОТ,
Навчальної групи ІП-14
Бабіч Д. В.

Перевірив:
Професор кафедри ОТ ФІОТ
Писарчук О. О.

Київ 2024

I. Мета:

Виявити дослідити та узагальнити особливості формування та перетворення координат площинних (2d) та просторових (3d) об’єктів.

II. Завдання:

Завдання II рівня

Здійснити синтез математичних моделей та розробити програмний скрипт, що реалізує базові операції 3D перетворень над геометричними примітивами: аксонометрична проекція будь-якого типу та з циклічне обертання (анімація) 3D графічного об’єкту навколо будь-якої обраної внутрішньої віссю. Траєкторію обертання не відображати. Для розробки використовувати матричні операції. Вхідна матриця координат кутів геометричної фігури має бути розширеною.

Таблиця 1.1 – Варіант завдання

Варіант (день народження)	**Технічні умови**	**Графічна фігура**
9	**Динаміка фігури:** графічна фігура з’являється та гасне, змінює колір контуру та заливки. **Обрати самостійно:** бібліотеку, розмір графічного вікна, розмір фігури, параметри зміни положення фігури, кольорову гамму усіх графічних об’єктів. Всі операції перетворень мають здійснюватись у межах графічного вікна.	Піраміда з чотирикутною основою

III. Результати виконання лабораторної роботи.

3.1. Синтезована математична модель перетворень об’єктів у 3Д просторі.

Для виконання обертання побудованого 3Д об’єкту необхідно було використати матриці трансформацій. Матриці трансформацій використовуються для виконання різних видів перетворень у просторі, таких як обертання, масштабування та перенесення. Вони є основою для багатьох операцій в графіці та комп’ютерному моделюванні. Також ці матриці можуть бути комбіновані для виконання складених перетворень. Наприклад, можна спочатку обернути об’єкт, потім масштабувати його, а потім перенести його до нового місця. Комбінація цих перетворень може бути представлена однією матрицею, яка є результатом множення окремих матриць перетворень, проте у цьому випадку особливу увагу потрібно зважати на порядок, у якому виконуються перетворення, має значення, оскільки множення матриць не є комутативним.

Матриця обертання, довкола осі X, де θ – заданий кут обертання.

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

Матриця обертання, довкола осі Y, де θ – заданий кут обертання.

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

Матриця обертання, довкола осі Z, де θ – заданий кут обертання.

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

3.2. Результати архітектурного проектування та їх опис.

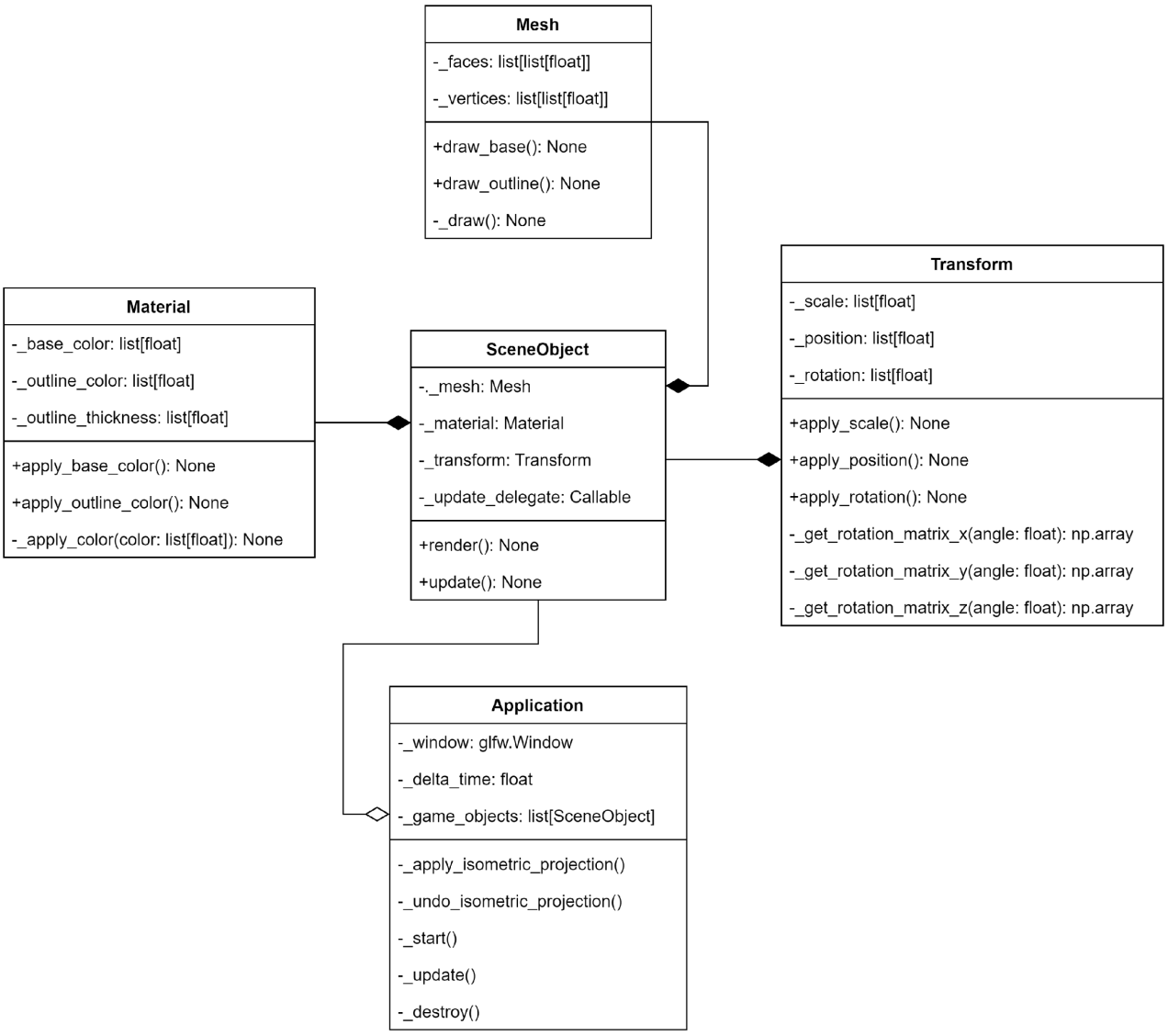

Рисунок 1.1 – UML-діаграма класів

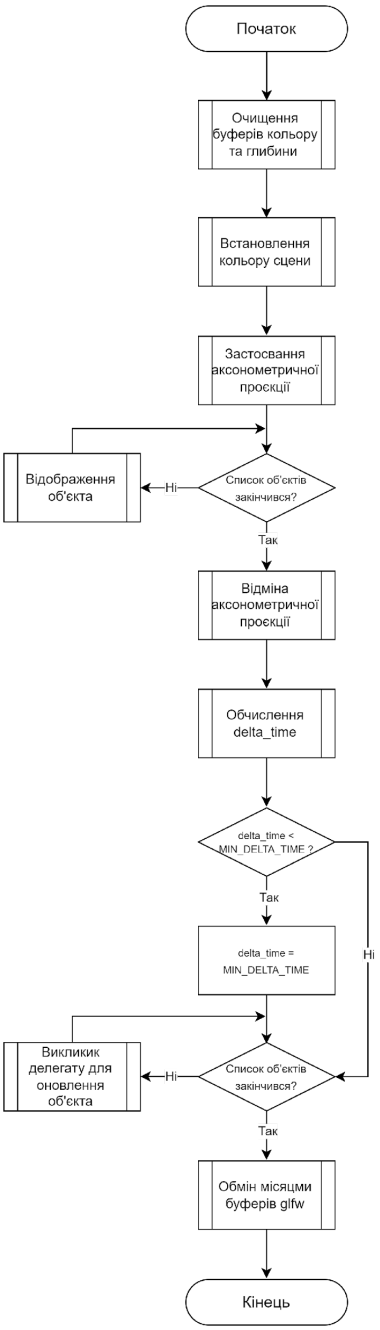

Рисунок 1.2 – Діаграма головного рантайм-циклу застосунку

У наведеній діаграмі класів позначені основні приватні та публічні члени, за допомогою яких і функціонує створене програмне забезпечення. Особливу увагу варто звернути на клас SceneObject, який інкапсулює представлення кожного об’єкту у контексті віртуальної сцени у OpenGL. Цей клас має створені об’єкт класу Transform, який інкапсулює логіку представлення об’єкта у контексті простору 3Д сцени та має збережені значення позиції, розмірів та обертання об’єкту (обертання здійснюється за допомогою наведених методів отримання матриць обертання для заданого кута – get_rotation_matrix_x, get_rotation_matrix_y, get_rotation_matrix_z).

Клас Material інкапсулює представлення за допомогою списку вершин та поверхонь об’єкта і за кожен умовний draw call виконує відображення об’єкта двічі (перший – за допомогою моду glPolygonMode(GL_FRONT_AND_BACK, GL_FILL), що означає відображення повноцінних заповнених полігонів, а другий – glPolygonMode(GL_FRONT_AND_BACK, GL_FILL), що є відображення того ж мешу, але тільки ліній, які і з’єднують вершини між собою).

Окремо варто зазначити логіку делегату update_delegate з SceneObject, який є делегатом, який приймає методи зазначеної сигнатури, які викликаються кожен оновлений кадр, тим самим для кожного об’єкта у сцені можуть виконуватися дії зазначені у цьому методі.

У якості аксонометричної проекції було використано підтипу аксонометричної проекції – ізометричну проекцію, яка утворена використанням комбінованої матриці обертання за віссю X та віссю Y. Ізометрична проекція – це різновид аксонометричної проекції, при якій у відображенні тривимірного об’єкта на площину коефіцієнт спотворення по всіх трьох осях однаковий та представлені в однаково в єдиному масштабі, це означає, що лінії, паралельні одній осі, залишаються паралельними на малюнку, створюючи тривимірне зображення, яке зберігає пропорції та просторову орієнтацію вихідного об’єкта. В ізометричній прямокутній проекції проєктуючі промені перпендикулярні до аксонометричної площини та утворюють між собою кути у 120°.

3.3. Опис структури проекту програми.

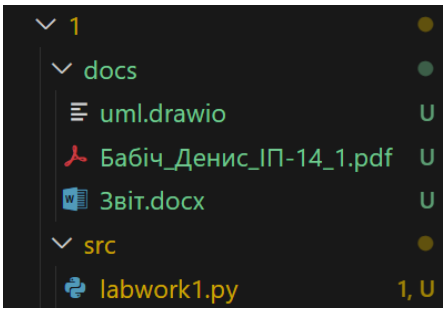

Рисунок 1.3 – Структура проєкту

У директорії src зберігається labwork1.py, який і є модулем з вихідним кодом, директорія docs – зберігає файли звіту у форматі pdf, docx та drawio – файл з діаграмами.

3.4. Результати роботи програми відповідно до завдання.

Чотирикутна піраміда здійснює постійне обертання, змінює колір заливки та колір граней.

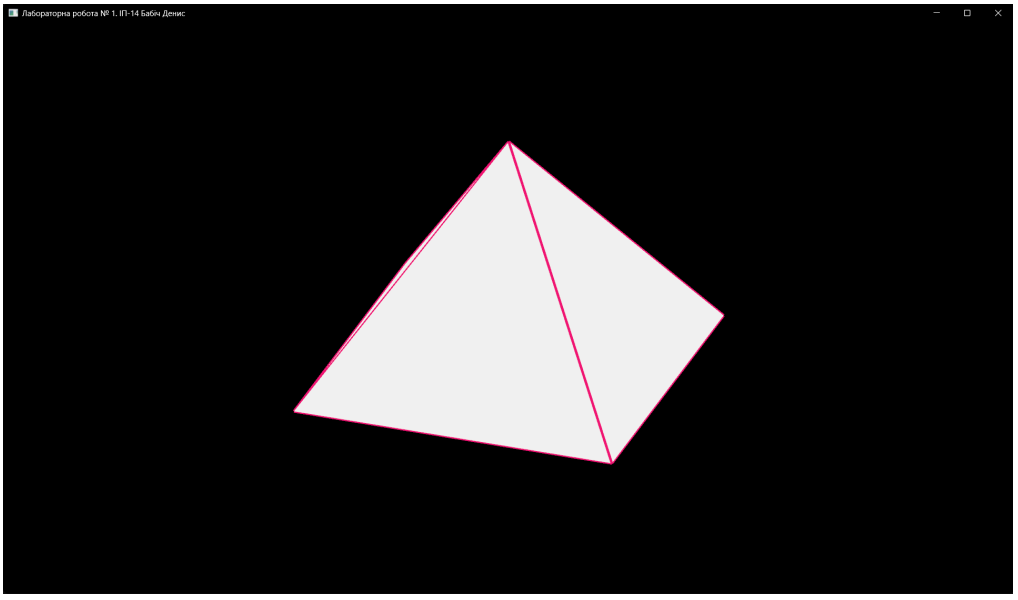

Рисунок 1.4 – Приклад виконання роботи

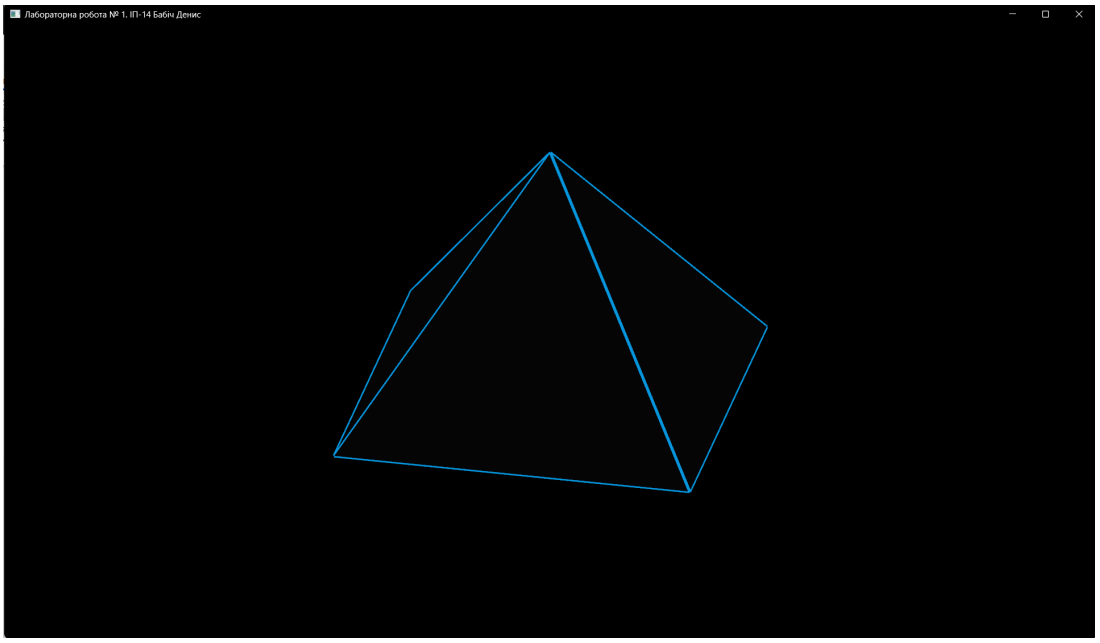

Рисунок 1.5 – Приклад виконання роботи

3.5. Програмний код, що забезпечує отримання результату.

```
import glfw
import numpy as np
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *
from datetime import datetime
from typing import Callable, ForwardRef

class Transform:

    _VECTOR3_X_INDEX = 0
    _VECTOR3_Y_INDEX = 1
    _VECTOR3_Z_INDEX = 2

    @property
    def position(self) -> list[float]:
        return self._position

    @position.setter
    def position(self, value: list[float]) -> None:
        self._position = value

    @property
    def rotation(self) -> list[float]:
        return self._rotation

    @rotation.setter
    def rotation(self, value: list[float]) -> None:
        self._rotation = value
```

```
    @property
    def scale(self) -> list[float]:
        return self._scale

    @scale.setter
    def scale(self, value: list[float]) -> None:
        self._scale = value

    def __init__(self, position: list[float], rotation: list[float], scale: list[float]) -> None:
        self._scale = scale
        self._position = position
        self._rotation = rotation

    def apply_scale(self) -> None:
                                                glScalef(self.scale[Transform._VECTOR3_X_INDEX],
self.scale[Transform._VECTOR3_Y_INDEX], self.scale[Transform._VECTOR3_Z_INDEX])

    def apply_position(self) -> None:
                                            glTranslatef(self.position[Transform._VECTOR3_X_INDEX],
self.position[Transform._VECTOR3_Y_INDEX],
self.position[Transform._VECTOR3_Z_INDEX])

    def apply_rotation(self) -> None:
        ANGLE_THRESHOLD = 360.0


glMultMatrixf(self._get_rotation_matrix_x(np.radians(self._rotation[Transform._VECTOR3_X_
INDEX] % ANGLE_THRESHOLD)))

glMultMatrixf(self._get_rotation_matrix_y(np.radians(self._rotation[Transform._VECTOR3_Y_
INDEX] % ANGLE_THRESHOLD)))

glMultMatrixf(self._get_rotation_matrix_z(np.radians(self._rotation[Transform._VECTOR3_Z_
INDEX] % ANGLE_THRESHOLD)))

    def _get_rotation_matrix_x(self, angle: float) -> np.array:
        return np.array([[1, 0, 0, 0],
                         [0, np.cos(angle), -np.sin(angle), 0],
                         [0, np.sin(angle), np.cos(angle), 0],
                         [0, 0, 0, 1]], dtype = np.float32)

    def _get_rotation_matrix_y(self, angle: float) -> np.array:
        return np.array([[np.cos(angle), 0, np.sin(angle), 0],
                         [0, 1, 0, 0],
                         [-np.sin(angle), 0, np.cos(angle), 0],
                         [0, 0, 0, 1]], dtype = np.float32)

    def _get_rotation_matrix_z(self, angle: float) -> np.array:
        return np.array([[np.cos(angle), -np.sin(angle), 0, 0],
                         [np.sin(angle), np.cos(angle), 0, 0],
                         [0, 0, 1, 0],
                         [0, 0, 0, 1]], dtype = np.float32)
```

```
class Mesh:

    def __init__(self, vertices: list[list[float]], faces: list[list[float]]) -> None:
        self._faces = faces
        self._vertices = vertices

    def draw_base(self) -> None:
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL)
        self._draw()

    def draw_outline(self) -> None:
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)
        self._draw()

    def _draw(self) -> None:
        for face in self._faces:
            glBegin(GL_POLYGON)

            for vertex in face:
                glVertex3fv(self._vertices[vertex - 1])

            glEnd()

class Material:

    _COLOR_RED_CHANNEL = 0
    _COLOR_GREEN_CHANNEL = 1
    _COLOR_BLUE_CHANNEL = 2
    _COLOR_ALPHA_CHANNEL = 3

    @property
    def base_color(self) -> list[float]:
        return self._base_color

    @base_color.setter
    def base_color(self, value: list[float]) -> None:
        self._base_color = value

    @property
    def outline_color(self) -> list[float]:
        return self._outline_color

    @outline_color.setter
    def outline_color(self, value: list[float]) -> None:
        self._outline_color = value

      def __init__(self, base_color: list[float], outline_color: list[float], outline_thickness: float =
1.0) -> None:
        self._base_color = base_color
        self._outline_color = outline_color
        self._outline_thickness = outline_thickness
```

```
    def apply_base_color(self) -> None:
        self._apply_color(self._base_color)

    def apply_outline_color(self) -> None:
        glLineWidth(self._outline_thickness)
        self._apply_color(self._outline_color)

    def _apply_color(self, color: list[float]) -> None:
                                                glColor4f(color[self._COLOR_RED_CHANNEL],
color[self._COLOR_GREEN_CHANNEL],           color[self._COLOR_BLUE_CHANNEL],
color[self._COLOR_ALPHA_CHANNEL])

class SceneObject:

    @property
    def transform(self) -> Transform:
        return self._transform

    @property
    def material(self) -> Material:
        return self._material

       def __init__(self, transform: Transform, mesh: Mesh, material: Material, update_delegate:
Callable[[ForwardRef("SceneObject"), float], None]) -> None:
        self._mesh = mesh
        self._material = material
        self._transform = transform
        self._update_delegate = update_delegate

    def render(self) -> None:
        glPushMatrix()

        self._transform.apply_position()
        self._transform.apply_rotation()
        self._transform.apply_scale()
        self._material.apply_base_color()
        self._mesh.draw_base()
        self._material.apply_outline_color()
        self._mesh.draw_outline()

        glFlush()

        glPopMatrix()

    def update(self, delta_time: float) -> None:
        self._update_delegate(self, delta_time)

class Application:

    _VIEWPORT_INITIAL_WIDTH = 1920
    _VIEWPORT_INITIAL_HEIGHT = 1080
```

```
    _VIEWPORT_OFFSET_X = 0
    _VIEWPORT_OFFSET_Y = 0

    _ANTI_ALIASING_SAMPLE_COUNT = 5

    _ISOMETRIC_PROJECTION_SCALER = 0.7
    _ISOMETRIC_PROJECTION_VIEWPORT_SCALE = 3.0
    _ISOMETRIC_PROJECTION_ANGLE = np.radians(-45.0)
    _ISOMETRIC_PROJECTION_FORESHORTENING = np.radians(35.264)
    _MIN_DELTA_TIME = 0.005

    def __init__(self) -> None:

        if not glfw.init():
            return

        glfw.window_hint(glfw.DOUBLEBUFFER, glfw.TRUE)
        glfw.window_hint(glfw.SAMPLES, self._ANTI_ALIASING_SAMPLE_COUNT)

                    self._window = glfw.create_window(self._VIEWPORT_INITIAL_WIDTH,
self._VIEWPORT_INITIAL_HEIGHT, "Лабораторна робота № 1. ІП-14 Бабіч Денис", None,
None)

        if not self._window:
            glfw.terminate()
            return

        glfw.make_context_current(self._window)
        glEnable(GL_DEPTH_TEST)
        glEnable(GL_MULTISAMPLE)
        self._start()

        while not glfw.window_should_close(self._window):
            self._update()
            glfw.poll_events()
        self._destroy()

    def _apply_isometric_projection(self, scale: float) -> None:
        glMatrixMode(GL_PROJECTION)
        glPushMatrix()
        glLoadIdentity()
        left = -scale
        right = scale
        bottom = -scale * self._ISOMETRIC_PROJECTION_SCALER
        top = scale * self._ISOMETRIC_PROJECTION_SCALER
        near = -scale
        far = scale
        glMultMatrixf( [2.0 / (right - left), 0, 0, -(right + left) / (right - left),
                    0, 2.0 / (top - bottom), 0, -(top + bottom) / (top - bottom),
                    0, 0, -2.0 / (far - near), -(far + near) / (far - near),
                    0, 0, 0, 1] )
```

```
        glMatrixMode(GL_MODELVIEW)

        glPushMatrix()
        glLoadIdentity()

        glMultTransposeMatrixf([[1, 0, 0, 0],
                                [0, np.cos(self._ISOMETRIC_PROJECTION_FORESHORTENING),
-np.sin(self._ISOMETRIC_PROJECTION_FORESHORTENING), 0],
                                [0, np.sin(self._ISOMETRIC_PROJECTION_FORESHORTENING),
np.cos(self._ISOMETRIC_PROJECTION_FORESHORTENING), 0],
                        [0, 0, 0, 1] ])

                glMultTransposeMatrixf([[np.cos(self._ISOMETRIC_PROJECTION_ANGLE),  0,
np.sin(self._ISOMETRIC_PROJECTION_ANGLE), 0],
                        [0, 1, 0, 0],
                                                [-np.sin(self._ISOMETRIC_PROJECTION_ANGLE),  0,
np.cos(self._ISOMETRIC_PROJECTION_ANGLE), 0],
                        [0, 0, 0, 1]])

    def _undo_isometric_projection(self) -> None:
        glMatrixMode(GL_MODELVIEW)
        glPopMatrix()

        glMatrixMode(GL_PROJECTION)
        glPopMatrix()

    def _start(self) -> None:
        self._game_objects = list()

        pyramid_transform = Transform([0.0, 0.5, 0.0], [0.0, 0.0, 0.0], [1.0, 1.0, 1.0])

        pyramid_vertices = [ [1, -1, -1],   # A
                        [1, -1, 1],    # B
                        [-1, -1, 1],   # C
                        [-1, -1, -1],  # D
                        [0, 1, 0] ]    # E (apex)

        pyramid_faces = [ [1, 2, 3, 4],  # Base (ABCD)
                    [1, 2, 5],     # Side (ABE)
                    [2, 3, 5],     # Side (BCE)
                    [3, 4, 5],     # Side (CDE)
                    [4, 1, 5] ]    # Side (DAE)

        pyramid_mesh = Mesh(pyramid_vertices, pyramid_faces)

        pyramid_material = Material([1.0, 1.0, 1.0, 1.0], [0.0, 0.0, 0.0, 1.0], 5)

                pyramid = SceneObject(pyramid_transform, pyramid_mesh, pyramid_material,
pyramid_update)

        self._game_objects.append(pyramid)
```

```
    def _update(self) -> None:
        width, height = glfw.get_framebuffer_size(self._window)

        glViewport(self._VIEWPORT_OFFSET_X, self._VIEWPORT_OFFSET_Y, width, height)

        start_time = glfw.get_time()

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

        glClearColor(0.0, 0.0, 0.0, 1.0)


self._apply_isometric_projection(self._ISOMETRIC_PROJECTION_VIEWPORT_SCALE)

        for game_object in self._game_objects:
            game_object.render()

        self._undo_isometric_projection()

        delta_time = (glfw.get_time() - start_time)

        if delta_time < Application._MIN_DELTA_TIME:
            delta_time = Application._MIN_DELTA_TIME

        for game_object in self._game_objects:
            game_object.update(delta_time)

        glfw.swap_buffers(self._window)

    def _destroy(self) -> None:
        glfw.terminate()

def pyramid_update(scene_object: SceneObject, delta_time: float) -> None:
    Y = 1
    COLOR_ALPHA_CHANNEL = 1.0
    COLOR_MAX_INTENSITY = 255

    ANGLE_ROTATION_SPEED = 120

    timestamp = datetime.now().timestamp()

    previous_rotation = scene_object.transform.rotation
     scene_object.transform.rotation = [0, previous_rotation[Y] + (ANGLE_ROTATION_SPEED
* delta_time), 0]

    COLOR_BASE_SHIFT_SPEED = 1

     gradient_black_white = np.interp((np.sin(timestamp * COLOR_BASE_SHIFT_SPEED)), [-1,
1], [0, 1])
              scene_object.material.base_color    =    [gradient_black_white,    gradient_black_white,
gradient_black_white, COLOR_ALPHA_CHANNEL]
```

```
    COLOR_OUTLINE_SHIFT_SPEED = 1

    COLOR_RED_CHANNEL_OFFSET = 0
    COLOR_GREEN_CHANNEL_OFFSET = 2 * np.pi / 3
    COLOR_BLUE_CHANNEL_OFFSET = 4 * np.pi / 3

     color_red_channel = int(np.interp(np.sin((timestamp * COLOR_OUTLINE_SHIFT_SPEED) + COLOR_RED_CHANNEL_OFFSET), [-1, 1], [0, 255])) / COLOR_MAX_INTENSITY
                        color_green_channel = int(np.interp(np.sin((timestamp * COLOR_OUTLINE_SHIFT_SPEED) + COLOR_GREEN_CHANNEL_OFFSET), [-1, 1], [0, 255])) / COLOR_MAX_INTENSITY
    color_blue_channel = int(np.interp(np.sin((timestamp * COLOR_OUTLINE_SHIFT_SPEED) + COLOR_BLUE_CHANNEL_OFFSET), [-1, 1], [0, 255])) / COLOR_MAX_INTENSITY

            scene_object.material.outline_color = [color_red_channel, color_green_channel, color_blue_channel, COLOR_ALPHA_CHANNEL]

def main() -> None:
    instance = Application()

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

IV. Висновки.

У цій лабораторній роботі було розроблено програмне забезпечення, яке використовує OpenGL для створення віртуальної 3D сцени та слугує демонстрацією застосування мови Python для створення подібних базових застосунків. Було створено класи SceneObject та Transform для інкапсуляції логіки представлення об’єктів у 3D просторі, включаючи позицію, розміри та обертання. Клас Material використовується для відображення об’єктів за допомогою списку вершин та поверхонь. Було використано ізометричну проекцію для аксонометричного відображення об’єктів. Крім того, було впроваджено концепцію дельта-часу для збереження змін, які відбуваються зі зміною часу. Загалом, ця робота демонструє ефективне використання OpenGL та об’єктно-орієнтованого програмування для створення віртуальних 3D сцен.

Виконав: ІП-14 Бабіч Д. В.