**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**

1. **Мета:**

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

1. **Завдання:**

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 |

1. **Результати виконання лабораторної роботи.**
2. **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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1  0  0 | 0  cos(θ)  sin(θ) | 0  -sin(θ)  cos(θ) |  |

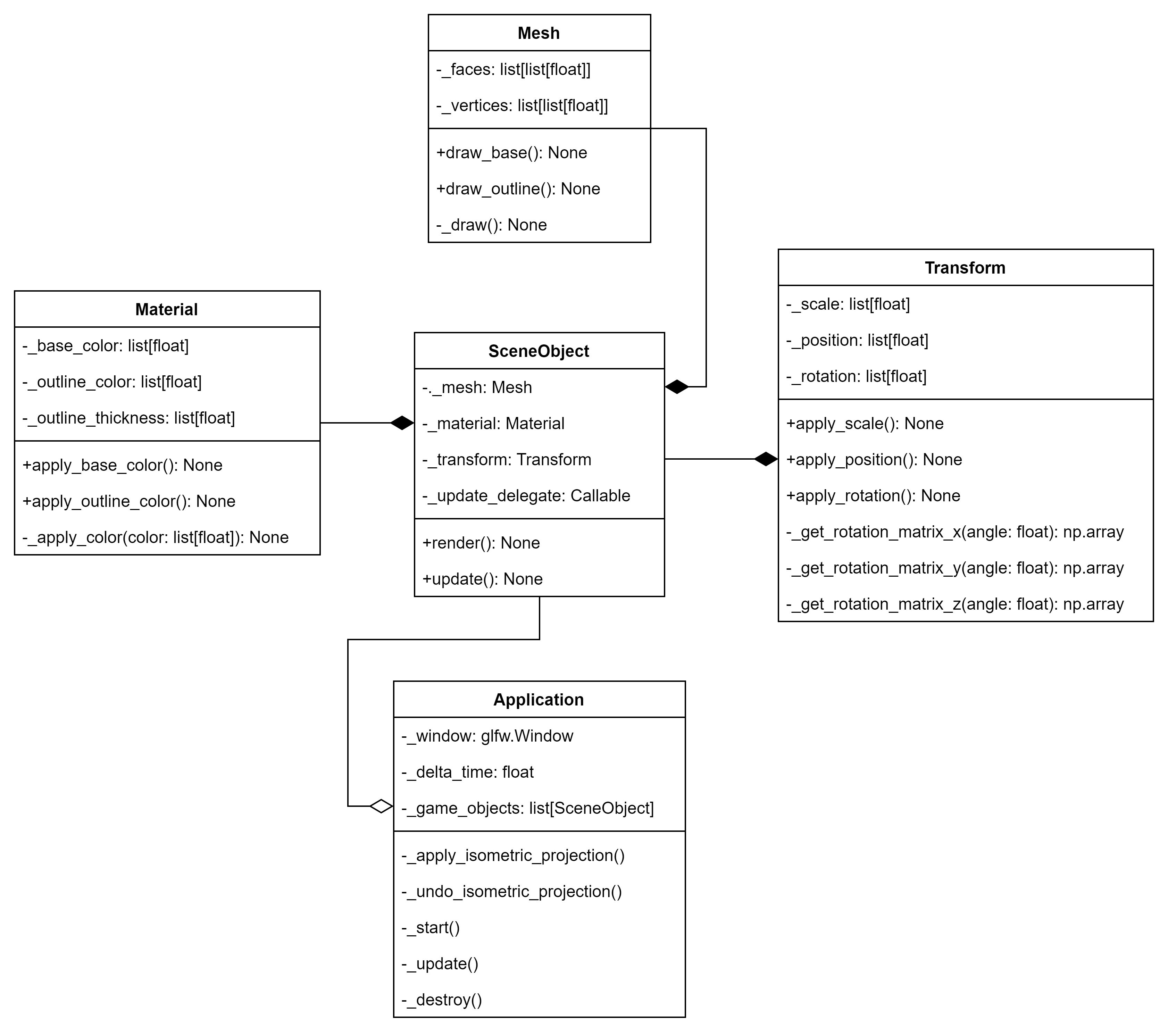
Матриця обертання, довкола осі Y, де θ – заданий кут обертання.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | cos(θ)  0  -sin(θ) | 0  1  0 | sin(θ)  0  cos(θ) |  |

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | cos(θ)  sin(θ)  0 | -sin(θ)  cos(θ)  0 | 0  0  1 |  |

1. **Результати архітектурного проектування та їх опис.**

Рисунок 1.1 – UML-діаграма класів

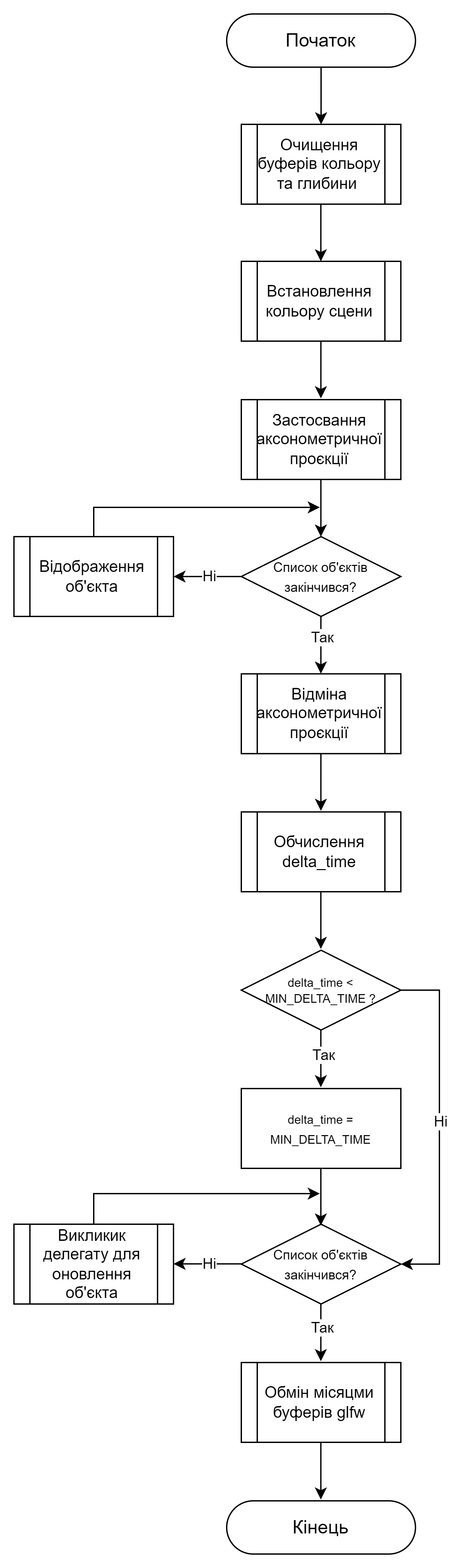


Рисунок 1.2 – Діаграма головного рантайм-циклу застосунку

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°.

1. **Опис структури проекту програми.**

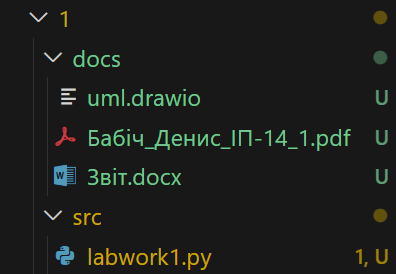


Рисунок 1.3 – Структура проєкту

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

1. **Результати роботи програми відповідно до завдання.**

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

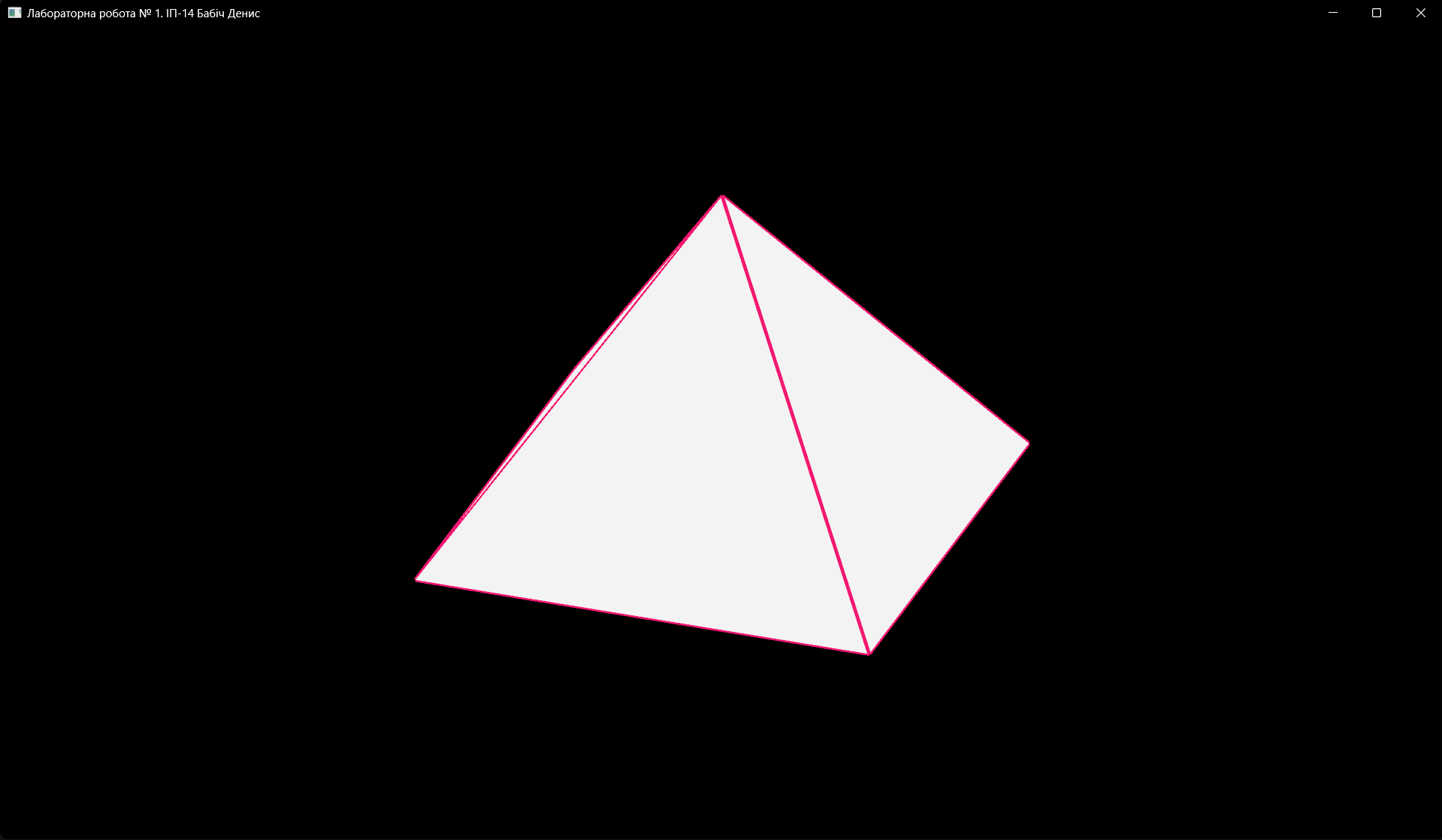


Рисунок 1.4 – Приклад виконання роботи

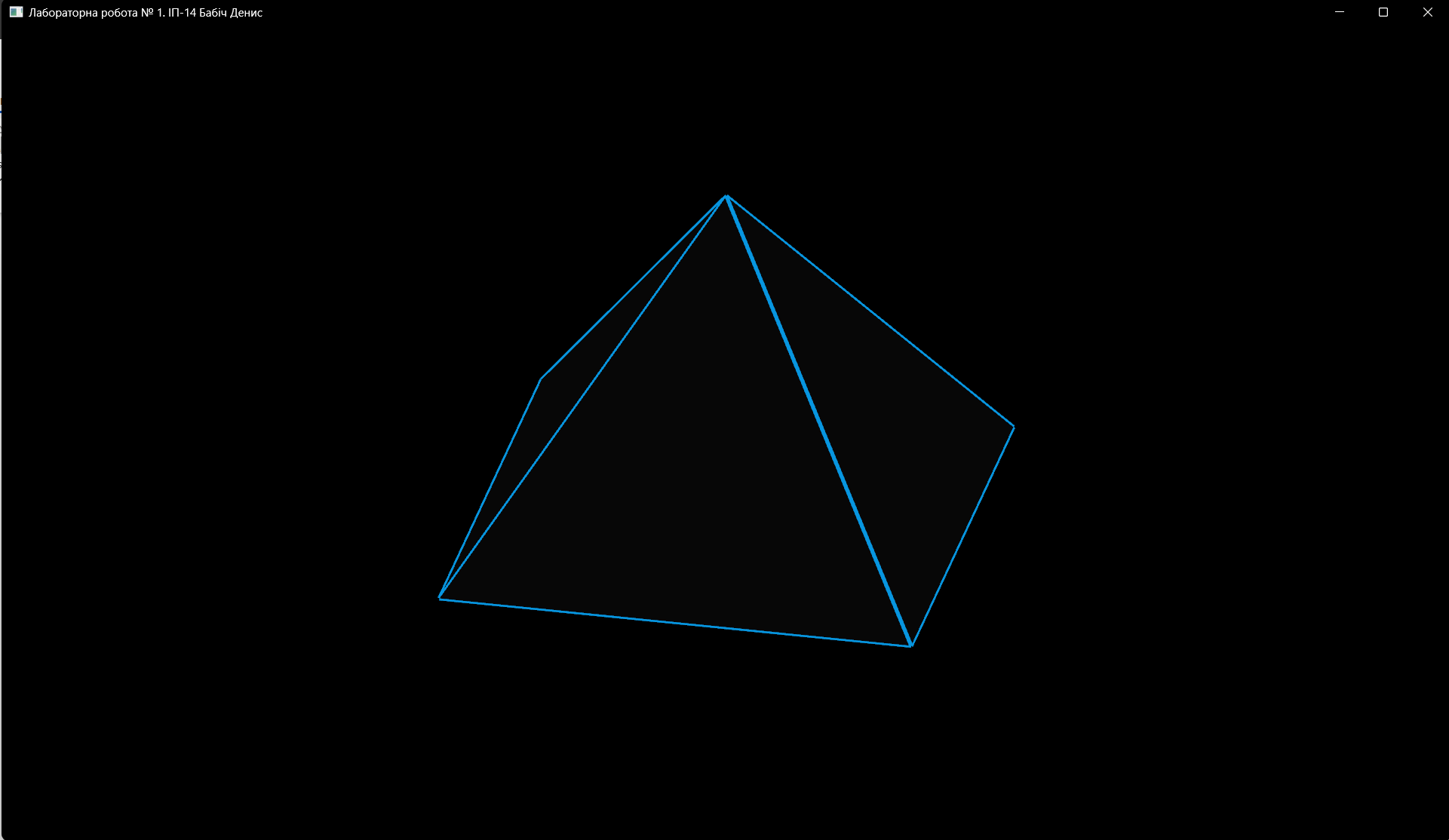


Рисунок 1.5 – Приклад виконання роботи

1. **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()

1. **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.