НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ Кафедра інформатики та програмної інженерії

ДИСЦИПЛІНА

«КОМП’ЮТЕРНА ГРАФІКА ТА ОБРОБКА ЗОБРАЖЕНЬ»

ЗВІТ

З лабораторної роботи №4

Тема: «2D-текстури. Підготовка та завантаження зображень. Текстурні координати, текстурна матриця»

Роботу виконала студентка

Групи ІС – 34

Ященко Олександра Олегівна

Викладач: к.т.н., с.н.с.

Щебланін Юрій Миколайович

Київ 2025

ВСТУП

Текстури є одним з ключових елементів сучасної комп'ютерної графіки, що дозволяють надавати тривимірним об'єктам реалістичний вигляд без значного збільшення геометричної складності моделей. Замість створення детальної геометрії для кожної поверхні, текстури дозволяють "наклеїти" растрові зображення на полігональні моделі, імітуючи складні матеріали, візерунки та деталі.

Бібліотека OpenGL надає потужні засоби для роботи з текстурами, підтримуючи різні типи текстурних об'єктів: одновимірні, двовимірні, тривимірні та кубічні. Двовимірні текстури є найпоширенішими і використовуються для накладання зображень на плоскі та об'ємні поверхні. Процес текстурування включає підготовку зображення, його завантаження у видеопам'ять, налаштування параметрів фільтрації та повторення, а також встановлення відповідності між точками текстури і вершинами геометричних примітивів.

Текстурні координати визначають, яка частина зображення відповідає кожній вершині об'єкта. Використання текстурної матриці дозволяє виконувати трансформації текстури (масштабування, обертання, зміщення) без зміни геометрії об'єкта, що забезпечує гнучкість у створенні візуальних ефектів.

Окрім завантаження готових зображень з файлів, існує можливість процедурної генерації текстур безпосередньо у програмі на основі математичних формул. Це дозволяє створювати різноманітні візерунки та патерни без необхідності зберігання великих графічних файлів.

У цій лабораторній роботі досліджуються основні принципи роботи з 2D-текстурами в OpenGL: підготовка текстурних зображень у потрібному форматі, завантаження їх з файлів, створення процедурних текстур, налаштування параметрів текстурування та використання текстурної матриці для контролю відображення текстури на поверхнях тривимірних об'єктів.

ЗМІСТ

ВСТУП

ОСНОВНА ЧАСТИНА

Мета лабораторної роботи

1. Теоретичні відомості

2. Завдання 3.1. Підготовка текстури

3. Завдання 3.2. Текстурований куб

4. Завдання 3.3. Шахівниця на кубі

5. Завдання 3.4. Візерунок

ВИСНОВКИ

ПЕРЕЛІК ПОСИЛАНЬ

ОСНОВНА ЧАСТИНА

**Мета лабораторної роботи:**

Вивчити способи роботи з 2D-текстурами в OpenGL: завантаження текстур з файлів, створення процедурних текстур, налаштування параметрів текстурування, використання текстурної матриці для масштабування.

**1. Теоретичні відомості**

Текстура – це растрове зображення, яке накладається на поверхню тривимірного об'єкта для підвищення візуальної деталізації. Кожен піксель текстури називається текселем (texel). OpenGL підтримує різні типи текстур: одновимірні (1D), двовимірні (2D), тривимірні (3D) та кубічні (cube maps).

Текстурні координати (s, t) визначають відповідність між точками текстури та вершинами об'єкта. Вони задаються в нормалізованому діапазоні від 0.0 до 1.0, де (0.0, 0.0) відповідає нижньому лівому куту текстури, а (1.0, 1.0) – верхньому правому.

Основні параметри текстурування включають режими фільтрації та повторення. Фільтрація визначає, як обчислюється колір при масштабуванні текстури: GL\_NEAREST використовує найближчий тексел (швидко, але з артефактами), GL\_LINEAR виконує інтерполяцію між сусідніми текселями (повільніше, але згладжено). Режими повторення контролюють поведінку текстури за межами координат [0,1]: GL\_REPEAT повторює текстуру, GL\_CLAMP продовжує крайні пікселі.

Текстурна матриця – це окрема матриця трансформації в OpenGL, яка множиться на текстурні координати перед їх використанням. Це дозволяє виконувати масштабування, обертання та зміщення текстури без зміни координат вершин об'єкта, забезпечуючи гнучкість у створенні візуальних ефектів.

**2. Завдання 3.1. Підготовка текстури**

Завдання полягає у підготовці та збереженні зображення текстури у форматі BMP з розмірами 512×512 пікселів та глибиною кольору 24 біти.

Розміри текстури обрано як степінь двійки (512 = 2⁹), що є оптимальним для OpenGL. Для створення тестових текстур використано бібліотеку PIL (Pillow), яка дозволяє програмно генерувати зображення.

Створення тестової текстури з кольоровими квадрантами виконується наступним чином:

from PIL import Image, ImageDraw, ImageFont

import numpy as np

def create\_test\_texture():

img = Image.new('RGB', (512, 512), color=(200, 200, 200))

draw = ImageDraw.Draw(img)

colors = [

(255, 220, 180),

(180, 50, 50),

(200, 100, 100),

(255, 240, 200),

]

draw.rectangle([0, 0, 255, 255], fill=colors[0])

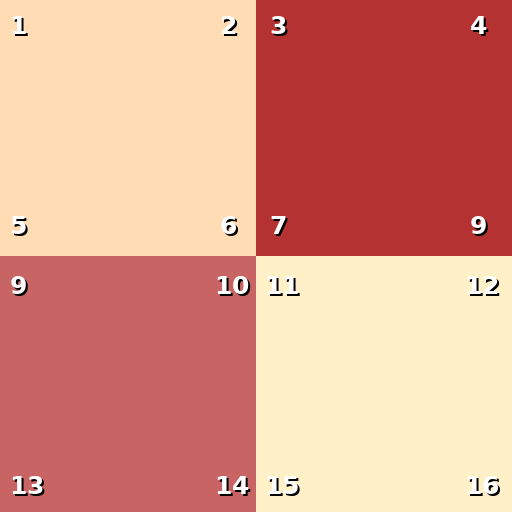
draw.rectangle([256, 0, 511, 255], fill=colors[1])

draw.rectangle([0, 256, 255, 511], fill=colors[2])

draw.rectangle([256, 256, 511, 511], fill=colors[3])

img.save('texture.bmp')

Текстура розділена на чотири квадранти різних кольорів, що дозволяє візуально контролювати правильність накладання текстури на грані куба. Додатково на текстуру можуть бути нанесені цифрові мітки для ідентифікації орієнтації кожної грані.



**3. Завдання 3.2. Текстурований куб**

Завдання полягає у доповненні проекту процедурою завантаження текстури з файлу та накладанні текстури на всі грані куба.

Завантаження текстури виконується за допомогою бібліотеки PIL, яка читає BMP-файл, змінює розмір до потрібних параметрів, конвертує у формат RGB та перетворює в масив NumPy:

def load\_bmp\_texture(self, filename):

try:

img = Image.open(filename)

img = img.resize((self.TW, self.TH))

img = img.convert('RGB')

img\_data = np.array(img, dtype=np.uint8)

self.arrayRGB = np.flipud(img\_data)

return True

Функція flipud() виконує вертикальне відображення масиву, оскільки PIL використовує систему координат з початком у верхньому лівому куті, тоді як OpenGL – у нижньому лівому.

Налаштування параметрів текстури в OpenGL включає створення текстурного об'єкта, завантаження даних у відеопам'ять та встановлення режимів фільтрації і повторення:

def setup\_texture(self):

glEnable(GL\_TEXTURE\_2D)

if self.texture\_id is not None:

glDeleteTextures([self.texture\_id])

self.texture\_id = glGenTextures(1)

glBindTexture(GL\_TEXTURE\_2D, self.texture\_id)

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, self.TW, self.TH, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, self.arrayRGB)

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR)

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR)

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT)

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT)

glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_DECAL)

glDisable(GL\_TEXTURE\_2D)

Функція glGenTextures() створює унікальний ідентифікатор текстури, glBindTexture() робить його активним для подальших операцій, glTexImage2D() завантажує дані зображення у текстурну пам'ять відеокарти. Параметр GL\_DECAL вказує, що використовується тільки колір текстури без змішування з власним кольором об'єкта.

Для накладання текстури на куб кожній вершині призначаються текстурні координати за допомогою glTexCoord2f() перед викликом glVertex3f():

glBegin(GL\_QUADS)

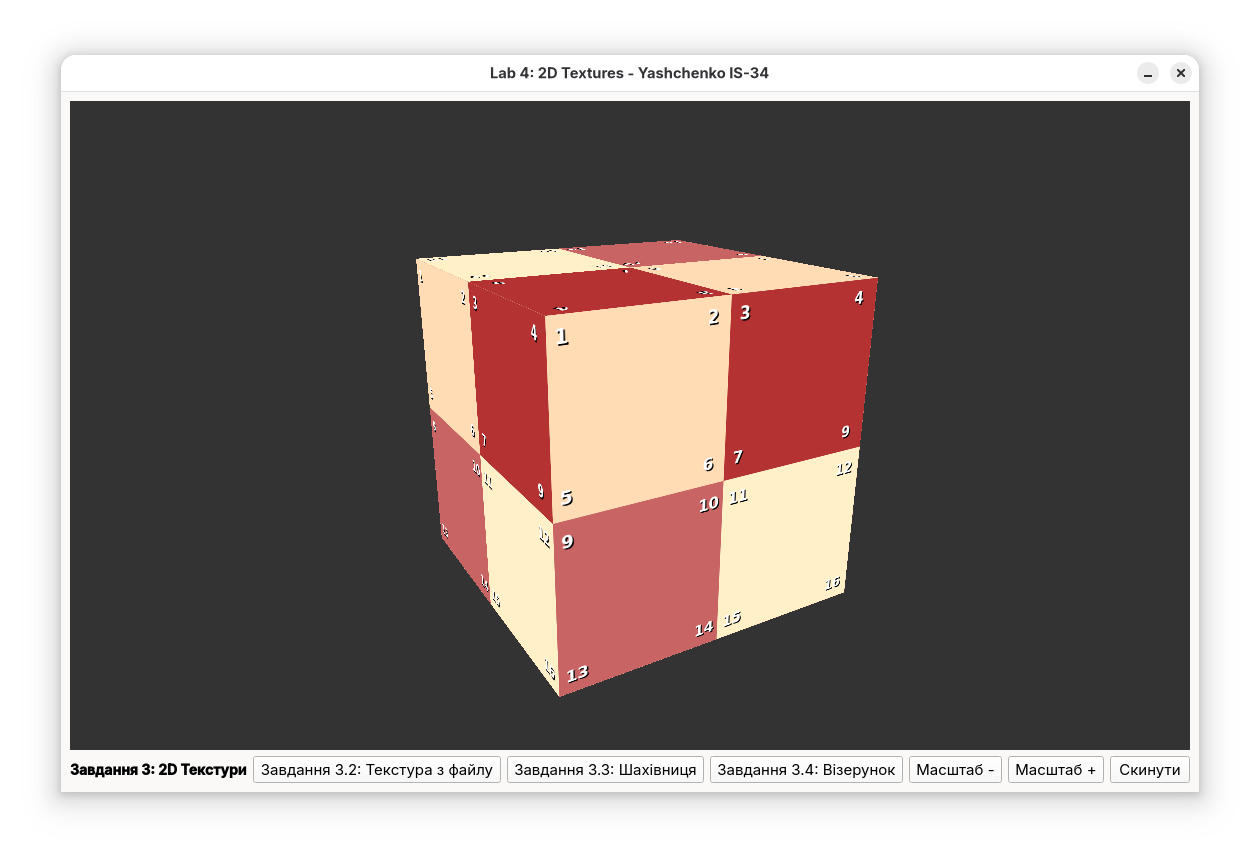
glTexCoord2f(1, 1); glVertex3f( h, h, h)

glTexCoord2f(0, 1); glVertex3f(-h, h, h)

glTexCoord2f(0, 0); glVertex3f(-h, -h, h)

glTexCoord2f(1, 0); glVertex3f( h, -h, h)

Аналогічно задаються текстурні координати для всіх шести граней куба, забезпечуючи правильне відображення текстури на кожній поверхні.

**4. Завдання 3.3. Шахівниця на кубі**

Завдання полягає у створенні текстури шахівниці з двох матеріалів (світлий та темний горіх) та відображенні на гранях куба з різним масштабуванням.

Для імітації текстури деревини обрано кольори: світлий горіх RGB(210, 180, 140) – теплий бежевий відтінок, темний горіх RGB(101, 67, 33) – насичений коричневий колір.

Процедурна генерація шахівниці виконується шляхом розбиття текстури на чотири квадранти, де діагонально протилежні квадранти мають однаковий колір:

def calculate\_checkerboard\_texture(self):

light\_color = np.array([210, 180, 140], dtype=np.uint8)

dark\_color = np.array([101, 67, 33], dtype=np.uint8)

for i in range(self.TW):

for j in range(self.TH):

if (i < self.TW // 2 and j < self.TH // 2) or (i >= self.TW // 2 and j >= self.TH // 2):

self.arrayRGB[j, i] = dark\_color

else:

self.arrayRGB[j, i] = light\_color

Умова перевіряє положення пікселя: якщо він знаходиться в лівому нижньому або правому верхньому квадранті, присвоюється темний колір, інакше – світлий. Це створює базовий патерн шахівниці 2×2.

Для повторення шахівниці на гранях куба використовується масштабування текстурної матриці:

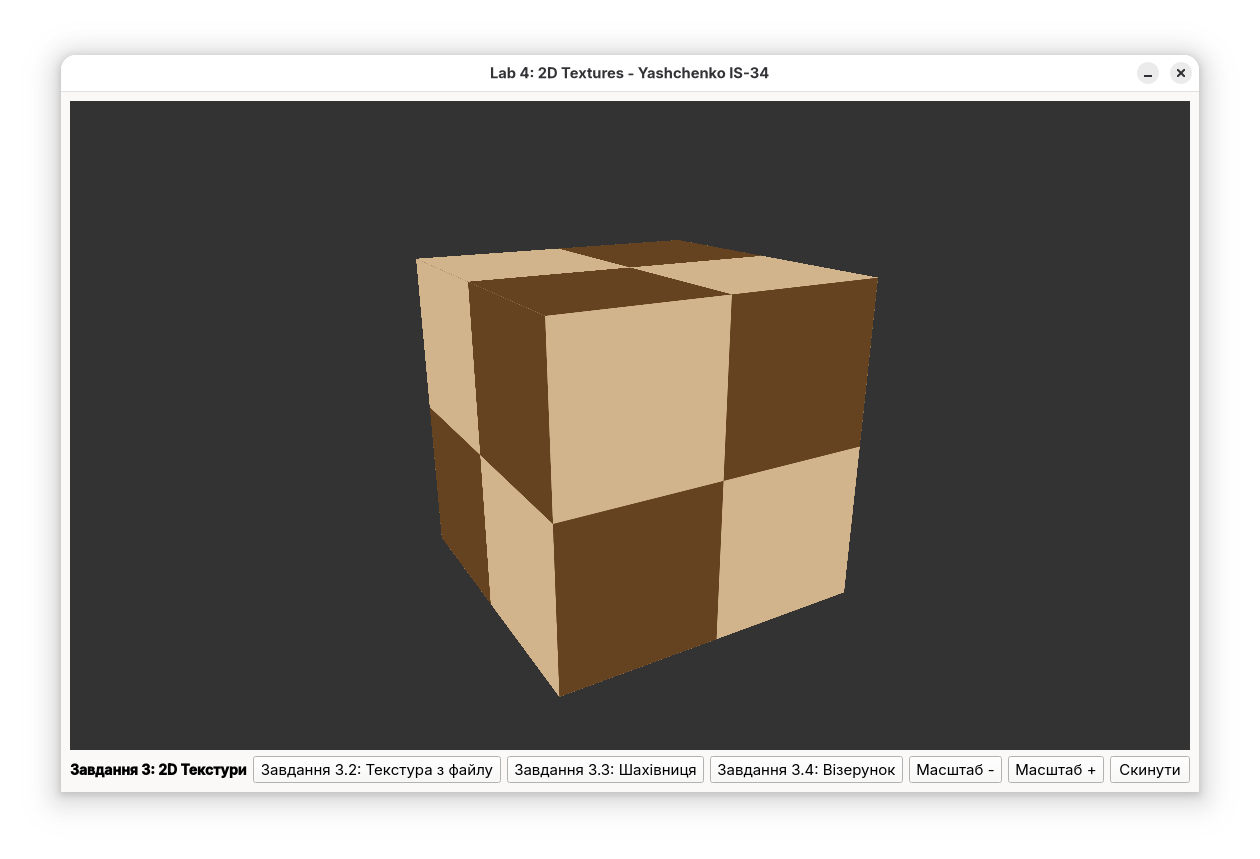
glMatrixMode(GL\_TEXTURE)

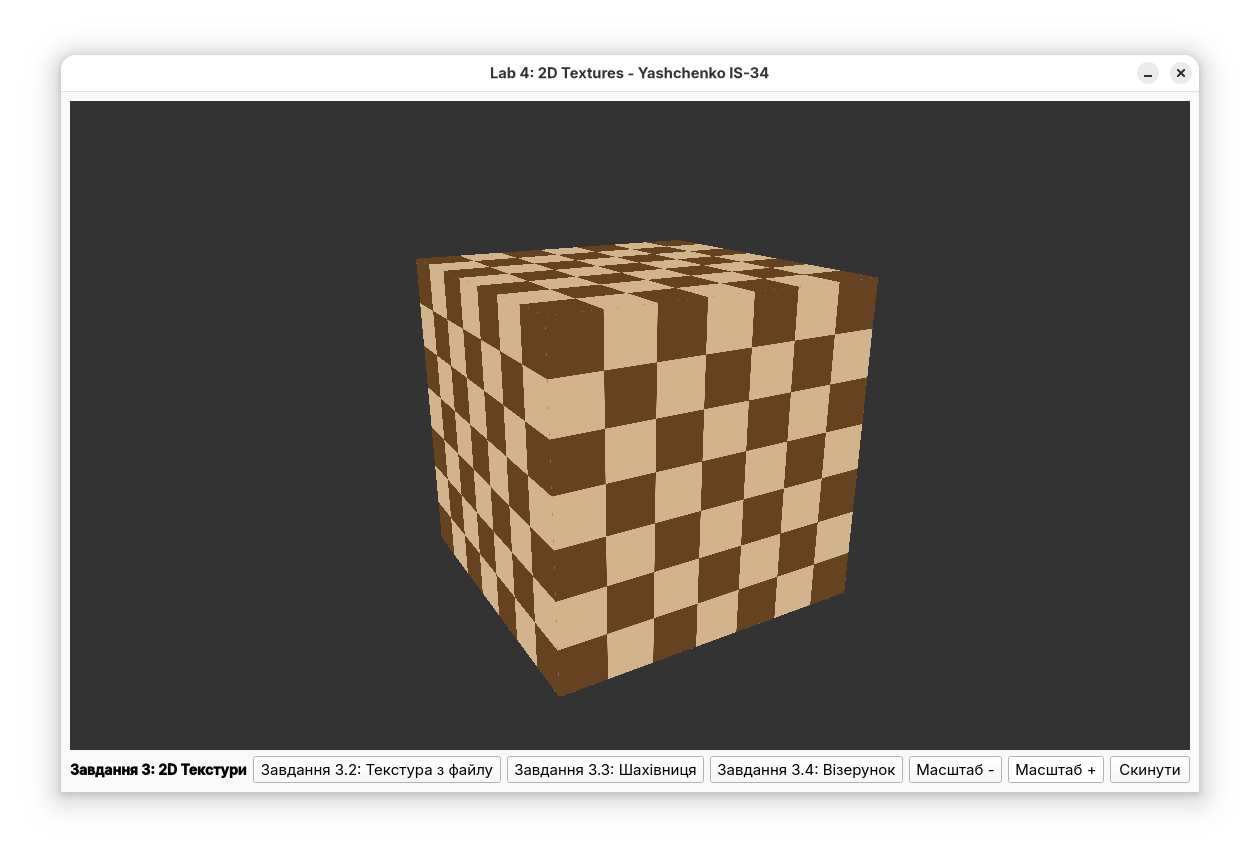
glLoadIdentity()

glScalef(self.mult, self.mult, self.mult)

glMatrixMode(GL\_MODELVIEW)

Параметр mult визначає коефіцієнт масштабування: при mult = 1.0 на кожній грані відображається одна шахівниця 2×2, при mult = 2.0 – чотири шахівниці (утворюючи сітку 4×4 клітинки), при mult = 4.0 – шістнадцять шахівниць (сітка 8×8 клітинок). Режим GL\_REPEAT забезпечує безшовне повторення текстури.





**5. Завдання 3.4. Візерунок**

Завдання полягає у створенні власного візерунку на основі математичних формул.

Обрано спіральний візерунок з радіальними секторами, який генерується на основі полярних координат та тригонометричних функцій. Спочатку декартові координати пікселів перетворюються в нормалізовану форму:

x = (i - self.TW // 2) / self.TW \* 8

y = (j - self.TH // 2) / self.TH \* 8

Множення на 8 визначає масштаб візерунку. Далі обчислюються полярні координати:

r = math.sqrt(x \* x + y \* y)

fi = math.atan(y / x) if x != 0.0 else math.pi / 2

Радіус r визначає відстань від центру текстури, кут φ – напрямок. Формула візерунку комбінує кутову та радіальну компоненти:

value = abs(math.cos(8 \* fi - r))

Множник 8 перед кутом φ визначає кількість радіальних секторів (променів), а віднімання r створює ефект спіралі, яка закручується від центру до країв. Абсолютне значення косинуса забезпечує симетричність візерунку.

На основі обчисленого значення вибирається колір пікселя:

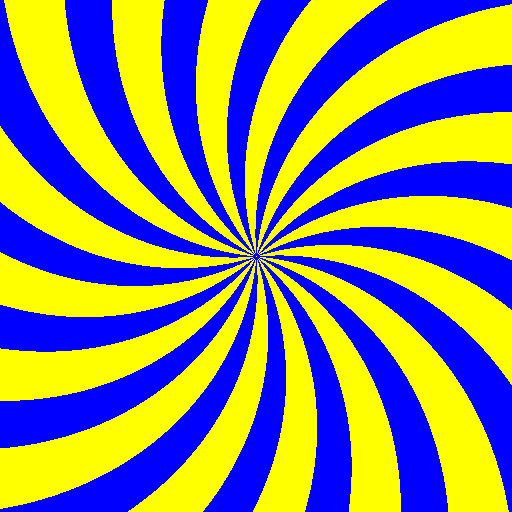
if (value % 1.0) < 0.75:

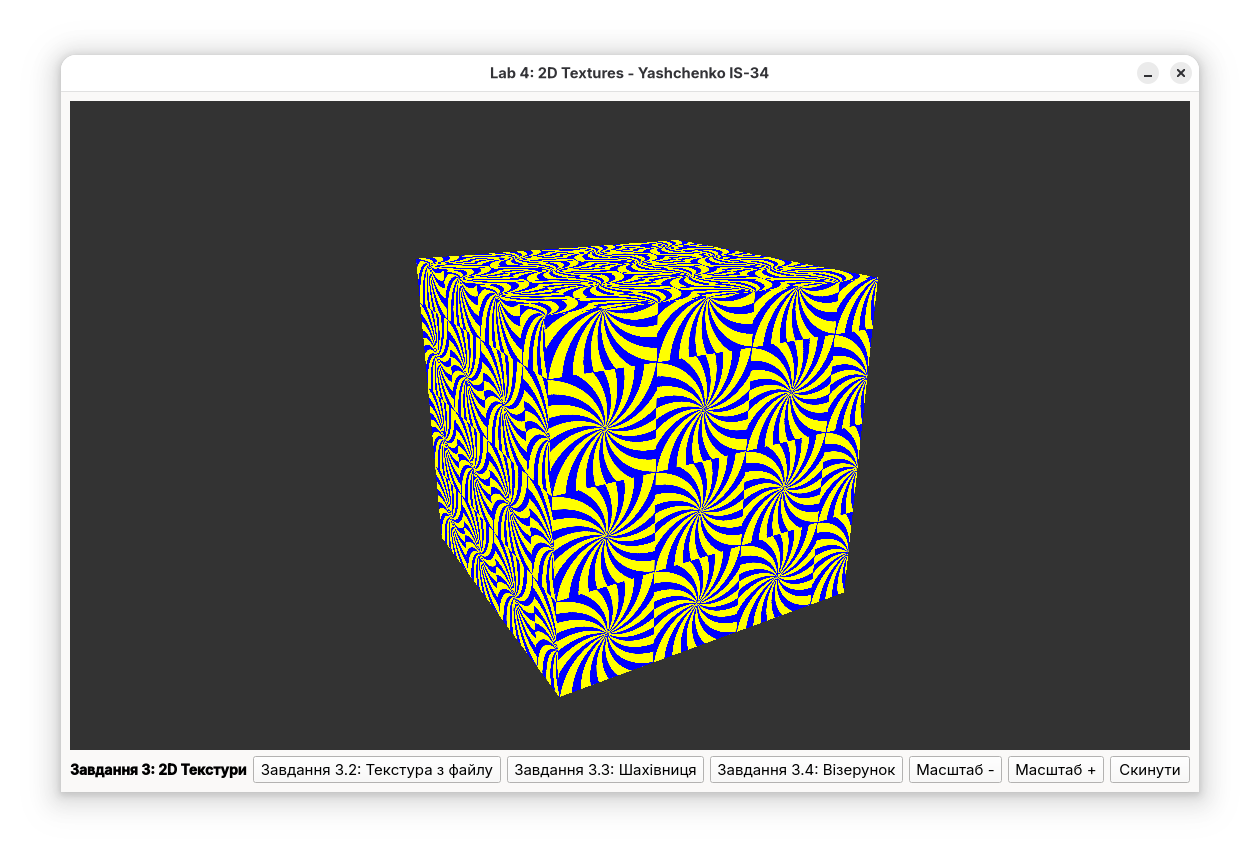
self.arrayRGB[j, i] = [255, 255, 0]

else:

self.arrayRGB[j, i] = [0, 0, 255]

Поріг 0.75 визначає співвідношення площ жовтих та синіх областей. Жовтий колір формує основні спіральні промені, синій заповнює проміжки між ними, створюючи контрастний візерунок.





ВИСНОВКИ

У ході виконання лабораторної роботи №4 було успішно вивчено основні принципи роботи з 2D-текстурами в бібліотеці OpenGL.

Освоєно процес завантаження текстур з файлів формату BMP, що включає читання зображення за допомогою бібліотеки PIL, конвертацію системи координат, передачу даних у текстурну пам'ять відеокарти через функцію glTexImage2D(). Реалізовано коректне налаштування параметрів текстурування: режимів фільтрації (GL\_LINEAR для згладженого відображення), режимів повторення (GL\_REPEAT для безшовного тайлінгу) та режимів змішування (GL\_DECAL для використання тільки кольору текстури).

Вивчено принципи зв'язування текстурних координат з вершинами тривимірних об'єктів. Для кожної вершини куба задаються координати у діапазоні [0,1], що визначають відповідність між точками текстури та точками поверхні. Правильне призначення координат забезпечує коректне відображення текстури на всіх шести гранях куба.

Реалізовано процедурну генерацію текстур без використання зовнішніх графічних файлів. Створено два типи процедурних текстур: шахівницю на основі логічних умов розбиття площини та спіральний візерунок на основі полярних координат і тригонометричних функцій. Процедурна генерація дозволяє легко змінювати параметри візерунків та економить пам'ять.

Досліджено використання текстурної матриці для трансформації текстурних координат. Масштабування через glScalef() дозволяє контролювати кількість повторень текстури на поверхні без зміни геометрії об'єкта. Продемонстровано, що зміна параметра масштабування від 1.0 до 4.0 збільшує кількість повторень шахівниці відповідно від однієї до шістнадцяти.

Створено повнофункціональний графічний додаток на основі PyQt6 з інтерактивним інтерфейсом, який дозволяє перемикатися між різними режимами текстурування, змінювати масштаб та обертати об'єкт за допомогою миші. Отримані практичні навички є фундаментальними для роботи в галузі комп'ютерної графіки та використовуються в розробці ігор, системах 3D-візуалізації, архітектурному моделюванні.

ПЕРЕЛІК ПОСИЛАНЬ

1. OpenGL Programming Guide: The Official Guide to Learning OpenGL / D. Shreiner, G. Sellers, J. Kessenich, B. Licea-Kane. – 9th ed. – Addison-Wesley Professional, 2017. – 984 p.

2. OpenGL Reference Manual [Електронний ресурс]. – Режим доступу: <https://www.opengl.org/sdk/docs/tutorials/>

3. PyOpenGL Documentation [Електронний ресурс]. – Режим доступу: <http://pyopengl.sourceforge.net/documentation/index.html>

4. Qt for Python Documentation [Електронний ресурс]. – Режим доступу: <https://doc.qt.io/qtforpython/>

5. Pillow Documentation [Електронний ресурс]. – Режим доступу: <https://pillow.readthedocs.io/>