**Міністерство освіти і науки України**

**Національний технічний університет України «КПІ» імені Ігоря Сікорського**

**Кафедра обчислювальної техніки ФІОТ**

**ЗВІТ**

**з лабораторної роботи №2**

**з навчальної дисципліни «Computer Vision»**

**Тема:**

**ДОСЛІДЖЕННЯ АЛГОРИТМІВ ФОРМУВАННЯ ТА ОБРОБКИ РАСТРОВИХ ЦИФРОВИХ ЗОБРАЖЕНЬ**

**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 реалізації алгоритмів растрової цифрових зображень на прикладі застосування алгоритмів растеризації, побудови складних 3D растрових об’єктів та застосування технологій корекції характеристик кольору окремих растрів цифрових зображень.

1. **Завдання:**

**Завдання ІІІ рівня.**

Реалізувати розробку програмного скрипта, що реалізує корекцію кольору цифрового растрового зображення з переліку: зміна яскравості, відтінки сірого, негатив, сепія – в градієнтах: діагональ (будь-який напрям); від центру; до центра. Обробку реалізувати на рівні матриці растра. Зображення обрати самостійно.

1. **Результати виконання лабораторної роботи.**
2. **Синтезована математична модель перетворень матриці растрового зображення.**
   1. **Зміна яскравості**

Процес зміни рівня яскравості зображення відбувається за рахунок множення значення кольору кожного пікселя растру на множник, який є коефіцієнтом значення яскравості. Також, це значення нормалізується між 0 та 255 (відповідно мінімально можливе значення каналу кольору та максимальне).

Де a – фактор зміни яскравості, r – канал червоного кольору, g – канал зеленого кольору, b r – канал синього кольору, 0 – мінімально допустиме значення, 255 – максимально допустиме, clip – функція нормалізації значення у заданому проміжку.

* 1. **Відтінки сірого**

Для кожного пікселя отримуються значення червоного, зеленого та синього кольорів, потім їх значення замінюються на середнє всіх трьох колірних каналів. Це результат використовується відразу для всіх трьох каналів, щоб кожен піксель мав однакове значення кольору у відтінках сірого.

* 1. **Негатив**

Ефект негативу досягається шляхом віднімання від максимально можливого значення кольору на каналі (255) поточного значення.

Де a – фактор зміни яскравості, r – канал червоного кольору, g – канал зеленого кольору, b r – канал синього кольору, 0 – мінімально допустиме значення, 255 – максимально допустиме, clip – функція нормалізації значення у заданому проміжку.

* 1. **Сепія**

Процес застосування ефекту сепії до зображення полягає у використанні спеціальної матриці фільтрації, яка змінює кольори зображення, надаючи йому відтінки сепії. Вибір типу сепії впливає на характер отриманих кольорів. Після отримання обробленого значення кольору пікселя, воно нормалізується до діапазону від 0 до 255, щоб забезпечити коректність кольорів.

Де *pixel* – старе значення кольору пікселя, *pixel’* – нове, – транспонована колірна матриця. Матриці фільтрації для ефекту сепії мають конкретні значення, оскільки вони відображають кольорові тони, які характерні для цього ефекту. Ці значення вибираються таким чином, щоб змінити кольори зображення, надаючи їм характерні для сепії відтінки.

* 1. **Шум**

Процес застосування шуму до зображення полягає в додаванні випадкового шуму до кольорів пікселів. Цей шум генерується випадковим чином з урахуванням заданої інтенсивності. Після додавання шуму до значень кольорів пікселів, вони нормалізуються до діапазону від 0 до 255, щоб забезпечити коректність кольорів у зображенні.

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

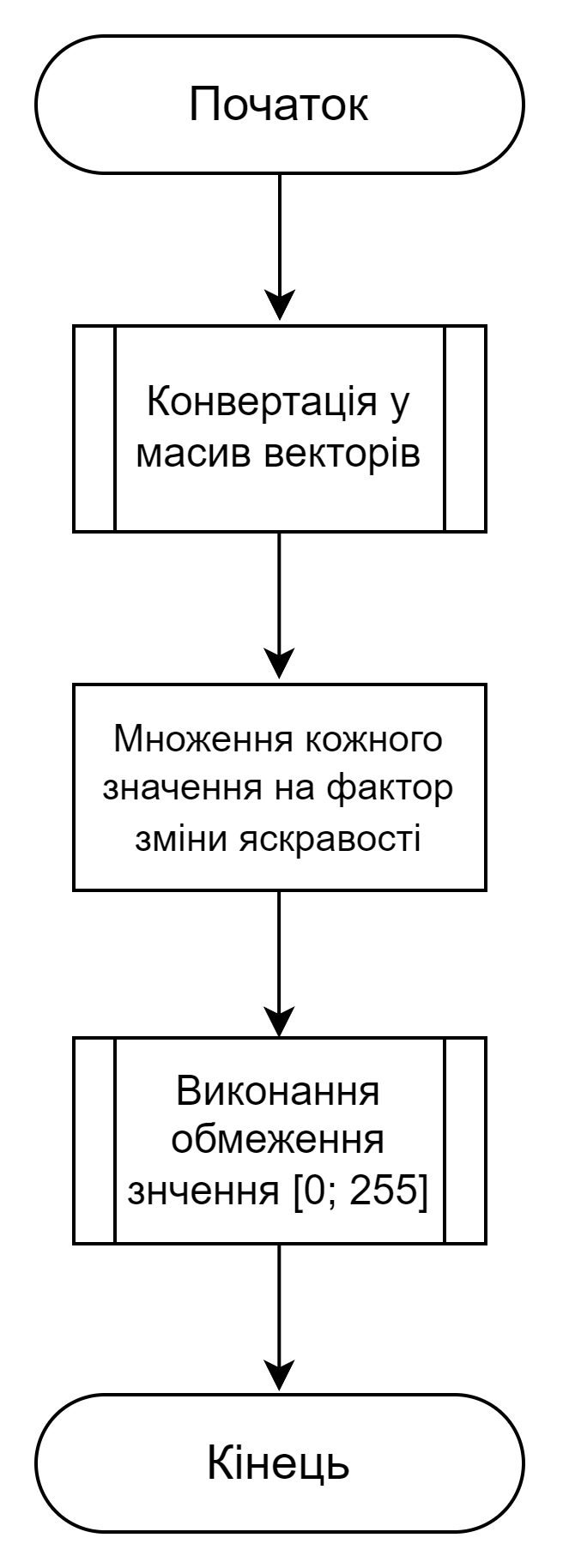


Рисунок 1.1 – Діаграма виконання зміни яскравості

1. Конвертація у масив. Для того щоб математично обробити зображення, спочатку виконується конвертація у формат, з яким можна зручно виконувати операції чисел з векторами.
2. Кожен піксель масиву зображення множиться на коефіцієнт factor, щоб змінити його яскравість. Це дозволяє збільшити або зменшити яскравість зображення в залежності від значення factor.
3. Після зміни яскравості можуть з'явитися значення пікселів, які виходять за межі діапазону [0, 255]. Саме з цією метою використовується для обмеження цих значень, щоб вони залишилися в допустимому діапазоні.

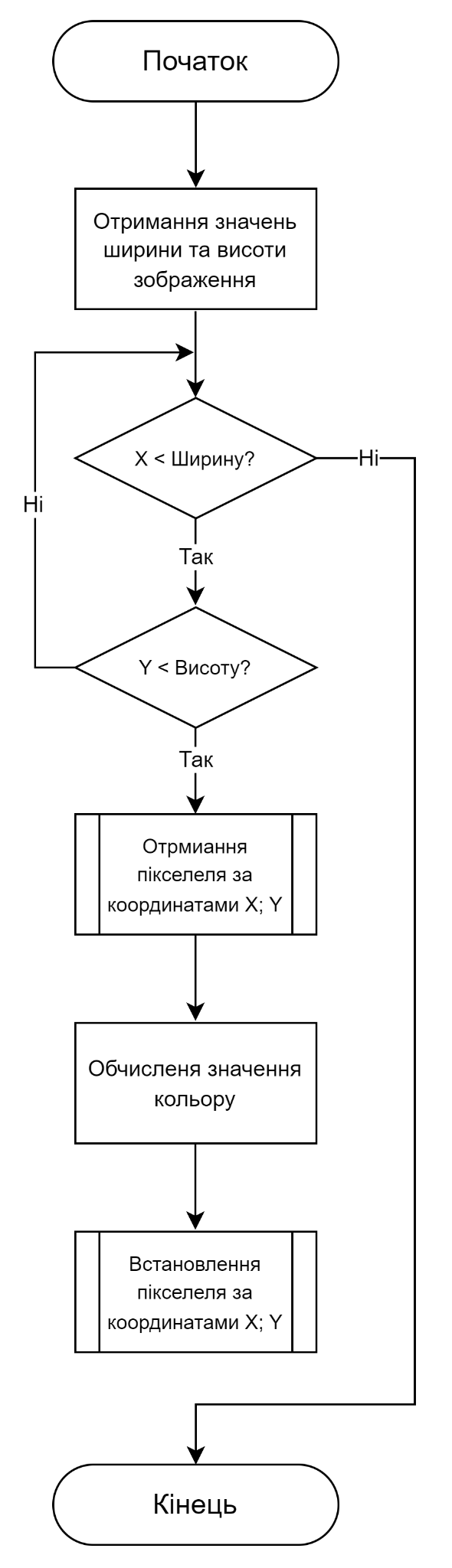


Рисунок 1.2 – Діаграма встановлення відтінків сірого

1. Отримання розмірів та створення копії зображення.
2. За допомогою вкладених циклів обробляється кожен піксель у зображенні.
3. Для кожного пікселя отримуються значення червоного, зеленого та синього каналів.
4. Значення кожного каналу комбінуються в середнє значення, щоб отримати відтінок сірого.
5. Оновлені значення пікселя встановлюються у скопійованому зображенні.



Рисунок 1.3 – Діаграма додавання ефекту негативу

1. Конвертація у масив. Для того щоб математично обробити зображення, спочатку виконується конвертація у формат, з яким можна зручно виконувати операції чисел з векторами.
2. Обчислення нового кольору пікселя.



Рисунок 1.4 – Діаграма додавання ефекту сепії

1. Конвертація зображення у вектор.
2. Зображення трансформується за допомогою множення матриці тону сепії на значення пікселів зображення.
3. Отримані значення пікселів обмежуються мінімальним та максимальним значеннями каналів кольору.



Рисунок 1.5 – Діаграма додавання ефекту шуму

1. Конвертація зображення у вектор.
2. Генерується шум за допомогою випадкових значень, які рівномірно розподілені від від від’ємного значення інтенсивності і до додатного.
3. Застосування шуму.
4. Отримані значення пікселів обмежуються мінімальним та максимальним значеннями каналів кольору.
5. **Опис структури проекту програми.**

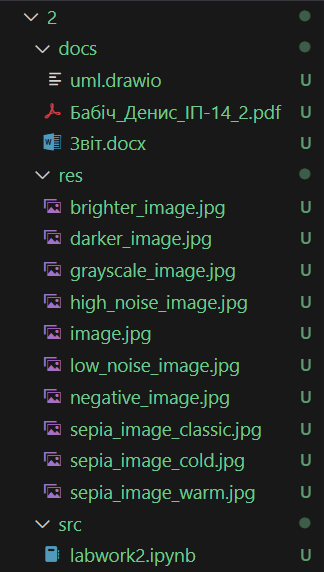


Рисунок 1.6 – Структура проєкту

У директорії src зберігається labwork2.py, який є юпітер-записником з вихідним кодом, директорія docs – зберігає файли звіту у форматі pdf, docx та drawio (файл з діаграмами), директорія res – використанні та отримані зображення під час виконання роботи.

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



Рисунок 1.7 – Початкове зображення



Рисунок 1.8 – Зображення зі збільшеною яскравістю



Рисунок 1.9 – Зображення зі зменшеною яскравістю



Рисунок 1.10 – Зображення у відтінках сірого



Рисунок 1.11 – Зображення з ефектом негативу



Рисунок 1.12 – Зображення з ефектом сепії

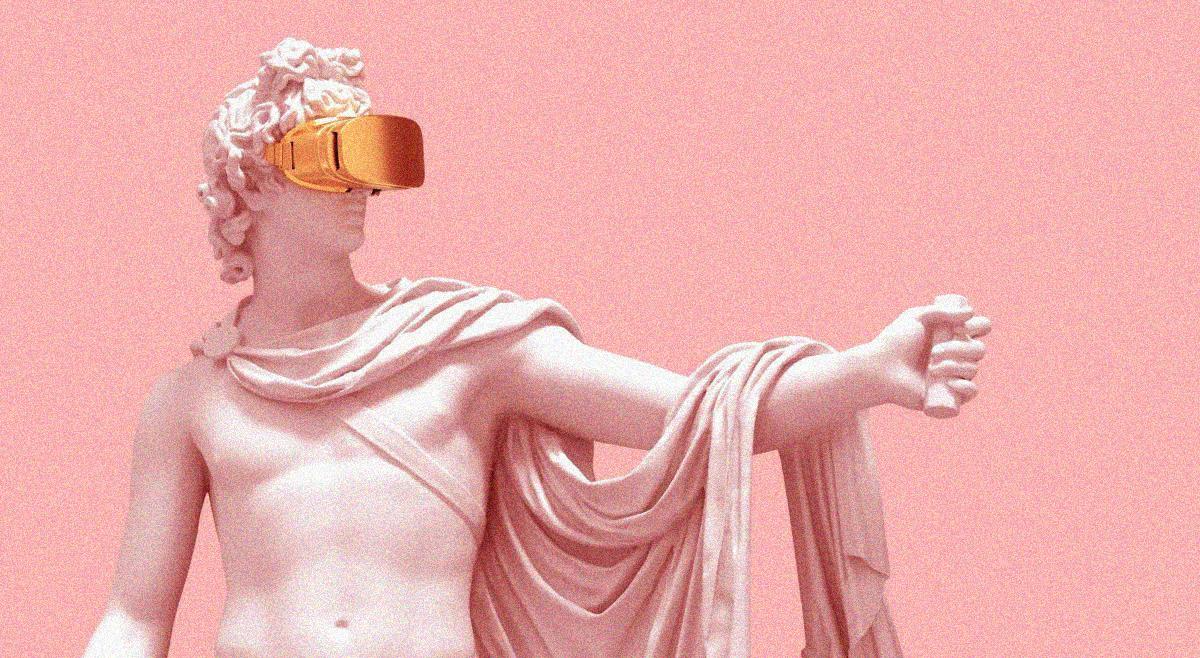


Рисунок 1.13 – Зображення з додаванням шуму

1. **Програмний код, що забезпечує отримання результату.**

# %% [markdown]

# # Лабораторна робота № 2. ІП-14 Бабіч Денис

# %% [markdown]

# ## Імпортування необхідних модулів

# %%

import numpy as np

from PIL import Image

from enum import Enum

# %% [markdown]

# ## Cтворення допоміжних функцій

# %%

def read\_image(path: str) -> Image:

return Image.open(path).convert("RGB")

def view\_image(image: Image) -> None:

image.show()

def save\_image(image: Image, file\_path: str) -> None:

image.save(file\_path)

# %% [markdown]

# ## Виконання завдання

# %% [markdown]

# ### Підготовка до виконання

# %%

image = read\_image("../res/image.jpg")

view\_image(image)

# %% [markdown]

# ### Постобробка: Зміна яскравості

# %% [markdown]

# #### Функція

# %%

def change\_brightness(image: Image, factor: float) -> Image:

COLOR\_CHANNEL\_MIN\_VALUE = 0

COLOR\_CHANNEL\_MAX\_VALUE = 255

processed\_image = np.array(image, dtype = np.float32)

processed\_image \*= factor

processed\_image = np.clip(processed\_image, COLOR\_CHANNEL\_MIN\_VALUE, COLOR\_CHANNEL\_MAX\_VALUE)

return Image.fromarray(processed\_image.astype(np.uint8))

# %% [markdown]

# #### Приклад використання № 1

# %%

brighter\_image = change\_brightness(image, 1.5)

view\_image(brighter\_image)

save\_image(brighter\_image, "../res/brighter\_image.jpg")

# %% [markdown]

# #### Приклад використання № 2

# %%

darker\_image = change\_brightness(image, 0.5)

view\_image(darker\_image)

save\_image(darker\_image, "../res/darker\_image.jpg")

# %% [markdown]

# ### Постобробка: Відтінки сірого

# %% [markdown]

# #### Функція

# %%

def apply\_grayscale(image: Image) -> Image:

width, height = image.size

processed\_image = image.copy()

for x in range(width):

for y in range(height):

r, g, b = image.getpixel((x, y))

gray\_value = int((r + g + b) / 3)

processed\_image.putpixel((x, y), (gray\_value, gray\_value, gray\_value))

return processed\_image

# %% [markdown]

# #### Приклад використання

# %%

grayscale\_image = apply\_grayscale(image)

view\_image(grayscale\_image)

save\_image(grayscale\_image, "../res/grayscale\_image.jpg")

# %% [markdown]

# ### Постобробка: Негатив

# %% [markdown]

# #### Функція

# %%

def apply\_negative(image: Image) -> Image:

COLOR\_CHANNEL\_MAX\_VALUE = 255

processed\_image = np.array(image)

processed\_image = COLOR\_CHANNEL\_MAX\_VALUE - processed\_image

return Image.fromarray(processed\_image.astype(np.uint8))

# %% [markdown]

# #### Приклад використання

# %%

negative\_image = apply\_negative(image)

view\_image(negative\_image)

save\_image(negative\_image, "../res/negative\_image.jpg")

# %% [markdown]

# ### Постобробка: Сепія в градієнтах (від центру, або до центра)

# %% [markdown]

# #### Функція

# %%

class SepiaTone(Enum):

CLASSIC = [[0.393, 0.769, 0.189],

[0.349, 0.686, 0.168],

[0.272, 0.534, 0.131]]

WARM = [[0.3588, 0.7044, 0.1368],

[0.299, 0.587, 0.114],

[0.2392, 0.4696, 0.0912]]

COLD = [[0.3, 0.7, 0.1],

[0.25, 0.6, 0.08],

[0.2, 0.5, 0.05]]

def apply\_sepia(image: Image, tone: SepiaTone = SepiaTone.CLASSIC) -> Image:

COLOR\_CHANNEL\_MIN\_VALUE = 0

COLOR\_CHANNEL\_MAX\_VALUE = 255

processed\_image = np.array(image)

sepia\_filter\_matrix = np.array(tone.value)

processed\_image = np.dot(processed\_image, sepia\_filter\_matrix.T)

processed\_image = np.clip(processed\_image, COLOR\_CHANNEL\_MIN\_VALUE, COLOR\_CHANNEL\_MAX\_VALUE)

return Image.fromarray(processed\_image.astype(np.uint8))

# %% [markdown]

# #### Приклад використання № 1

# %%

sepia\_image\_classic = apply\_sepia(image, SepiaTone.CLASSIC)

view\_image(sepia\_image\_classic)

save\_image(sepia\_image\_classic, "../res/sepia\_image\_classic.jpg")

# %% [markdown]

# #### Приклад використання № 2

# %%

sepia\_image\_warm = apply\_sepia(image, SepiaTone.WARM)

view\_image(sepia\_image\_warm)

save\_image(sepia\_image\_warm, "../res/sepia\_image\_warm.jpg")

# %% [markdown]

# #### Приклад використання № 3

# %%

sepia\_image\_cold = apply\_sepia(image, SepiaTone.COLD)

view\_image(sepia\_image\_cold)

save\_image(sepia\_image\_cold, "../res/sepia\_image\_cold.jpg")

# %% [markdown]

# ### Постобробка: Шум

# %% [markdown]

# #### Функція

# %%

def apply\_noise(image: Image, intensity: float) -> Image:

COLOR\_CHANNEL\_MIN\_VALUE = 0

COLOR\_CHANNEL\_MAX\_VALUE = 255

processed\_image = np.array(image)

noise\_filter = np.random.uniform(low = -intensity, high = intensity, size = processed\_image.shape)

processed\_image = np.clip(processed\_image + noise\_filter, COLOR\_CHANNEL\_MIN\_VALUE, COLOR\_CHANNEL\_MAX\_VALUE)

return Image.fromarray(processed\_image.astype(np.uint8))

# %% [markdown]

# #### Приклад використання № 1

# %%

low\_noise\_image = apply\_noise(image, 30)

view\_image(low\_noise\_image)

save\_image(low\_noise\_image, "../res/low\_noise\_image.jpg")

# %% [markdown]

# #### Приклад використання № 2

# %%

high\_noise\_image = apply\_noise(image, 100)

view\_image(high\_noise\_image)

save\_image(high\_noise\_image, "../res/high\_noise\_image.jpg")

1. **Висновки.**

У цій лабораторній роботі було розроблено програмне забезпечення, яке демонструє застосування різних методів обробки зображень. Було використано різні техніки, такі як зміна яскравості, перетворення у відтінки сірого, створення ефекту негативу, застосування ефекту сепії та додавання шуму. Кожна техніка була реалізована за допомогою конкретних наданих математичних формул та алгоритмів. Ця робота демонструє ефективне використання математичних та програмних навичок для обробки зображень на рівні матриці растрового зображення.

Виконав: ІП-14 Бабіч Д. В.