**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-11

Трикош І. В.

**Перевірив:**

Професор кафедри ОТ ФІОТ

Писарчук О.О.

**Київ 2024**

## **І. 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.

## **ІІ. Завдання:**

**Завдання ІІІ рівня**

Реалізувати розробку програмного скрипта, що реалізує корекцію кольору цифрового растрового зображення з переліку: негатив – в градієнтах: діагональ (від лівого верхнього кута до правого нижнього); від центру; до центру. Обробку реалізувати на рівні матриці растра. Зображення обрати самостійно.

## **ІІІ. Результати виконання лабораторної роботи.**

* 1. **Синтезована математична модель**

Відповідно до умов задачі синтезовано математичні моделі операцій над структурами вхідних графічних об’єктів.

Модель є реалізує негатив для зображення та застосовує градієнти по діагоналі, від центру та до центру.

Реалізація негативу є наступною:

де T – матриця для перетворення у негатив (заповнена максимальним значенням RGB), P – піксель, який треба перетворити (r, g, b – колір в RGB), G – відповідний градієнт для пікселя.

Градієнт по діагоналі від лівого верхнього кута до правого нижнього реалізовується наступним чином:

де i, j – координати відповідного пікселя, max\_value – сума ширини та висоти зображення.

Градієнт від центру реалізовується наступним чином:

де distance – відстань від центру зображення до відповідного пікселя, max\_distance – відстань від центру зображення до нижнього правого кута.

Градієнт до центру реалізовується наступним чином:

де distance – відстань від центру зображення до відповідного пікселя, max\_distance – відстань від центру зображення до нижнього правого кута.

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

На рис. 1 зображена блок-схема до моделі:



Рисунок 1 – Блок схема алгоритму

Робота алгоритму розпочинається із зчитування зображення за допомогою Pillow і отримання інформації про зображення (саме зображення, його draw об’єкт, ширина, висота та матриця його пікселів з RGB). Ці дії виконуються у блоці 1 блок-схеми алгоритму рис.1.

У блоці 2 реалізується створення значень градієнту за допомогою функції, що передається у метод негативу. Таких функцій є 4: значення градієнта за замовчуванням (градієнт дорівнює 1), градієнт по діагоналі (значення градієнта варіюється від 0 для верхнього лівого кута до 1 для нижнього правого кута), градієнт від центру (значення градієнта варіюється від 0 для центру до 1 для кутів зображення) та градієнт до центру (значення градієнта варіюється від 0 для кутів зображення до 1 для центру).

У блоці 3 відбувається застосування негативу з градієнтом для кожного пікселя зображення.

У блоці 4 реалізується відображення результуючого зображення з допомогою matplotlib.

У блоці 5 відбувається збереження результуючого зображення з допомогою вбудованого методу Pillow. На цьому виконання алгоритму завершено.

* 1. **Опис структури проекту програми**

Для реалізації розробленого алгоритму мовою програмування Python з використанням можливостей інтегрованого середовища PyCharm сформовано проєкт.

Проєкт базується на лінійній бізнес-логіці функціонального програмування та має таку структуру.

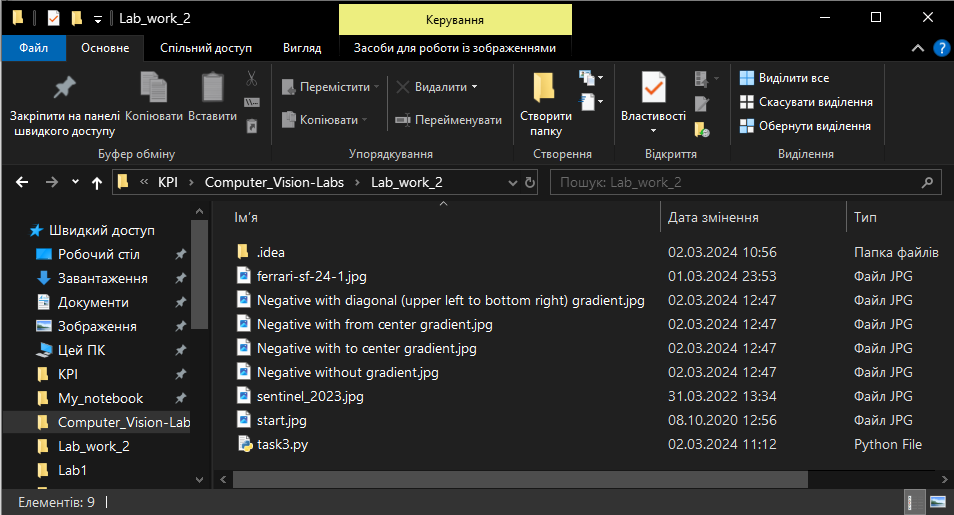


Рисунок 2 – Структура проєкту

Lab\_work\_2 – головний каталог проєкту

task3.py – файл програмного коду

ferrari-sf-24-1.jpg, start.jpg, sentinel\_2023.jpg – зображення для тестування програми

Negative without gradient.jpg, Negative with diagonal (upper left to bottom right) gradient.jpg, Negative with from center gradient.jpg, Negative with to center gradient.jpg – результат негативів з градієнтами.

* 1. **Результати роботи програми відповідно до завдання (допускається у формі скриншотів)**

Результатом роботи програми є сукупність послідовності графічних вікон, що реалізують умови завдання лабораторної роботи.

1. Початкове зображення ferrari-sf-24-1.jpg:

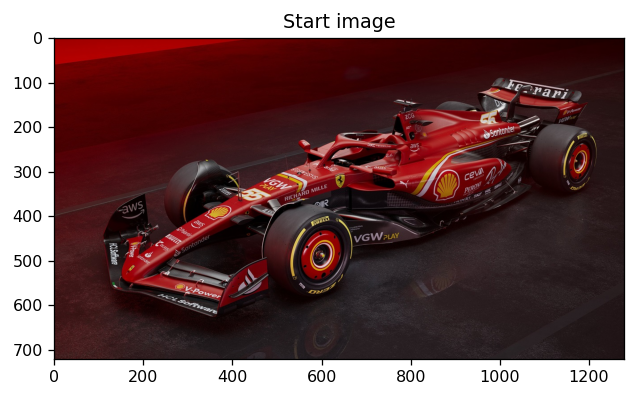


Рисунок 3 – Початкове зображення ferrari-sf-24-1.jpg

1. Негатив без градієнту:

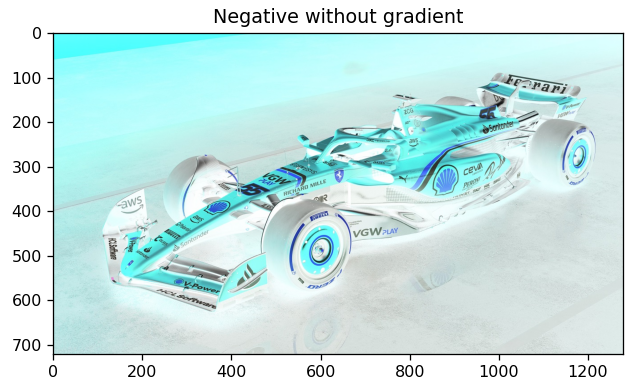


Рисунок 4 – Негатив без градієнту

1. Негатив з діагональним (верхній лівий кут – нижній правий кут) градієнтом:

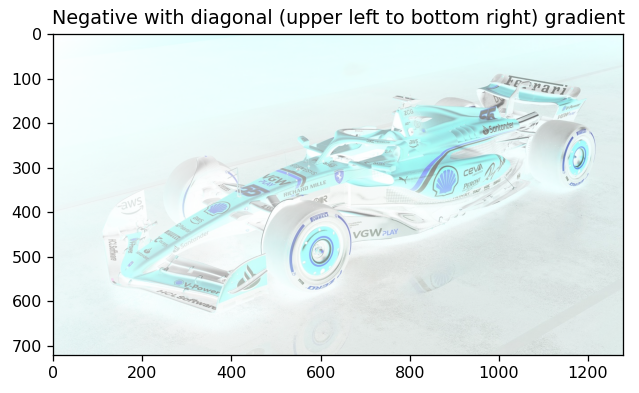


Рисунок 5 – Негатив з діагональним (верхній лівий кут – нижній правий кут) градієнтом

1. Негатив з градієнтом від центру:

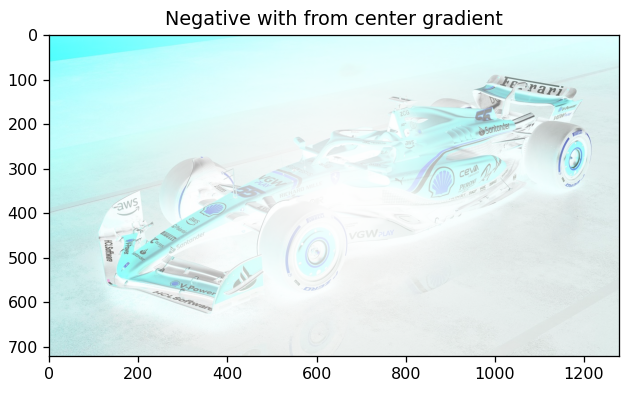


Рисунок 6 – Негатив з градієнтом від центру

1. Негатив з градієнтом до центру:

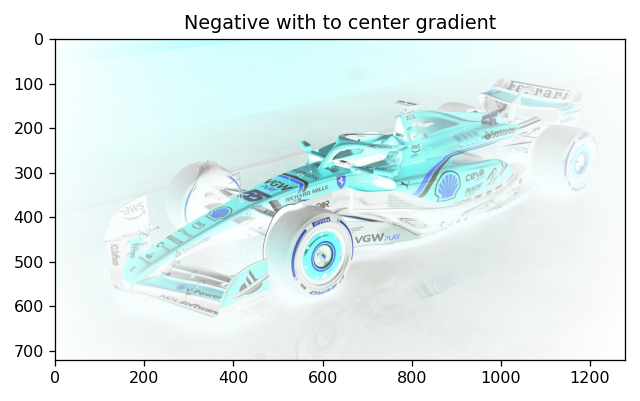


Рисунок 7 – Негатив з градієнтом до центру

Таким чином, представлені результати у повному обсязі відповідають завданню лабораторної роботи.

* 1. **Програмний код, що забезпечує отримання результату (допускається у формі скриншотів)**

Програмний код послідовно реалізує алгоритм рис.1 та спрямовано на отримання результатів, поданих на рис.3-7.

Для спрощення програмного коду і раціоналізації обчислень застосовано функціональні механізми створення підпрограм.

Обчислення проводились з матрицею растра зображення.

При цьому використано можливості Python бібліотек: numpy, Pillow, matplotlib.

Контексні коментарі пояснюють сутність окремих скриптів наведеного коду програми.

task3.py

*"""  
Цифрова обробка зображень: корекція кольору цифрового растрового зображення (негатив) в градієнтах:  
діагональ (верхній лівий до нижнього правого кута); від центру; до центру.  
"""  
  
import* numpy *as* np  
*from* PIL *import* Image, ImageDraw  
*from* matplotlib *import* pyplot *as* plt  
  
  
*def* print\_center(width, height, pixels, name):  
 *""" Вивід RGB центру зображення """* center\_x, center\_y = width // 2, height // 2  
 print(f"Image - {name}; coordinates - {center\_x}, {center\_y}; RGB - {pixels[center\_x, center\_y]}")  
  
  
*def* image\_read(file\_name):  
 *""" Завантаження зображення """* image = Image.open(file\_name)  
 draw = ImageDraw.Draw(image)  
 width = image.size[0]  
 height = image.size[1]  
 pixels = image.load()  
  
 plt.imshow(image)  
 plt.title("Start image")  
 plt.show()  
 print\_center(width, height, pixels, "Start image")  
  
 image\_info = {"image\_file": image, "image\_draw": draw, "image\_width": width, "image\_height": height, "image\_pixels": pixels}  
 *return* image\_info  
  
  
*def* get\_default\_gradient(img\_width, img\_height):  
 *""" Без градієнта (він дорівнює 1) """  
 return* np.full((img\_width, img\_height), 1)  
  
  
*def* get\_upper\_left\_to\_bottom\_right\_gradient(img\_width, img\_height):  
 *""" Значення градієнта варіюється від 0 (лівий верхній кут) до 1 (нижній правий кут) """* gradient = np.zeros((img\_width, img\_height))  
 max\_value = img\_width + img\_height  
 *for* i *in* range(img\_width):  
 *for* j *in* range(img\_height):  
 gradient[i, j] = (i + j) / max\_value  
 *return* gradient  
  
  
*def* get\_from\_center\_gradient(img\_width, img\_height):  
 *""" Значення градієнта варіюється від 0 (центр) до 1 (кути зображення) """* center\_x, center\_y = img\_width // 2, img\_height // 2  
 max\_distance = np.sqrt((img\_width - center\_x) \*\* 2 + (img\_height - center\_y) \*\* 2)  
 gradient = np.zeros((img\_width, img\_height))  
 *for* i *in* range(img\_width):  
 *for* j *in* range(img\_height):  
 distance = np.sqrt((i - center\_x) \*\* 2 + (j - center\_y) \*\* 2)  
 gradient[i, j] = distance / max\_distance  
 *return* gradient  
  
  
*def* get\_to\_center\_gradient(img\_width, img\_height):  
 *""" Значення градієнта варіюється від 0 (кути зображення) до 1 (центр) """* center\_x, center\_y = img\_width // 2, img\_height // 2  
 max\_distance = np.sqrt((img\_width - center\_x) \*\* 2 + (img\_height - center\_y) \*\* 2)  
 gradient = np.zeros((img\_width, img\_height))  
 *for* i *in* range(img\_width):  
 *for* j *in* range(img\_height):  
 distance = np.sqrt((i - center\_x) \*\* 2 + (j - center\_y) \*\* 2)  
 gradient[i, j] = 1 - distance / max\_distance  
 *return* gradient  
  
  
*def* apply\_negative(value, gradient):  
 *""" Розрахувати негатив """  
 return* int(255 - value \* gradient)  
  
  
*def* negative(file\_name\_init, gradient\_function, window\_title):  
 *""" Негатив """* image\_info = image\_read(file\_name\_init)  
 image = image\_info["image\_file"]  
 draw = image\_info["image\_draw"]  
 width = image\_info["image\_width"]  
 height = image\_info["image\_height"]  
 pixels = image\_info["image\_pixels"]  
  
 gradient = gradient\_function(width, height)  
 *for* i *in* range(width):  
 *for* j *in* range(height):  
 r, g, b = pixels[i, j]  
  
 gradient\_value = gradient[i, j]  
  
 modified\_r = apply\_negative(r, gradient\_value)  
 modified\_g = apply\_negative(g, gradient\_value)  
 modified\_b = apply\_negative(b, gradient\_value)  
 draw.point((i, j), (modified\_r, modified\_g, modified\_b))  
  
 plt.imshow(image)  
 plt.title(window\_title)  
 plt.show()  
 print\_center(width, height, pixels, window\_title)  
  
 result\_file\_name = window\_title + ".jpg"  
 image.save(result\_file\_name, "JPEG")  
 *return  
  
  
if* \_\_name\_\_ == "\_\_main\_\_":  
 init\_file = "ferrari-sf-24-1.jpg"  
 *# init\_file = "start.jpg"  
 # init\_file = "sentinel\_2023.jpg"* negative(init\_file, *lambda* width, height: get\_default\_gradient(width, height), "Negative without gradient")  
 negative(init\_file, *lambda* width, height: get\_upper\_left\_to\_bottom\_right\_gradient(width, height), "Negative with diagonal (upper left to bottom right) gradient")  
 negative(init\_file, *lambda* width, height: get\_from\_center\_gradient(width, height), "Negative with from center gradient")  
 negative(init\_file, *lambda* width, height: get\_to\_center\_gradient(width, height), "Negative with to center gradient")

## **IV. Висновки.**

У ході виконання лабораторної роботи було проведено дослідження технологій кореляції кольору окремих растрів цифрових зображень з використанням можливостей мови програмування Python та наступних операцій над ними: негатив та градієнт: діагональний (верхній лівий – правий нижній кути), від центру, до центру. Дослідження показали, що корекція кольору здійснюється за допомогою операцій над матрицею растрів цифрового зображення, тобто операція застосовується до кожного окремого пікселя зображення.

Виконав: студент Трикош І. В.