Національний технічний університет України «КПІ ім. Ігоря Сікорського»
Факультет Інформатики та Обчислювальної Техніки

Кафедра інформаційних систем та технологій

Лабораторна робота №2

з дисципліни «Технології Computer Vision»

на тему

«ДОСЛІДЖЕННЯ АЛГОРИТМІВ ФОРМУВАННЯ ТА ОБРОБКИ РАСТРОВИХ ЦИФРОВИХ ЗОБРАЖЕНЬ»

Виконала:
студентка групи ІС-12
Павлова Софія

Перевірив:
Баран Д. Р.

Київ – 2023

1. Постановка задачі

Мета роботи:

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.

Завдання III рівня:

Реалізувати розробку програмного скрипта, що реалізує корекцію кольору цифрового растрового зображення з переліку: зміна яскравості, відтінки сірого, негатив, сепія – в градієнті діагональ. Обробку реалізувати на рівні матриці растра. Зображення обрати самостійно.

2. Виконання

2.1. Зміна яскравості

Математична модель

Яскравість (Brightness) визначає на скільки колір пікселю відрізняються від червоного кольору. В колірному просторі RGB, яскравість – середнє арифметичне μ червоного, зеленого і синього кольорів пікселей.

Зміна яскравості растрового зображення на коефіцієнт α відбувається за моделлю:

$$R' = min(max(R \cdot \alpha, 0), 255),$$
$$G' = min(max(G \cdot \alpha, 0), 255),$$
$$B' = min(max(B \cdot \alpha, 0), 255),$$

де min та max функції забезпечують те, щоб значення кожного кольору знаходилося в межах від 0 до 255.

Алгоритм програми

Розробимо скрипкове рішення, яке буде змінювати градієнт зміни яскравості, розрахований як відстань між двома точками за алгоритмом ***Брезенхема***.

Рисунок 1 – Блок схема програми для зміни яскравості

Програмна реалізація

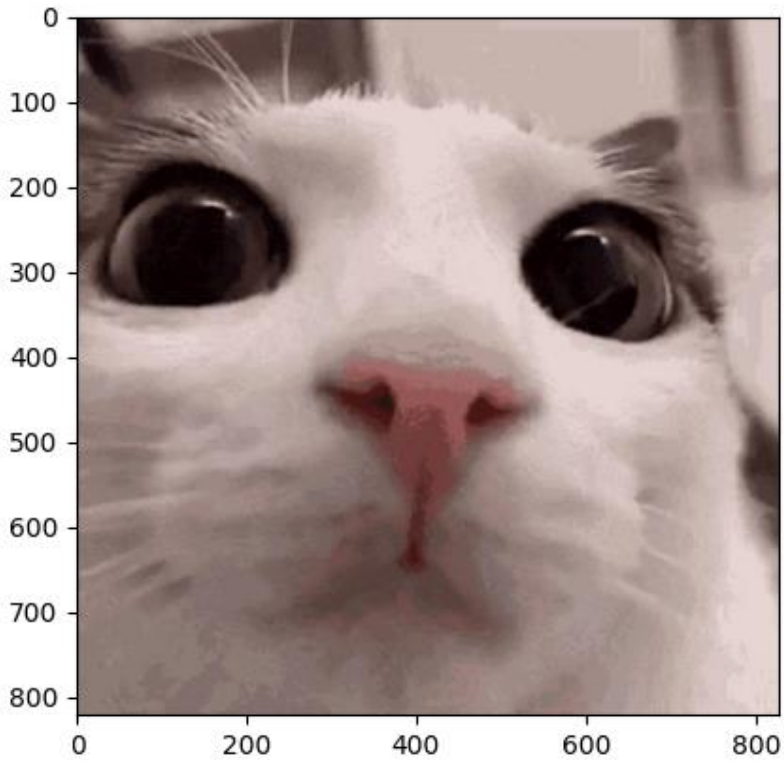
Реалізуємо функцію зчитування зображення як матриці пікселей.

Лістинг коду:

```
# Зчитування файлу зображення
def image_read(file_name: str) -> None:
    # Відкриття файлу зображення
    image = Image.open(file_name)
    # Створення інструменту для малювання
    draw = ImageDraw.Draw(image)
    # Визначення ширини картинки
    width = image.size[0]
    # Визначення висоти картинки
    height = image.size[1]
    # Отримання значень пікселей для картинки
    pix = image.load()
    print('\nПочаткове зображення')
    print(f'red = {pix[1, 1][0]}\tgreen = {pix[1, 1][1]}\tblue = {pix[1, 1][2]}')
    plt.imshow(image)
    plt.show()
    image_info = {"image_file": image, "image_draw": draw, "image_width": width,
"image_height": height,
                  "image_pix": pix}

    return image_info
```

Результат:

Бачимо візуалізацію зображення та його RGB характеристику.




```
Початкове зображення
red = 42    green = 26  blue = 26
```

Рисунок 2 – Початкове зображення

Для алгоритму ***Брезенхема*** задамо 2 точки по діагоналі зображення: *верхній лівий кут* $(0, 0)$ та *нижній правий кут* $(width - 1, height - 1)$. Алгоритм проведе через них пряму і вздовж цієї прямої буде за градієнтом змінювати яскравість кожного пікселю.

Додамо можливість користувачу визначати діапазон зміни яскравості.

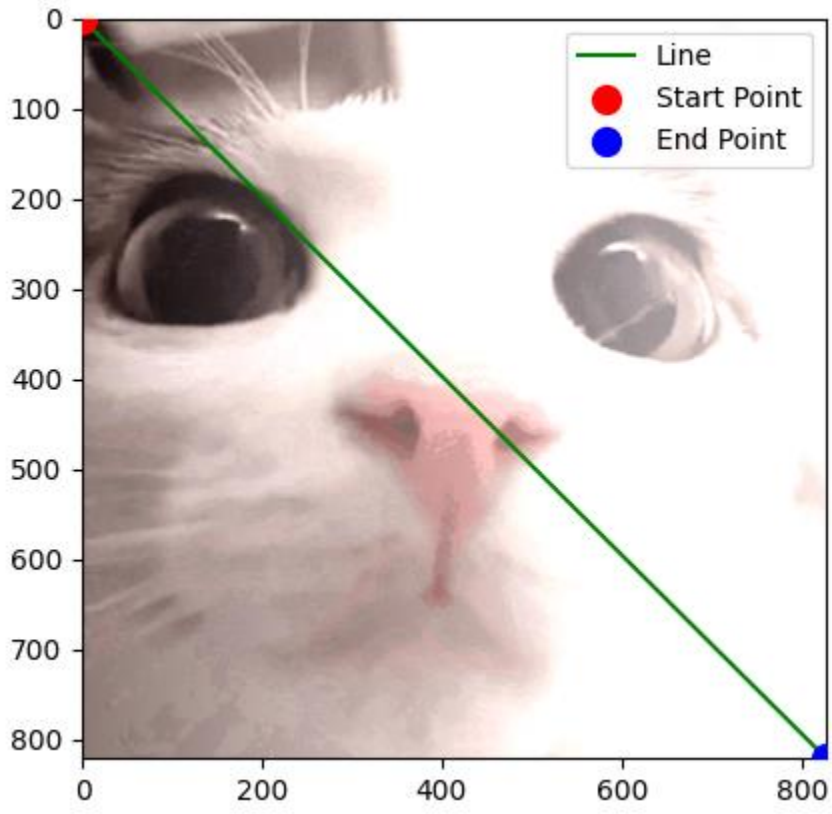
Лістинг коду:

```
# Зміна яскравості з використанням алгоритму Брезенхема
def brightness_change_with_gradient(file_name_start: str, file_name_stop: str) -> None:
    image_info = image_read(file_name_start)
    image = image_info["image_file"]
    draw = image_info["image_draw"]
    width = image_info["image_width"]
    height = image_info["image_height"]
    pix = image_info["image_pix"]

    # Задайте початкові та кінцеві точки для лінії зміни яскравості
    start_point = (0, 0)  # Лівий верхній кут
    end_point = (width - 1, height - 1)  # Правий нижній кут

    # Розрахуйте коефіцієнт зміни яскравості за градієнтом
    print('\nУведіть діапазон зміни яскравості')
    max_brightness_change = int(input('factor:'))
    brightness_gradient = max_brightness_change / (width + height)  # Градієнт зміни
яскравості

    # Прокладання лінії від початкової до кінцевої точки з використанням алгоритму
Брезенхема по діагоналі
    x1, y1 = start_point
    x2, y2 = end_point
    dx = abs(x2 - x1)
    dy = abs(y2 - y1)
    sx = 1 if x1 < x2 else -1
    sy = 1 if y1 < y2 else -1
    err = dx - dy

    x = x1
    y = y1
    while x != x2 or y != y2:
        # Зміна яскравості кожного пікселя на діагоналі відповідно до його розташування
вздовж лінії
        factor = brightness_gradient * (x - x1 + y - y1)  # Збільшення яскравості
        for j in range(height):
            # Додавання яскравості до кожного пікселя на діагоналі
            if 0 <= x < width and 0 <= j < height:  # Перевірка меж зображення
                a = int(min(max(pix[x, j][0] + factor, 0), 255))
                b = int(min(max(pix[x, j][1] + factor, 0), 255))
                c = int(min(max(pix[x, j][2] + factor, 0), 255))
                draw.point((x, j), (a, b, c))

        e2 = 2 * err
        if e2 > -dy:
            err -= dy
            x += sx
        if e2 < dx:
            err += dx
            y += sy
```

```
    plt.imshow(image)
    plt.plot([start_point[0], end_point[0]], [start_point[1], end_point[1]],
color='green', label='Line')    # Відображення лінії між точками start_point та end_point
    plt.scatter(*start_point, color='red', label='Start Point', s=100, zorder=3)   #
Відображення точки start_point
    plt.scatter(*end_point, color='blue', label='End Point', s=100, zorder=3)   #
Відображення точки end_point
    plt.legend()  # Відображення легенди з поясненням кольорів точок
    plt.show()
    print('\nКінцеве зображення')
    print(f'red = {pix[1, 1][0]}\tgreen = {pix[1, 1][1]}\tblue = {pix[1, 1][2]}')
    image.save(file_name_stop, "JPEG")
    del draw

    return
```

Результат:

Якщо ввести діапазон зміни яскравості = 200 (+200), отримаємо наступне зображення.




```
Кінцеве зображення
red = 42    green = 26  blue = 26
```

Рисунок 3 – Зміна яскравості із застосуванням градієнту

Червона та сині точки – початок та кінець прямої за якою розраховувався градієнт. Зелена пряма – діагональ зображення, уздовж якої по стовпцях змінювалась яскравість пікселів.

2.2. Відтінки сірого

Математична модель

Для перетворення кольорового зображення у відтінки сірого застосовується формула середнього значення яскравості кольорів. Кожен піксель перетворюється шляхом обчислення середнього значення його каналів яскравості S, і нове значення для кожного каналу встановлюється як S.

Формально, формула для перетворення кольорового пікселя I_{ij} у відтінок сірого може бути записана так:

$$S_{ij} = \frac{R_{ij} + G_{ij} + B_{ij}}{3},$$

де S_{ij} – нове значення яскравості для пікселя I_{ij}, а R_{ij}, G_{ij} та B_{ij} – значення червоного, зеленого і синього каналів відповідно.

Алгоритм програми

Розробимо скрипкове рішення, яке буде змінювати градієнт відтінків сірого, розрахований як відстань між двома точками за алгоритмом ***Брезенхема***.

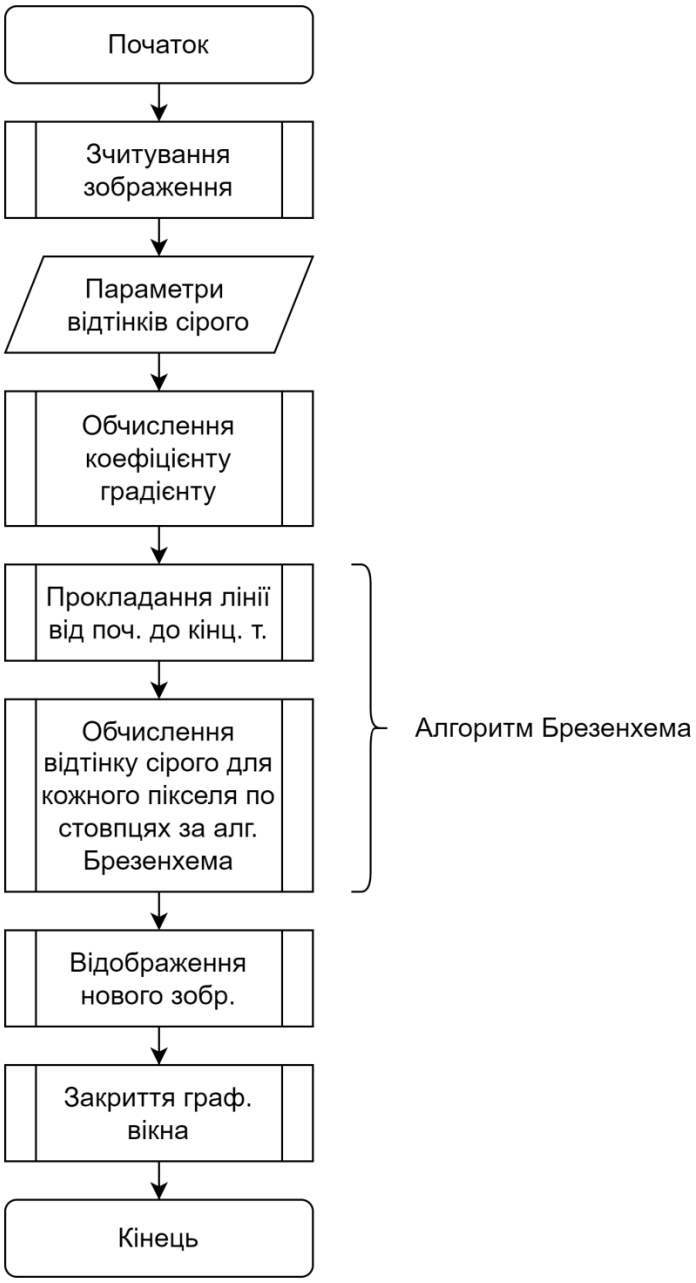


Рисунок 4 – Блок схема програми для відтінків сірого

Програмна реалізація

Реалізуємо функцію з використанням алгоритму Брезенхема для відтворення градієнту аналогічно до зміни яскравості.

Лістинг коду:

```
# Зміна відтінків сірого з використанням алгоритму Брезенхема
def shades_of_gray_with_gradient(file_name_start: str, file_name_stop: str) -> None:
    image_info = image_read(file_name_start)
    image = image_info["image_file"]
```

```
    draw = image_info["image_draw"]
    width = image_info["image_width"]
    height = image_info["image_height"]
    pix = image_info["image_pix"]

    # Задайте початкові та кінцеві точки для лінії градієнта
    start_point = (0, 0)  # Лівий верхній кут
    end_point = (width - 1, height - 1)  # Правий нижній кут

    # Розрахунок коефіцієнта зміни відтінків сірого за градієнтом
    max_gray_change = 255  # Максимальний відтінок сірого
    print('\nУведіть фактор сили градієнту (0-1)')
    gradient_factor = float(input('factor:'))
    gray_gradient = gradient_factor * max_gray_change / (width + height)  # Градієнт
зміни відтінків сірого

    # Прокладання лінії від початкової до кінцевої точки з використанням алгоритму
Брезенхема
    x1, y1 = start_point
    x2, y2 = end_point
    dx = abs(x2 - x1)
    dy = abs(y2 - y1)
    sx = 1 if x1 < x2 else -1
    sy = 1 if y1 < y2 else -1
    err = dx - dy

    x = x1
    y = y1
    while x != x2 or y != y2:
        # Зміна відтінку сірого кожного пікселя вздовж лінії градієнта
        factor = gray_gradient * (x - x1 + y - y1)  # Зміна відтінку сірого
        for j in range(height):
            # Застосування градієнту відтінків сірого до кожного пікселя на діагоналі
            if 0 <= x < width and 0 <= j < height:  # Перевірка меж зображення
                r, g, b = pix[x, j]
                gray = int((r + g + b) / 3)  # Відтінок сірого
                new_gray = int(min(max(gray + factor, 0), 255))  # Новий відтінок сірого
з градієнтом
                draw.point((x, j), (new_gray, new_gray, new_gray))

        e2 = 2 * err
        if e2 > -dy:
            err -= dy
            x += sx
        if e2 < dx:
            err += dx
            y += sy

    plt.imshow(image)
    plt.plot([start_point[0], end_point[0]], [start_point[1], end_point[1]],
color='green', label='Line')
    plt.scatter(*start_point, color='red', label='Start Point', s=100, zorder=3)
    plt.scatter(*end_point, color='blue', label='End Point', s=100, zorder=3)
    plt.legend()
    plt.show()
    print('\nКінцеве зображення')
    print(f'red = {pix[1, 1][0]}\tgreen = {pix[1, 1][1]}\tblue = {pix[1, 1][2]}')
    image.save(file_name_stop, "JPEG")
    del draw

    return
```

Результат:

При виборі фактору сили градієнту $= 0.5$ бачимо наступне накладання на зображення градієнту відтінків сірого (зліва направо відтінки сірого стають світлішими).

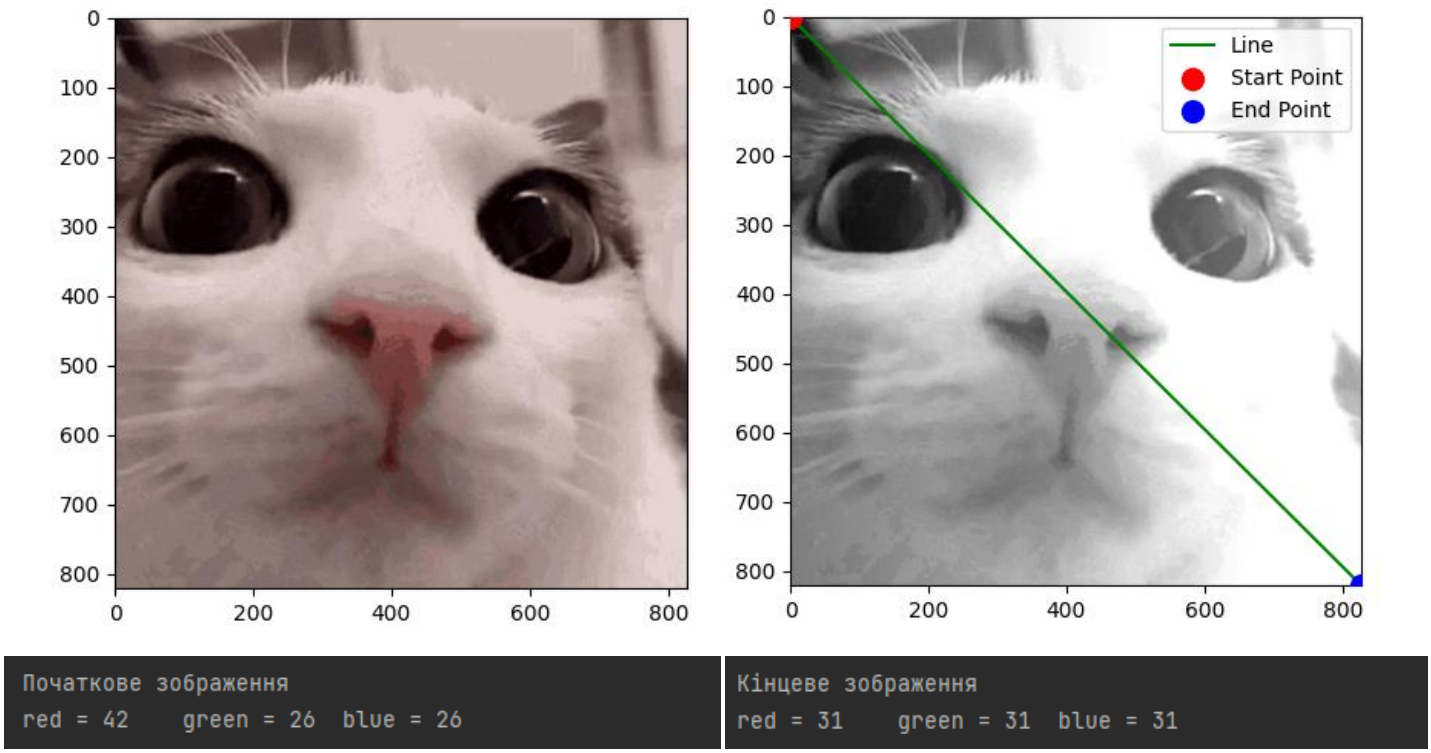


Рисунок 5 – Відтінки сірого із застосуванням градієнту

2.3. Негатив

Математична модель

Математична модель для перетворення кольорового растрового зображення в негатив у матричному представленні полягає в обчисленні доповнення кожного кольорового каналу до максимального можливого значення (255). Це можна виразити наступним чином:

$$new\_R = 255 - old\_R,$$

$$new\_G = 255 - old\_G,$$

$$new\_B = 255 - old\_B,$$

де new_R, new_G та new_B – нові значення червоного, зеленого та синього каналів відповідно, а old_R, old_G та old_B – старі значення червоного, зеленого та синього каналів відповідно.

Алгоритм програми

Розробимо скрипкове рішення, яке буде застосовувати негатив по градієнту до зображення (градієнт розрахований як відстань між двома точками за алгоритмом ***Брезенхема***).

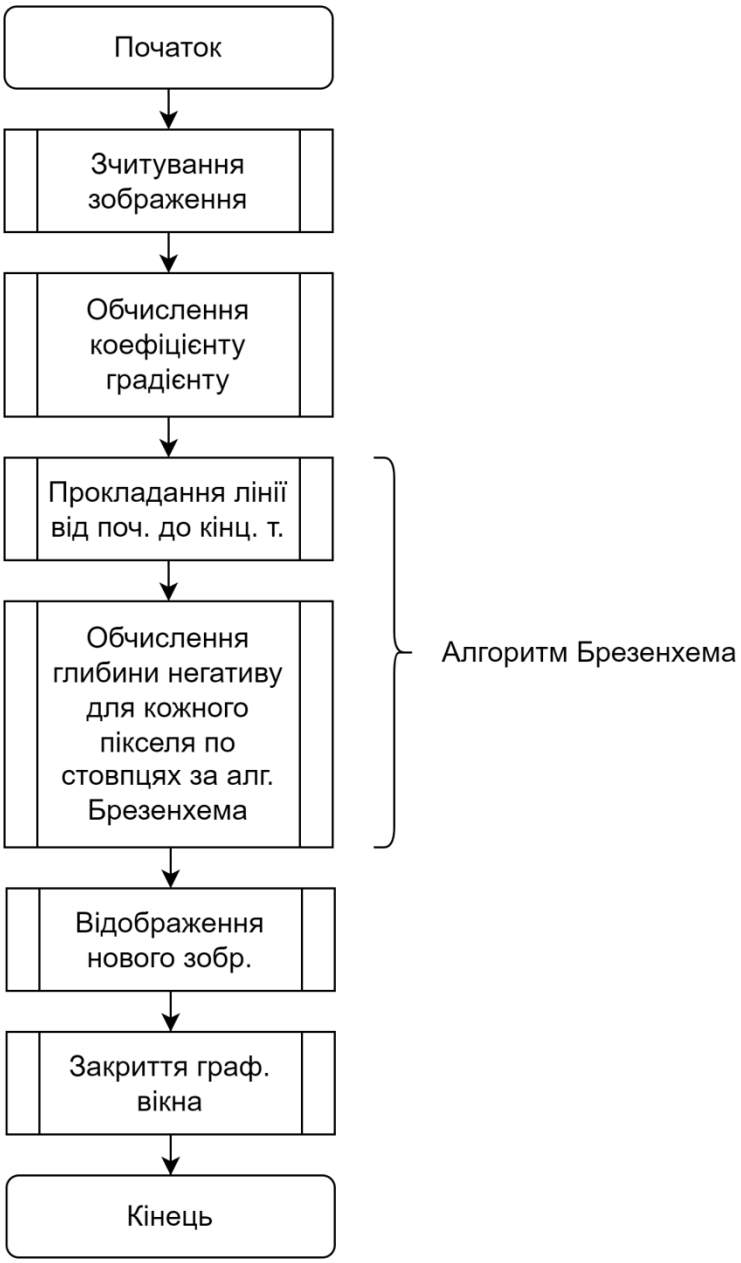


Рисунок 6 – Блок схема програми для негативу

Програмна реалізація

Реалізуємо функцію з використанням алгоритму Брезенхема для відтворення градієнту. Так, як переведення зображення в негатив – це однозначна операція, ніяких коефіцієнтів для цього застосовувати не будемо. Натомість градієнт реалізуємо як висвітлення зображення.

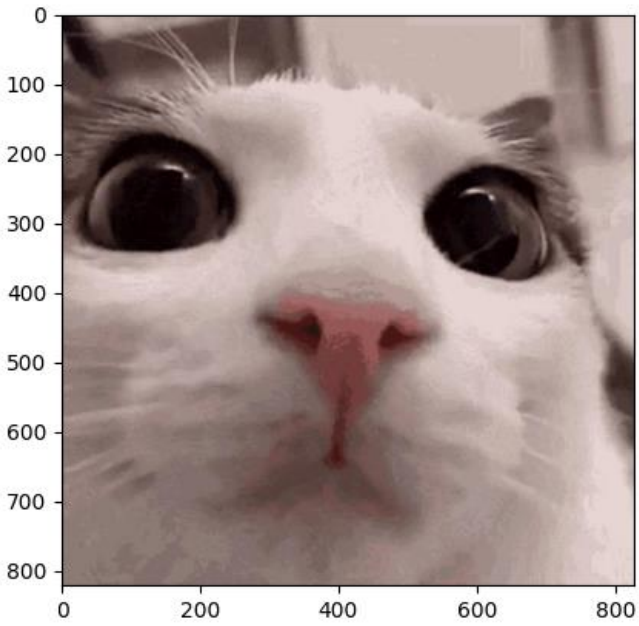
Лістинг коду:

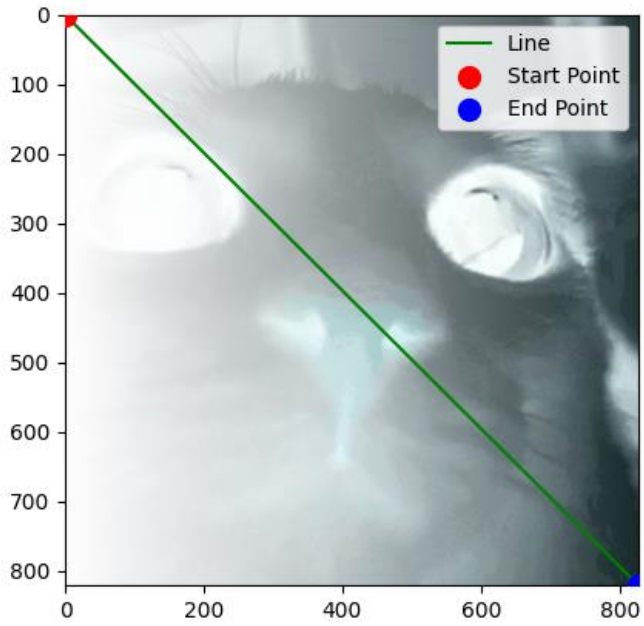
```
# Зміна негативу з використанням алгоритму Брезенхема
def negative_with_gradient(file_name_start: str, file_name_stop: str) -> None:
    image_info = image_read(file_name_start)
    image = image_info["image_file"]
    draw = image_info["image_draw"]
    width = image_info["image_width"]
    height = image_info["image_height"]
    pix = image_info["image_pix"]

    # Задайте початкове та кінцеве значення глибини негативу
    start_depth = 0  # Початкова глибина негативу
    end_depth = 255  # Кінцева глибина негативу (повний негатив)

    # Задайте початкову та кінцеву точки для лінії градієнту
    start_point = (0, 0)  # Лівий верхній кут
    end_point = (width - 1, height - 1)  # Правий нижній кут

    # Прокладання лінії від початкової до кінцевої точки з використанням алгоритму
Брезенхема
    x1, y1 = start_point
    x2, y2 = end_point
    dx = abs(x2 - x1)
    dy = abs(y2 - y1)
    sx = 1 if x1 < x2 else -1
    sy = 1 if y1 < y2 else -1
    err = dx - dy

    x = x1
    y = y1
    while x != x2 or y != y2:
        # Обчислення глибини негативу для поточного пікселя вздовж лінії
        current_depth = start_depth + (end_depth - start_depth) * (x - x1 + y - y1) /
(width + height)
        for j in range(height):
            # Застосування глибини негативу до кожного пікселя на діагоналі
            if 0 <= x < width and 0 <= j < height:  # Перевірка меж зображення
                r, g, b = pix[x, j]
                new_r = min(255 - r * (current_depth / 255), 255)
                new_g = min(255 - g * (current_depth / 255), 255)
                new_b = min(255 - b * (current_depth / 255), 255)
                draw.point((x, j), (int(new_r), int(new_g), int(new_b)))

        e2 = 2 * err
        if e2 > -dy:
            err -= dy
            x += sx
        if e2 < dx:
            err += dx
```

```
            y += sy

    plt.imshow(image)
    plt.plot([start_point[0], end_point[0]], [start_point[1], end_point[1]],
color='green', label='Line')
    plt.scatter(*start_point, color='red', label='Start Point', s=100, zorder=3)
    plt.scatter(*end_point, color='blue', label='End Point', s=100, zorder=3)
    plt.legend()
    plt.show()

    print('\nКінцеве зображення')
    print(f'red = {pix[1, 1][0]}\tgreen = {pix[1, 1][1]}\tblue = {pix[1, 1][2]}')
    image.save(file_name_stop, "JPEG")
    del draw

    return
```

Результат:

Бачимо, що зображення перетворено у негатив і н нього накладений градієнт яскравості.






```
Початкове зображення
red = 42    green = 26  blue = 26
```

```
Кінцеве зображення
red = 254   green = 254 blue = 254
```

Рисунок 7 – Негатив із застосуванням градієнту

За кольоровою характеристикою пікселів бачимо, що зображення дійсно змінилось на негативне.

2.4. Сепія

Математична модель

Перетворення кольорового зображення в сепію полягає в заміні кожного кольору на новий кольор, що відповідає тону сепії. Математично це можна виразити наступним чином:

Обчислюється середнє значення кольорів:

$$Gray = \frac{R + G + B}{3},$$

Застосовується формула переведення у відтінки сепії:

$$new\_R = Gray + depth \cdot 2,$$

$$new\_G = Gray + depth,$$

$$new\_B = Gray,$$

де $depth$ – параметр, який визначає глибину сепії, тобто наскільки темнішим буде кожен піксель відносно оригіналу.

Алгоритм програми

Розробимо скрипкове рішення, яке буде застосовувати сепію по градієнту до чорнобілого зображення (градієнт розрахований як відстань між двома точками за алгоритмом ***Брезенхема***).

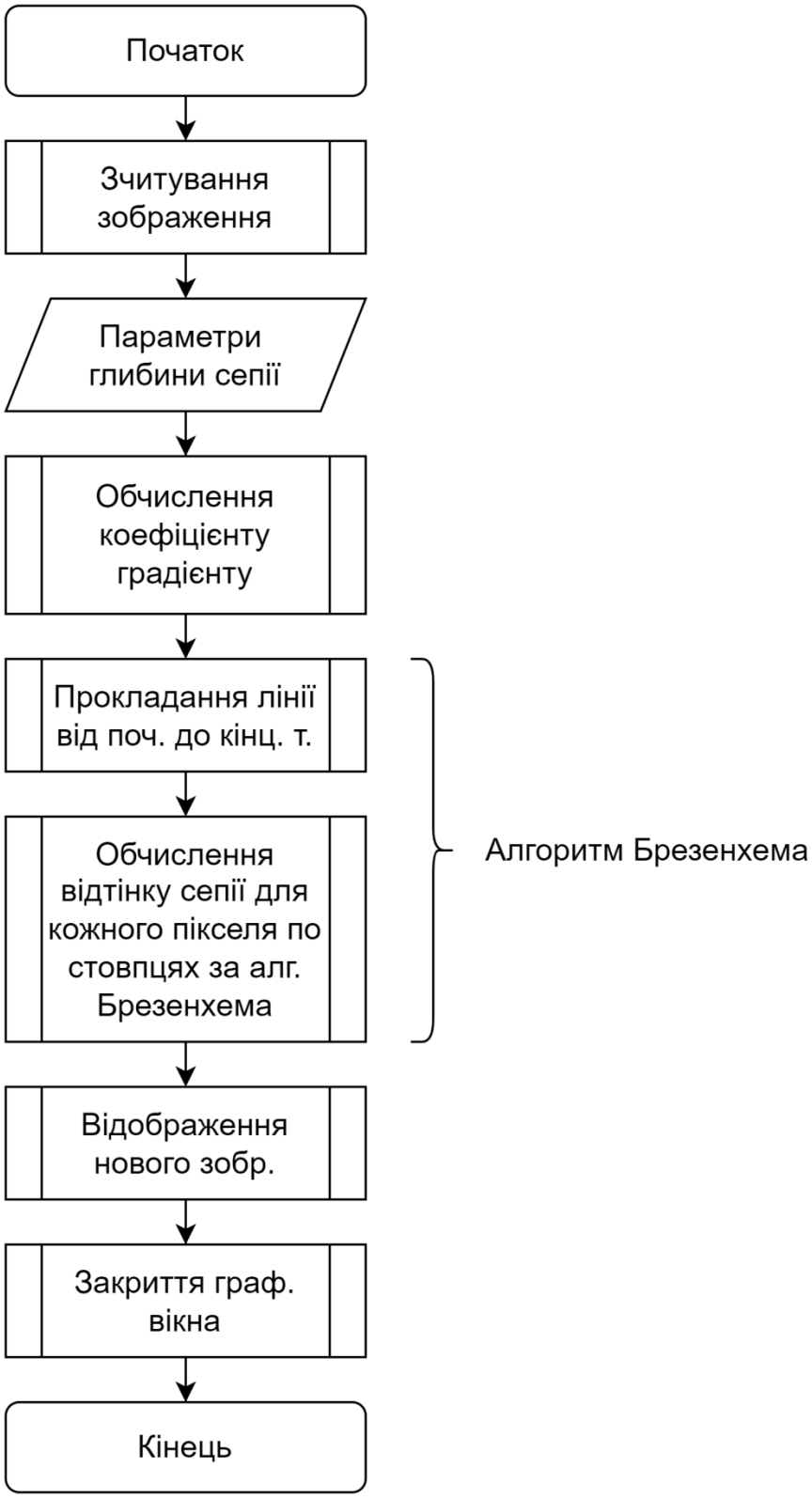


Рисунок 8 – Блок схема програми для сепії

Програмна реалізація

Реалізуємо функцію з використанням алгоритму Брезенхема для відтворення градієнту. Так, як сепія працює з середнім арифметичним каналів кольорів, виконаємо градієнт між чб зображенням та сепією.

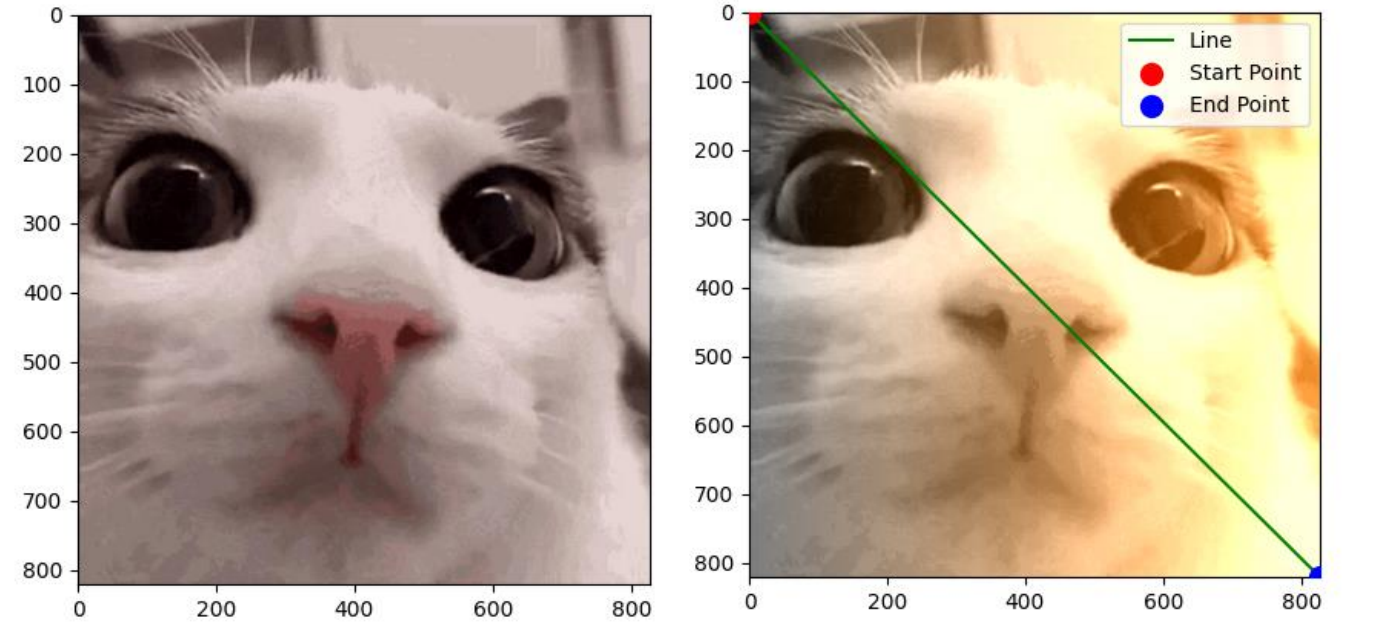
Лістинг коду:

```
# Зміна глибини сепії з використанням алгоритму Брезенхема
def sepia_with_gradient(file_name_start: str, file_name_stop: str) -> None:
    # Зчитування зображення та необхідних параметрів
    image_info = image_read(file_name_start)
    image = image_info["image_file"]
    draw = image_info["image_draw"]
    width = image_info["image_width"]
    height = image_info["image_height"]
    pix = image_info["image_pix"]

    # Задання початкової та кінцевої глибини сепії
    start_depth = 0  # Початкова глибина сепії
    print('\nУведіть діапазон зміни сепії')
    end_depth = int(input('depth:'))  # Кінцева глибина сепії

    # Задання початкової та кінцевої точок для лінії градієнту
    start_point = (0, 0)  # Лівий верхній кут
    end_point = (width - 1, height - 1)  # Правий нижній кут

    # Розрахунок коефіцієнта градієнту сепії
    gradient = (end_depth - start_depth) / (width + height)

    # Прокладання лінії від початкової до кінцевої точки за допомогою алгоритму
Брезенхема
    x1, y1 = start_point
    x2, y2 = end_point
    dx = abs(x2 - x1)
    dy = abs(y2 - y1)
    sx = 1 if x1 < x2 else -1
    sy = 1 if y1 < y2 else -1
    err = dx - dy

    x = x1
    y = y1
    while x != x2 or y != y2:
        # Обчислення глибини сепії для поточного пікселя вздовж лінії
        current_depth = start_depth + (end_depth - start_depth) * (x - x1 + y - y1) /
(width + height)

        # Застосування глибини сепії до кожного пікселя на діагоналі
        for j in range(height):
            if 0 <= x < width and 0 <= j < height:  # Перевірка меж зображення
                r, g, b = pix[x, j]
                gray = (r + g + b) // 3  # усереднення кольорів
                sepia_r = min(gray + current_depth * 2, 255)
                sepia_g = min(gray + current_depth, 255)
                sepia_b = min(gray, 255)
                draw.point((x, j), (int(sepia_r), int(sepia_g), int(sepia_b)))

        # Оновлення значення помилки та координат
```

```
        e2 = 2 * err
        if e2 > -dy:
            err -= dy
            x += sx
        if e2 < dx:
            err += dx
            y += sy

    # Відображення зображення та лінії градієнту
    plt.imshow(image)
    plt.plot([start_point[0], end_point[0]], [start_point[1], end_point[1]],
color='green', label='Line')
    plt.scatter(*start_point, color='red', label='Start Point', s=100, zorder=3)
    plt.scatter(*end_point, color='blue', label='End Point', s=100, zorder=3)
    plt.legend()
    plt.show()

    # Виведення інформації про кінцеве зображення та збереження його
    print('\nКінцеве зображення')
    print(f'red = {pix[1, 1][0]}\tgreen = {pix[1, 1][1]}\tblue = {pix[1, 1][2]}')
    image.save(file_name_stop, "JPEG")
    del draw

    return
```

Результат:

При глибині сепії $= 100$, бачимо як зображення з чб по градієнту перетворюється у сепію.




```
Початкове зображення
red = 42    green = 26  blue = 26
```

```
Кінцеве зображення
red = 31    green = 31  blue = 31
```

Рисунок 9 – Сепія із застосуванням градієнту

2.5. Аналіз отриманих результатів

Реалізовано програмний скрипт, що реалізує корекцію кольору цифрового растрового зображення – підтверджено рисунками.

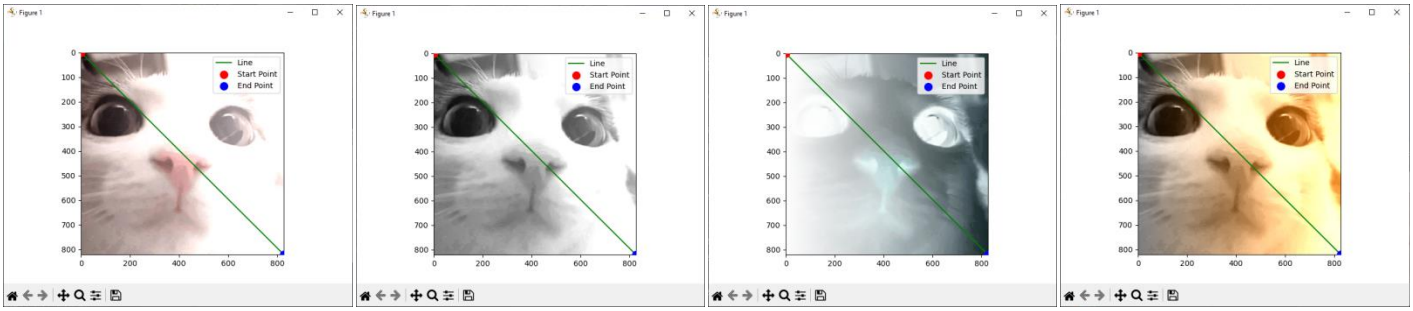


Рисунок 10 – Підтвердження реалізації програми для кольорової корекції зображення

Застосовано градієнти для кожного перетворення зображення – підтверджено рисунками та скриптом.

Обробка зображень реалізована на рівні матриць – підтверджено скриптом.

Висновок:

У результаті виконання лабораторної роботи отримано практичні та теоретичні навички роботи з растровими зображеннями.

Реалізовано програмний скрипт, який викорує корекцію кольору цифрового растрового зображення, а саме змінює його яскравість, переводить зображення у відтінки сірого, негатив та сепію.

Кожна операція накладається на початкове зображення градієнтом, який розраховується залежно від відстані по діагоналі зображення (градієнт зліва на право) за алгоритмом Брезенхема.