Dithering چیست؟

دیترینگ همانند LUT یکی از روش‌های کاهشBanding Effect (کاهش پیوستگی رنگ‌ها و بوجود آمدن مرزها)در شرایط کاهش رزولوشن(کمتر کردن بیت‌های نگهداری‌شده برای هر پیکسل به دلیل فشرده‌سازی و ...) رنگ است. در این روش با کاهش اطلاعات مکانی تصویر اثر کاهش رزولوشن جبران می‌شود.

یک استفاده معمول از dithering ، تبدیل تصویر در مقیاس خاکستری(gray scale) به سیاه و سفید است ، به گونه ای که تراکم نقاط سیاه در تصویر جدید ، تقریباً به سطح خاکستری متوسط در تصویر اصلی نزدیک باشد.

دیترینگ از روش Halftone برای افزایش سطوح رنگ کمک می‌گیرد(توزیع نقاط سیاه و سفید با چگالی‌های مختلف)

دو مورد از الگوریتم‌های Dithering را نام برده و طرز کار آنها را شرح دهید.

روش پایه‌ی دیترینگ بر پایه‌ی تکنیک halftone : در این روش یک ماتریس دیترینگ nXn با درایه های بین 0 تا $n^2 - 1$ در نظر می‌گیریم لذا از نظر تعداد نقاط سیاه موجود $n^2 + 1$ حالت داریم. لذا محدوده‌ی رنگی خود را به $n^2 + 1$ حالت تقسیم کرده و برای هر کدام یکی از حالت های ماتریس دیترینگ را چاپ می‌کنیم.

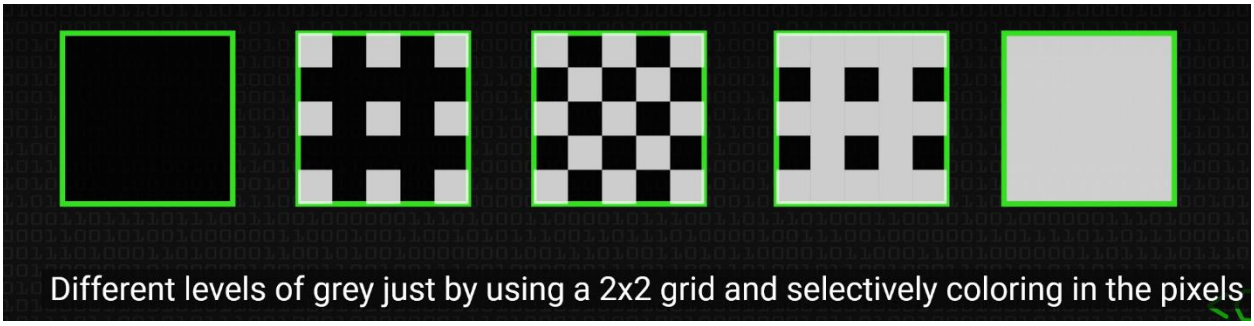
روش Ordered Dithering: برای جلوگیری از افزایش تعداد پیکسل‌ها که در روش قبل اتفاق می‌افتاد می توانیم ماتریس دیترینگ را بر روی تصویر بلغزانیم و اگر عدد حساب شده‌ی ما(z) کمتر از مقدار گفته‌شده در ماتریس دیترینگ بود در آن محل یک نقطه‌ی سیاه چاپ کنیم.

در الگوریتم Ordered dithering پنجره‌ی لغزان چه سایزهایی می‌تواند داشته باشد؟

N^2

تاثیر سایز پنجره‌ی لغزان در الگوریتم Ordered dithering را با مثالی توضیح دهید.

هر چه ماتریس بزرگتر باشد سطوح gray متعددتری را می توانیم تولید کنیم مثلا اگر سایز ماتریس ۲X۲ باشد آنگاه قادر به تولید ۵ سطح gray خواهیم بود.



گزارش

در ابتدا تابعی برای باز کردن و تبدیل عکس به GrayScale نوشته شده است که با استفاده از کتابخانه ی PIL عکس را باز کرده و در آرایه ی numpy ذخیره می‌کنیم که اگر تصویر را مجموعه از چند ردیف که شامل پیکسل‌ها هستند فرض کنیم مقدار نشان داده شده در تصویر یک سطر از تصویر است که هر پیکسل با ۳ مقدار R, G, B نشان داده شده است. حال با استفاده از فرمول زیر هر پیکسل را به یک مقدار بین ۰ تا ۲۵۵ نگاشت می‌کنیم.

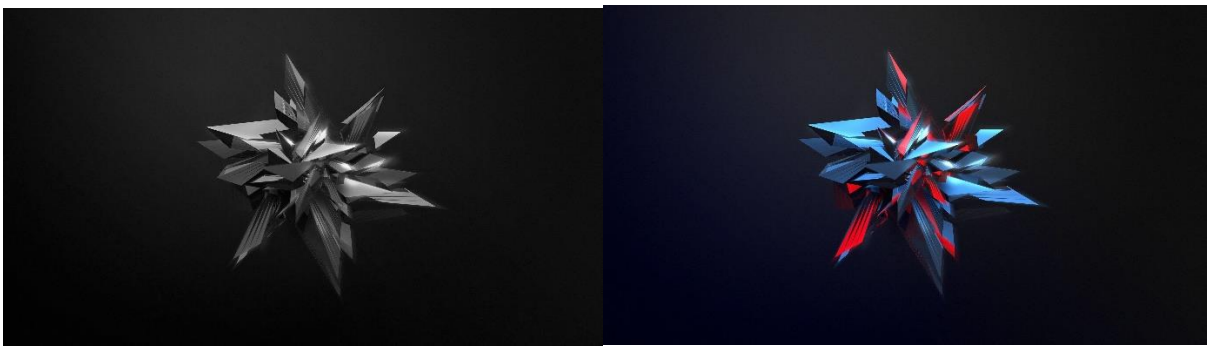
$$Y' = 0.2989\ R + 0.5870\ G + 0.1140\ B$$

عکس جدید را با استفاده از PIL ذخیره می کنیم

```python
from PIL import Image
import numpy as np


def create_gray_scale_image(address):
    # opening the image in store in numpy array
    image = np.array(Image.open(address))
    print('Here is first row of image (each row is a set of pixels showed by R G B values)\n')
    print(image[0])

    # converting image to gray-scale (transform each pixel to one number in range (0,255))
    # Y' = 0.2989 R + 0.5870 G + 0.1140 B (formula in stackOverflow)
    image_array = [[(j[0] * 299 / 1000) + (j[1] * 587 / 1000) + (j[2] * 114 / 1000) for j in r] for r in image]

    # storing the gray-scale image in numpy array
    gray_scale_image = np.array(image_array)

    # saving the gray-level image
    Image.fromarray(gray_scale_image).convert('L').save('GrayScale Of Your Image.png')

    # normalizing matrix (each element should be between (0 , 1))
    return gray_scale_image * (1 / 255)
```

```
Here is image (each row is a pixel showed by R G B values)
[[11 10 28]
 [11 10 28]
 [10  9 27]
 ...
 [15 15 15]
 [17 17 17]
 [18 18 18]]
```

خروجی به شکل زیر خواهد بود.



با استفاده از فرمول زیر به طور بازگشتی ماتریس دیترینگ را بر حسب سایز پنجره تولید می‌کنیم

https://en.wikipedia.org/wiki/Ordered_dithering

$$\mathbf{M}_{2n} = \frac{1}{(2n)^2} \times \begin{bmatrix} (2n)^2 \times \mathbf{M}_n & (2n)^2 \times \mathbf{M}_n + 2 \\ (2n)^2 \times \mathbf{M}_n + 3 & (2n)^2 \times \mathbf{M}_n + 1 \end{bmatrix}$$

```python
# matrix created recursively in a way https://en.wikipedia.org/wiki/Ordered_dithering says.
def create_dither_matrix(window_size):
    if window_size == 1:
        return np.array([[0]])
    else:
        element_0_0 = (window_size ** 2) * create_dither_matrix(int(window_size / 2))
        element_0_1 = (window_size ** 2) * create_dither_matrix(int(window_size / 2)) + 2
        element_1_0 = (window_size ** 2) * create_dither_matrix(int(window_size / 2)) + 3
        element_1_1 = (window_size ** 2) * create_dither_matrix(int(window_size / 2)) + 1
        column1 = np.concatenate((element_0_0, element_1_0), axis=0)
        column2 = np.concatenate((element_0_1, element_1_1), axis=0)
        result_matrix = (1 / window_size ** 2) * np.concatenate((column1, column2), axis=1)
        return result_matrix
```

برای پیاده‌سازی الگوریتم Ordered Dithering کافی است مانند جزوه عمل کنیم:

Algorithm 3.1 Ordered Dither

```
begin
    for x = 0 to x_max          // columns
        for y = 0 to y_max      // rows
            i = x mod n
            j = y mod n
            // I(x, y) is the input, O(x, y) is the output, D is the dither matrix.
            if I(x, y) > D(i, j)
                O(x, y) = 1;
            else
                O(x, y) = 0;
end
```

عکس دیتر شده با الگوریتم با سایز پنجره ی ۲ در پایین آمده است: