دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

درس: سیستم‌های چندرسانه‌ای

استاد درس: دکتر خرسندی

گزارش پروژه نهایی

نگارنده: میلاد اسرافیلیان
شماره دانشجویی: ۹۷۳۱۰۰۷

تیر ۱۴۰۰

سوالات تشریحی

سوال ۱) Dithering چیست؟

همانطور که میدانیم یکی از روش‌های کاهش حجم تصاویر تبدیل تصاویر ۲۴ بیتی به تصاویر با تعداد بیت کمتر برای نشان دادن هر پیکسل است.(کوانتیزاسیون رنگ) این کار اگرچه تعداد بیت ها برای هر پیکسل را کمتر میکند اما اثر منفی‌ای نیز دارد که به آن اثر بندینگ[1] گویند. در این اثر به دلیل کاهش تعداد بیت‌ها سایه‌های مشخصی که جداکننده رنگ‌های مختلف اند ظاهر می‌شوند. یکی از راه‌حل ها استفاده از دیترینگ است. دیترینگ از اثر halftone برای افزایش سطوح رنگ(عمق رنگ) از نظر بصری استفاده می‌کند. دیترینگ جزئیات اطلاعات مکانی تصویر را کاهش می‌دهد تا اثر کاهش رزولوشن را جبران کند.

سوال ۲) دو مورد از الگوریتم های دیترینگ را نام برده و طرز کار آن ها را تشریح کنید.

۱- ordered dithering:

در این روش پس از تشکیل ماتریس دیترینگ، آن را بر روی تصویر می‌لغزانیم اگر مقدار پیکسل موجود از مقداری که در ماتریس ذیتر هست بیشتر بود یک نقطه سفید و اگر کمتر بود یک نقطه سیاه چاپ میکنیم.

از مزیت‌های این روش سریعتر بودن آن نسبت به بقیه است اما با توجه به اینکه ارور کوانتیزاسیون را پخش نمی‌کنیم نتایج بدتری را بدست می‌آوریم.

۲- floyd-steinberg:

در این روش نیز پس از تشکیل ماتریس دیترینگ، آن را روی تصویر می‌لغزانیم و نزدیکترین رنگ را انتخاب میکنیم(اگر فقط دو رنگ سفید و سیاه داشته باشیم مثل روش قبل عمل میکنیم). اما تفاوت این روش با روش قبلی استفاده از یک ماتریس distribution برای پخش کردن ارور کوانتیزاسیون در همسایه‌های آن پیکسل که هنوز پردازش نشده‌اند، است. این ماتریس به شکل زیر است:

[1] Banding effect

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & \frac{7}{16} \\ \frac{3}{16} & \frac{5}{16} & \frac{1}{16} \end{bmatrix} \quad (1)$$

در واقع پس از کوانتیزاسیون ارور(مقدار کوانتیزه - مقدار اصلی) را محاسبه کرده و با ضرایب موجود در ماتریس آن را با پیکسل‌های اطراف جمع می‌کنیم و حال مقادیر بعدی را کوانتیزه می‌کنیم.

استفاده از این روش با اینکه سرعت کمتری نسبت به روش قبل‌تر دارد اما تصاویر بهتری را نمایش می‌دهد.

سوال ۳) در الگوریتم ordered dithering پنجره لغزان چه سایزهایی می‌تواند داشته باشد؟

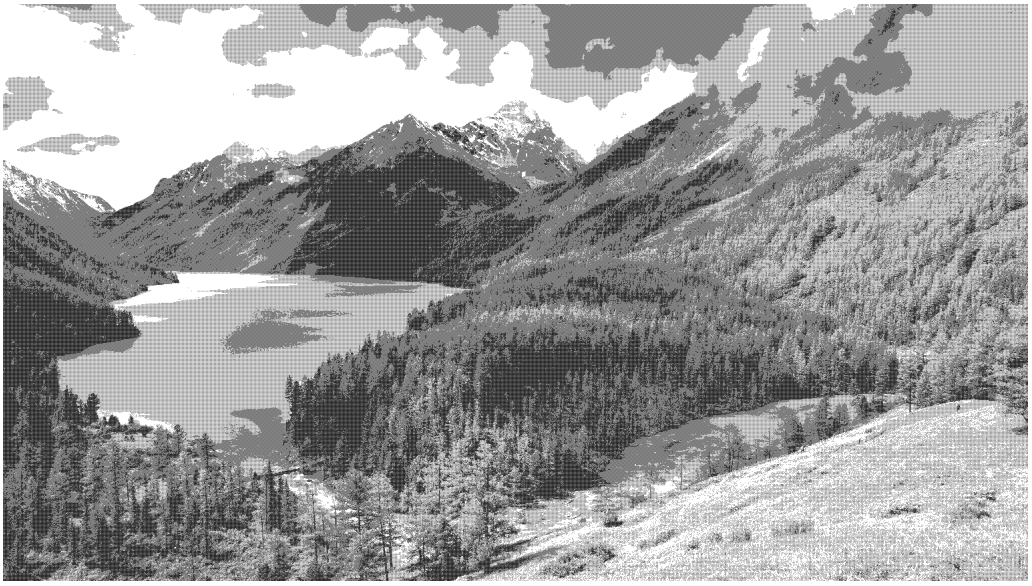
پنجره لغزان یک پنجره مربعی است(در واقع همان ماتریس دیترینگ مربعی است) و اندازه هر طرف از آن باید توانی از ۲ باشد. منبع

سوال ۴) تاثیر سایز پنجره لغزان در الگوریتم ordered dithering را با یک مثال توضیح دهید.

در شکل ۱ تصویر خاکستری یک منظره را مشاهده می‌کنیم. با استفاده از الگوریتم ordered dithering با تغییر سایز پنجره تصاویر زیر به وجود آمده اند.

همانطور که در شکل‌ها مشاهده می‌شود با افزایش سایز پنجره اثر بندینگ کمتر می‌شود و تصویر دیتر شده به تصویر اصلی نزدیکتر خواهد بود.

شکل ۱: تصویر خاکستری یک منظره بدون دیترینگ



شکل ۲: تصویر خاکستری یک منظره با اندازه پنجره ۲

شکل ۳: تصویر خاکستری یک منظره با اندازه پنجره ۴

شکل ۴: تصویر خاکستری یک منظره با اندازه پنجره ۱۶

شکل ۵: تصویر خاکستری یک منظره با اندازه پنجره ۶۴

گزارش کد

در این پروژه از دو کتابخانه Pillow برای باز کردن و ذخیره تصاویر و numpy برای انجام عملیات‌های مختلف روی ماتریس پیکسل‌های تصاویر استفاده کرده‌ایم.

از تابع dither matrix برای تشکیل ماتریس دیتر و پنجره لغزان استفاده می‌کنیم.(شکل ۶) این تابع با توجه به تعریف ماتریس دیتر به شکل بازگشتی تعریف شده است.

```python
def dither_matrix(n:int):
    if n == 1:
        return np.array([[0]])
    else:
        first = (n ** 2) * dither_matrix(int(n/2))
        second = (n ** 2) * dither_matrix(int(n/2)) + 2
        third = (n ** 2) * dither_matrix(int(n/2)) + 3
        fourth = (n ** 2) * dither_matrix(int(n/2)) + 1
        first_col = np.concatenate((first, third), axis=0)
        second_col = np.concatenate((second, fourth), axis=0)
        return (1/n**2) * np.concatenate((first_col, second_col), axis=1)
```

شکل ۶: تابع برای ساخت ماتریس دیتر با اندازه n

تعریف ماتریس دیتر به شکل زیر است:

$$M_{۲n} = \frac{۱}{(۲n)^۲} * \begin{bmatrix} (۲n)^۲ * M_n & (۲n)^۲ * M_n + ۲ \\ (۲n)^۲ * M_n + ۳ & (۲n)^۲ * M_n + ۱ \end{bmatrix} \quad (۲)$$

از تابع get image برای باز کردن تصویر با src داده شده و دریافت پیکسل های آن و تبدیل آن به grayscale به صورت دستی استفاده می‌کنیم.(شکل ۷) سپس عکس خاکستری حاصل را ذخیره می‌کنیم تا بتوانیم با حالت دیتر شده مقایسه کنیم و پیکسل‌های حاصل را به صورت نرمال شده برمی‌گردانیم.

تبدیل به grayscale طبق فرمول زیر انجام می‌شود:

$$L = R * ۰٪۲۹۹ + G * ۰٪۵۸۷ + B * ۰٪۱۱۴ \quad (۳)$$

```python
def get_image(src:str):
    img = np.array(Image.open(src))
    img_arr = [[(j[0] * 299/1000) + (j[1] * 587/1000) + (j[2] * 114/1000) for j in r] for r in img]
    img_gray = np.array(img_arr)
    Image.fromarray(img_gray).convert('L').save('gray-scale.png')
    return img_gray * (1/255)
```

شکل ۷: تابع برای باز کردن عکس و تبدیل پیکسل‌ها به grayscale

تابع ordered dithering در واقع اجرا کننده الگوریتم ordered dithering است که با لغزاندن ماتریس دیتر ورودی بر روی ماتریس پیکسل‌های تصویر اگر مقدار پیکسل از مقدار ماتریس بزرگتر باشد عدد ۲۵۵ و در غیر این صورت عدد ۰ را در نظر می‌گیرد و آن را در انتها ذخیره میکند.(شکل ۸) این کوانتیزاسیون سبب کاهش حجم شده و از افت شدید کیفیت هم جلوگیری میکند.

```python
def ordered_dithering(img_pixel:np.array, dither_m:np.array):
    n = np.size(dither_m, axis=0)
    x_max = np.size(img_pixel, axis=1)
    y_max = np.size(img_pixel, axis=0)
    for x in range(x_max):
        for y in range(y_max):
            i = x % n
            j = y % n
            if img_pixel[y][x] > dither_m[i][j]:
                img_pixel[y][x] = 255
            else:
                img_pixel[y][x] = 0

    Image.fromarray(img_pixel).convert('L').save('dithered.png', bit=1)
```

شکل ۸: تابع برای اجرای ordered dithering