به نام خدا

گزارش پروژه پایانی – درس سیستم‌های چندرسانه‌ای

نام: نیکی پورآذین

شماره دانشجویی: ۹۷۳۱۱۱۰

سوالات تشریحی:

سوال اول: دیترینگ چیست؟

در computer graphics، دیترینگ تکنیکی برای پردازش تصویر است که از آن برای ایجاد نوعی عمق ظاهری در تصاویری استفاده می‌شود که color palette محدودی دارند. این محدودیت معمولا به دلیل کاهش حجم تصویر یا محدودیت رسانه نمایش تصویر است و دیترینگ، به طور معمول در پردازش داده‌های صوتی و تصویری دیجیتال استفاده‌های زیادی دارد. از این روش برای تبدیل تصاویر grayscale به تصاویر سیاه و سفید (باینری) نیز استفاده می‌شود.

در واقع نوعی به کار گیری عمدی نویز است و برای تصادفی کردن خطای کوانتیزاسیون و جلوگیری از تشکیل الگوهای مقیاس بزرگ در آن (مانند اثر banding رنگ‌ها) استفاده می‌شود. برای این کار از تکنیک‌هایی مانند Halftone استفاده می‌شود. دیترینگ جزییات اطلاعات مکانی تصویر را کاهش می دهد تا اثر کاهش رزولوشن رنگ را جبران کند

سوال دوم: دو مورد از الگوریتم های Dithering را نام برده و طرز کار آن ها را تشریح کنید.

الگوریتم‌های Dithering انواع متفاوتی دارند. مثلا یک نوع از آنها نوع ordered است که تفاوت آن با نوع غیر ordered، در این است که تفاوتی در تعداد پیکسل‌ها ایجاد نمی‌کند. هر کدام از این انواع هم شامل روش‌های متعدد و گوناگونی هستند.

در ادامه به دو روش از بین روش‌های مشهور dithering اشاره میکنیم و توضیح مختصری در مورد هر یک می‌دهیم.

الگوریتم Floyd–Steinberg (FS) dithering ، از نوع الگوریتم‌های Error diffusion:

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

خطای کوانتیزاسیون را روی خانه‌های همسایه‌اش که هنوز پردازش نشده‌اند، پخش می‌کند. به این ترتیب از گوشه بالا چپ شروع کنیم، هربار مقدار اختلاف مقداری که نسبت داده‌ایم و مقدار واقعی را به عنوان خطا در نظر گرفته و مطابق ضرایبی که نمونه آن در ماتریس درون تصویر آمده، به پیکسل‌های همسایه اضافه می‌کنیم تا خطا پخش شود.

الگوریتم Stucki dithering، از نوع الگوریتم‌های Error diffusion:

این الگوریتم بر مبنای الگوریتم Minimized average error dithering(خطا را به جای اینکه مثل الگوریتم FS بین ۴ خانه تقسیم کند، بین ۱۲ خانه تقسیم می‌کند.) است اما کمی سریعتر است.

الگوریتم Burkes dithering، از نوع الگوریتم‌های Error diffusion:

نسخه ساده شده‌ی الگوریتم Stucki است که سرعتش بیشتر اما شفافیت در خروجی آن کمتر است.

سوال سوم: در الگورتیم dithering Ordered پنجره لغزان چه سایز هایی می تواند داشته باشد؟

می‌تواند سایزهای متفاوتی داشته باشد، "معمولا" سایز آن را توانی از ۲ انتخاب می‌کنند. اگر سایز توانی از دو باشد، می‌توان جدول مربوط به آن را به طور بازگشتی تولید کرد و اگر اینطور نباشد باید از روش دیگری استفاده کرد.

همچنین سایز آن باید بزرگتر یا مساویِ نسبت تعداد رنگ‌های مبدا (target) به رنگ‌های مقصد (خروجی) باشد. مثلا از یک تصویر grayscale با ۲۵۶ رنگ به تصویر سیاه و سفید با ۲ رنگ باید اندازه پنجره حداقل ۱۲۸ باشد (منظور اینجا از اندازه طول ضلع نیست، مثلا برای ۱۲۸ میتوان از پنجره‌ی ۸ در ۱۶ یا بزرگتر استفاده کرد. اندازه پنجره می‌تواند مربع نباشد اما مربع نبودن باعث ایجاد حالتی شبیه خطهایی در راستای طول پنجره می‌شود. در تصویری که در کنار این قسمت متن قرار گرفته دو نمونه به این صورت مشاهده می‌شود.)

(Reduced to 12 colors using 4x1 Ordered Dithering - 12KB)

(Reduced to 64 colors using 4x4 Ordered Dithering - 14KB)

نکته بعد در مورد محدودیت سایز پنجره این است که بهتر است اندازه تصویر به آن بخش‌پذیر باشد. در بعضی روش‌ها بخش‌پذیر نبودن ممکن است در حواشی اعوجاج ایجاد کند. از این نظر، معمولا توان‌های دو اعداد مناسبی هستند که معمولا اندازه‌های تصاویر به آنها بخش‌پذیر است.

همچنین مزیت دیگر توان دو بودن اندازه این است که برای اندازه‌های توان ۲، Bayer ثابت کرده است که الگوی بهینه‌ای وجود دارد که برای نویز تصویر به ما بهترین خروجی را می‌دهد. برای مثال، چند مورد از این جداول بهینه در ادامه آمده‌است.

$$\frac{1}{16} \times \begin{bmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{bmatrix} \qquad \frac{1}{9} \times \begin{bmatrix} 0 & 7 & 3 \\ 6 & 5 & 2 \\ 4 & 1 & 8 \end{bmatrix} \qquad \frac{1}{4} \times \begin{bmatrix} 0 & 2 \\ 3 & 1 \end{bmatrix}$$

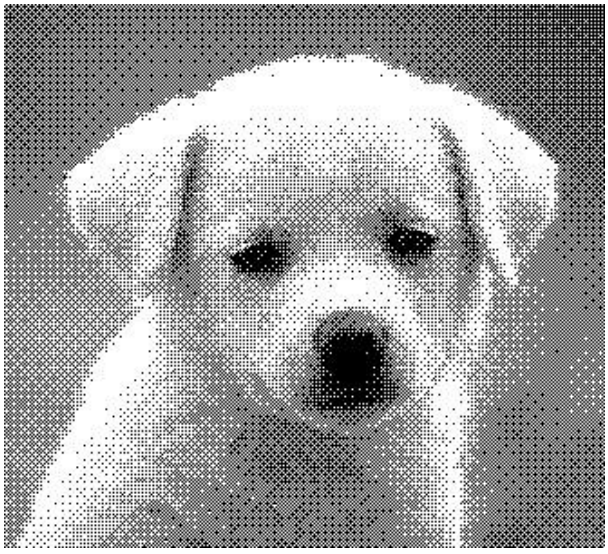
سوال چهارم: تاثیر سایز پنجره لغزان در الگوریتم dithering Ordered را با مثالی توضیح دهید.

با کمک برنامه‌ای که نوشته شده مثال را ایجاد میکنیم. در ادامه تصویر سیاه و سفید از یک سگ آمده است. یک بار آن را با پنجره لغزانی با سایز n=2 و بار دیگر با سایز n=8، dither میکنیم. این دو تصویر با جداول بهینه که در سوال قبل در رابطه با آنها توضیح دادیم، تولید شده‌اند.

(نمونه ای دیگر از تاثیر سایز آن در سوال قبل در رابطه با مربع بودن یا نبودن نیز آورده شد.)

تصویر سیاه و سفید

پس از دیتر کردن، دو تصویر زیر تولید می‌گردد:



تهیه شده با n=8

تهیه شده با n=2

همانطور که میبینیم، با افزایش n (افزایش سایز پنجره لغزان)، دقت و شفافیت تصویر بیشتر شده اما تصاویر با n کمتر، علی رغم داشتن banding بیشتر در رنگ‌هایشان تا حدودی smooth ترند. تاثیر افزایش n در پوزه‌ی سگِ شکل سمت راست و وضوح بیشتر آن به خوبی دیده می‌شود. به نظر می‌آید زیاد کردن سایز پنجره لغزان تا یک حد بالایی خوب باشد چرا که یکی از کران‌های بالای آن مثلا این است که اگر از خود عکس هم بزرگتر شود، دقت را کم خواهد کرد و در واقع می‌شود گفت بدیهیست که فایده‌ی آن در زمانیست که سایز آن کمتر از تصویر اصلی باشد.

بخش دوم: پیاده‌سازی ← گزارش کد:

توضیحات درباره‌ی فایل‌های ارسالی:

در کنار این گزارش دو برنامه به زبان پایتون قرار دارد. فایل اصلی و خواسته‌ی سوال فایل FinalProj.py است و فایل دیگر، FinalProjWithSetoutVals.py، صرفا حالات خاصی را به طور بهینه تولید می‌کند و به منظور مقایسه، نوشته شده‌است. (برای اجرای آن نیازی به دادن n به صورت ورودی نیس و فقط متغیر filename باید مطابق توضیحات بخش بعد، مقداردهی شود.)

فلدر data نیز نمونه اجرا شده‌ی کد FinalProj.py را در بر دارد.

نحوه‌ی اجرای برنامه:

پس از اجرای FinalProj.py، باید در کنار آن فلدری با نام data داشته باشیم که فایل تصویر را در آن قرار دهیم. درون برنامه، متغیر filename نام آن فایل تصویر (بدون آدرس) در خود نگه میدارد. مثال:

```
file_name = "Lenna.png"
```

پس از انتخاب فایل، برنامه ورودی n را در رابطه با سایز پنجره لغزان دریافت می‌کند.

خروجی برنامه تصویر grayscale و dithered است که هم نمایش داده می‌شوند و هم در فایلهایی با نام‌هایی شبیه فایل اصلی که به آنها به ترتیب _gs و _nd1 اضافه شده، در فرمت jpg ذخیره می‌گردد.

نحوه‌ی عملکرد برنامه:

پس از خواندن n، برنامه ابتدا تصویر را به grayscale تبدیل می‌کند. برای این کار پس از جدا کردن حالات خاص و رفع آنها، مولفه‌های r-g-b را برای هر عکس استخراج کرده و با استفاده از ضرایب زیر، آنها مقدار آن پیکسل در حالت grayscale را محاسبه خواهیم کرد.

```
The coefficients: 0.299(R)  0.587 (G)  0.114 (B)
```

پس از اینکه تصویر grayscale را تولید کرده، نمایش داده و ذخیره کردیم، با استفاده از n، جدولی n*n را با مقادیر ۰ تا n^2-1 پر میکنیم. برای این کار از یک جایگشت دلخواه استفاده شده است.

(اگر n توان ۲ باشد، جایگشت بهینه نیز وجود دارد که در FinalProjWithSetoutVals.py مثال آن آورده شده است.)

پس از ساخت dithering_table، در ادامه با استفاده از مقادیر این جدول و مقادیر پیکسل در تصویر، دقیقا مشابه با الگوریتم اسلاید ۲۷ فصل ۳ عمل میکنیم. همچنین چون رنگ‌ها مقادیری از ۰ تا ۲۵۵ دارند اما مقادیر جدول ممکن است در این بازه نباشد، از یک scale برای برابر کردن این range ها استفاده شده. در واقع مقادیر scale شده را مقایسه کرده و با توجه به اینکه کدام بزرگتر است، به آن خانه یکی از دو مقدار ۰ یا ۱ را نسبت می‌دهیم.

پس از تولید تصویر جدید دیتر شده، آن را نمایش داده و ذخیره می‌کنیم.

نمونه‌ای از خروجی حاصل از اجرای برنامه:

تصویر اصلی

تصویر grayscale

تصویر خروجی dithering