گزارش تمرین سری پنجم

اصول پردازش تصویر

رضا اکبریان بافقی - ۹۵۱۰۰۰۶۱

۱

در این سوال هدف کمینه کردن مقدار زیر بود:

$$\hat{v} = \underset{v}{\operatorname{argmin}} \sum_{i \in S. j \in S \cap N_i} \left[(v_i - v_j) - (s_i - s_j) \right]^{۲} + \sum_{i \in S. j \in -S \cap N_i} \left[(v_i - t_j) - (s_i - s_j) \right]^{۲}$$

در واقع می‌خواهیم در ناحیه Ω گرادیان نقاطی که می‌گذاریم تا حد امکان شبیه گرادیان نقاط متناظر در عکس منبع باشد.
هم‌چنین می‌خواهیم که در مرز ناحیه که با $\partial\Omega$ نشان می‌دهیم، نقاطی که می‌گذاریم با نقاط عکس هدف، تا حد ممکن، منطبق باشد.
در واقع اگر $I(x.y)$ نقاطی باشند که به دنبال آن هستیم. برای نقاطی که کاملا در ناحیه Ω هستند به دنبال نقاط زیر هستیم:

$$\nabla^{۲} I(x.y) = \nabla^{۲} S(x.y)$$

و برای نقاطی که در ناحیه $\partial\Omega$ یعنی در مرز قرار می‌گیرند به دنبال نقاط زیر هستیم:

$$I(x.y) = T(x.y)$$

پس می‌توانیم مسئله را به شکل ضرب ماتریسی $Ax = b$ دربیاوریم. که در آن مقادیر x ما همان مقادیر مجهولی است که به دنبال آن می‌گردیم. A و b را باید بسازیم تا معادلات بالا را حل کند. می‌دانیم که عبارت $\nabla^{۲} I(x.y)$ برابر مقدار زیر است.

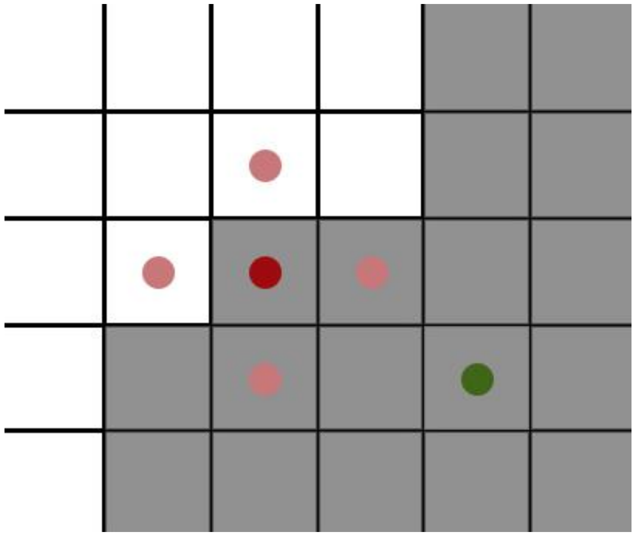
$$\nabla^{۲} I(x.y) = ۴ \times I(x.y) - I(x - ۱.y) - I(x + ۱.y) - I(x.y + ۱) - I(x.y - ۱)$$

پس A برای نقاط داخلی Ω باید عبارت را محاسبه کند. یعنی برای خود آن نقطه ضریب ۴ داشته باشد و در همان ردیف برای ۴ تا همسایه‌هایش ضریب ۱− داشته‌باشد. در این‌جا برای درایه b متناظر با آن نقطه هم باید $\nabla^{۲} S(x.y)$ را محاسبه کنیم.
اما برای نقاط مرزی شرایط فرق می‌کند. در نقطه‌های مرزی باید عبارت $I(x.y) = T(x.y)$ هم برقرار باشد. در اینجا ما برای نقاطی که همسایه‌هایشان بیرون از ناحیه Ω قرار می‌گیرند باید بجای $I(x.y)$ متناظرش $T(x.y)$ را قرار دهیم. به عنوان مثال اگر نقطه‌ای مرزی، همسایه‌های بالا و سمت چپش بیرون از ناحیه Ω بیافتاد باید عبارت زیر را برای آن محاسبه کنیم.

$$\nabla^{۲} I(x.y) = ۴ \times I(x.y) - I(x + ۱.y) - I(x.y + ۱) - T(x - ۱.y) - T(x.y - ۱)$$

بنابراین درایه‌های A که متناظر با این نقاط هستند، نباید درایه ۱− برای همسایه‌هایی که بیرون از ناحیه می‌افتند داشته باشند.
حال در b متناظر با آن نقطه باید $\nabla^{۲} S(x.y)$ با $T(x.y)$ همسایه‌هایی که بیرون از ناحیه هستند جمع شود. برای مثال در نمونه بالا داریم:

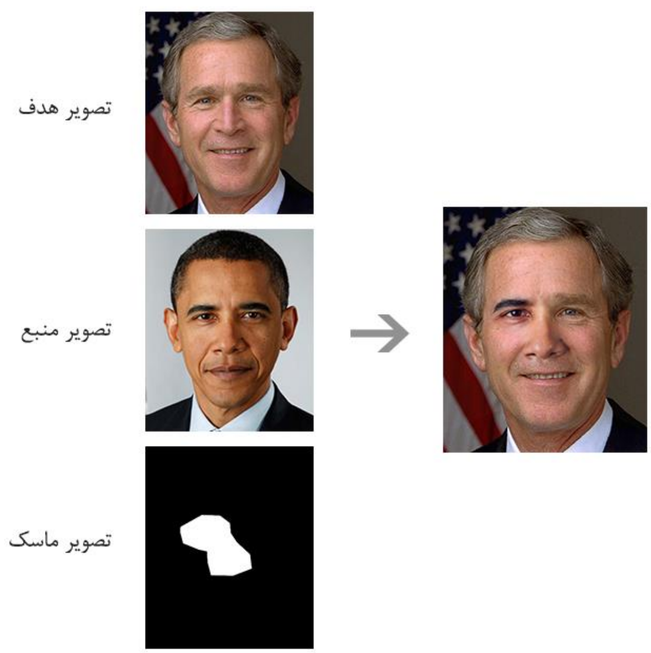
$$۴ \times I(x.y) - I(x + ۱.y) - I(x.y + ۱) = \nabla^{۲} S(x.y) + T(x - ۱.y) + T(x.y - ۱)$$



شکل ۱ در این شکل نقاط خاکستری ناحیه Ω را مشخص کرده است که در آن نقطه سبز به عنوان نقطه داخلی و نقطه قرمز پررنگ به همراه همسایه‌هایش به عنوان نقطه‌ای مرزی مشخص شده است.

بعد از آن‌که A و b را ساختیم، باید معادله $Ax = b$ را حل کنیم. چون این ماتریس A یک ماتریس اسپارس می‌باشد، بنابراین در زمان کمتری این معادله حل می‌شود. از توابع آماده برای حل این معادله استفاده می‌کنیم. حال از x به‌دست آمده استفاده می‌کنیم و این روشنایی‌های به‌دست آمده را در نقاط متناظرشان قرار می‌دهیم.

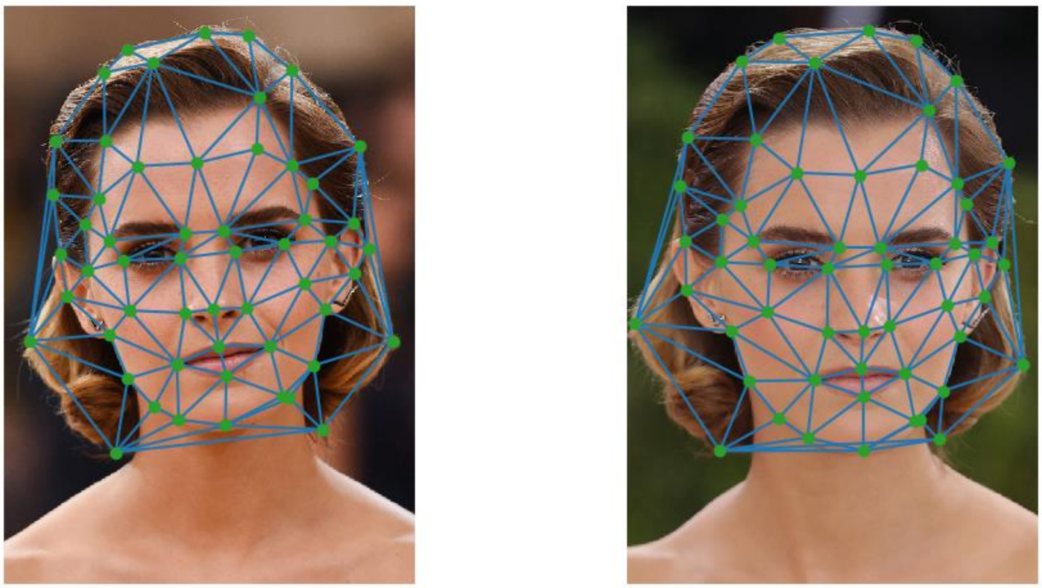
این کار را باید برای هر کانال رنگی به‌طور مجزا انجام بدهیم و سپس آن‌ها را باهم ترکیب کنیم. یکی از نتایج به دست آمده در ادامه قابل مشاهده است.



شکل ۲ تصویر حاصل از اعمال poisson blending

۲

ابتدا تعدادی نقطه متناظر در دو تصویر انتخاب شده‌اند. اگر می‌خواهید خودتان نقطه انتخاب کنید دکمه r را فشار دهید و شروع به انتخاب نقطه با کلیک چپ کنید. وگرنه با زدن کلید esc، پنجره را ببندید تا برنامه با نقاط از قبل انتخاب شده خروجی بدهد. نقاط انتخابی پیش‌فرض را می‌توانید در شکل ۳ مشاهده کنید.



شکل ۳ نقاط متناظر انتخاب شده در دو شکل و مثلث‌بندی آن‌ها

ابتدا مثلث‌بندی را در نقاط یکی از تصاویر انجام می‌دهیم. چون نقاط متناظر با هم انتخاب شده‌اند پس با اعمال آن مثلث‌بندی به نقاط تصویر دیگر، مثلث‌های متناظر با هم به دست می‌آیند. در واقع تابع زیر آبجکت `tri` را به ما می‌دهد که درونش آرایه `simplices` وجود دارد که در آن نشان داده که کدام نقاط باهم یک مثلث تشکیل داده‌اند.

```
tri = Delaunay(u)
```

سپس با انتخاب m، تعداد مراحل مورفینگ را تعیین می‌کنیم. من در این سوال این مقدار را برابر ۱۰ قرار داده‌ام. در هر مرحله دو مثلث متناظر با هم انتخاب می‌کنیم و محاسبه می‌کنیم برای رسیدن به مثلث مورد نظر (که از طریق جمع وزن دار مختصات دو مثلث قبلاً به‌دست آورده‌ایم) چه تبدیلی باید انجام دهیم. این عمل را تابع `warpTriangle` انجام می‌دهد.

در تابع `warp` ابتدا با استفاده از این مختصات دو مثلث ماتریس `Affine` را به‌دست می‌آوریم و روی تصویر اول ورودی تابع اعمال می‌کنیم. سپس سعی در ساختن یک ماسک برای جدا کردن مثلث نهایی از تصویر وارپ شده می‌کنیم. با استفاده از تابع `fillConvexPoly` آن قسمتی که توسط مثلث نهایی احاطه شده است را به صورت یک ماتریس به‌دست می‌آوریم. سپس آن قسمت از تصویر را می‌بریم و به تصویر نهایی‌مان در این مرحله اضافه می‌کنیم.

سپس خروجی این تصاویر `warp` شده را به‌طور وزن‌دار با هم جمع می‌کنیم و به لیست تصاویرمان اضافه می‌کنیم. این لیست تصاویر را بعداً تبدیل به فایل گیف می‌کنیم. خروجی در فایل `im2.gif` قابل مشاهده می‌باشد.