دانشکده مهندسی کامپیوتر

پایان نامه جهت اخذ درجه کارشناسی

نمایش های مختلف ماتریس اسپارس

و کاربرد آن در پردازش تصویر

استاد پروژه:

.....

توسط:

.....

.....

مقدمه:

مجموعه عملیات و روش هایی که برای کاهش عیوب و افزایش کیفیت ظاهری تصویر مورد استفاده قرار می گیرد، پردازش تصویر نامیده می شود.حوزه های مختلف پردازش تصویر را می توان شامل بهبود تصاویر مختلف پزشکی مانند آشکار سازی تومور های مغز یا پهنای رگ های خونی و ... ، افزایش کیفیت تصاویر حاصل از ادوات نمایشی مانند تصاویر تلویزیونی و ویدیویی، ارتقا متون و شکل های مخابره شده در رسانه های مختلف مانند شبکه و فاکس و همچنین بهبود کیفیت روش های کنترل توسط بینایی ماشین و درک واقعی تر مناظر توسط ربات ها دانست.

اگرچه حوزه ی کار با تصاویر بسیار گسترده است ولی عموما محدوده ی مورد توجه در چهار زمینه ی بهبود کیفیت ، بازسازی تصاویر مختل شده، فشرده سازی تصویر و درک تصویر توسط ماشین متمرکز می گردد. در اینجا سه تکنیک اول بررسی خواهد شد.

از آنجایی که برای کار روی تصاویر با پیکسل ها سروکار داریم و هر پیکسل نشان دهنده ی یک عنصر از یک آرایه ی دوبعدی است، کار روی تصاویر همواره با کار روی ماتریس ها عجین شده است. ماتریس اسپارس یا ماتریس خلوت ، ماتریسی است که درایه های صفر آن زیاد باشد و در نتیجه ذخیره ی عناصر صفر مقرون به صرفه نیست و همواره سعی در کاهش ذخیره ی این عناصر است تا بتوان عملیات ماتریسی را سریع تر انجام داد. در کار با تصویر با اینگونه ماتریس ها زیاد برخورد می کنیم . در این پروژه ابتدا تکنیک ها و روش های مختلف پردازش تصویر را معرفی می کنیم. در بخش بعد الگوریتم های موازی را شرح می دهیم که در GPU کاربرد دارند و با معماری موازی آشنا می گردیم. در بخش سوم برخی از الگوریتم های مربوط به ماتریس خلوت را مورد بررسی قرار می دهیم و در نهایت در بخش چهارم کاربرد این ماتریس ها را در پردازش تصویر معرفی خواهیم نمود.

و در آخر، پیاده سازی یکی از ا لگوریتم های مبحث فشرده سازی را روی تصاویر باینری، انجام خواهیم داد و با یکی از الگوریتم های فشرده سازی مربوط به تصاویر باینری به نام Run length coding مقایسه خواهیم نمود.

بخش اول

روش های پردازش تصویر

توجه و روی آوردن به روش های پردازش تصاویر به اوایل سال ۱۹۲۰ باز می گردد، زمانی که عکس های دیجیتال برای اولین بار توسط کابل های زیردریایی از نیویورک به لندن فرستاده شد.با این حال، کاربرد مفهوم پردازش تصویر تا اواسط ۱۹۶۰ گسترش وپیشرفت چندانی نیافت. در ۱۹۶۰ بود که کامپیوتر های نسل سوم دیجیتال به بازار آمد که می توانست سرعت و حافظه بالای مورد نیاز برای پیاده سازی الگوریتم های پردازش تصویر رافراهم کند.

از آن پس، تجربه در این زمینه گسترش یافت. مطالعات و تحقیقات زیادی در این موضوع در علوم مختلف از جمله : مهندسی، علوم کامپیوتر، علوم اطلاعات، فیزیک، شیمی، بیولوژی و داروسازی انجام شد.

نتیجه ی این تلاش ها در تکنیک های پردازش تصویر در مسائل مختلف - از بهبود کیفیت و بازیابی تصاویر گرفته تا پردازش اثر انگشت در مسائل تجاری – خود رانشان داد.

در این فصل بر آنیم که تکنیک ها و روش های مختلف پردازش تصویر را معرفی و بررسی کنیم. اما پیش از پرداختن به روش ها ، برخی تعاریف پایه را ذکر خواهیم کرد.

۱-۱ تصویر دیجیتالی:

تصویر به عنوان ترجمه image نشانگر یک شکل دو بعدی می باشد که توسط یک وسیله ی حساس به نور مانند دوربین به وجود آمده باشد. اما picture (عکس) نشانگر هر گونه شکل دو بعدی مانند یک تابلوی نقاشی و یا یک دست نوشته است. مقصود از تصویر دیجیتال ، digital image می باشد.

یک **تصویر** را می توان توسط تابع دوبعدی $f(x,y)$ نشان داد که در آن x و y را مختصات مکانی و مقدار f در هر نقطه را شدت روشنایی تصویر درآن نقطه می نامند. اصطلاح سطح خاکستری نیز به شدت روشنایی تصاویر مونوکروم (monochrome) اطلاق میشود . تصاویر رنگی نیز از تعدادی تصویر دوبعدی تشکیل می شود.

زمانی که مقادیر x و y و مقدار $f(x,y)$ با مقادیر گسسته و محدود بیان شوند ، تصویر را یک **تصویر دیجیتالی** می نامند. دیجیتال کردن مقادیر x و y را Sampling و دیجیتال کردن مقدار $f(x,y)$ را quantization گویند.

برای نمایش یک تصویر M * N از یک آرایه دو بعدی (ماتریس) که M سطر و N ستون دارد استفاده می کنیم . مقدار هر عنصر از آرایه نشان دهنده ی شدت روشنایی تصویر در آن نقطه است. در تمام توابعی که پیاده سازی می شود ، هر عنصر آرایه یک مقدار ۸ بیتی است که می تواند مقداری بین ۰ و ۲۵۵ داشته باشد. مقدار صفر نشان دهنده ی رنگ تیره (سیاه) و مقدار ۲۵۵ نشان دهنده رنگ روشن (سفید) است.

به عنوان مثال تصویر زیر که سایز آن ۲۶۵×۲۸۸ است از یک ماتریس که دارای ۲۸۸ سطر و ۲۶۵ ستون است برای نمایش تصویر استفاده می کند

شکل ۱-۱

هر پیکسل از این تصویر نیز مقداری بین ۰ و ۲۵۵ دارد . نقاط روشن مقادیری نزدیک به ۲۵۵ و نقاط تیره مقادیر نزدیک به ۰ دارد. همه ی توابع پردازش تصویر از این مقادیر استفاده کرده و اعمال لازم را بر روی تصویر انجام می دهند.

۲-۱ تعریف رنگ و ویژگی های آن:

برای ارایه ی یک تعریف صحیح از رنگ باید علاوه بر پدیده های فیزیکی و قوانین حاکم بر آن، نتیجه ی حاصل از این پدیده های فیزیکی که ذهنی می باشد را نیز در نظر گرفت.

از دیدگاه فیزیکی ایجاد رنگ به ۳ عامل بستگی دارد که عبارتند از :

۱)منبع نوری که جسم را روشن می کند.
۲)جسم که به وسیله منع نوری روشن می شود.
۳)چشم و مغز که رنگ را دریافت می کند.

اگرچه بهترین دریافت کننده ای که می تواند رنگ را بسنجد و در مورد آن دریک لحظه قضاوت نماید چشم و مغز انسان می باشد، اما به جز چشم نور یاب های دیگری مانند فتو تیوپها و فتوسلها نیز در سنجش رنگ توسط دستگاه ها به کار می روند. جهت ایجاد رنگ های متفاوت، منبع نوری باید علاوه بر انرژی مناسب، توزیع کافی در طیف مریی بین ۳۸۰ تا ۷۶۰ نانومتر را داشته باشد و مشاهده کننده نیز از بینایی رنگی معمول و نرمالی برخوردار باشد. به علاوه محیط مشاهده نیز از فضای مناسبی برای تشخیص جسم برخوردار باشد.
بدیهی است که با تغییر هر یک از سه عامل اصلی ایجاد کننده ی رنگ یعنی منبع نوری، جسم و مشاهده کننده تغییراتی دررنگ ظاهر شده ایجاد خواهد شد.
به سیستم هایی که بیان و تنظیم رنگ را ارایه می دهند " فضای رنگ " گویند. در ادامه به تعریف چند سیستم فضای رنگ رایج می پردازیم.

۱-۲-۱ فضای رنگ HSV :

به منظور بررسی رفتار یک انسان در موردرنگ و تقسیم بندی آنان فرض می گردد که شخصی که هیچ تجربه قبلی راجع به رنگ ندارد قصد دارد سنگ هایی با رنگ های مختلف را طبقه بندی نموده و از لحاظ رنگ آن ها را منظم و نامگذاری نماید. فرض می شود اولین کار شخص جدا کردن سنگ های رنگی از سنگ های غیر رنگی مانند سیاه و سفید و خاکستری باشد.
در میان سنگ های غیر رنگی می توان ردیف منطقی از رنگ های سفید و خاکستری روشن و خاکستری تیره و سیاه ایجاد نمود و یا به عبارت دیگر در میزان روشنایی آن ها تفکیک قایل شد. نام دیگر آن **کیفیت ارزش*** می باشد.
در مورد سنگ های رنگی می توان آن ها را ابتدا از نظر **ته رنگ** یا **فام**** از یکدیگر جدا ساخت. یعنی آنها را به رنگ های قرمز و آبی و زرد و نارنجی و غیره تقسیم بندی نمود و در هر طبقه ی رنگی نیز مجددا آنها را در دسته های کوچکتری مانند قرمز ته آبی و یا ته زرد و... قرار داد. علاوه بر آن هر سری از رنگ ها با فام مشخص را می توان دوباره بنا به کم رنگی مانند سنگ های آکروماتیک مجددا تقسیم بندی کرد. مثلا یک سری سنگ های با فام قرمز می تواند از صورتی کمرنگ تا قرمز گیلاسی تقسیم بندی شود. دراین صورت هر سنگ قرمز دراین سری از لحاظ کمرنگی می تواند یک مشابه در سری سنگ های خاکستری آکروماتیک داشته باشد. علاوه بر دو مولفه ی (ارزش و فام) که شخص در تفکیک رنگ ها انتخاب نموده، مولفه دیگری نیز برای تشخیص موجود است.

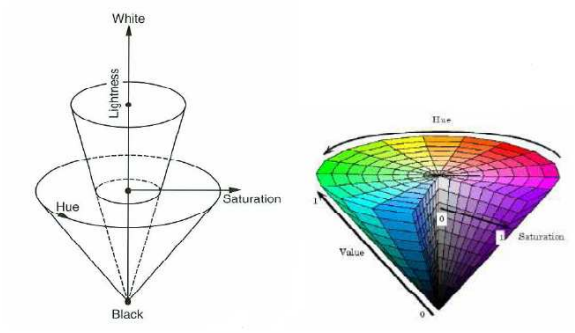
---
* value
** Hue

مثلا اگر یک سنگ قرمز آجری با یک سنگ درخشان قرمز گوجه فرنگی مقایسه شود اختلافی در فام و ارزش(روشنایی) مشاهده نمی شود در واقع هیچکدام زردتر یا آبی تر از دیگری نیست و به علاوه از لحاظ روشنایی نیز تفاوتی وجود ندارد و با یک خاکستری در سری آکروماتیک معادلند. ولی هرکسی تفاوت آن ها را تشخیص می دهد.

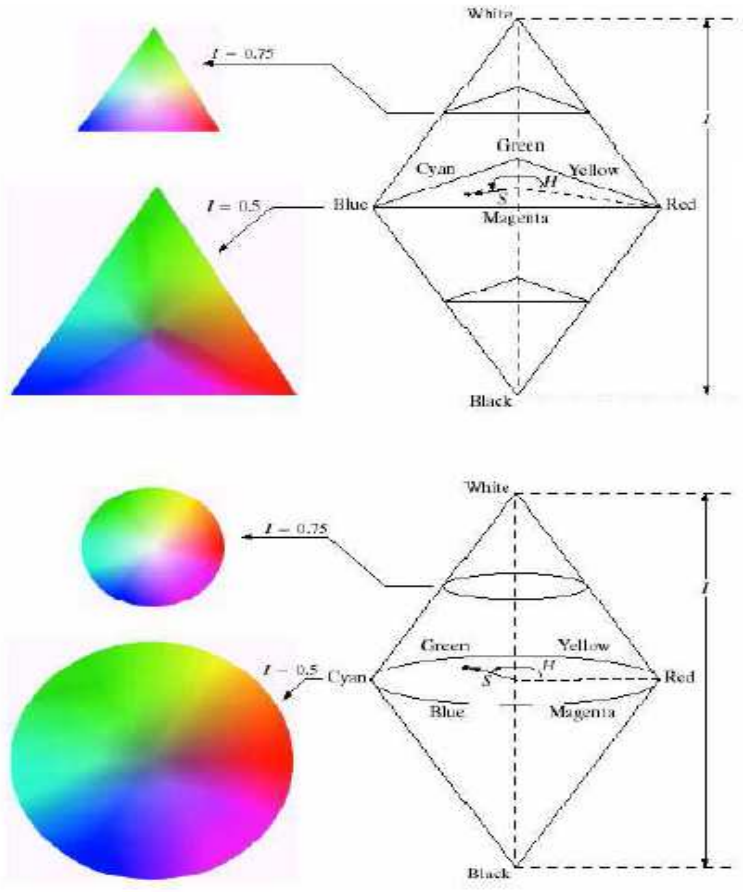
مولفه سوم در اینجا مشخص می شود و آن **خلوص رنگ**یا **اشباع رنگ*** نام دارد.

در شکل ۲-۱ سیستم رنگ یا فضای رنگ HSV نمایش داده شده که بر اساس همین سه مولفه تعریف می شود.

* saturation



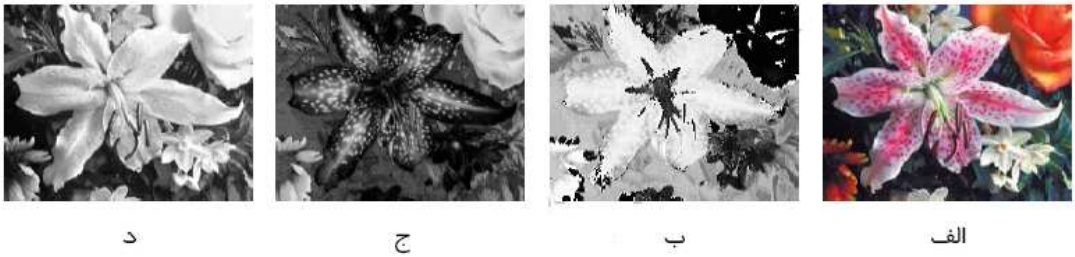
ارتباط مؤلفه های سیستم HSV.



شکل ۲-۱

چگونگی تغییرات پارامترهای مختلف رنگ.

شکل ۳-۱



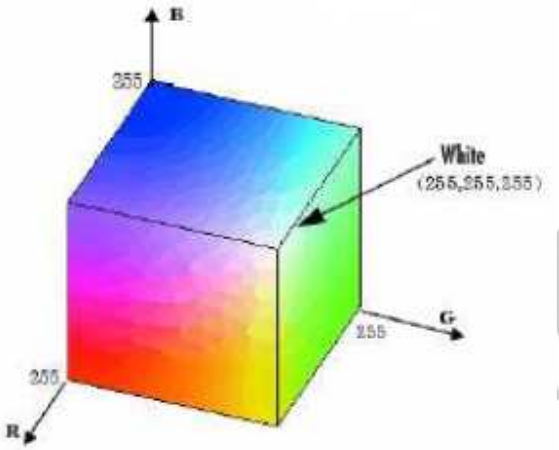
الف- تصویر رنگی، ب- و تصویر مقادیر مؤلفه فام و ب- مؤلفهٔ خلوص و ج- مؤلفهٔ روشنائی(ارزش) از مؤلفه‌های HSV به صورت جداگانه.

شکل ۴-۱

۱-۲-۲ فضای رنگ RGB :

در این سیستم فرض می شود که هررنگ در یک فضای سه بعدی از سه مولفه رنگی مستقل قرمز ، سبز و آبی تشکیل شده است.

فضای رنگ RGB متداول ترین فضای رنگ به کار گرفته شده در پردازش تصویر می باشد. دوربین های رنگی ، پویشگرها و صفحه های نمایشی در اغلب موارد دارای سیگنال های ورودی و خروجی تعریف شده در این فضای رنگ می باشند. مقادیر خاکستری در روی قطر اصلی تعریف می شود که سیاه در مختصات R=0,G=0,B=0 و سفید در نقطه ای با مختصات R=G=B=max تعریف می شود که max=255 و دلیل آن استفاده از ۱ بایت برای هر رنگ است.
اشکال عمده ی این فضا، همبستگی زیاد بین مولفه های رنگی است. به نحوی که همبستگی R-B حدود ۰/۷۸ و برای R-G حدود ۰/۹۴ می باشد.



فضای مکعبی رنگ ها که سیاه در مبدأ قرار دارد

شکل۵-۱

۳-۱پردازش تصویر (Image Processing)

پردازش تصاویر امروزه بیشتر به موضوع پردازش تصویر دیجیتال گفته می‌شود که شاخه‌ای از دانش رایانه است که با پردازش سیگنال دیجیتال که نماینده تصاویر برداشته شده با دوربین دیجیتال یا پویش شده توسط پویشگر هستند سر و کار دارد. پردازش تصاویر دارای دو شاخه عمدهٔ بهبود تصاویر و بینایی ماشین است. بهبود تصاویر دربرگیرندهٔ روش هایی چون استفاده از فیلتر محوکننده و افزایش تضاد برای بهتر کردن کیفیت دیداری تصاویر و اطمینان از نمایش درست آنها در محیط مقصد (مانند چاپگر یا نمایشگر رایانه) است، در حالی که بینایی ماشین به روش هایی می‌پردازد که به کمک آنها می‌توان معنی و محتوای تصاویر را درک کرد تا از آنها در کارهایی چون رباتیک استفاده شود.
در واقع اگر a(m,n) یک پیکسل در تصویر باشد ، این پیکسل بعد از عملیات پردازش به b(m,n) تبدیل خواهد شد. گذشته از روش های ارتباط دو تصویر قبل و بعد از پردازش، فرآیند پردازش در دو سطح کلی مقدماتی و پیشرفته صورت می پذیرد. در سطح مقدماتی ، هدف به دست آوردن اطلاعات تصویر و بهبود ظاهر آن توسط انسان می باشد. و شامل

حذف نویز، جداسازی اجسام از زمینه ی تصویر، رمزگذاری و فشرده سازی می باشد. سطح پیشرفته استفاده از ا طلاعات تصویرجهت استفاده در کامپیوتر می باشد که به بینایی ماشین تعبیر می شود.
دراینجا به تکنیک های مختلف پردازش تصاویر در سطح مقدماتی خواهیم پرداخت .

۱-۳-۱ بهبود کیفیت تصویر(image enhancement) :

بهبود کیفیت ظاهری تصویر از مباحث مهم در پردازش تصویر می باشد که به منظور کار در هر گیرنده ای می تواند مورد استفاده قرار گیرد. مواقعی پیش می آید که جزییات تصویر به دلیل نورپردازی نا مناسب یا اشکالات مختلف نا شی از تصویر برداری نامناسب کیفیت ظاهری مطلوبی ندارد که در این صورت می توان با استفاده از روش های مختلف پردازش، آن ها را بهبود بخشید. به علاوه ممکن است که در اثر مخابره ی تصویر، نویز مختصری بر روی آن تاثیر گذاشته باشد که در این صورت باید توان نویز را کاهش داد.
به طور کلی می توان روش های بهبود ظاهر تصویر را به دو گروه تقسیم کرد :
الف: روشهایی که مبتنی بر مقادیر روشنایی اصلی تصویر بوده و پردازش درحوزه مکان* صورت می گیرد.
ب:روش هایی که مبتنی بر تبدیلات تصویر می باشد و پردازش در حوزه تبدیل** (مانند فرکانس) صورت پذیرفته و سپس با تبدیل معکوس تصویر قابل رؤیت و دریافت است.

۱-۱-۳-۱بهبود کیفیت تصویر در حوزه مکان :

اصطلاح حوزه مکان به کل پیکسل های تشکیل دهنده ی تصویر اشاره دارد و روش های حوزه مکان روش هایی هستند که به طور مستقیم بر روی پیکسل ها کار می کنند. توابع پردازش تصویر در حوزه مکان را می توان به صورت:

$$g(x,y)=T[f(x,y)]$$

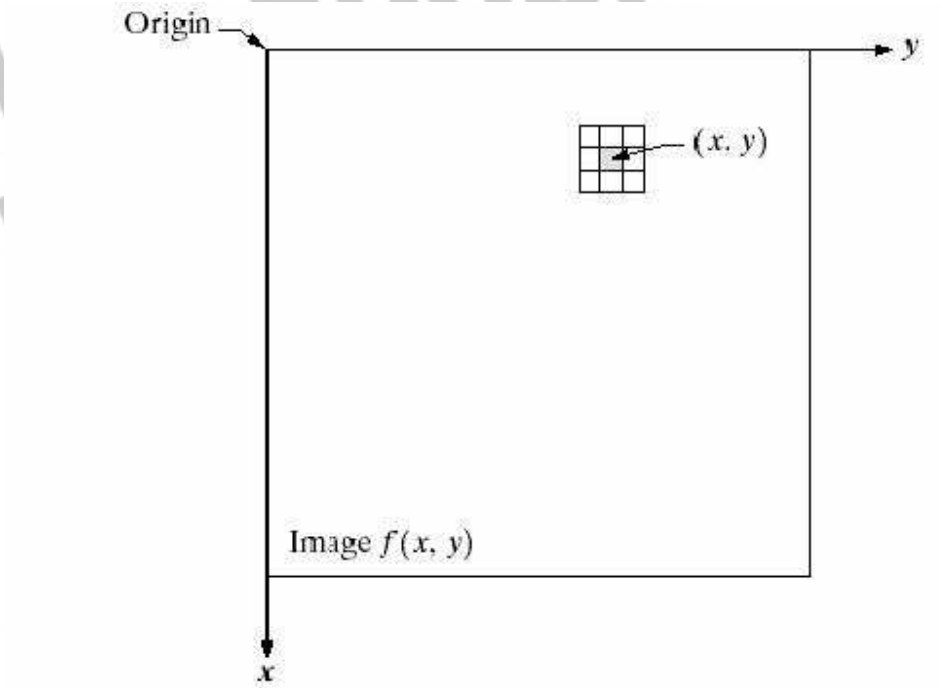
بیان کرد که $f(x,y)$ مربوط به تصویر ورودی و $g(x,y)$ مربوط به پیکسل متنا ظر آن در تصویر پردازش شده است و T یک عملگر روی f است که در یک همسایگی پیکسل (x,y) تعریف می شود.

---

* spatial domain

** frequency domain

همانطور که درشکل ۶-۱ نشان داده شده، روش تعریف همسایگی حول (x,y) استفاده از زیر تصویر های کوچک مربعی یا مستطیلی به مرکز (x,y) می باشد. مرکز زیر تصویر مثلا با شروع از گوشه چپ بالای تصویر پیکسل به پیکسل جا به جا می شود و در هر نقطه (x,y) با استفاده از T مقدار g تعیین می شود.

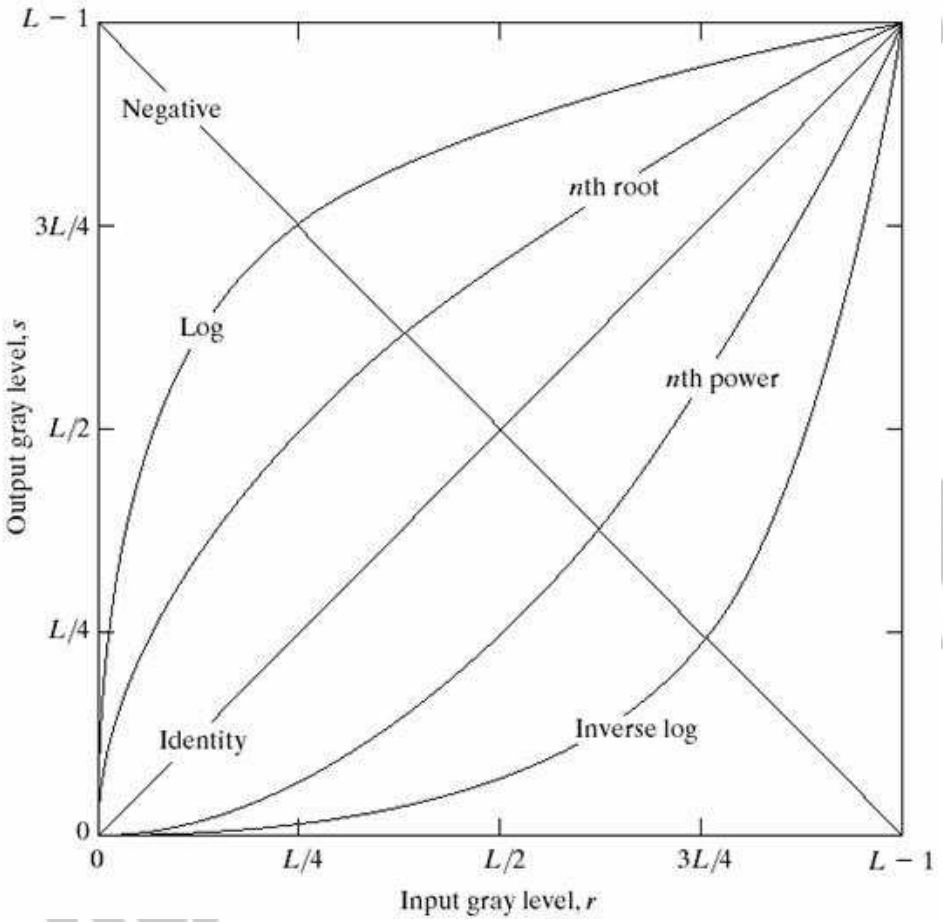


شکل۶-۱

تکنیک های بهبود تصویر را با بررسی توابع تبدیل سطوح خاکستری که مبتنی بر شدت روشنایی یک نقطه هستند شروع می کنیم. تابع تبدیل این توابع می تواندخطی یا غیر خطی باشد.
نکته مهم دراین روش ها که روش نقطه ای نامیده می شوند آن است که مقدار روشنایی هر پیکسل فقط و فقط بستگی به روشنایی پیکسل متنا ظر در تصویر اصلی دارد. در ادامه ی این بحث شدت روشنایی پیکسل ها قبل و بعد از پردازش را به ترتیب با r و s نمایش می دهیم.

برای آشنایی با سطوح خاکستری شکل ۷-۱ را در نظر بگیرید. این شکل سه نوع اصلی از توابع که اغلب برای بهبود تصویر به کار می روند را نشان می دهد. این توابع عبارتند از :

خطی (تبدیل های منفی و همانی)
لگاریتمی(تبدیل های لگاریتم و لگاریتم معکوس)
نمایی (تبدیلات توان n ام و ریشه n ام)



Linear:
Negative, Identity
Logarithmic:
Log, Inverse Log
Power-Law:
*n*th power, *n*th root

شکل۷-۱

تابع همانی تابع کم اهمیتی است که در آن شدت روشنایی خروجی با روشنایی ورودی برابر است و تنها برای کامل بودن شکل آورده شده است.

قرینه ی یک تصویر با سطوح خاکستری در محدوده [0,L-1] با استفاده از تبدیل منفی نشان داده شده در شکل، با رابطه

S=L-1-r

به دست می آید. هدف این است که ترتیب سیاه به سفید عکس شود طوری که با افزایش شدت روشنایی ورودی روشنایی تصویر خروجی کاهش یابد. این تبدیلات بیشتر در تصاویر پزشکی کاربرد دارد.

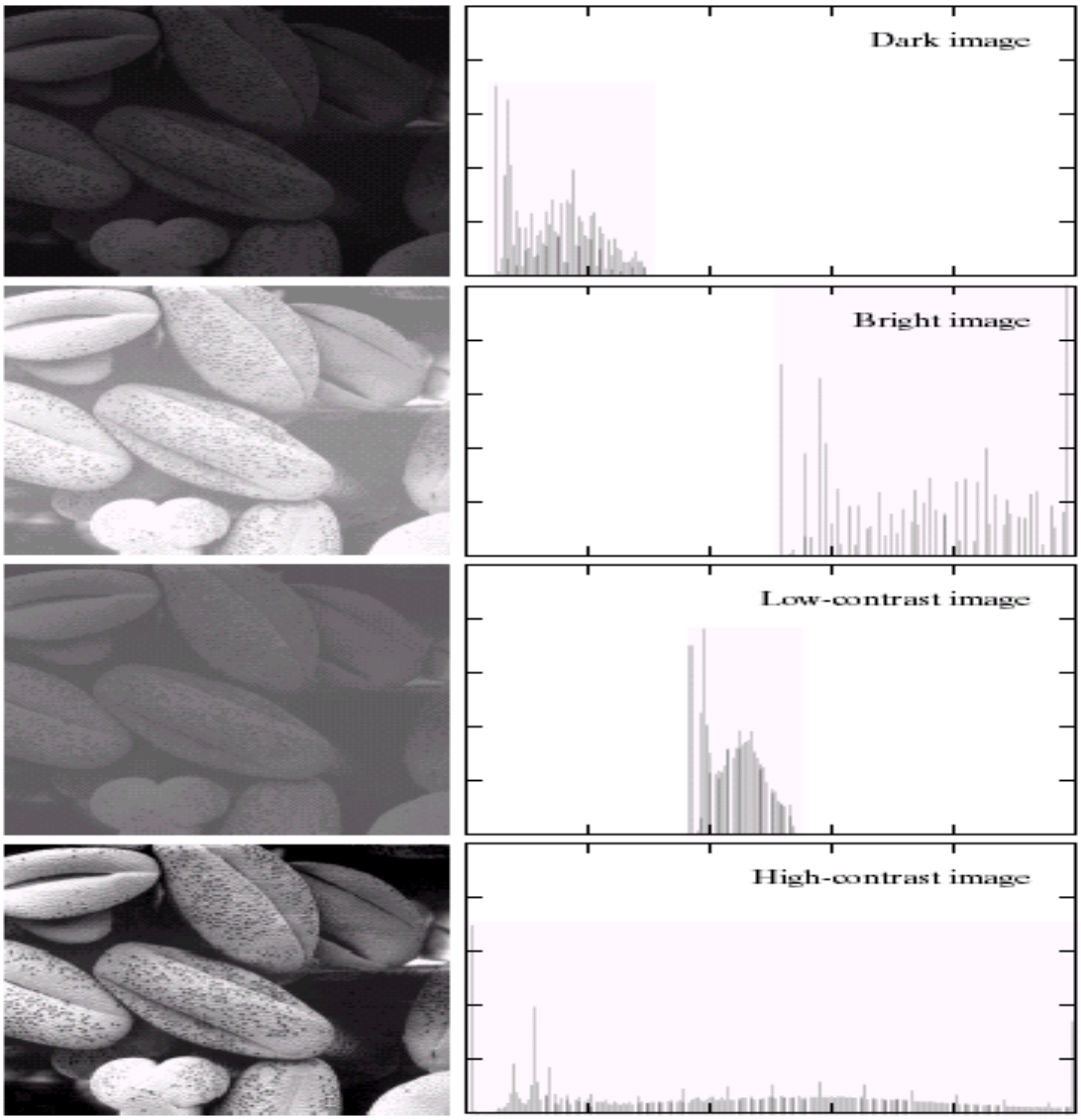
فرم کلی تبدیلات لگاریتمی نشان داده شده در شکل به صورت

$$S=c \log (1+r)$$

است که c ثابت مقیاس بوده و فرض می شود $r \geq 0$.
با استفاده از این تبدیل محدوده تغییرات روشنایی به سوی مقادیر روشن تر فشرده می گردد. این روش برای واضح ساختن تصاویر تاریک می تواند مناسب باشد.

تبدیل نمایی دارای فرم کلی $s=cr^{\gamma}$ می باشد که ثابت های c و γ مثبت هستند.
در این روش مقادیر روشنای بیشتر به سمت سطوح تاریک پیش می رود.

یکی دیگر از تبدیلات، تبدیلات هیستوگرام است که برا ی توضیح آن لازم است ابتدا تعریفی از هیستوگرام داشته باشیم.
هیستوگرام : تعداد نقاطی از تصویر که روشنایی یکسانی دارند را نمایش می هد. هر پیکسل از تصویر دارای روشنایی r_i می باشد. جهت رسم هیستوگرام ، تعداد تمام پیکسل های دارای روشنایی r_i شمرده می شود که با k_i نشان داده می شود. سپس مقادیر k_i برحسب r_i رسم می شود. به شکل حاصل هیستگرام گویند. معمولا هیستوگرام به صورت میله ای رسم می شود. ولی می توان فقط پوش آن را در نظر گرفت و به طور پیوسته رسم نمود. در صورتیکه مقادیر k_i بر تعداد کل نقاط موجود در تصویر تقسیم شود، مقادیر آن متناسب با تابع توزیع احتمال یک متغیر تصادفی خواهد بود. چگونگی توزیع هیستوگرام می تواند نشانگر توصیف کلی از سطوح روشنایی تصویر باشد. برای چشم بشر، بهترین رؤیت زمانی اتفاق می افتد که هیستوگرام دارای توزیع یکنواخت باشد. در شکل ۸-۱ چند تصویر مختلف و هیستوگرام آن نشان داده شده است.



شکل۸-۱

همانطور که گفته شد تغییر هیستوگرام یکی دیگر از روش های بهبود تصاویر در حوزه مکان است. در این روش، هدف یافتن توابع انتقال مناسب به منظور تغییر هیستوگرام تصویر در جهت مطلوب می باشد، هرچند باید همواره نکات زیر را مد نظر داشت:

- محدوده ی تغییرات رو شنایی همچنان ثابت باقی بماند.
- ترتیب نقاط تصویر جدید مشابه تصویر اصلی بماند. یعنی نقطه ای که تاریکترین بوده همچنان تاریکترین بماند.

نکات فوق سبب می شود که رعایت شرایط زیر برای تابع انتقال $s=T(r)$ ضروری باشد:

الف- رابطه ی انتقال باید دارای خاصیت تابع باشد، یعنی به ازای هر r، فقط و فقط یک مقدار s محاسبه گردد. هر چند لزومی ندارد که تابع یک به یک نیز باشد.

ب- تابع انتقال $T(x)$ در فاصله ی $0 \leq r \leq L-1$ به طور یکنوا فزایش یابد چنانکه:

$$r_1 \leq r_2 \Leftrightarrow T(r_1) \leq T(r_2)$$

در اثر این شرط ترتیب روشنایی نقاط در تصویر اصلی و جدید یکسان خواهد بود.

ج- اگر $0 \leq r < L$ باشد مقادیر روشنایی جدید نقاط نیز لازم است که در همان محدوده باشد. یعنی $0 \leq s=T(r) < L$. در نتیجه مقادیر حداقل و حداکثر سطوح روشنایی تغییر نیافته و در همان محدوده باقی می ماند.

همانطور که ذکر شد مقادیر تابع هیستوگرام می تواند شکل تابع احتمال را داشته باشد. پس اگر تعداد نقاط با روشنایی r_i ، برابر k_i باشد، در یک تصویر $N*N$ می توان تابع چگالی احتمال را به صورت رابطه زیر تعریف کرد:

$$\rho_r(r_i) = k_i / N^2 \quad , \quad 0 \leq r_i \leq L \qquad 0 \leq i \leq M$$

که M تعداد سطوح روشنایی متغیر گسسته ی r می باشد. در نتیجه مساله تغییر هیستوگرام ، به تغییر تابع چگالی احتمال منجر می گردد.

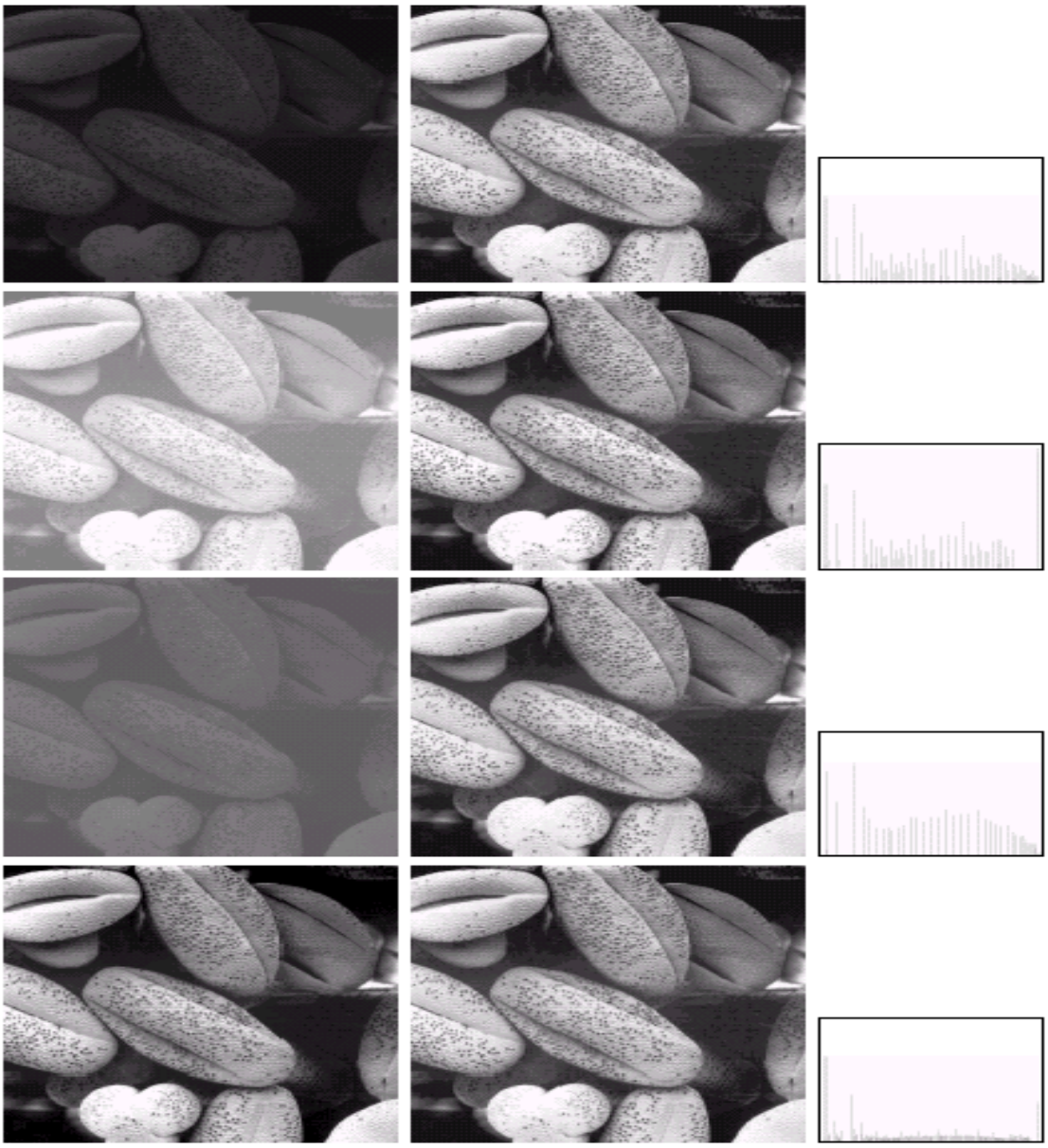
در روش یکسان سازی هیستوگرام ، هدف تغییر هیستوگرام تصویر اصلی به شکل یک هیستوگرام با توزیع یکنواخت می باشد و تابع انتقال آن به صورت :

$$s_i = T(r_i) = \sum_{j=1}^{i} k_j / N^2 = \sum_{j=1}^{i} \rho_r(r_j) \qquad 0 \leq s_i \leq L \qquad 0 \leq i \leq M$$

که با رابطه فوق ، تابع $T(r)$ در شرایط ذکر شده ، صدق می کند.

روش فوق، روش متداول بهبود کیفیت تصویر می باشد. روشن ترین نقطه ی تصویر به بالاترین سطح روشنایی ممکن تغییر می یابد و نتیجه ی آن تقریب هیستوگرام جدید به تابع یکنواخت می باشد.

درشکل ۹-۱ چند نمونه ی مختلف از یک تصویر و تصاویر یکسان ساز شده ی آن به همراه هیستوگرام های مربوطه نشان داده شده است.



شکل ۹-۱

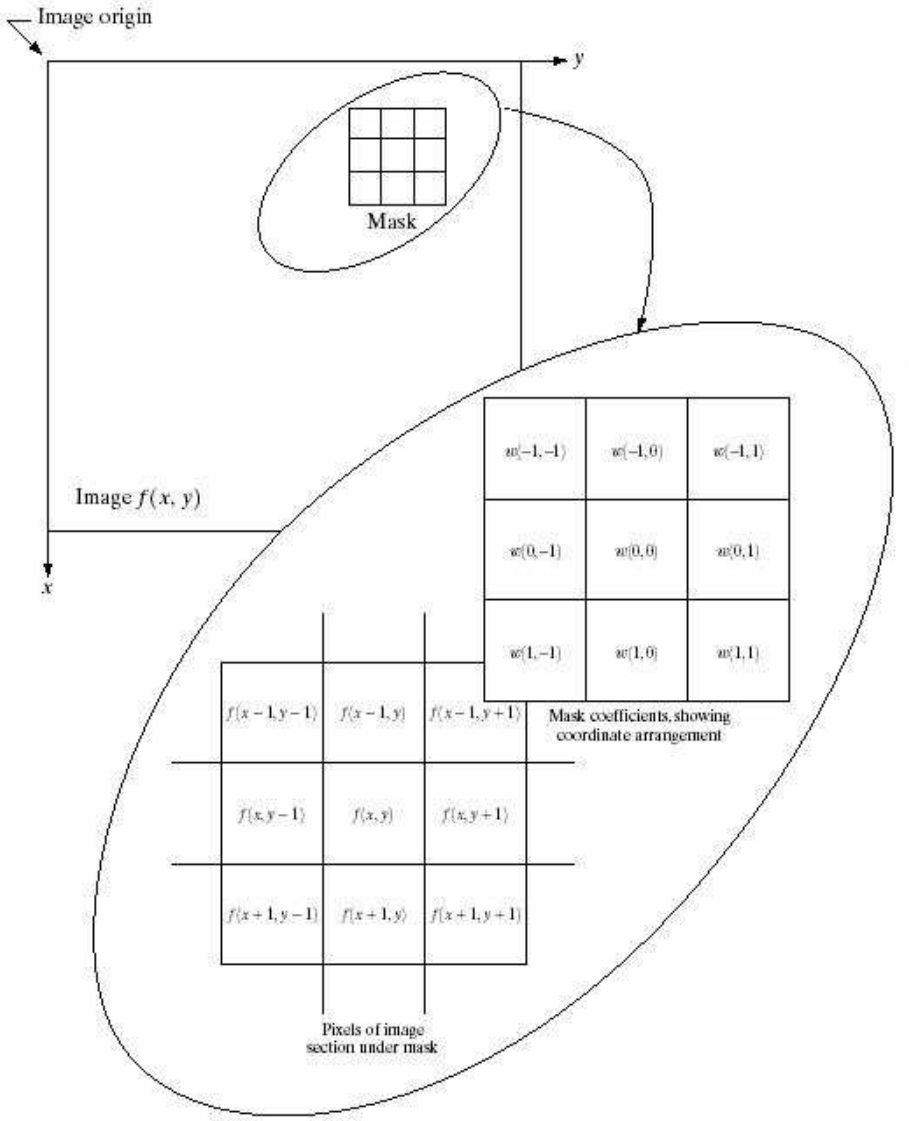
عملیات حسابی- منطقی به روی تصاویر به شیوه ی پیکسل به پیکسل بین ۲ یا چند تصویر اجرا می شود. (البته به جز عملگر NOT که تنها روی یک تصویر اعمال می شود.) به عنوان مثال تفاضل ۲ تصویر، تصویر جدیدی را ایجاد می کند که پیکسلی که در نقطه ی (x,y) قرار دارد حاصل اختلاف پیکسل های واقع در همان مکان درتصاویر اولیه است. همانطور که می دانید AND,OR,NOT قادر به پیاده سازی هر عمل منطقی می باشند پس پیاده سازی همین سه عمل منطقی کافی است.

همانطور که قبلا گفتیم در برخی از روش ها مقدار روشنایی جدید پیکسل $g(x,y)$ به مقدار روشنایی $f(x,y)$ و پیکسل های همسایه ی آن بستگی دارد. این زیر تصویر را **ما سک** یا **پنجره** گویند و به این روش ها فیلتر مکانی گویند. نوع پردازش می تواند خطی یا غیر خطی باشد.

طرز کار این روش در شکل ۱۰-۱ نمایش داده شده است. معمولا روال کار به این صورت است که مرکز ثقل پنجره یعنی $w(0,0)$ به روی پیکسل $f(x,y)$ قرار گرفته و مقدار پیکسل متناظر با مرکز پنجره برای تصویر جدید، $g(x,y)$ ، با توجه به پیکسل های همسایه و وزن های پنجره محاسبه می گردد. در پردازش های خطی استفاده از یک پنجره جهت وزن دهی به پیکسل مورد نظر و همسایگان آن به نحوی صورت می گیرد که مجموع وزن دهی شده به عنوان مقدار جدید تصویر در نظر گرفته می شود. در شکل۱۰-۱ نتیجه فیلتر خطی ، R ، در نقطه ی (x,y) با رابطه ی زیر محاسبه می گردد.

$$R=w(-1,1)f(x-1,y-1)+w(-1,0)f(x-1,y)+\ldots+w(0,0)f(x,y)+ \quad \ldots$$
$$+w(1,0)f(x+1,y)+w(1,1)f(x+1,y+1)$$

برای فیلتر های غیر خطی می توان دو مورد فیلتر بیشینه و فیلتر کمینه را نام برد. که اولی برای یافتن روشن ترین نقاط همسایگی و دومی برای منظور مخالف به کار می رود.



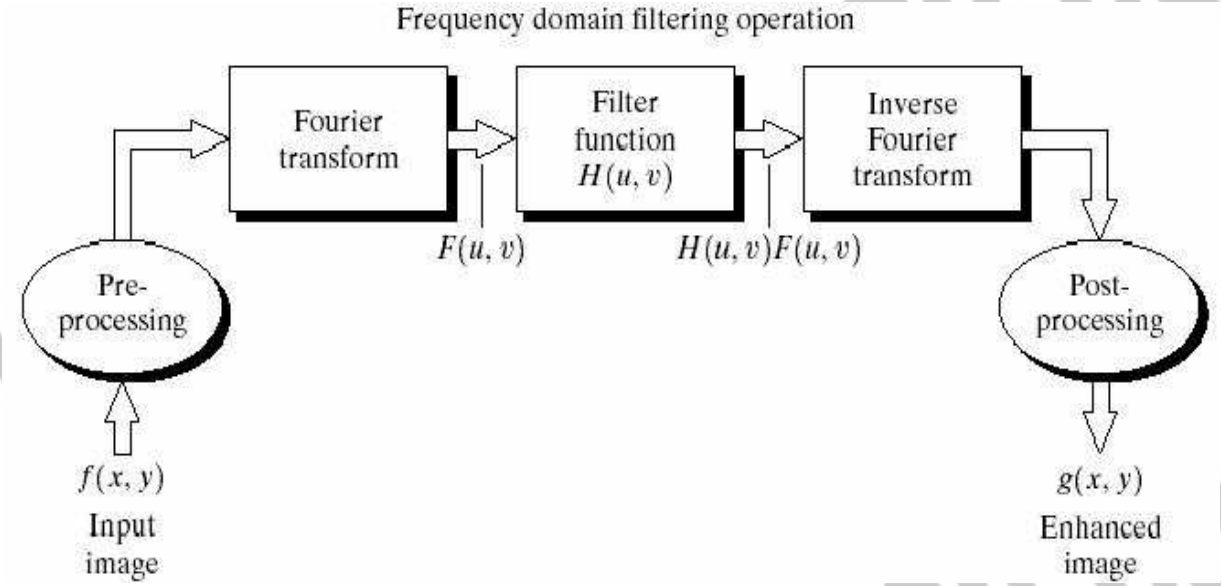
شکل ۱۰-۱

۲-۱-۳-۱ بهبود کیفیت تصویر در حوزه فرکانس :

پردازش درحوزه فرکانس از جمله روش هایی می باشد که مقدار هر پیکسل در تصویر پردازش شده را به تمام پیکسل های موجود در تصویر اصلی مرتبط می سازد. در این مورد، مشخصه ی مطلوبی در حوزه فرکانس به صورت$H(u,v)$ در نظر گرفته می شود.

$$F(u, v) = \mathcal{F}\{f(x, y)\}$$
$$G(u, v) = F(u, v)\, H(u, v)$$
$$g(x, y) = \mathcal{F}^{-1}\{G(u,v)\}$$

بلوک دیاگرام نشان داده شده در شکل ۱۱-۱ نحوه ی پردازش را نشان می دهد. در ادامه به بررسی دو فیلتر پایین گذر و بالا گذر در حوزه فرکانس می پردازیم.



شکل ۱۱-۱

لبه ها و سایر تغییرات سریع در سطوح خاکستری تصویر (نظیر نویز) به میزان زیادی در محتوای فرکانس بالای تبدیل فوریه ی تصویر سهیم هستند. بنابراین مات کردن (هموار کردن) در حوزه فرکانس، با تضعیف محدوده ی مشخصی از مولفه های فرکانس بالای تبدیل فوریه ی تصویر حاصل می گردد. هدف، انتخاب تابع تبدیل $H(u,v)$ است که با تضعیف مولفه های فرکانس بالای $F(u,v)$ ، $G(u,v)$ را بدهد. به این نوع فیلترها، پایین گذر می گویند. به فیلترهای پایین گذر به دلیل کارکرد خاص آنان فیلترهای هموارساز نیز می گویند.

یک فیلتر پایین گذر ایده آل دو بعدی (ILPF) فیلتری است که تابع انتقال آن در رابطه ی

$$H(u, v) = \begin{cases} 1 & \text{if } D(u, v) \le D_0 \\ 0 & \text{if } D(u, v) > D_0 \end{cases}$$

صدق کند. در این رابطه D_0 یک کمیت غیر منفی معین است و $D(u,v)$ فاصله ی نقطه ی (u,v) تا مبدا صفحه ی فرکانسی می باشد ؛ یعنی

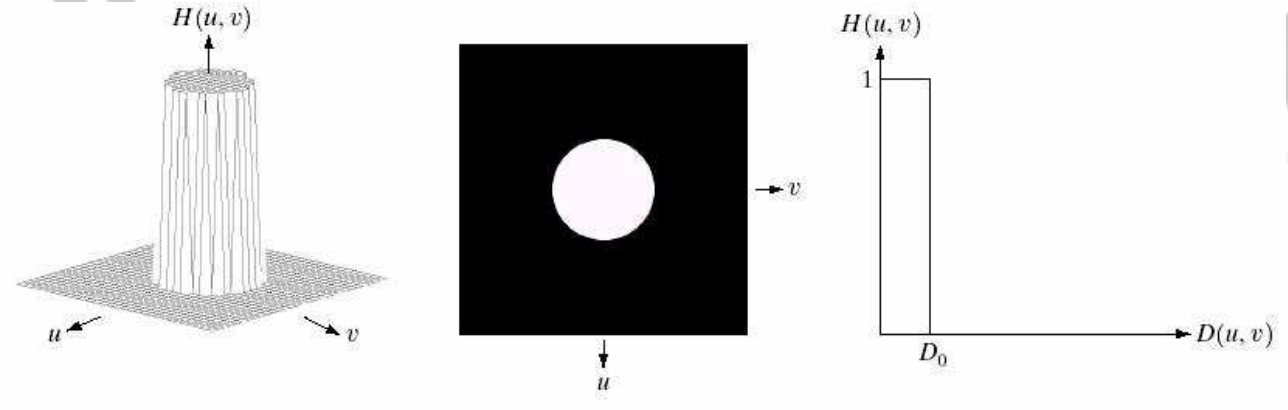
$$D(u, v) = (u^2 + v^2)^{1/2}$$

شکل۱۲-۱ پرسپکتیو سه بعدی، نمایش فیلتر به صورت تصویر و سطح مقطع شعاعی $H(u,v)$ را به صورت تابعی از u و v نشان می دهد. ایده آل بودن فیلتر بدین معناست که تمام فرکانس های درون دایره با شعاع D_0 بدون تضعیف عبور داده می شوند، در حالیکه تمام فرکانس های خارج آن دایره کاملا تضعیف می شوند.

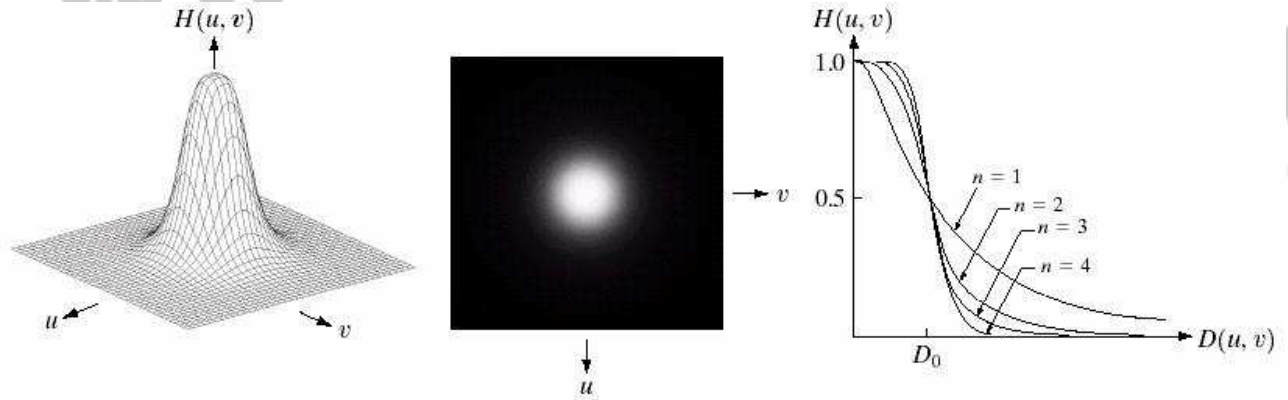
تابع انتقال فیلتر پاین گذر باتروورث (BLPF) از مرتبه ی n و با فرکانس قطع در فاصله ی D_0 از مبدا با رابطه ی

$$H(u,v) = \frac{1}{1+[D(u,v)/D_0]^{2n}}$$

تعریف می گردد. شکل۱۳-۱ تابع انتقال BLPF را نشان می دهد.



شکل ۱۲-۱



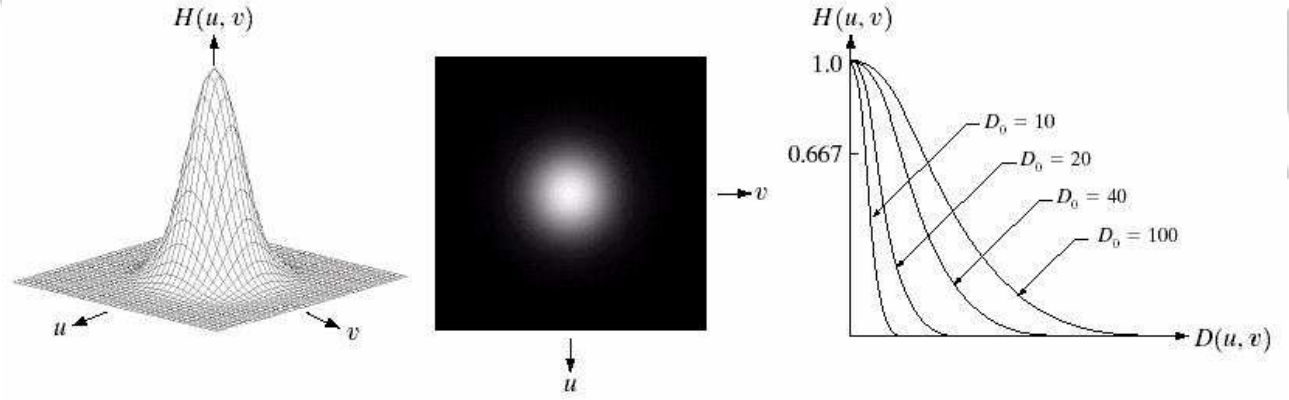
شکل ۱۳-۱

بر خلاف ILPF ، تابع انتقال BLPF ناپیوستگی تیز که باعث تعیین فرکانس قطع مشخصی بین فرکانس های عبور داده شده و فیلتر شده گردد، ندارد. برای فیلتر هایی که توابع انتقال با تغییرات آرام دارند، معمولا منحنی فرکانس قطع در نقاطی که $H(u,v)$ برابر کسر معینی از مقدار بیشینه اش باشد تعریف می شود. در معادله ی فوق، وقتی $D(u,v)= D_0$ باشد، $H(u,v)=0.5$ (۵۰ درصد کمتر از مقدار بیشینه) می باشد. مقدار دیگری که عموما استفاده می شود، $1/\sqrt{2}$ برابر مقدار بیشینه ی $H(u,v)$ است.

یکی دیگر از فیلترهای پایین گذر که بیشترین کاربرد را دارد، فیلتر گوسی (GLPF) یا زنگوله ای شکل می باشد. به دلیل خاصیت توابع نمایی، شکل آن در حوزه ی فرکانس نیز به صورت گوسی باقی می ماند.

$$H(u,v) = e^{-D^2(u,v)/2D_0^2}$$

شکل ۱۴-۱ تابع انتقال GLPF را نشان می دهد.



شکل۱۴-۱

چون لبه ها و سایر تغییرات سریع در سطوح خاکستری با مولفه های فرکانس بالا مرتبط هستند، می توان با یک فرآیند فیلتر کردن بالاگذر که بدون تغییر اطلاعات فرکانس بالای تبدیل فوریه، مولفه های فرکانس پایین را تضعیف می کند، تصویر را تیز کرد. به فیلترهای بالاگذر فیلترهای تیزکننده نیز می گویند.

فیلتر بالاگذر ایده آل (IHPF) دوبعدی، فیلتری است که تابع انتقال آن در رابطه ی

$$H(u,v) = \begin{cases} 0 & if \quad D(u,v) \le D_0 \\ 1 & if \quad D(u,v) > D_0 \end{cases}$$

صدق می کند. در این معادله D_0 فاصله قطع از مبدا فرکانسی می باشد. این فیلتر متضاد فیلتر پایین گذر ایده آل می باشد، زیرا تمام فرکانس های درون دایره ای به شعاع D_0 را تضعیف می کند، در حالیکه تمام فرکانس های خارج از دایره را بدون تضعیف عبور می دهد.

تابع انتقال فیلتر بالاگذر باتروورث (BHPF) از مرتبه ی n و با فرکانس قطع به فاصله یD_0 از مبدا با رابطه ی

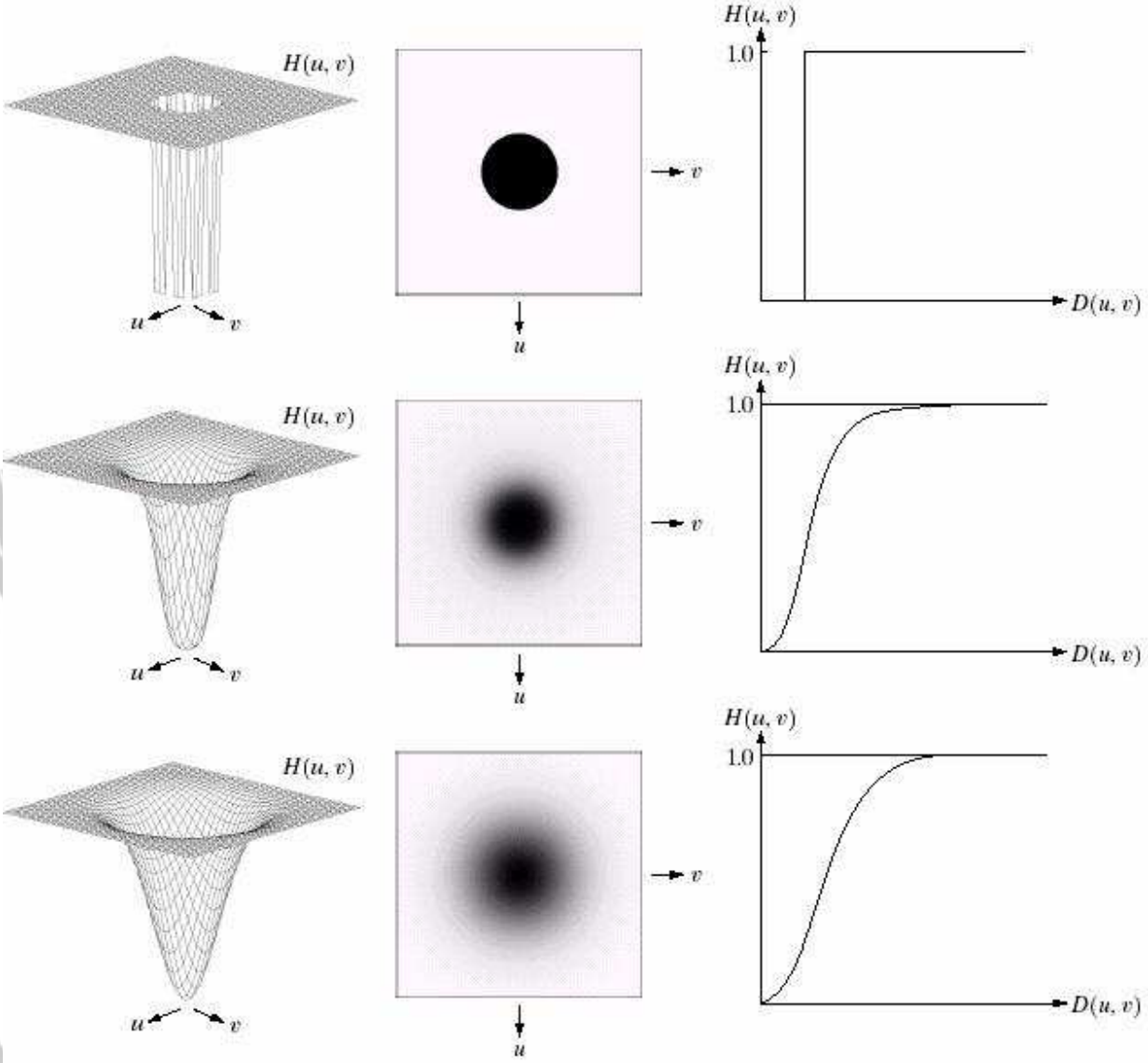
$$H(u,v)=\frac{1}{1+\left[D_\circ / D(u,v)\right]^{2n}}$$

تعریف می شود.

تابع انتقال فیلتر بالاگذر گوسی (GHPF) به صورت زیر تعریف می شود.

$$H(u,v)=1-e^{-D^2(u,v)/2D_\circ^2}$$

شکل ۱۵-۱ پرسپکتیو سه بعدی، نمایش فیلتر به صورت تصویر و سطح مقطع شعاعی هر سه فیلتر بالاگذر را نشان می دهد. ردیف بالا فیلتر ایده آل ، ردیف وسط فیلتر باتروورث و ردیف پایین فیلتر گوسی می باشد.



شکل۱۵-۱

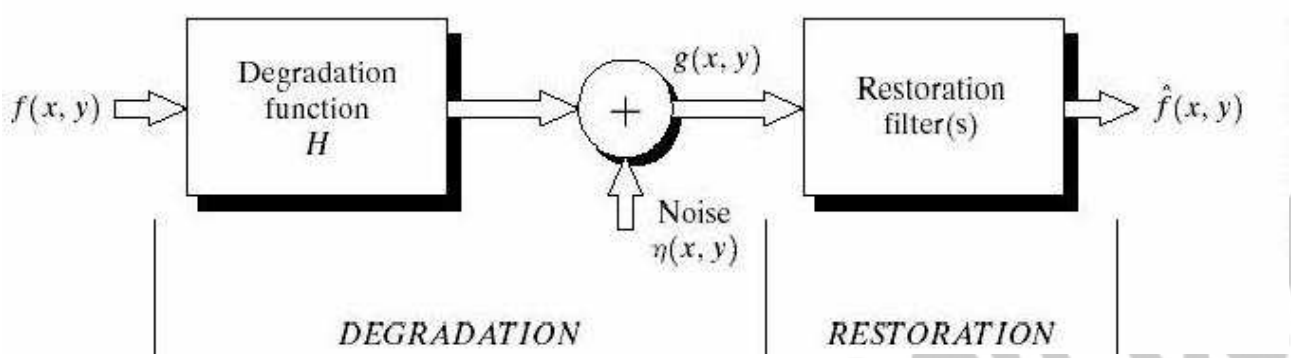
۲-۳-۱ بازسازی تصاویر:

یکی دیگر از تکنیک های پردازش تصویر، بازسازی تصاویر می باشد. هدف از بازسازی تصاویر استفاده از فرآیندهایی جهت به دست آوردن یک تصویر مطلوب از یک تصویر تخریب شده است. اگرچه در مواقعی نمی توان به تصویر مطلوب رسید ولی می توان تاثیر اختلال را کمینه کرد به نحوی که گاهی تیز کردن لبه های تصویر نیز می تواند کافی باشد. تخریب تصویر به علت تنظیم نامناسب دوربین، حرکت نسبی دوربین و جسم مورد تصویربرداری، انعکاس ناخواسته ی نورها از منابع غیر قابل کنترل، ایده آل نبودن سیستم های تصویر برداری و مخابراتی و ... صورت می گیرد.

روش های مختلفی در پردازش تصاویر تخریب شده به منظور بازسازی آنان موردتوجه قرار می گیرند که از جمله ی آن ها می توان به موارد زیر اشاره کرد :

- از بین بردن مات شدگی در تصویر
- حذف نویز در تصویر
- بهبود تمایز و دیگر معیارهای رؤیت تصویر

تعریف مدلی از تخریب می تواند در درک مفاهیم و همچنین به یافتن روابطی برای بازسازی تصویر کمک کند. در یک حالت کلی و بدون ایجاد خطا های بزرگ می توان یک سیستم تخریب کننده و بازسازی را به صورت شکل ۱۶-۱ مدل سازی نمود. همانطور که در شکل نشان داده شده است، تابع تخریب به همراه نویز جمع شونده بر روی تصویر ورودی $f(x,y)$ اعمال می شود تا تصویر تخریب شده ی $g(x,y)$ تشکیل شود. با معلوم بودن $g(x,y)$ و دانستن تابع تخریب H و نوع نویز جمع شونده $\eta(x,y)$ می توان $\hat{f}(x,y)$ ، تقریبی از تصویر اصلی را به دست آورد.



شکل۱۶-۱

اگر H یک فرآیند خطی و تغییر ناپذیر با مکان باشد، آنگاه تصویر تخریب شده در حوزه ی مکان با رابطه زیر مشخص می شود :

$g(x,y)=h(x,y)* f(x,y) +\eta(x,y)$

با فرض $H=1$ ، تنها با تخریب توسط نویز سروکار خواهیم داشت. در ادامه به بررسی چند تابع چگالی احتمال نویز ، که اهمیت بیشتری دارند می پردازیم.

نویز گوسی: به دلیل تشابه رابطه ی آن در هر دو حوزه ی مکان و فرکانس، نویز گوسی (نرمال) در عمل بسیار مورد استفاده قرار می گیرد. تابع چگالی احتمال (PDF) متغیر تصادفی z با توزیع نرمال با رابطه ی زیر مشخص می شود :

$$P(z) = \frac{1}{\sqrt{2\Pi\, 6}} e^{-(z-\mu)^2 / z\delta^2}$$

که z نشان دهنده ی سطح خاکستری، μ میانگینِ z و σ^2 واریانس **آن** می باشد. نمودار این تابع در شکل ۱۷-۱ نشان داده شده است.

نویز ریلی: تابع چگالی احتمال نویز ریلی به صورت زیر تعریف می شود:

$$P(z) = \begin{cases} \frac{2}{b}(z-a)e^{-(z-a)^2/b} & for \quad z \geq a \\ 0 & for \quad z < a \end{cases}$$

و میانگین و واریانس آن با روابط زیر تعیین می گردد.

$$\mu = a + \sqrt{\pi b / 4} \qquad \delta^2 = \frac{b(4-\pi)}{4}$$

نمودار چگالی ریلی در شکل۱۷-۱ نشان داده شده است.

نویز ارلانگ (گاما) : تابع چگالی احتمال نویز گاما به صورت زیر تعریف می شود:

$$P(z) = \begin{cases} \frac{a^b z^{b-1}}{(b-1)} e^{-az} & for \quad z \geq 0 \\ 0 & for \quad z < 0 \end{cases}$$

در این رابطه $a > 0$ و b یک عدد صحیح مثبت می باشد. میانگین و واریانس این تابع چگالی در زیر آمده است و نمودار آن را در شکل ۱۷-۱ مشاهده می کنید.

$$\mu = \frac{b}{a} \qquad 6^2 = \frac{b}{a^2}$$

نویز نمایی : تابع چگالی، میانگین و واریانس نویز نمایی به صورت زیر است :

$$P(z)=\begin{cases} ae^{-az} & for \quad z\geq 0 \\ 0 & for \quad z<0 \end{cases}$$

$$\mu=\frac{1}{a} \qquad 6^2=\frac{1}{a^2}$$

نویز نمایی حالت خاصی از نویز گاما است، وقتی که b=1 باشد. شکل ۱۷-۱ نمودار تابع چگالی آن را نشان می دهد.

نویز یکنواخت : تابع چگالی ، میانگین و واریانس نویز نمایی به صورت زیر است :

$$p(z)=\begin{cases} \frac{1}{b-a} & if \quad a\leq z\leq b \\ 0 & otherwise \end{cases}$$

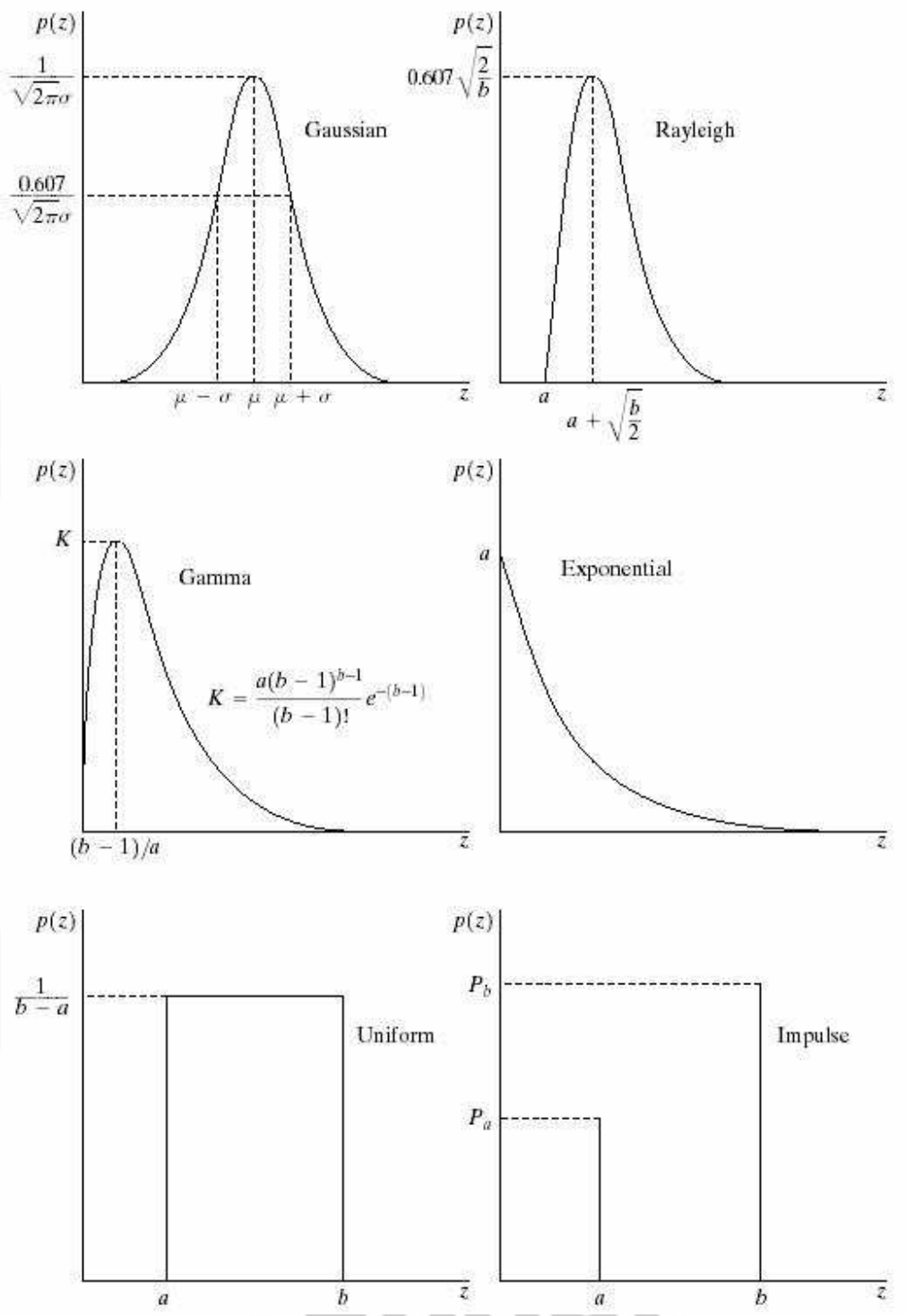
$$\mu=\frac{a+b}{2} \qquad 6^2=\frac{(b-a)^2}{12}$$

و نمودار تابع چگالی آن در شکل ۱۷-۱ آمده است.

نویز ضربه (نمک و فلفل) : تابع چگالی احتمال نویز ضربه به صورت زیر است :

$$p(z)=\begin{cases} P_a & for \quad z=a \\ P_b & for \quad z=b \\ 0 & otherwise \end{cases}$$

نمودار تابع چگالی آن در شکل ۱۷-۱ نشان داده شده است.



شکل۱۷-۱

در ادامه به بررسی چند فیلتر که جهت حذف نویز به کار می رود، خواهیم پرداخت.

فیلتر میانگین ریاضی : این فیلتر به صورت خطی عمل می کند و در نتیجه کارکرد آن مانند کانولوشن بوده و در حوزه ی فرکانس نیز قابل بررسی است.

$$\hat{f}(x,y)=\frac{1}{mn}\sum_{(s,t)\in S_{xy}} g(s,t)$$

که S_{xy} نشان دهنده ی مختصات زیر تصویر و m و n ابعاد زیر تصویر می باشد. در این صورت، تصویر جدید مقداری محو شدگی در مرزهای خود خواهد داشت. فیلتر میانگین در حوزه ی فرکانس مشابه یک فیلتر پایین گذر عمل می کند. هر چه میزان نویز بیشتر باشد، یا اشیاء درون تصویر از اندازه ی بزرگتری برخوردار باشند، لازم است از پنجره بزرگتری استفاده کرد که سبب افزایش خاصیت محو شدگی شده و کیفیت ظاهری تصویر را کاهش می دهد، هرچند که گاهی نسبت توان سیگنال به نویز را افزایش می دهد.

فیلتر میانگین هندسی : این فیلتر به صورت زیر تعریف می شود :

$$\hat{f}(x,y)=\left[\prod_{(s,t)\in S_{xy}} g(s,t)\right]^{\frac{1}{mn}}$$

فیلتر میانه: این فیلتر یکی از مؤثرترین فیلترهای غیرخطی جهت حذف نویز در تصاویر می باشد. پنجره ای مربعی به اندازه ی $(2k+1)\times(2k+1)$ - که k عددی طبیعی است – در نظر گرفته شده و مرکز ثقل آن به روی تک تک پیکسل ها ی تصویر شیفت داده می شود. آنگاه مقدار میانی موجود داخل پنجره به عنوان پاسخ خروجی برای پیکسل مرکزی در نظر گرفته می شود.

$$\hat{f}(x,y)=\underset{(s,t)\in S_{xy}}{median}\{g(s,t)\}$$

این فیلتر توانایی حذف نویز ضربه را دارد، هر چند توان کاهش نویز گوسی را به خوبی دارا نیست. از دیگر مزایای ویژه ی این فیلتر آن است که مقدار جدید روشنایی در تصویر ایجاد نمی کند. مشکل اصلی این فیلتر جابه جایی محل لبه به اندازه یک یا دو پیکسل می باشد.

فیلترهای بیشینه و کمینه : روابط این فیلترها به صورت زیر است :

$$\hat{f}(x,y)=\max_{(s,t)\in S_{xy}}\{g(s,t)\} \qquad \hat{f}(x,y)=\min_{(s,t)\in S_{xy}}\{g(s,t)\}$$

فیلتر نقطه میانی : این فیلتر به صورت زیر تعریف می شود :

$$\hat{f}(x,y)\frac{1}{2}\left[\max_{(s,t)\in s_{xy}}\{g(s,t)\}+\min_{(s,t)\in s_{xy}}\{g(s,t)\}\right]$$

۳-۳-۱ کدینگ و فشرده سازی تصویر:

یکی دیگر از روش ها ی پردازش تصویر، کدینگ و فشرده سازی می باشد که در این قسمت مختصرا به آن می پردازیم.
در این روش سعی داریم داده های اضافی را حذف کنیم. ذکر این نکته ضروری است که **داده** با **اطلاعات** متفاوت است. در واقع از داده استفاده می شود تا به اطلاعات برسیم. معمولا تعداد داده ها بیشتر از اطلاعات داخل آن هاست و ما در این تکنیک سعی میکنیم که داده هایی که اطلاعات ندارد و redundancy خوانده می شود را حذف کنیم.

به سه دلیل ممکن است از کدینگ استفاده شود:

- فشرده سازی اطلاعات تصویر برای کاهش حجم آنها
- انتقال یا ارسال تصاویر
- استخراج ویژگی ها

روش های فشرده سازی به دو گروه تقسیم می شود:

- Lossless: هیچگونه اطلاعاتی از بین نمی رود.
- Lossy: مقداری از اطلاعات از بین می رود ولی تصویر حاصل، تقریب خوبی ا ز تصویر اولیه است.

معیارهای فشرده سازی:

- سرعت فشرده سازی: سعی در افزایش سرعت فشرده سازی است. مخصوصا در موارد real time این نیاز افزایش می یابد.
- با احتمال خطای کم بتوان تصویر را فشرده کرد.
- دسترسی به تصویر به طور تصادفی : به این معنا که در تصاویر ویدیویی نیاز نباشد برای دسترسی به فریم n ام ابتدا به فریم n-1 ام دسترسی داشت.
- هزینه سخت افزاری : بتوان با صرف هزینه ی کم نتیجه خوبی به دست آورد.

برای انجام عملیات فشرده سازی از الگوریتم ها و روش های متفاوتی استفاده می شود که با توجه به معیار های مذکور از یک روش خاص و یا ترکیبی از روش ها، برای موارد متفاوت استفاده می شود.

از جمله الگوریتم‌های کدینگ، می توان به **کد هافمن** اشاره کرد. ایده‌ی اصلی در اینجا این است که برای سمبل‌ها، کلمات کد یونیک یا یکتا تعریف کنیم که تحت معیارهای فشرده سازی باشد.

روش:

قدم اول:

۱- ابتدا سطوح خاکستری را براساس احتمال وقوعشان مرتب می‌کنیم.

۲- سپس دو احتمال کوچکتر را با هم جمع می‌کنیم.

۳- مجدداً مرتب سازی (sort) می‌کنیم.

۴- تکرار مراحل تا رسیدن به تنها دو احتمال

قدم دوم:

به احتمال بیشتر کد صفر می دهیم و به احتمال کمتر کد ۱

به سمت عقب حرکت می‌کنیم و ۰ و ۱ اضافه می‌کنیم تا کلمات کد به دست بیاید.

با استفاده از این روش سطوح خاکستری با احتمال وقوع بیشتر با کد کوچکتری کد می‌شوند و آنهایی که احتمال وقوع کمتری دارند کلمات کد بزرگتری خواهند داشت که نهایتاً کل تصویر فشرده خواهد شد.

در زیر مثالی را مشاهده می کنید که اطلاعات را با استفاده از هافمن کد کرده است.

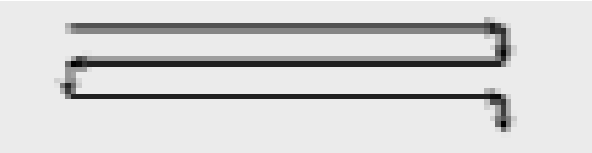
r_k	$p(r_k)$	node 1	node 2	node 3	node 4	node 5	node 6
1	0.4	→ 0.4	0.4	0.4	0.4	0.4	↘ 0.6
4	0.3	→ 0.3	0.3	0.3	0.3	0.3	↗ 0.4
0	0.1	→ 0.1	0.1	0.1	↘ 0.2	→ 0.3	
5	0.1	→ 0.1	0.1	0.1	↗ 0.1	↗	
3	0.05	→ 0.05	0.05	→ 0.1			
2	0.03	→ 0.03	→ 0.05	↗			
6	0.01	→ 0.02	↗				
7	0.01	↗					
	$L_{avg} = 3$						
r_k	code	node 1	node 2	node 3	node 4	node 5	node 6
1	1	1	1	1	1	1	↙ 0
4	00	00	00	00	00	00	↖ 1
0	011	011	011	011	↙ 010	← 01	
5	0100	0100	0100	0100	↖ 011	↙	
3	01010	01010	01010	← 0101			
2	010110	010110	← 01011	↙			
6	0101110	← 010111	↙				
7	0101111	↙					

یکی دیگر از روش‌ها Run – leagth codign است که به صورت زیر عمل می‌کند. هر کلمه‌ی کد از جفت (g , l) _ که g سطح خاکستری و l تعداد پیکسل‌ها با سطح خاکستری g است_ تشکیل شده است. درزیر مثا لی از جفت (g , l) ها است.

56 56 56 82 82 82 83 80
56 56 56 56 56 80 80 80

(56, 3)(82, 3)(83, 1) (80, 4)(56, 5).

کد به صورت سطر به سطر محاسبه می شود



شکل ۱۸-۱

در تصاویر باینری روش به این صورت است که اولین بیت را مطابق با اولین سطح خاکستری، صفر یا یک قرار می‌دهیم سپس تعداد سطوح خاکستری بعدی را به ترتیب می‌نویسیم. در شکل زیر ۱ برای سطح خاکستری سیاه و صفر برای سطح خاکستری سفید است.

شکل۱۹-۱

آنچه که به عنوان کد ذخیره می شود به این صورت است:

1,7,5,8,3,1…..

در واقع از یک آرایه ی یک بعدی برای کد کردن تصویر باینری که دو بعدی است استفاده می کنیم.

این روش در تصاویر باینری که از صفر و یک تشکیل شده (دو سطح سیاه و سفید) کاربرد دارد و مخصوصاً در فاکس استفاده می‌شود. از مشکلات آن انتشار خطا است. به این معنا که اگر در یک بیت اشتباه کنیم، این اشتباه در ادامه‌ی کار تاثیر می‌گذارد و این خطا اصطلاحاً منتشر می شود.

روش های مختلف دیگری نیز برای عملیات کدینگ وجود دارد که در اینجا تنها به ذکر نام چند نمونه از آنها اکتفا شده است.

۱- DCMP

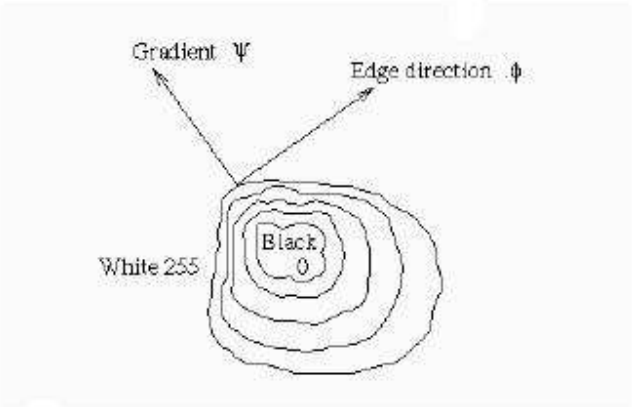
۲- Bit–plane coding

۳- pyscho

۴-۳-۱ قطعه بندی تصویر (Image segmentation) :

قطعه بندی تصویر مهمترین مرحله برای آنالیز تصویر می باشد. هدف این است که مجموعه ای از پیکسل ها که همبستگی شدیدی بین آن هاست به عنوان یک قطعه بگیریم. پس از این مرحله با تحلیل و آنالیز قطعات مختلف می توان اجزا و object ها را شناسایی کرد که مرحله تشخیص، مربوط به بینایی ماشین است.

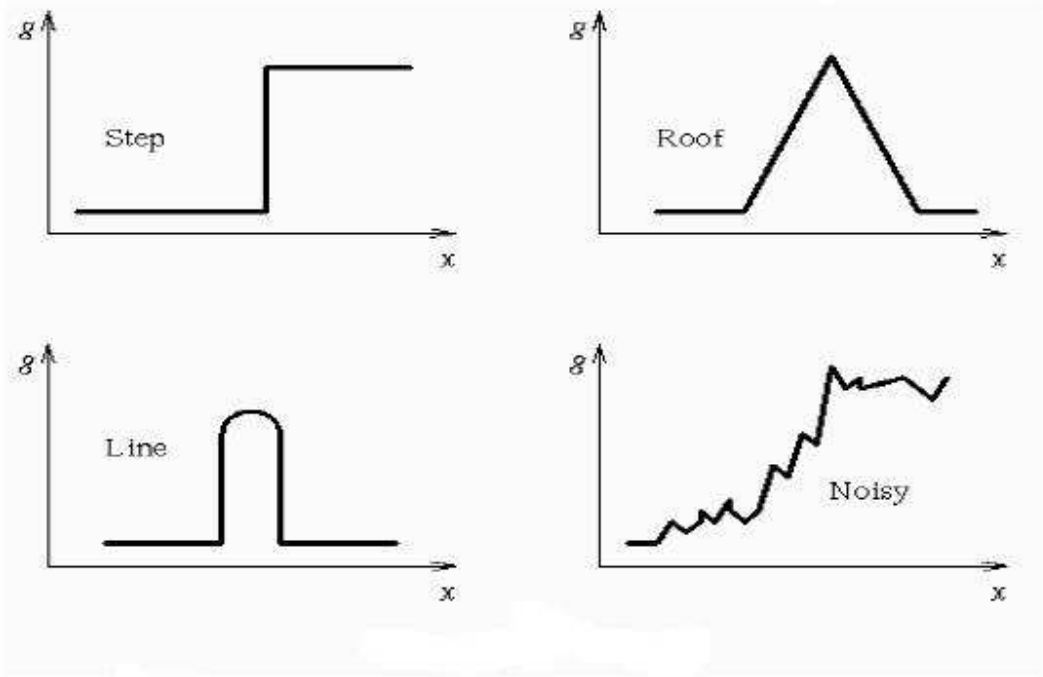
تشخیص لبه یا edge detection یکی از روش های قطعه بندی است. تغییرات فیزیکی به صورت تغییر رنگ و تغییر شدت روشنایی به صورت لبه در تصویر نمایان می شود. در محیطی با مقادیر پیوسته، مشتق، تغییرات ناگهانی و شدت آن را مشخص می کند. در محیط گسسته، محاسبات تغییرات نسبت به پیکسل های مجاور، تقریبی از مشتق را نمایان می سازد. در محاسبه ی بردار گرادیان، دو مؤلفه ی اندازه و جهت مهم است. باید توجه داشت که جهت لبه دارای چرخش 90° در جهت حرکت عقربه های ساعت نسبت جهت مشتق مطا بق شکل ۲۰-۱ می باشد.



نمائی از جهت‌های مشتق و لبه ها. مسیرهای بسته نشانگر نواحی با مقادیر روشنایی یکسان می‌باشند.

شکل ۲۰-۱

در شکل زیر نیز چند نمونه از لبه ها در حالت سیگنال پیوسته ی یک بعدی نشان داده شده است.



چند نمونه از تغییرات لبه.

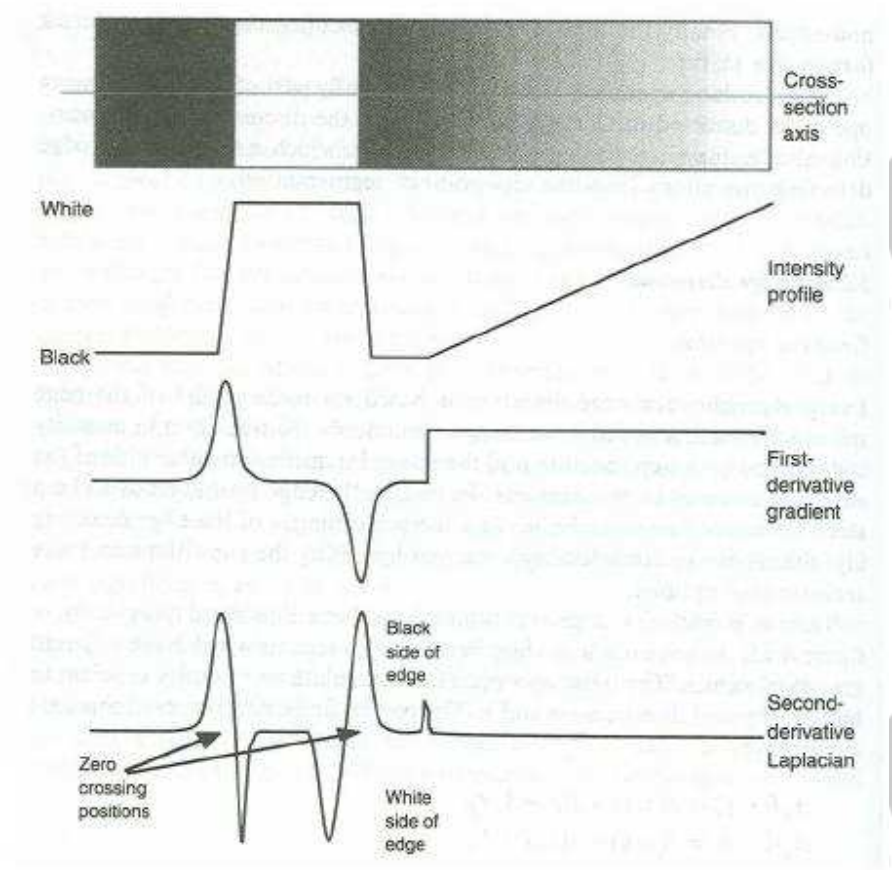
شکل ۲۱-۱

روش های زیادی برای تشخیص لبه وجو دارد. با این حال اکثریت متدهای مختلف در دو گروه طبقه بندی می شوند:

Gradient : این متد لبه ها را با استتفاده از جستجوی مینیمم و ماکزیمم در مشتتق اول یک تصویر، مشخص می کند.

Laplacian : این متد صفر ها را در مشتق دوم جستجو می کند تا لبه را بیابد.

در شکل زیر یک تصویر با تغییرات روشنایی به صورت افقی مشاهده می گردد. یک ردیف از تصویر مذکور به همراه مشتق اول و دوم آن نشان داده شده است.



تصویری با تغییرات روشنایی در راستای افق و مشتقات اول و دوم.

شکل ۲۲-۱

همانطور که در شکل مشخص شده است در نقطه ی مرزی یک لبه و یا در نقطه ی وسط بین دو ناحیه ی هموار دو طرف لبه، اندازه ی مشتق اول به ماکزیمم رسیده و مشتق دوم نیز یک نقطه ی عبور از صفر خواهد داشت. حال

تعدادی از الگوریتم های تشخیص لبه را در دو گروه Gradient (مشتق اول) و Laplacian (مشتق دوم) معرفی می کنیم.

۱-۴-۳-۱ روش های مبتنی بر از مشتق اول:

در حالت سیگنال دو بعدی، باید گرادیان *f(x,y)* براساس رابطه‌ی ۱ محاسبه گردد.

$$|\nabla f(x,y)| = \sqrt{\left(\frac{\partial f(x,y)}{\partial x}\right)^2 + \left(\frac{\partial f(x,y)}{\partial y}\right)^2} = \sqrt{(f'x(x,y))^2 + (f'y(x,y))^2} \qquad (۱)$$

$$\psi = \tan^{-1}\left(\frac{f'y(x,y)}{f'x(x,y)}\right), \Phi = \psi - \frac{\pi}{2}$$

تقریب گرادیان در مورد تصاویر گسسته به شکل‌های متفاوتی خواهد بود. معمولاً، دو دریچه، جهت تقریب گرادیان در جهت های افق و قائم در نظر گرفته می‌شود و محاسبات با استفاده از روابط زیر صورت می‌گیرد.

تقریب مشتق نسبت به جهت افق:

$$f_x(m,n) = f(m,n) * h_x(m,n)$$

تقریب مشتق نسبت به جهت عمودی:

$$f_x(m,n) = f(m,n) * h_x(m,n)$$

و

$$|\nabla f(m,n)| = \sqrt{{f_x}^2(m,n) + {f_y}^2(m,n)}$$

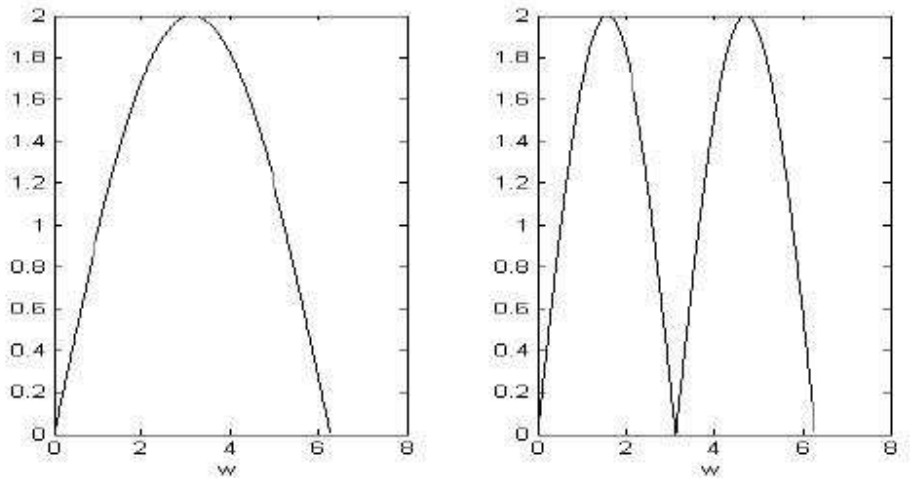
جهت توابع $h_x(m,n)$ و $h_y(m,n)$ شکل‌های مختلفی در نظر گرفته شده است که سبب پیدایش روش‌های مختلفی می‌شود. در تمام روش‌ها، مقدار $|\nabla f(m,n)|$ با استفاده از رابطه‌ی۱ محاسبه می‌گردد که در ادامه‌ی مطلب به چند روش آن اشاره می‌گردد.

فیلترهای مشتق گیر پایه

$$h_x(m,n)=[1 \quad -1], hy(m,n)=\begin{bmatrix}1\\-1\end{bmatrix}, ORhx(m,n)=[1 \quad 0 \quad -1\ ], hy(m,n)=\begin{bmatrix}1\\0\\-1\end{bmatrix}$$

$$h_x(m,n)=[1 \quad -1] \overset{\Im}{\Leftrightarrow} H_x(\omega)2\left|\sin(\omega/2)\right|.e^{j(\pi-\omega)/2}$$

$$h_x(m,n)=[1 \quad 0 \quad -1] \overset{\Im}{\Leftrightarrow} H_x(\omega)=2\left|\sin(\omega)\right|.e^{j\pi/2}$$



ب الف

الف- اندازهٔ پاسخ فرکانسی $|H(\omega)|=2|\mathrm{Sin}(\omega)|$ ب- اندازهٔ پاسخ فرکانسی $|H(\omega)|=2|\mathrm{Sin}(\omega/2)|$

شکل ۲۳-۱

شکل ۲۳-۱ نشان می دهد که فیلتر اول، فرکانس‌های بسیار بالا (حوالی π) را جداسازی می‌کند، در حالی که فیلتر دوم، فرکانس‌های میانی (حوالی $\pi/2$ و $3\pi/2$) را جدا سازی می‌نماید که در واقع مشابه فیلتر میان گذر عمل می‌کند. فیلتر اول به نویز که دارای مؤلفه های فرکانسی بسیار بالا است. شدیداً حساس بوده و در نتیجه نقشه‌ی لبه‌ی داده شده

بسیار نویزی خواهد بود، در حالی که بعضی از لبه های مهم و اصلی تصویر که ناشی از وجود نواحی طبیعی می‌باشند، ممکن است که آشکار نگردد.

فیلترهای مشتق گیر قطری

$$h_1(m,n)=\begin{bmatrix}0 & 1 & 1\\ -1 & 0 & 1\\ -1 & -1 & 0\end{bmatrix},\quad h_2(m,n)=\begin{bmatrix}1 & 1 & 0\\ 1 & 0 & -1\\ 0 & -1 & -1\end{bmatrix}$$

که $h_1(m,n)$ خطوط موجود در جهت ۴۵ درجه و $h2(m,n)$ لبه‌ی موجود در جهت ۴۵- درجه را آشکار می‌کند.

فیلترهای مشتق گیر prewitt

روابط حاکم بر این نوع فیلترها در راوابط زیر نشان داده شده اند.

$$h_x(m,n)=\begin{bmatrix}1 & 0 & -1\\ 1 & 0 & -1\\ 1 & 0 & -1\end{bmatrix},\quad h_y(m,n)=\begin{bmatrix}1 & 1 & 1\\ 0 & 0 & 0\\ -1 & -1 & -1\end{bmatrix},$$

$$h_{+45}(m,n)=\begin{bmatrix}1 & 1 & 0\\ 1 & 0 & -1\\ 0 & -1 & -1\end{bmatrix},\quad h_{-45}(m,n)=\begin{bmatrix}-1 & -1 & 0\\ -1 & 0 & 1\\ 0 & 1 & 1\end{bmatrix}$$

بطور مثال لازم است که توجه نمود h_x گرادیان در جهت افق را گرفته و وجود لبه در جهت قائم را مشخص می‌سازد. $h_{+45}(m,n)$ و $h_{-45}(m,n)$ نیز لبه‌های قطری را مشخص می‌کنند.

فیلترهای مشتق گیر sobel

$$h_x(m,n)=\begin{bmatrix}1 & 0 & -1\\ 2 & 0 & -2\\ 1 & 0 & -1\end{bmatrix},\quad h_y(m,n)=\begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix},$$

$$h_{+45}(m,n)=\begin{bmatrix}2 & 1 & 0\\ 1 & 0 & -1\\ 0 & -1 & -2\end{bmatrix},\quad h_{-45}(m,n)=\begin{bmatrix}0 & 1 & 2\\ -1 & 0 & 1\\ -2 & -1 & 0\end{bmatrix}$$

فیلترهای مشتق گیر Robert

$$h_x(m,n)=\begin{bmatrix}0 & 0 & 1\\ 0 & -1 & 0\\ 0 & 0 & 0\end{bmatrix},\quad h_y(m,n)=\begin{bmatrix}0 & 1 & 0\\ 0 & 0 & -1\\ 0 & 0 & 0\end{bmatrix},$$

در صورتی که هدف تهیه‌ی یک تصویر باینری از لبه‌ها (نقشه‌ی لبه) باشد لازم است مقدار مطلق مشتق با سطح آستانه‌ی مشخصی مقایسه گردد تا وجود یا عدم وجود لبه مشخص شود. در صورتی که سطح آستانه بزرگ در نظر گرفته شود، تغییرات کوچک آشکار نمی‌گردد، و در صورتی که سطح آستانه کوچک نیز اختیار گردد، نقشه لبه بسیار نویزی خواهد گشت. گاهی نیز ممکن است که یک لبه با پهنای زیادی آشکار گردد که در این صورت لازم است از روش‌های باریک سازی لبه استفاده شود. نتیجه‌ی آشکار سازی لبه‌های تصویر فیلمبردار توسط سه فیلتر فوق جهت مقایسه در شکل ۲۴-۱ نشان داده شده است که نشانگر نتایج تقریباً یکسان آنان می‌باشد.

الف ب ج

آشکارسازی لبه توسط روش های مختلف، با سطح آستانهٔ ۰/۱٤. الف- روش Sobel. ب- روش Prewitt. ج- روش Roberts. مقادیر روشنایی بین صفر و یک می‌باشد.

شکل ۲۴-۱

۲-۴-۳-۱ روش مبتنی بر مشتق دوم یا لاپلاس:

روش دیگر آشکارسازی لبه، استفاده از مشتق دوم و جداسازی نقاط عبور از صفر آن می باشد. این نقاط را می توان توسط تغییر علامت مشتق اول نیز تشخیص داد. در این روش اگر هیچ کنترل اولیه ای بر روی مقادیر مشتق اول و دوم وجود نداشته باشد، کوچکترین تغییر نویزی موجب عبور از صفر مشتق دوم و یا تغییر علامت مشتق اول خواهد شد. که نشانگر حساسیت بالای این روش به نویز است. برای تقریب لاپلاسین تصویر، در حالت پیوسته از رابطه ی زیر استفاده می شود:

$$\left|\nabla^2 f(x,y)\right| = \sqrt{\left(\frac{\partial f^2(x,y)}{\partial x^2}\right)^2 + \left(\frac{\partial f^2(x,y)}{\partial y^2}\right)^2}$$

در سیستم زمان گسسته از تقریب های مختلفی استفاده می شود . رابطه ی زیر یکی از آن ها است که بافرض این که یک قطعه از تصویر انتخاب و پیکسل ها را به صورت زیر شماره گذاری کرده باشیم، این رابطه به دست آمده است:

x1	*x2*	*x3*
x4	x5	x6
x7	x8	**x9**

L [f(x,y)]=4x5-(x2+x4+x6+x8)

وقتی ما سک مورد استفاده به صورت ماتریس زیر باشد

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

منظور از L همان لاپلاسین است.

مشاهده می شود که مجموع مقادیر داخل ماتریس برابر صفر می باشد، و نشانگر آن است که در نواحی هموار، مقدار آن صفر بوده که لبه نمی باشد و تنها نقاطی که عبور از صفر در آنان به وقوع پیوسته به عنوان لبه در نظر گرفته می شود.

الگوریتم Marr & Hildreth از جمله الگوریتم هایی است که بر اساس مشتق دوم طراحی شده که در اینجا تنها به ذکر نام آن بسنده شده است.

بخش دوم

الگوریتم های موازی

تکنیک های مربوط به حل مسایل با استفاده از کامپیوتر، که منجر به روشی گام به گام در حل آن مساله می شود را **الگوریتم** گویند.ساخت و طراحی الگوریتم های مختلف برای یک مساله ی خاص به دلیل پیچیدگی های متفاوت

آن ها، یا به بیان دیگر حافظه ی مصرفی و زمان اجرای متفاوت الگوریتم هاست. مسلما سعی می شود از الگوریتم هایی که سرعت بالاتری را تامین می کند استفاده شود. بسیاری از الگوریتم ها برای کامپیوترهای حاوی یک پردازنده طراحی شده اند که دنباله ی منفردی از دستورات را اجرا می کند. امروزه با کاهش شدید قیمت سخت افزار کامپیوتر، ساخت کامپیوترهای موازی افزایش یافته است. این کامپیوتر ها بیش از یک پردازنده دارند و همه ی پردازنده ها می توانند دستورات را به طور همزمان (موازی) اجرا کنند. الگوریتم های طراحی شده برای چنین کامپیوترهایی را الگوریتم های موازی گویند . در ادامه شرح بیشتری راجع به این الگوریتم ها و سخت افزارهای مربوطه خواهیم داشت.

۱-۲ الگوریتم های موازی:

برای انجام سریع کارها، می توان کار را به طور همزمان یا موازی بین چند نفر تقسیم کرد. این دقیقا همان کاری است که در کامپیوترها نیز قابل پیاده سازی و انجام است. کامپیوترها می توانند با واگذاری کارهای خود به چند پردازنده به طور موازی، کار را به انجام برسانند.

الگوریتم هایی که در دوره ی کارشناسی با آن سرو کار داریم اصولا طوری طراحی شده بودند که روی کامپیوترهای ترتیبی متداول قابل پیاده سازی باشند. چنین کامپیوترهایی یک پردازنده دارند که دستورات را به ترتیب اجرا می کنند. درست مثل اینکه خودتان به تنهایی، انجام کار را به طور ترتیبی و پشت سر هم بر عهده گیرید. چنانکه پس از اتمام کار اول، دومی را شروع کنید و ا لی آخر. این کامپیوترها بر اساس مدل ارایه شده توسط وان نیومن ساخته شده اند. همانطور که در شکل۱-۲ نشان داده شده است، این مدل از یک پردازنده موسوم به واحد پردازش مرکزی (CPU) و حافظه تشکیل شده است. این مدل یک سری دستورات را گرفته، روی یک سری از داده ها عمل می کند. چنین کامپیوترهایی را کامپیوترِ جریان تک دستوری جریان تک داده ای (SISD) می گویند و آن را عامه ی مردم به عنوان کامپیوترهای سریال می شناسند.

یک کامپیوتر ترتیبی متداول

شکل۱-۲

اگر کامپیوتر چندین پردازنده داشته باشد که به طور همزمان (موازی) دستورات را اجرا کنند، بسیاری از مسایل را می توان بسیار سریع تر حل کرد. قیمت پردازنده ها در سه دهه ی اخیر به شدت کاهش یافته است. در حال حاضر،

سرعت یک ریزپردازنده ی موجود در حد سریع ترین کامپیوترهای سریال است. ولی هزینه ی آن ها بسیار پایین تر است. بنابر این با استفاده از کامپیوترهای موازی، این امکان وجو دارد که با صرف هزینه ی کمتر، قدرت محاسباتی بیشتر از سریع ترین کامپیوتر های سریال به دست آورد. کاربردهای فراوا نی وجود دارد که می توا نند از محاسبه ی موازی بهره ببرند. کاربرد هایی در حیطه ی هوش مصنوعی، استنباط در شبکه ی عصبی، درک زبان های طبیعی، تشخیص گفتار و بینایی ماشین. کاربردهای دیگر شامل پردازش درخواست بانک اطلاعاتی، پیش بینی وضع هوا، کنترل آلودگی، تحلیل ساختار پروتیین ها و بسیاری از موارد دیگر می شود.

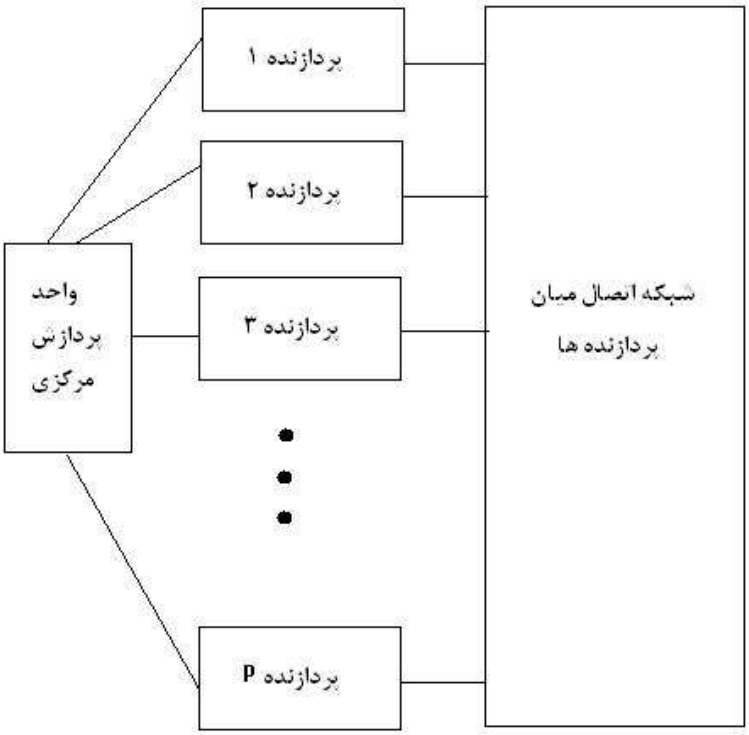
۲-۲ معماری موازی:

هر پردازنده در یک کامپیوتر موازی می تواند یا تحت کنترل یک واحد کنترل مرکزی و یا به طور مستقل تحت کنترل واحد کنترل خودش عمل کند. نوع اول را جریان تک دستوری جریان چند داده ای (SIMD) می نامند. شکل یک معماری SIMD را نشان می دهد. شبکه اتصال میان پردازنده ها که در این شکل تصویر شده است، سخت افزاری را نشان می دهد که پردازنده ها را قادر به برقراری ارتباط با یکدیگر می سازد. در معماری SIMD ، یک دستور به طور همزمان توسط همه ی واحد های پردازش تحت کنترل واحد پردازش مرکزی اجرا می گردد. لازم نیست همه ی پردازنده ها یک دستور را در هر چرخه اجرا کنند، هر پردازنده ی مفروض را می توان در هر چرخه از کار انداخت.

کامپیوترهای موازی که در آن ها هر پردازنده دارای واحد کنترل مختص به خود می باشد، کامپیوترهای جریان چند دستوری جریان چند داده ای (MIMD) نام دارند. کامپیوترهای MIMD هم سیستم عامل و هم برنامه را در هر پردازنده نگهداری می کنند.

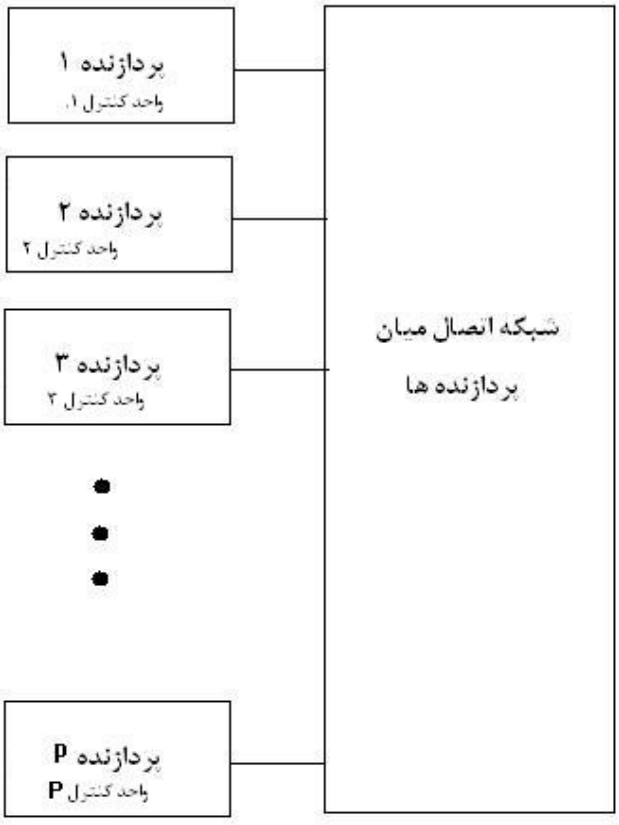
کامپیوترهای SIMD برای برنامه هایی منا سب هستند که در آن ها یک مجموعه دستورات مشترک روی عناصر متفاوتی از مجموعه داده ها اجرا می شوند. چنین برنامه هایی را برنامه های با داده های موازی می نا مند.

به طور کلی کامپیوتر های SIMD برای الگوریتم های موازی که نیاز به همزمان سازی دارد، از همه مناسب تر است. بسیاری از کامپیوترهای MIMD دارای سخت افزار اضافی هستند که آن ها را قادر به تقلید از کامپیوترهای SIMD می سازد.



SIMD

شکل ۲-۲



MIMD

شکل ۳-۲

نکته مهم در بررسی و تحلیل الگوریتم های موازی این است که مقدار کل کار انجام شده توسط ا لگوریتم را تحلیل نمی کنیم بلکه مقدار کل کار انجام شده توسط یک پردازنده را تحلیل می کنیم زیرا این ایده ی خوبی از سرعت پردازش ورودی توسط کامپیوتر را ارایه می دهد.

بخش سوم

ماتریس اسپارس(خلوت)

ماتریس اسپارس یا خلوت، ماتریسی است که میزان صفرهای آن زیاد باشد. برای ذخیره ی عناصر ماتریس در حافظه ی کامپیوتر، از مفهوم آرایه ی خطی استفاده می شود. وقتی ماتریس اسپارس به صورت full matrix ذخیره شود همه ی عناصر آن – شامل عناصر صفر و غیر صفر – در آرایه ذخیره می شوند که به نوعی اسراف حافظه است. برای جلوگیری از اسراف در حافظه و پردازش های اضافی فرم ها و الگوریتم های خاصی تعریف و استفاده می شود. استفاده از این الگوریتم ها، پیچیدگی محاسباتی و زمانی را کاهش می دهد. در ادامه ی این فصل چند نمونه از این الگوریتم ها را معرفی خواهیم کرد . پیاده سازی این الگوریتم ها می تواند کارایی application ها را افزایش دهد. به خصوص در نمایش درخت، نمایش سه بعدی یک object، اتصالات شبکه، الکترونیک، محاسبات ریاضی، ذخیره ی تصویر و فایل و فشرده سازی فایل.

الگوریتم های پیشنهادی حوزه و برد بزرگی دارند و نه تنها در محاسبات بلکه در شاخه های علوم، الکترونیک و گرافیک نیز کاربرد دارند.

در زیر چند ا لگوریتم را بررسی می کنیم. این بررسی صرفا برای نشان دادن این مطلب است که با استفاده از ماتریس اسپارس عملیات پردازش بسیار سریع تر و با صرفه تر از زمانی است که ماتریس اصطلاحا full matrix باشد.

۱-۳ الگوریتم های ذخیره ماتریس اسپارس:

۱-۳-۱ Coordinate storage (COO)

در این روش به جای ذخیره همه عناصر فقط عناصر غیر صفر را ذخیره می کنیم . عناصر غیر صفر را در آرایه ی val ، اندیس سطرعنصر مورد نظر را در خانه متناظرش در آرایه row_ind ، اندیس ستون عنصر مورد نظر را در خانه متناظرش در آرایه col_ind ذخیره می کنیم .

6	0	9	0	0	4	0	0
0	0	0	0	0	4	0	0
0	5	0	0	0	0	0	0
0	0	3	5	8	0	0	0
0	0	0	0	6	0	0	0
0	0	0	0	0	5	0	0
0	0	0	0	0	4	3	0
0	0	0	0	0	0	2	2

Matrix A

val()	6	9	4	4	5	3	5	8	6	5	4	3	2	2
row_ind()	1	1	1	2	3	4	4	4	5	6	7	7	8	8
col_ind()	1	3	6	6	2	3	4	5	5	6	6	7	7	8

تحلیل زمان اجرای ذخیره سازی :

الگوریتم ذخیره سازی از درجه n^2 می باشد .

تحلیل حافظه مصرفی :

با توجه به اینکه هر کدام از ۳ آرایه ، N_{nze} (تعداد عناصر غیر صفر) عنصر در خود ذخیره می کنند پس میزان حافظه مصرفی از درجه $3N_{nze}$ خواهد بود .

۲-۳-۱ روش CRS

این روش یکی از عمومی ترین روش های ذخیره ماتریس اسپارس است. در این روش به ۳ آرایه ی یک بعدی نیاز داریم : یک آرایه ی val از نوع اعشاری (float) برای ذخیره ی مقادیر عناصر غیر صفر ، یک آرایه col_ind از نوع اعداد صحیح (int) که برای ذخیره شماره ستون عناصر غیر صفر در ماتریس اصلی به کار می رود و یک آریه row_ptr از نوع اعداد صحیح ، که اعداد ذخیره شده در این ماتریس مکان عناصری در آرایه ی val نشان می دهد که سطر جدیدی را در ماتریس اصلی شروع می کنند.

عناصر غیر صفر ماتریس را به ترتیب سطری در آرایه val پشت سر هم قرار می دهیم . تعداد عناصر این آرایه برابر با N_{nze} خواهد بود . شماره ستون هر عنصر غیر صفر را در خانه متناظر آن عنصر را در آرایه ی col_ind به همان ترتیب قرار می دهیم . به قسمی که:

if value_list (k) = $a_{i,j}$ then col_ind(k)=j

تعداد عناصر این آرایه برابر با nnz خواهد بود .

در آرایه ی row_ptr عنصر آخر را برابر با تعداد عناصر غیر صفر به علاوه ی ۱ قرار می دهیم. این عنصر برای کنترل حلقه ی loop در ضرب ماتریس به کار می رود . تعداد عناصر این آرایه حداکثر برابر با تعداد سطرها به اضافه یک یعنی n+۱ است .

بنابراین در این روش به جای n^2 خانه لازم در روش معمولی برای ذخیره سازی ، تنها به 2nnz+n+1 خانه حافظه نیاز داریم .

برای مثال ماتریس اسپارس A

$$A = \begin{bmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{bmatrix}.$$

به شکل زیر در ساختمان داده مذکور ذخیره می شود .

val	10	-2	3	9	3	7	8	7	3 ...-9	13	4	2	-1	
col_ind	1	5	1	2	6	2	3	4	1 ...-5	6	2	5	6	

row_ptr	1	3	6	9	13	17	20

برای برگرداندن ماتریس به شکل ماتریس اصلی شماره ی ستون عناصر را در ماتریس col_ind داریم و بوسیله ماتریس row_ptr نیز می توان شماره ی سطر ها را بدست آورد .

پیاده سازی :

```
private void CRS (int[,] A,int n,int Nnze)
{
bool flag = false;
int k = 0;
int index = 0;
int num =Nnze;
int[] value_list = new int[num];
int[] column_index = new int[num];
int[] row_ptr = new int[n + 1];

//  obtain value_list , column_index ,  row_ptr
for (int I = 0; I < n; i++)
    {

    flag = true;
   for (int j = 0; j < n; j++)
       if (A[I, j] != 0)
       {
           value_list[k] = A[I, j];
           column_index[k] = j;
           if (flag == true)
          {
          row_ptr[index] = k;
            index ++;
```

```
                flag = false;
            }
                k++;
        } // end if
      }
 row_ptr[index] = k;      امین n+1 برای
}
```

تحلیل زمان اجرای ذخیره سازی :

همان طور که مشاهده می شود الگوریتم ذخیره سازی ماتریس اسپارس به روش CRS از درجه n^2 است .

تحلیل حافظه مصرفی :

بنابراین در این روش به جای n^2 خانه لازم در روش معمولی برای ذخیره سازی ، تنها به $2N_{nze}+n+1$ خانه حافظه نیاز داریم .

$2N_{nze}$: فضای مورد نیاز برای val و col_ind

$n+1$: فضای مورد نیاز برای row_ptr

ضرب :

برای محاسبه w * A + d = d داریم :

```
void CRS_Mult(const double val[],int col_idx[],int row ptr[],int n,
const double w[], double d[])
{
  for (int i = 0; i <= n - 1; i++)
      {
         int tmp = 0;
         for (int j = row_ptr[i]; j <= row_ptr[i + 1] -1; j++)
            tmp += value_list[j] * W[column_index[j]];

         d[i] = tmp;
      }
return;
}
```

تحلیل زمان اجرای ضرب :

تعداد دفعاتی که عمل ضرب انجام می شود دقیقا برابر است با تعداد عناصر غیر صفر .

تعداد دستیابی به حافظه : 4 بار که ۱ بار آن دستیابی غیر مستقیم است .

حسن این روش این است که اولا هیچ فرضی برای نوع توزیع عناصر غیر صفر در نظر نمی گیرند و ثانیا هیچ عنصر صفری را ذخیره نمی کنند .

۳-۳-۱ روش CCS

همانند روش CRS است با این تفاوت که به ترتیب ستونی عناصر را ذخیره می کنیم . به عبارت دیگر $CCS=CRS\ (A^T)$.

در این روش به ۳ آرایه یک بعدی نیاز داریم : یک آرایه val از نوع اعشاری (float) برای ذخیره مقادیر عناصر غیر صفر ، یک آرایهrow__ind از نوع اعداد صحیح (int) که برای ذخیره ی اندیس سطر عناصر غیر صفر در ماتریس اصلی به کار می رود و یک آریه col_ptr از نوع اعداد صحیح، که اعداد ذخیره شده در این ماتریس مکان عناصری در آرایه ی val نشان می دهد که ستون جدیدی را در ماتریس اصلی شروع می کنند.

عناصر غیر صفر ماتریس را به ترتیب ستونی در آرایه ی val پشت سر هم قرار می دهیم . تعداد عناصر این آرایه برابر با N_{nze} خواهد بود . شماره ی سطر هر عنصر غیر صفر را در خانه ی متناظر آن عنصر را در آرایه ی row_ind به همان ترتیب قرار می دهیم . به قسمی که:

if value_list (k) = $a_{i,j}$ then row_ind(k)=i

تعداد عناصر این آرایه برابر با N_{nze} خواهد بود .

در آرایه col_ptr عنصر آخر را برابر با تعداد عناصر غیر صفر +۱ قرار می دهیم این عنصر برای کنترل حلقه loop در ضرب ماتریس به کار می رود . تعداد عناصر این آرایه حداکثر برابر با تعداد سطرها به اضافه یک یعنی n+۱ است .

بنابراین در این روش به جای n^2 خانه لازم در روش معمولی برای ذخیره سازی ، تنها به 2nnz+n+1 خانه حافظه نیاز داریم .

برای مثال اگر

$$A = \begin{bmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{bmatrix}.$$

با اعمال روش CCS خواهیم داشت:

val	10	3	3	9	7	8	4	8	8 ···-9	2	3	13	-1		
row_ind	1	2	4	2	3	5	6	3	4 ···-5	6	2	5	6		

col_ptr	1	4	8	10	13	17	20

تحلیل حافظه مصرفی :

در این روش به جای n^2 خانه لازم در روش معمولی برای ذخیره سازی ، تنها به $2N_{nze}+n+1$ خانه حافظه نیاز داریم .

$2N_{nze}$: فضای مورد نیاز برای val وrow_ind

$n+1$: فضای مورد نیاز برای col_ptr

ضرب :

```
void CCS_Mult(double[] value_list ,int[] row_index,int[] column_ptr
,int n,double[] w, double[] d)
   {
      float[] d = new float[4];
      for (int l = 0; l < n; l++)
          d[l] = 0;

      for (int i = 0; i < n; i++)
         {
           for (int j = column_ptr[i]; j <= column_ptr[i + 1] - 1;
j++)
            d[row_index[j]] += value_list[j] * W[i];

   }
```

تحلیل زمان اجرای ضرب :

تعداد دفعاتی که عمل ضرب انجام می شود دقیقا برابر است با تعداد عناصر غیر صفر .

تعداد دستیابی به حافظه : 4 بار که ۱ بار آن دستیابی غیر مستقیم است .

حسن این روش این است که اولا هیچ فرضی برای نوع توزیع عناصر غیر صفر در نظر نمی گیرند و در ثانی هیچ عنصر صفری را ذخیره نمی کنند .

Compresses diagonal storage (CDS) ۱-۳ -۴

این روش مناسب ماتریس هایی است که عناصر آن به ترتیب قطری در ماتریس قرار گرفته اند این ماتریس 2n-1 قطر دارد . در این روش ماتریس را به صورت قطری ذخیره می کنیم. اندیس هر قطر با اندیس k شناخته می شود که k=j-i بنابراین k مقادیری بین n-1 , 1-n دارد .

$$A = \begin{array}{c} \\ \boldsymbol{d}_{-1} \\ \boldsymbol{d}_{-2} \\ . \\ . \\ . \\ \boldsymbol{d}_{1-n} \end{array} \begin{array}{c} \begin{array}{cccccccc} \boldsymbol{d}_0 & \boldsymbol{d}_1 & \boldsymbol{d}_2 & . & . & . & \boldsymbol{d}_{n-1} \end{array} \\ \begin{bmatrix} a_{11} & a_{12} & a_{13} & . & . & . & a_{1n} \\ a_{21} & a_{22} & a_{23} & & & & . \\ a_{31} & a_{32} & a_{33} & & & & . \\ . & & & . & & & . \\ . & & & & . & & . \\ . & & & & & . & . \\ a_{n1} & . & . & . & . & . & a_{nn} \end{bmatrix} \end{array}$$

اگر داشته باشیم .

$$A = \begin{pmatrix} 10 & -3 & 0 & 0 & 0 & 0 \\ 3 & 9 & 6 & 0 & 0 & 0 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 0 & 0 & 8 & 7 & 5 & 0 \\ 0 & 0 & 0 & 9 & 9 & 13 \\ 0 & 0 & 0 & 0 & 2 & -1 \end{pmatrix} . \qquad (4.2)$$

$$val(i, j) = a_{i,i+j} \qquad (4.3)$$

به شکل زیر ذخیره می کنیم .

val(:,-1)	0	3	7	8	9	2
val(:, 0)	10	9	8	7	9	-1
val(:,+1)	-3	6	7	5	13	0

اگر ماتریس از نوع متقارن باشد می توان فقط قسمت بالا یا پایین آن را ذخیره کرد .
همان طور که مشاهده می شود عناصر صفر هم ممکن است ذخیره شود .
اما در روش فوق از لحاظ حافظه مصرفی بستگی به نوع توزیع عناصر غیر صفر ماتریس دارد . اگر به صورت قطری توزیع شده بودند یعنی بیشترین چگالی عناصر غیر صفر در قطر ها باشد و تعداد قطر هایی که چگالی عناصر غیر صفر در آنها

پایین است کم باشد . روش خوبی برای ذخیره سازی است و بهتر از CRS خواهد بود چون به جای ذخیره ی دو آرایه که مشخص کننده ی سطر و ستون باشد از یک آرایه برای ذخیره ی اندیس قطر استفاده خواهیم کرد . ولی اگر تعداد صفرهای قطر ها زیاد باشد اصلا روش موثری نخواهد بود .

۱-۳-۵ (JDS) Jagged Diagonal Format

$$\begin{bmatrix} a_{11} & a_{12} & 0 & a_{14} & 0 & 0 \\ 0 & a_{22} & a_{23} & 0 & a_{25} & 0 \\ a_{31} & 0 & a_{33} & a_{34} & 0 & 0 \\ 0 & a_{42} & 0 & a_{44} & a_{45} & a_{46} \\ 0 & 0 & 0 & 0 & a_{55} & a_{56} \\ 0 & 0 & 0 & 0 & a_{65} & a_{66} \end{bmatrix}$$

ابتدا عناصر غیر صفر را به سمت چپ شیفت داده داریم ، A_{crs} بدست می آید :

$$A_{crs} = \begin{bmatrix} a_{11} & a_{12} & a_{14} & 0 & 0 & 0 \\ a_{22} & a_{23} & a_{25} & 0 & 0 & 0 \\ a_{31} & a_{33} & a_{34} & 0 & 0 & 0 \\ a_{42} & a_{44} & a_{45} & a_{46} & 0 & 0 \\ a_{55} & a_{56} & 0 & 0 & 0 & 0 \\ a_{65} & a_{66} & 0 & 0 & 0 & 0 \end{bmatrix}$$

همزمان اندیس ستون عناصر غیر صفر در ماتریس اصلی را در یک آرایه ی مجزا به اسم column_index ، یادداشت می کنیم (چون جای ستون ها نسبت به ماتریس اصلی عوض شده است).

با تعویض مکان سطرها به ترتیب کاهشی تعداد عناصر غیر صفر از بالا به پایین A_{jsd} را خواهیم داشت :

$$A_{jds} = \begin{bmatrix} a_{42} & a_{44} & a_{45} & a_{46} & 0 & 0 \\ a_{11} & a_{12} & a_{14} & 0 & 0 & 0 \\ a_{22} & a_{23} & a_{25} & 0 & 0 & 0 \\ a_{31} & a_{33} & a_{34} & 0 & 0 & 0 \\ a_{55} & a_{56} & 0 & 0 & 0 & 0 \\ a_{65} & a_{66} & 0 & 0 & 0 & 0 \end{bmatrix}$$

همزمان با تعویض مکان سطرها، یک آرایه ی جدید به اسم perm_vector به وجود آورده ، اندیس سطرها در ماتریس اصلی را یادداشت می کنیم (چون جای سطرها نسبت به ماتریس اصلی عوض شده است).

همچنن نیاز به یک آرایه یک بعدی دیگر جهت ذخیره ی اندیس عنصر شروع کننده ی ستون جدید ، به نام Start_position است . از این آرایه در ضرب استفاده می شود.

حال عناصر را در ماتریس یک بعدی value_list (از نوع اعشاری) به ترتیب ستونی (ابتدا ستون ۱ ، بعد ستون ۲ و ....) ذخیره می کنیم (nnz)

.

$value\_list$	a_{42}	a_{11}	a_{22}	a_{31}	a_{55}	$a_{65;}$	a_{44}	a_{12}	a_{23}	a_{33}	a_{56}	$a_{66;}$	a_{45}	a_{14}	a_{25}	$a_{34;}$	a_{46}

چون اندیس ستون هر یک از عناصر تغییر کرده لذا باید شماره ستون تک تک عناصر قبل از جابجایی را ذخیره کنیم لذا داریم : (nnz)

$column\_indexes$	2	1	2	1	5	5;	4	2	3	3	6	6;	5	4	5	4;	6

از آنجایی که ما ترتیب سطر ها را نیز عوض کرده ایم برای برگشت نتیجه به ترتیب اولیه باید سطر ها را نگه داریم لذا :

$perm\_vector$	4	2	3	1	5	6

واضح است که اندازه ی آرایه ی perm_vector برابر n است .

همچنن نیاز به یک ماتریس یک بعدی دیگر جهت ذخیره اندیس عنصر شروع کننده ستون جدید مانند زیر:

$start\_positions$	1	7	13	17	18

که تعداد عناصر این آرایه برابر است با تعداد عنصر های سطری که بیشترین تعداد عنصر را دارد . عنصر آخر برای این است که حلقه loop را در عمل ضرب کنترل کنیم .

به دلیل اینکه جای سطر ها عوض شده است در نهایت باید سطرهای ماتریس حاصل ضرب را به جای اول بر گردانیم .

```
for i ← 1 to N
    Temp[perm vector[i]] := Y [i]
for i ← 1 to N
Y [i] := Temp[i]
```

بنابراین داریم :

$$S_{jds} = (N_{nze} * 2) + (N_{nze} * 1) + N + N_{jd}$$

ضرب:

```
int disp = 0;
for (int i = 0; i < num_jdiag; i++)
     for (int j = 0; j < (start_possition[i + 1] - start_possition[i]); j++)
      {
       Y2[j] = Y2[j] + valaue_list[disp] * X[column_indexes[disp]];
       disp = disp + 1;
       }
```

می دانیم در ضرب معمولی Ax = y تمام عناصر ستون شماره i ماتریس A در عنصر i از بردار x ضرب می شوند چون عناصر به ترتیب ستونی در ماتریس vlaue_list ذخیره شده است، الگوریتم فوق دقیقا طبق همین ایده نوشته شده است .

تحلیل زمان اجرای ضرب:

برای لود ۴ ماتریس باید ۴ بار به حافظه دسترسی پیدا کرد که یک بار آن دستیابی غیر مستقیم است . همچنین باید در آخر کار دوباره ترتیب سطرهای ماتریس حاصل را به حالت ماتریس اصلی برگرداند. پس علاوه بر تعداد دسترسی زیاد ، برگشت سطرها هم مورد نیاز است .

تحلیل زمان اجرای ذخیره سازی :

الگوریتم شیفت سطری (شیفت به چپ) از مرتبه n^3 است .

The transpose jagged diagonal format ۱-۳-۶

در این روش عناصر غیر صفر را به صورت ستونی به سمت بالا شیفت می دهیم و همزمان اندیس سطر عناصر غیر صفر در ماتریس اصلی را در یک آرایه ی مجزا به اسم row_index ، یادداشت می کنیم (چون جای سطر ها نسبت به ماتریس اصلی عوض شده است).

بنابراین

$$\begin{bmatrix} a_{11} & a_{12} & 0 & a_{14} & 0 & 0 \\ 0 & a_{22} & a_{23} & 0 & a_{25} & 0 \\ a_{31} & 0 & a_{33} & a_{34} & 0 & 0 \\ 0 & a_{42} & 0 & a_{44} & a_{45} & a_{46} \\ 0 & 0 & 0 & 0 & a_{55} & a_{56} \\ 0 & 0 & 0 & 0 & a_{65} & a_{66} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix}$$

تبدیل می شود به :

$$\begin{bmatrix} a_{11} & a_{12} & a_{23} & a_{14} & a_{25} & a_{46} \\ a_{31} & a_{22} & a_{33} & a_{34} & a_{45} & a_{56} \\ 0 & a_{42} & 0 & a_{44} & a_{55} & a_{66} \\ 0 & 0 & 0 & 0 & a_{65} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix}$$

سپس جای ستونهای ماتریس بدست آمده در مرحله ی قبل را به ترتیب کاهشی تعداد عناصر غیر صفر از چپ به راست عوض می کنیم (مرتب می کنیم) . همچنین در این مرحله ترتیب عناصر بردار x را نیز با فرض اینکه سطری از ماتریس A است، عوض می کنیم

$$\begin{bmatrix} a_{25} & a_{12} & a_{14} & a_{46} & a_{11} & a_{23} \\ a_{45} & a_{22} & a_{34} & a_{56} & a_{31} & a_{33} \\ a_{55} & a_{42} & a_{44} & a_{66} & 0 & 0 \\ a_{65} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_5 \\ x_2 \\ x_4 \\ x_6 \\ x_1 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix}$$

از آنجایی که ترتیب عناصر بردار x را عوض کرده ایم در نتیجه نیازی به ذخیره ی جابجایی ستون ها نیست . حالا عناصرغیر صفر را سطر به سطر (به هرکدام از این سطر ها transpose jagged diagonal (tjd) گویند) در آرایه ی value_list ذخیره می کنیم . همچنین مکان عناصری در در آرایه ی value_list که شروع کننده ی tjd هستند را در آرایه ی statrt_position ذخیره می کنیم به اضافه یک عنصر که مقدار آن برابر است با $N_{nze}+1$.

value_list	a_{25}	a_{12}	a_{14}	a_{46}	a_{11}	$a_{23;}$	a_{45}	a_{22}	a_{34}	a_{56}	a_{31}	$a_{33;}$	a_{55}	a_{42}	a_{44}	$a_{66;}$	a_{65}

row_indexes	2	1	1	4	1	2	4	2	3	5	3	3	5	4	4	6	6

start_position	1	7	13	17	18

X	x_5	x_2	x_4	x_6	x_1	x_3

تحلیل حافظه مصرفی :

$S_{tjds}=(2 * N_{nze}) + N_{tjd}$

N_{nze} : ذخیره عناصر غیر صفر

N_{nze} : ذخیره اندیس ها

N_{tjds} : تعداد tjd

ضرب :

```
Ax = y
```

```
m ←  1
for i←1 to num_tjdiag
   k←1
   for j := start_position[i] to start_position[i +1] - 1
       p←row_indexes[m]
      Y [p] += value_list[j] * X[k]
      k←k +1
            m ←m+1
   endfor
endfor.
```

تحلیل زمان اجرای ضرب:

تعداد دفعاتی که عمل ضرب انجام می شود دقیقا برابر است با تعداد عناصر غیر صفر .

تعداد دستیابی به حافظه : ۴ بار که ۱ بار آن دستیابی غیر مستقیم است .

Robs Alorithm ۱-۳-۷

ماتریس A_{n*m} را در نظر بگیرید ، ماتریس A را به شکل زیر فشرده می کنیم:

در سطر صفر داریم:

A[0][0]= m+n
A[0][0]= m-n

از سطر ۱ به بعد داریم :

A[0][i]=i*n+j

که i مقدار سطر ، j مقدار ستون و n تعداد کل ستون ها است .

A[1][i]=actual none-zero value

به عنوان مثال برای ماتریس زیر

$$\begin{bmatrix} a_{11} & a_{12} & 0 & a_{14} & 0 & 0 \\ 0 & a_{22} & a_{23} & 0 & a_{25} & 0 \\ a_{31} & 0 & a_{33} & a_{34} & 0 & 0 \\ 0 & a_{42} & 0 & a_{44} & a_{45} & a_{46} \\ 0 & 0 & 0 & 0 & a_{55} & a_{56} \\ 0 & 0 & 0 & 0 & a_{65} & a_{66} \end{bmatrix}$$

به شکل زیر نوشته خواهد شد :

۱۲	۰
۷	a_{11}
۸	a_{12}
۱۰	a_{14}
۱۴	a_{22}
⋮	⋮
۴۱	a_{65}
۴۲	a_{66}

```
for(i=0;i<Nnnz;i++)
{
printf(''Enter the Row value'');
scanf(''%d'',&r);
printf(''Enter the Column value'');
scanf(''%d'',&c);
printf(''Enter the value of element'');
scanf(''%d'',&val);
tr[i+1][1]=val; // non zero value
tr[i+1][0]=((r)*tc)+c; // tc is total column in the matrix
if(max_row< r)
max_row=r;
if(max_col< c)
max_col=c;
}
printf(''nnnt Effective matrix (NEW)...nn'');
for(i=0;i< nz þ 1;i++)
{
for(j=0;j<2;j++)
printf(''%4d'',tr[i][j]);
printf(''nn'');
}
printf(''nn n nnt. . . . . .Sparse Matrix (NEW). . . . . .'');
```

```
printf(''nn. . . . . .nn n
nnt'');
size= tr[0][1];
for(i=1;i < nz+1;i++)
{
r_c=tr[i][0];
r=r_c/size;
c=r_c%size;
val=tr[i][1];
sp[r][c]=val;
}
```

برای بدست آوردن (تولید دوباره) ماتریس اصلی درایم :

RC_value=A[i][0] , i=1,2,...
Row_value = RC_value /n
column_value = RC_value MOD n
value = A[i][1]

بخش چهارم

کاربرد ماتریس اسپارس در پردازش تصاویر

همانطور که در بخش سوم گفته شد الگوریتم های پیشنهادی برای ماتریس اسپارس حوزه و برد بزرگی داشته و نه تنها در محاسبات بلکه در شاخه های مختلف علوم ، الکترونیک و گرافیک کاربرد دارند.

برای ذخیره ی تصاویر اسپارس به صورت فشرده از هر یک ا ز الگوریتم های بخش سوم می توان استفاده کرد. همچنین برای ضرب ماتریس اسپارس در یک بردار (که در عملیات پردازش بسیار مورد استفاده است) روش ها و الگوریتم های

مختلفی وجود دارد که برخی از آن ها در فصل سوم معرفی گردید. در این فصل یک الگوریتم موازی برای ضرب ماتریس اسپارس در یک بردار معرفی خواهد شد که بر روی GPU که ساختاری موازی دارد قابل پیاده سازی است.
در این فصل به طور خاص به سراغ sparse matrix solver که بر روی GPU قابل اجرا است خواهیم رفت . پیش از بررسی ، اطلاعاتی مقدماتی راجع به GPU ارایه خواهد شد. سپس با استفاده از الگوریتم Robs که در فصل سوم بررسی شد یکی از تکنیک های پردازش تصویر را پیاده سازی خواهیم کرد. تکنیک مورد بررسی، تکنیک کدینگ و فشرده سازی می باشد و در ادامه، الگوریتم پیاده سازی شده را با الگوریتم Run-length coding مقایسه می کنیم و محاسن و معایب هریک را نشان می دهیم.

۴-۱ Graphic Processing Unit (GPU):

GPU که گاهی VPU (Visual Processing Unit) نیز گفته می شود، پردازنده ی خاص گرافیکی با ساختاری موازی، برای کامپیوترهای شخصی،workstation وgame console می باشد. GPU های مدرن امروزی در دستکاری و نمایش گرافیک کامپیوتری بسیار کارا هستند. همچنین به خاطر ساختار موازی برای پیاده سازی الگوریتم های پیچیده بسیار مؤثرتر از CPU می باشند. یک GPU می تواند در بالای کارت ویدیو بنشیند و یا به طور مستقیم روی مادربورد ملحق شود. بیش از ۹۰٪ کامپیوترهای دارای GPU از مورد دوم یعنی به صورت الحاقی روی مادربورد استفاده می کنند در صورتیکه نقطه ی مقابل آن بسیار پر قدرت تر است.
یک GPU تعدادی از عملیات پایه ای گرافیکی را بسیار پرسرعت تر از ترسیم های با استفاده از CPU اجرا می کند. استفاده ی عمده و اصلی این پردازنده ها در بازی های سه بعدی و ساختارهای سه بعدی می باشد.
GPU ها ی امروزی، از نسل چیپ های گرافیکی monolithic هستند که در اوایل دهه ی ۱۹۸۰ و ۱۹۹۰ ساخته شد.این چیپ ها اصولا ترسیمات گرافیکی را پشتیبانی نمی کردند. از مثال های اولیه ی GPU،
ANTIC co-processor است که در Atari800 و Atari5200 استفاده شد.از سال ۲۰۰۰ به بعد بود که با ظهور OpenGL و DirectX ، GPU ها programming shading را نیز به قابلیت های دیگرشان افزودند به طوریکه با اجرای یک برنامه ی کوتاه یک پیکسل از یک بافت تصویر ورودی، می توانست پردازش شود، همچنین بردارهای هندسی می توانستند با یک برنامه ی کوتاه پیش از آن که روی screen طرح ریزی شوند مورد پردازش قرار گیرند.
GeForce 3 اولین محصول با چنین قابلیت هایی بود که توسط شرکت NVIDIA ساخته شد.
هم اکنون GPU های موازی با CPU ها به مقابله و رقابت پرداخته اند به طوریکه با ساخت GPU های همه منظوره (GPGPU) جایگاه خود را در زمینه های مختلف اکتشاف نفت، پردازش تصویر علمی، و حتی ارزش گذاری سهام یافته اند .شرکت های پیشتاز در زمینه ی ساخت GPU دو شرکت AMD و NVIDIA هستند.
پس از ذکر این تاریخچه ی کوتاه به سراغ پردازش تصویر روی GPU می رویم.

۴-۲ پردازش تصویر و GPU:

پردازش تصویر موضوع پایه ای در تسریع GPU به شمار می آید . بسیاری از تکنیک های پردازش تصویر دارای بخش هایی است که شامل محاسبات بر روی تعداد زیادی پیکسل است.این موضوع پردازش تصویر را موضوعی پایه ای برای تسریع این سخت افزار موازی، نموده است. از تکنیک های قابل اجرا روی GPU می توان به تشخیص لبه، منطقه، بافت، اشیا، تشخیص چهره و غیره اشاره کرد.و همه ی این موارد می تواند بر روی تناوب های ویدیویی به کار رود.

در اینجا sparse matrix solver را بررسی خواهیم کرد که برای ضرب ماتریس های اسپارس به کار می رود. این الگوریتم موازی با استفاده از GPU که ساختار موازی (SIMD) دارد قابل پیاده سازی است.

جبر خطی به طور گسترده برای الگوریتم های قطعه بندی تصاویر(image segmentation) مورد استفاده قرار می گیرد. دو زمینه ی اساسی در این باره که مورد توجه است عملیات پایه(مانند جمع و ضرب) و سیستم های solving linear است. عملیات پایه شامل ضرب ماتریس های حجیم (متراکم) و جمع است که در اینجا بحث نمی شود. مورد دوم در ادامه بررسی شده است.

در مرکز این الگوریتم معادله ی فریبنده ی ساده ای به چشم می خورد: $Ax=y$. و برای حل آن نیاز به به حل یک سیستم اسپارس است . استفاده از الگوریتم های ماتریس های حجیم بسیار پر هزینه خواهد بود و این در حالی است که الگوریتم های اسپارس با هزینه ی کم و کارایی بالا برای پیاده سازی موجود می باشد. هرچند پیاده سازی الگوریتم های ماتریس حجیم ساده تر می باشد اما کارایی و سرعت مسایل اصلی در عملیات پردازش است ودر نتیجه برای افزایش آن ها الگوریتم های اسپارس بیشتر مورد توجه می باشد.

برای حل معادله ی بالا، هنگامی که A ماتریسی است خلوت، و x به صورت یک بردار می باشد الگوریتم به این صورت عمل می کند:

-عناصر قطر اصلی در A_i^x ذخیره می شود

-عناصر غیر صفر غیر قطری به صورت segment ، که هر segment شامل عناصر یک سطر است در A_j^a ذخیره میگردد.

-بردارx در X^x ذخیره می شود.

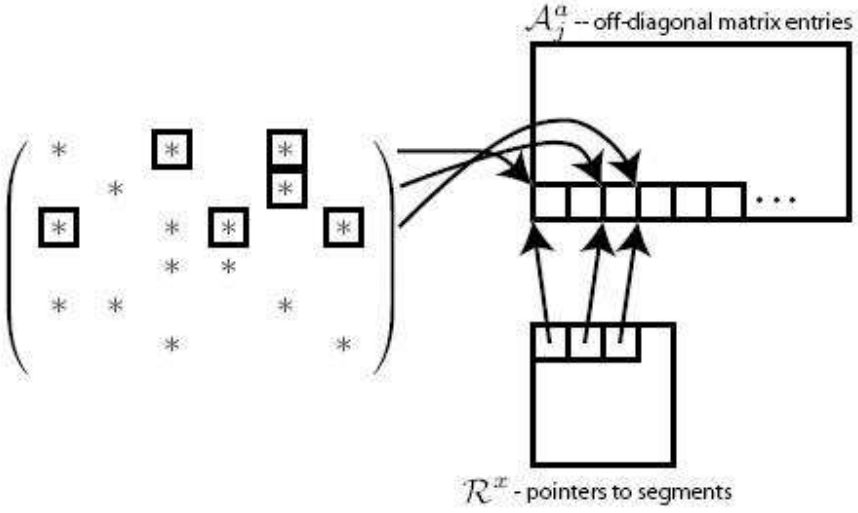
-دو مکان دیگر شامل اشاره گر داریم یکی C^a که اشاره گر به بردار است و دیگری R^x که اشاره گری به ابتدای هر segment ذخیره شده برای عناصر غیر قطری غیر صفر اشاره دارد.

با توجه به ساختار موازی GPU و نحوه ی ذخیره سازی، y به صورت زیر محاسبه می شود:

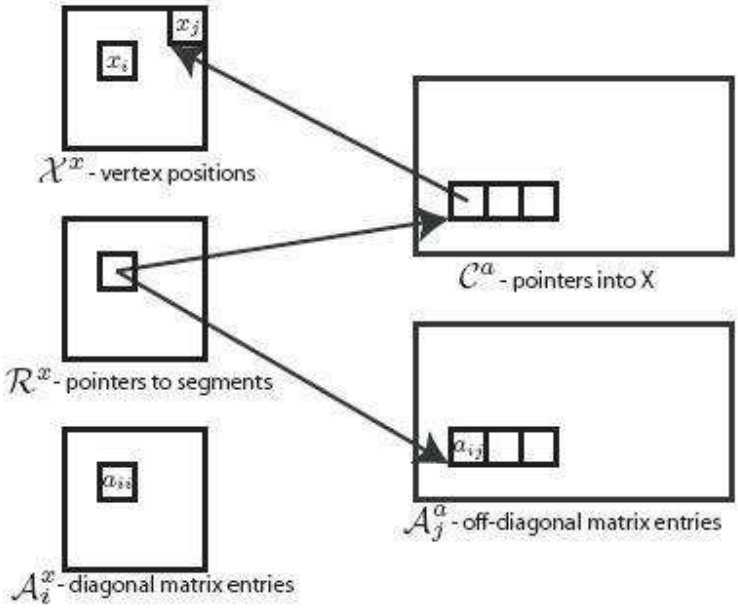
$$j = \mathcal{R}^x[i]$$
$$\mathcal{Y}^x[i] = \mathcal{A}_i^x[i] * \mathcal{X}^x[i] + \sum_{c=0}^{k_i-1} \mathcal{A}_j^a[j+c] * \mathcal{X}^x[\mathcal{C}^a[j+c]],$$

که y نهایتا ساختاری همانند x خواهد داشت.

نحوه ی ذخیره سازی با توجه به توضیحات، می تواند شکلی شبیه شکل زیر داشته باشد:



شکل۱-۴



شکل۲-۴

این متد سرعت را بسیار افزایش داده و مخصوصا برای پردازش های real time مؤثر است.

۴-۳ مقایسه ی دو الگوریتم:

حال به سراغ تصاویر باینری می رویم و کاربرد ماتریس اسپارس را در فشرده سازی این تصاویر مورد بررسی قرار می دهیم. الگوریتمی ارایه خواهیم کرد که تصاویر باینری را به طور مؤثر و تحت معیارهای فشرده سازی که در فصل ۲ بحث شد کد خواهد کرد. پس از آن این الگوریتم را با الگوریتم run length coding مقایسه خواهیم کرد و معایب و مزایای هر یک را بر خوا هیم شمرد.

تصاویر باینری همانطور که از نامشان مشخص است از دو سطح صفر(سیاه) و یک (سفید) تشکیل شده اند. برای ذخیره ی این تصاویر روش های مختلفی وجود دارد.run length coding یکی از این روش ها است که در فصل سوم معرفی شد. در صورتیکه تصویر باینری ما اسپارس باشد یا به بیان دیگر سطح صفر آن بیش از سطح یک آن باشد می توان از ماهیت اسپارس بودن تصویر بهره جست و تنها عناصر غیر صفر (که در تصاویر باینری صرفا یک است) را ذخیره نمود. به این صورت عمل می کنیم که آدرس اولین عنصر غیر صفر از هر تکه از هر سطر را به همراه عنصر پایانی غیر صفر همان قطعه در یک آرایه ی دو بعدی nz×۲ که nz تعداد قسمت های مجزای غیر صفر(یک) تصویر به علاوه ی یک سرآیند برای مشخص کردن ابعاد تصویر اولیه است ذخیره می کنیم. سطر اول این آرایه شامل سرآیند m+n و m-n است تا هنگام کدگشایی بتوان تصویر اولیه را با همان ابعاد بازگرداند.

قدم های کار در مثال زیر نشان داده شده است:

اگر ماتریس زیر را پیکسل های یک تصویر باینری فرض کنیم داریم:

$$\begin{bmatrix} 0&0&0&0&0&0 \\ 1&1&0&0&0&0 \\ 0&0&0&1&0&0 \\ 0&1&0&0&0&0 \\ 1&1&1&0&0&1 \\ 0&0&0&0&0&0 \end{bmatrix}$$

12,0
7,8
16,16
20,20
25,27
30,30

در نتیجه یک ماتریس کد 6×2 داریم که به جای ذخیره ی 6×6 در حافظه ذخیره خواهد شد

در فشرده سازی مفهومی به نام CR به معنای نسبت فشرده سازی وجود دارد که به صورت زیر تعریف می شود:

CR=n1/n2

که n1 حجم تصویر اولیه و n2 حجم تصویر فشرده است.

در این تصویر داریم:

CR=36/12=3

در صورتیکه از روش Run length coding با احتساب سرآیندی مشابه روش بالا داریم:

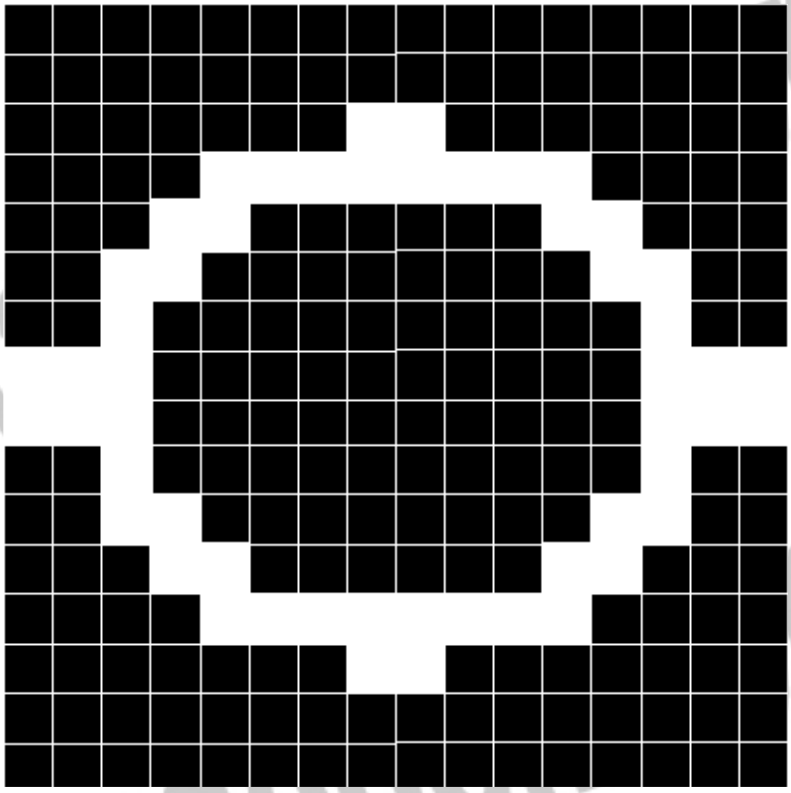
12,0,0,10,2,6,1,4,1,4,1,2,3,6

که در این حالت n2=1×13 است و در نتیجه CR=36/13=2.769

پس میزان فشرده سازی الگوریتم اول برای این تصویر بیشتر از دومی می باشد.

از محاسن روش اول نسبت به Run length coding ، عدم وجود انتشار خطا است. یکی از محاسن دیگر CR بزرگتر آن است. اما CR بستگی به تصویر دارد و این برتری یک برتری کلی نیست.

تصویر زیر را در نظر می گیریم و CR را برای دو روش مذکور به دست می آوریم:



اگر هر یک از مربع های تصویر بالا را یک پیکسل فرض کنیم با ستفاده از روش مذکور خواهیم داشت:

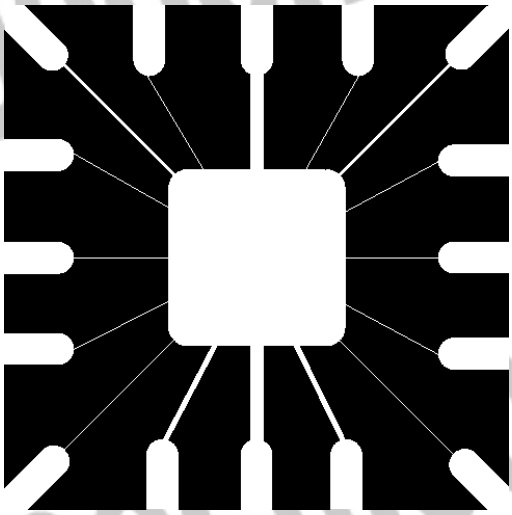
32 ,0
40 ,41

53 ,60
68 ,69
76 ,77
83 ,84
93 ,94
99 ,99
110,110
.
.
.
216,217

ماتریس فشرده ی حاصل یک ماتریس $21\times2=42$ می باشد .
با این کار به جای ذخیره ی $16\times16=256$ عنصر تنها 42 عنصر ذخیره کرده ایم.
$CR=256/42\approx6.095$
در صورتیکه از Run length coding برای کد کردن این تصویر استفاده کنیم و از سرایندی مشابه الگوریتم قبل برای باز گرداندن تصویر بهره گیریم خواهیم داشت:

32,0,0,39,2,11,8,7,2,6,2,5...,2,39

که کد مذکور یک آرایه ی 42×1 است و در نتیجه CR دقیقا با روش قبل برابر است.
و یا تصویر زیر که با استفاده از نرم افزار MATLAB و دو روش فوق فشرده شده است:



ابعاد این تصویر 486×468 پیکسل است. با استفاده از الگوریتم اول ماتریس فشرده ی حاصل ابعاد 2×2154 دارد در صورتیکه با Run length coding این میزان تا 1×4044 کاهش می یابد. در این تصویر خاص مشاهده می کنیم که CR نسبت به Run length coding کمتر است.

این یکی از محدودیت های الگوریتم ماست. در واقع در تصاویری که میزان تکه ها یا قسمت های سفید هر سطر یکی بیشتر از تکه های سیاه آن است، CR روش Run engthcoding مناسب تر است. علت را می توان اینطور توضیح داد که در هر سطر به جای ذخیره ی b+w که b تعداد تکه های سیاه وw تعداد تکه های سفید است، با روش ما 2w است که حداقل یکی بیشتر از روش Run length است .

اما در مورد تصاویری که چنین حالتی ندارند و در ضمن اسپارس نیز هستند CR روش ما بیشتر از روش Run length است. در جدول زیر چند نمونه تصویر بررسی شده به همراه CR آن ها آورده شده است. این تصاویر در ضمیمه ۲ و همچنین در CD همراه پروژه موجود است.

نام تصویر	size	Run lenght	Runrobs	CR-1	CR-2
Bi18.bmp	576×400	1×15862	7930×2	14.525	14.527
Bi19.bmp	463×502	1×15622	7810×2	14.878	14.880
Bi8.bmp	102×97	1×434	216×2	22.797	22.902
Bi4.bmp	158×187	1×932	465×2	31.701	31.769
Bi3.bmp	500×300	1×912	455×2	164.473	164.835
Bi5.bmp	281×261	1×5628	2813×2	13.031	13.036
ebookblack	377×324	1×2768	1383×2	44.128	44.160

جدول ۱-۴

که CR-1 مربوط به Run length و CR-2 مربوط به روش پیشنهادی است.

در زمان پیاده سازی " زمان" اجرای عملیات فشرده سازی و رمز گشایی را در تصاویر مختلف بررسی کردیم و میانگین زمانی را برای هر تصویر به طور جداگانه محاسبه نمودیم. مشاهده شد که زمان اجرای فشرده سازی و کد گشایی الگوریتم ما تقریبا در تمام موارد تست شده بهتر از زمان اجرای الگوریتم Run length coding بوده است. این در حالی است که از لحاظ پیچیدگی دو الگوریتم وضعیت مشابهی دارند.

این زمان ها در جدول زیر برای چند نمونه ی تست شده نشان داده شده است که در آن tlc وtld به ترتیب مربوط به عملیات فشرده سازی و کد گشایی(decompress) با Run length و trc و trd برای موارد مشابه مربوط به الگوریتم ماست.

نام تصویر	tlc	tld	trc	trd
Bi18	2.2003	1.1997	0.5282	0.1641
Bi19	2.0795	1.2813	0.5064	0.1656
Bi8	0.089	0.531	0.0046	0.0123

Bi4	0.188	0.161	0.0126	0.0095
Bi3	0.7044	0.8234	0.0311	0.0141
Bi5	0.3845	0.2596	0.1373	0.0596
ebookblack	0.658	0.657	0.042	0.0297

جدول ۲-۴

کد برنامه فشرده سازی و کد گشایی الگویتم های مذکور در ضمیمه ۱ آمده است.

پیچیدگی هر دو مورد در فشرده سازی O(mn) وقتی m تعداد سطر و n تعداد ستون باشد، است.

مقایسه ی دو الگوریتم در جدول زیر آمده است:

Run length	Runrobs
زمان کد کردن زیادتر است	زمان کد کردن کوتاه تر است
زمان دکد زیادتر است	زمان دکد کوتاه تر است
کلی است	برای موارد اسپارس کارایی دارد
انتشار خطا دارد	خطا منتشر نمی شود
CR در اکثر موارد به جز موارد استثنای ذکر شده،کمتر است	CR در اکثر موارد به جز موارد استثنای ذکر شده، بیشتر است

جدول ۳-۴

نتیجه گیری:

در عملیات پردازش تصویر ماتریس ها ی اسپارس سرعت عملیات را بالا می برند. در صورتیکه عملیات بر روی پردازنده های موازی انجام گیرد انجام این عملیات با سرعت بسیار بالا تری انجام می شود. Sparse matrix solver از جمله نمونه هایی است که بر روی پردازنده های موازی قابل پیاده سازی است و برا ی ضرب یک ماتریس اسپارس در یک بردار طراحی شده است. از دیگر کاربردهای اسپارس که در این پروژه بررسی شد فشرده سازی تصاویر باینری بود. الگوریتمی پیشنهاد شد و با Run length coding مقایسه گردید. نتیجه ی مقایسه در جدول ۳-۴ مطرح شد. دیدیم که هرچند با استفاده از این الگوریتم کلیت را از دست می دهیم ، اما در مواردی که تصویر باینری ما اسپارس باشد در اکثر موارد CR بالاتر، زمان اجرای کوتاه تر و عدم وجود انتشار خطا حاصل اجرای الگوریتم پیشنهادی خواهد بود.

ضمیمه ۱

کد matlab مربوط به پیاده سازی دو الگوریتم run length coding و runrobs برای فشرده سازی:

Run length coding

compress

```
function rnl=rlncompress(I);
[m n]=size(I);
I1=I;
rnl=[m+n m-n];
if abs(I1(1,1))>=0.5
   rnl=[rnl 1];
   flag=1;
end
if abs(I1(1,1))<0.5
   rnl=[rnl 0];
   flag=0;
end
counter=0;
i=1;
varib=[];
if mod(m,2)==0
   m2=m/2;
   m2=m2+1;
else
   m2=m+1;
```

```
   m2=m2/2;
end
for k=1:m2-1
   for j=1:n
      if(I1(i,j)==flag)
         counter=counter+1;
      else

         rnl=[rnl counter];
         flag=~flag;
         counter=1;
      end
   end
   i=i+1;
    for j=n:-1:1
      if(I1(i,j)==flag)
         counter=counter+1;
      else

         rnl=[rnl counter];
         flag=~flag;
         counter=1;
      end
    end
   i=i+1;
end
if i<=m

i=m;
if mod(m,2)~=0
   for j=1:n
      if(I1(i,j)==flag)
         counter=counter+1;
      else
         rnl=[rnl counter];
```

```
            flag=~flag;
            counter=1;
        end
    end
else
    for j=n:-1:1
        if(I1(i,j)==flag)
            counter=counter+1;
        else

            rnl=[rnl counter];
            flag=~flag;
            counter=1;
        end
    end
end
end
rnl=[rnl counter];

%.....................................................................
Decompress


function I2=runrobsdecompress(cd);

m1=(cd(1,1)+cd(1,2))/2;
n1=cd(1,1)-m1;
I2=zeros(m1,n1);
[m2 n2]=size(cd);
m=m1;
n=n1;
for i=2:m2

    rc=cd(i,1);
    r=int32(rc/n);
```

```
    c=mod(rc,n);
    if c==0 | c==1
        c=n;
    end
 %.....................
    rc2=cd(i,2);
    r2=int32(rc2/n);

    c2=mod(rc2,n);
    if c2==0 | c2==1
        c2=n;
    end
    for ir=r:r2
        for ic=c:c2
            I2(ir,ic)=1;
        end
    end
end
I2=I2(1:m1, 1:n1);
```

Runlength :

```
function runlength(filename)
I=imread(filename);
[m n]=size(I);
I1=im2double(I);
subplot(1,3,1);
imshow(I),title(size(I));
tic
rnl=rlncompress(I1);
t=toc;
imwrite(rnl,'code.bmp');
subplot(1,3,2);
```

```
imshow('code.bmp'), title(size(rnl));
%.............................decode
tic
I2=rlndecompress(rnl);
td=toc;
imwrite(I2,'ss.bmp');
subplot(1,3,3);
imshow('ss.bmp'),title(size(I2));
Runrobs:
Compress:

function cd=runrobscompress(I);
[m n]=size(I);
cd=[m+n m-n];
I1=I;
zrow=zeros(1,n);
zcol=zeros(m+2,1);
I1=[zrow ;I1 ;zrow];
I1=[zcol I1 zcol];
FRST=[];
SECND=[];
for i=2:m+1
   for j=2:n+1
      if  (I1(i,j)>0.5)  & (I1(i,j-1)<0.5)
         fr=i*n+j;
         FRST=[FRST ; fr];

      end
      if (I1(i,j)>0.5)  & (I1(i,j+1)<0.5)
         sec=i*n+j;
         SECND=[SECND ; sec];
      end
   end
end
ANSWER=[FRST SECND];
```

```
cd=[cd ; ANSWER];
```

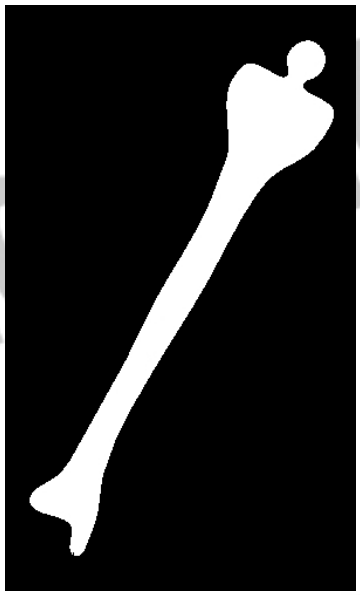
Decompress:

```
function I2=runrobsdecompress(cd);

m1=(cd(1,1)+cd(1,2))/2;
n1=cd(1,1)-m1;
I2=zeros(m1,n1);
[m2 n2]=size(cd);
m=m1;
n=n1;
for i=2:m2

   rc=cd(i,1);
   r=int32(rc/n);
   c=mod(rc,n);
   if c==0 | c==1
      c=n;
   end
 %.....................
   rc2=cd(i,2);
   r2=int32(rc2/n);

   c2=mod(rc2,n);
   if c2==0 | c2==1
      c2=n;
   end
   for ir=r:r2
      for ic=c:c2
         I2(ir,ic)=1;
      end
   end
end
```

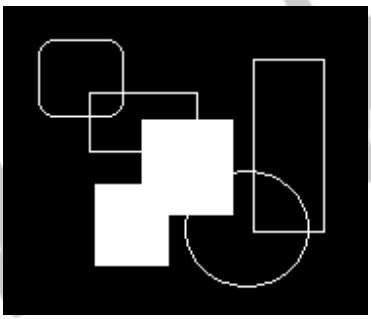
```
I2=I2(1:m1, 1:n1);

Runrobs:

function runrobs(filename);
I=imread(filename);

[m n]=size(I);

I1=im2double(I);
subplot(1,3,1);
imshow(I),title(size(I));
tic
cd=runrobscompress(I1);
t=toc;
imwrite(cd,'sprs.bmp');
subplot(1,3,2);
imshow('sprs.bmp'),title(size(cd));
tic
I2=runrobsdecompress(cd);
td=toc;
imwrite(I2,'ss.bmp');
subplot(1,3,3);
imshow('ss.bmp'),title(size(I2));
```

ضمیمه ۲

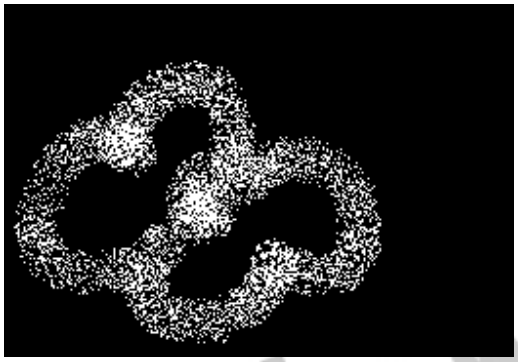
تصاویر تست شده در بخش چهارم



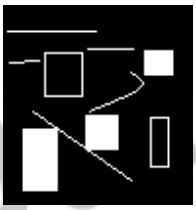
Bi3.bmp



Bi4.bmp



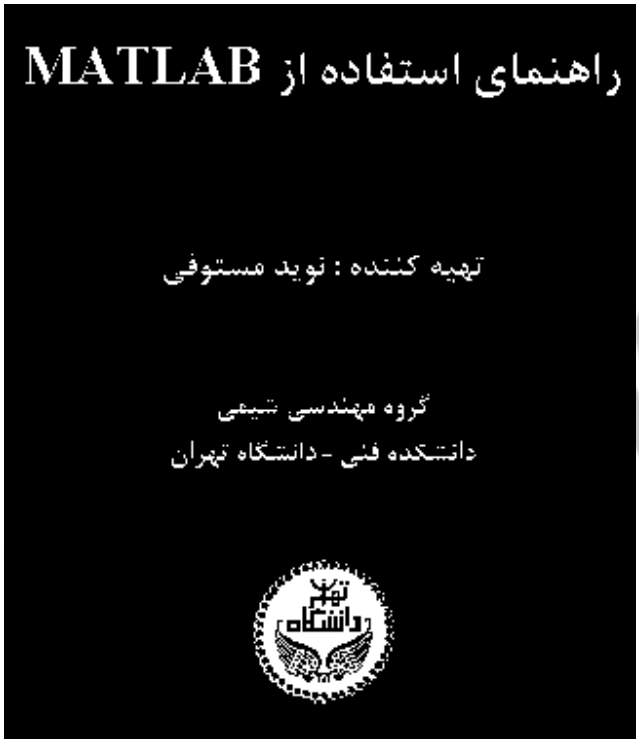
Bi5.bmp



bi8.bmp

Bi18.bmp

bi19.bmp



Ebookblack.bmp

منابع:

کتاب:

۱- مقدمه ای بر پردازش تصاویر دیجیتال (جلد اول) تالیف دکتر فرح ترکمنی آذر

۲- Digital Image Processing BY Rafael C.Gonzalez/Paul Wintz

۳- طراحی الگوریتم ها با شبه کدهای C++ ریچارد نیپولیتان- کیومرث نعیمی پور ترجمه ی عین الله جعفرنژاد قمی

مقالات

1)Image Processing on the GPU
IkkJin Ahn ikkjin@gmail.com, Michael Lehr michael.lehr@gmail.com,, Paul Turner turnerpd@seas.upenn.edu University of Pennsylvania GPU Programming and Arch
itecture February 27, 2005

2)Sparse Matrix Solvers on the GPU: Conjugate Gradients and Multigrid

Jeff Bolz, Ian Farmer, Eitan Grinspun and Peter Schröder

اینترنت:

1-http://en.wikipedia.org/wiki/Graphics_processing_unit
2-www.cs.columbia.edu/cg/pdfs/28_**GPU**Sim.pdf
3-www.hamedhabibi.com

پایان نامه:

۱ - راه اندازی آزمایشگاه پردازش تصویر با نرم افزار Matlab (محمد حسین عطار)
۲- بررسی روش های نمایش و پردازش ماتریس اسپارس و مقایسه ی آن ها (سید مصطفی طباطبایی پور)