نویسنده:Lord_Viper

مقدمه ای بر پردازش تصاویر

۱.اطلاعات رنگها:Vector Representation

۲.استفاده از فیلترها

۳.تبدیل به شیاه سفید(نور و کنتراست)

۴.تغییر اندازه و چرخش عکس

۵.ترکیب

۶.یافتن خواص-سرعت

۷.تعدادی فیلتر معمولی

۸.بالا بردن پیچیدگی FFT

۹.تشخیص حرکت

۱۰.تشخیص شکل

۱.اطلاعات رنگها:Vector Representation

مشخصا اصلی ترین و مقدماتی ترین راه برای نمایش رنگهای دیجیتال در حافظه کامپیوتر استفاده از Bitmap میباشد.یک bitmap از رایه ای از پیکسلها تشکیل شده است.هر پیکسل دارای یک مقدار خاص میباشد که بیان کننده رنگ ان پیکسل می باشد.ازتجزیه این مقدار ۳ عدد بدست می اید که مقدار رنگهای اصلی قرمز-ابی-سبز در رنگ پیکسل می باشد.هر رنگ قایل نمایشبرای چشم انسان به این صورت می باشد.تجزیه هر رنگ به رنگهای اصلی مقداری از ۰ تا ۲۵۵ بدست می اید به صور مثال از تجزیه رنگ سفید صورت زیر حاصل می شود R=255,G=255,B=255 و رنگ سیاه برابر است با RGB(0,0,0) یا صورتی روشن برابر است با RGB(255,0,255) .به بیان دیگر یک غکس از یک ارایه ۲ بعدی از

رنگها(پیکسل) تشکیل می شود.که هر کدام از انها در ۳ بایت قرار میگیرد برای نمایش ۳ رنگ اصلی هر رنگ در ۱ بایت(۱ بایت میتواند ۰تا ۲۵۵ را در خود جای دهد)این کار به عکس این قابلیت را میدهد که در مجموع میتواند ۲۵۵*۲۵۵*۲۵۵=۱۶.۸ میلیون رنگ مختلف داشته باشد.این روش به RGB Encoding معروف می باشد.

استفاده از مقدار ۰تا۲۵۵ دارای ۲ دلیل خوب می باشد چشم انسان تنها قابلیت دید بیشتر از ۲۵۶ حالت از یک رنگ را ندارد ۲۵۶/۱=۰.۳۹% برای یک رنگ.این بدان معنی است که یک عکس نمی تواند بیشتر از ۲۵۶ حالت خاکستری نمایش دهد (۲۵۶ سایه خاکستری بین سیاه و سفید)بنابر این

بنابر این مقدار ۲۵۶ کافی می باشد.دلیل دوم مقدار ۱ بایت در حافظه میتواند مقدار ۰تا۲۵۶ را در خود جای دهد.

برخلاف سیگنالهای صوتی که کد انها بر مبنای محدوده زمان می باشد.به همین دلیل انالیز اطلاعات خام عکس از ازاطلاعات فایل صوتی سرراست تر و راحتر است.به همین دلیل ما میتوانیم فیلترهای زیادی را بر روی عکس بدون تغییر و تبدیل اطلاعات ان انجام دهیم البته این کاردر نهایت برای سیگنالهای صوتی نیز امکان پذیر است. دربخش اول مقداری افکت ساده و فیلتربدون تغییر در اطلاعات عکس و فقط با انالیز ردیفها پیکسل قرار دارد.به بعدهای استاندارد رزولیشن گفته میشود bitmap های استاندارد دارای ۵۰۰ سطر و ۵۰۰ ستون هستند.این رزولیشن برای تلویزیونهای انالوگ و برنامه های معمولی کافی می باشد.شما به این وسیله به راحتی میتوانید اندازه حافظه مورد نیاز یک bitmap را حساب کنید .ما ۵۰۰*۵۰۰ پیکسل داریم که هر پیکسل ۳ بایت میباشد پس bitmap ما باید ۷۵۰kb حجم داشته باشد.توجه داشته باشید این حجمی نیست که bitmap روی هارد دیسک اشغال میکند بلکه حجمی است که هنگام پردازش در حافظه اشغال میکند زیرا الگوریتمهای فشرده سازی پیشرفته مانند jpg2000 میتواند حجم یک عکس را تا ۵۰ بار کاهش دهد بدون افت کیفیت محسوس. پس واقعا رندر کردن روان ۳۰عکس در ثانیه نیاز به پهنای باند sec/10moدارد که این محدودیت دسترسی به اطلاعات و تبدیل انها در رم در پردازش تصاویر بسیار مهم می باشدو گاهی اوقات محدود بودن cpuالبته امروزه با امدن سیستمهای قدرتمند این مشکلات دیگر تقریبا حل شده است .برای نوشتن کدها از زبان قدرتمند دلفی استفاده میکنیم که دراین زمینه قابلیتهای زیادی در اختیار ما قرار میدهد برای استفاده از فایلهای Bitmap در دلفی از کلاس Tbitmap استفاده می شود قدم اول بارگزاری فایل bitmap در حافظه می باشد از سربرگ Dialogs یک OpenPictureDialog و یک Button روی فورم قرار دهید حالا کد زیر را در رویداد OnClick دکمه قرار دهید

```
procedure TForm1.FormCreate(Sender :TObject);

var

bitmap:TBitmap;

begin

  if OpenPictureDialog1.Execute then

  begin
```

```
   bitmap=:TBitmap.Create;
   bitmap.LoadFromFile(OpenPictureDialog1.FileName);
  end;
end;
end.
```

در کد فوق ما یک متغییر از نوع Tbitmap تعریف کردیم و یک نمونه از ان را ایجاد کردیم و با استفاده از متد LoadFromFile فایل مورد نظر را در ان لود کردیم

نمایش برداری رنگها

همانطور که در bitmap دیدید رنگها در ۳ بایت نمایشداده می شوند که میتوان انها را به ۳ رنگ اصلی تجزیه کرد .اشکار است که در محاسبات ریاضی در تفسیر رنگها به صورت برداری در یک فضای ۳ بعدی هر محور نماینده یک رنگ اصلی می باشد که این برای محاسبات هندسی رنگهای ما مفید میباشد مانند مقیاس –حاصل ضرب عددی-برامدگی-چرخش-فاصله.این روشها برای تعدادی از فیلترهایی که ما استفاده میکنیم کاربرد دارد

در دلفی برای نمایش رنگ از Tcolor استفاده میشود به صورت زیر

```
var
  cl:TColor;
begin
 cl=:clOlive;
 cl=:$0000FF00;
```

به ۲ صورت میتوان به مقدار Tcolor مقدار داد ۱ از طریق رنگهای رزرو شده مثل ClRed یا با استفاده از مقدار hex ان رنگ مانند $0000FF00 برای استخراج رنگهای اصلی از توابع Get Color Value استفاده می شود

```
var
  r,g,b:Byte;
begin
```

r=:GetRValue(cl); رنگ قرمز

g=:GetGValue(cl); رنگ سبز

b=:GetBValue(cl);رنگ ابی

نمایش برداری رنگها در فضای ۳ بعدی

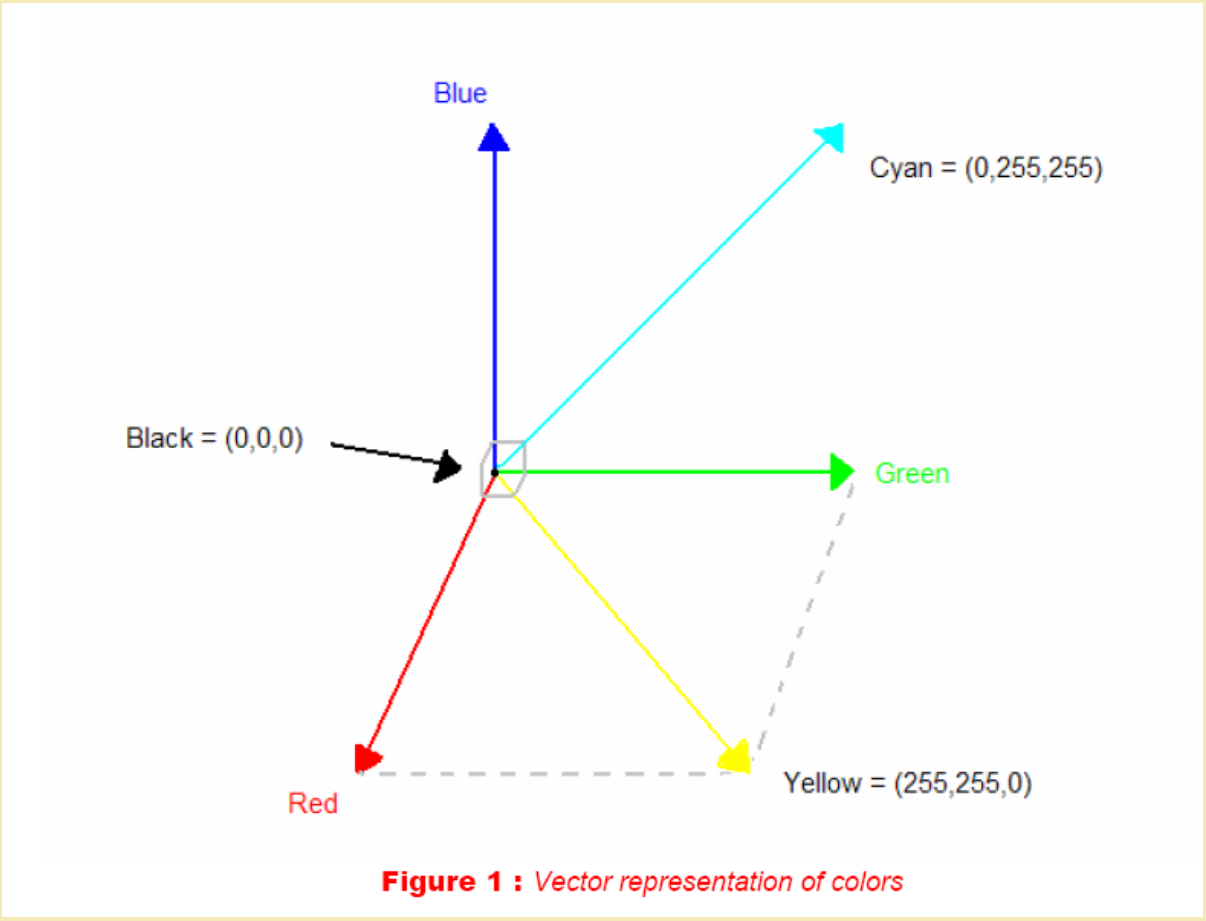


Figure 1 : *Vector representation of colors*

برای پیمایش یک Bitmap ما از یک حلقه تو در تو استفاده میکنیم چون bitmap یک ارایه ۲ بعدی از پیکسلهاست .

```
procedure TForm1.Button1Click(Sender :TObject);
var
  bitmap:TBitmap;
  cl:TColor;
  r,g,b:Byte;
  row,col:Integer;
begin
```

```
  if OpenPictureDialog1.Execute then
 begin
  bitmap=:TBitmap.Create;
  bitmap.LoadFromFile(OpenPictureDialog1.FileName);
  for row=:0 to bitmap.Width-1 do
  begin
   for col=:0 to bitmap.Height-1 do
    begin
    cl=:bitmap.Canvas.Pixels[row,col];
     r=:GetRValue(cl);
     g=:GetGValue(cl);
     b=:GetBValue(cl);
    end;
  end;
  bitmap.Free;
 end;
end;
```

استفاده از فیلتر های فوری

تشخیص لبه تصاویر

ابتدا باید از خود بپرسیم که چگونه میتوانیم تفاوت بین ۲ رنگ را بوسیله محاسبات هندسی بین بردار رنگها کم کنیم بیایید اینگونه در نظر بگیریم C1=(R1,G1,B1) و C2=(R2,G2,b2) حدفاصل بین رنگها از فرمول زیر بدست میاید

$$D(C1,C2)=\sqrt{(R1-R2)^2+(G1-G2)^2+(B1-B2)^2}$$

(R1)

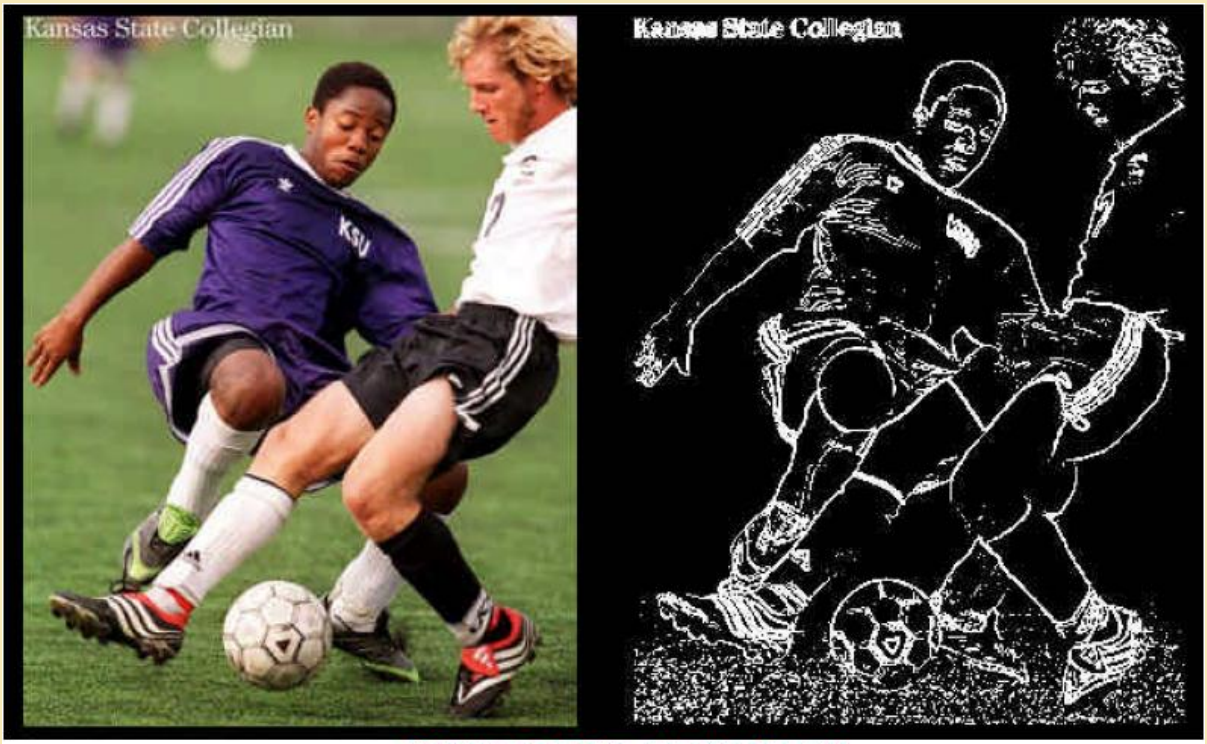
این فرمول مارا به سوی ایجاد اولین فیلتر میبرد بنام Edge Detection این فیلتر لبه های اشکال در تصاویر را به صورت لبه های سفید در پس زمینه سیاه ایجاد میکند .ایده ان بسیار ساده می باشد در این تکنیک رنگها با هم مقایسه می شوند اگر رنگفعلی با رنگ مجاور تفاوت زیادی داشته باشد یعنی لبه یک شیی می باشد و باید به رنگ سفید دراید و در غیر این صورت باید به رنگ سیاه دراید به

بیان دیگر هر پیکسل با پیکسل سمت راست و پایین خود مقایسه می شود زیرا یک عکس شامل یک ارایه ۲ بعدی میباشد اگر شما یک عکس راه راه ابی و قرمز داشته باشید و تنها به صورت افقی عمل پردازش را انجام دهید هیچ گاه لبه ای در ان نخواهید یافت اگر هر پیکسل را با پیکسل مجاور سمت راست خود مقایسه کنید پس برای هر پیکسل به ۲ مقایسه نیازمندید برای پیاده سازی این الگوریتم ابتدا برای هر پیکسل (i,j) در Bitmap کارهای زیر را انجام میدهیم ابتدا رنگهای اصلی هر پیکسل را جدا میکنیم (R,g,b)سپس رنگ مجاور راست (r1,g1,b1) و رنگ مجاور پایین (r2,g2,b2) سپس تفاوت D(c,c1) و D(c,c2) را مقایسه میکنیم برای R1 واگرD(c,c1) یا D(c,c2) مقدارشان بیشتر از پارامتر K باشد لبه تشخیص داده می شود

```
procedure TForm1.Button1Click(Sender :TObject);

var

  bitmap:TBitmap;

  cl,cl1,cl2:TColor;

  r,g,b,r1,r2,g1,g2,b1,b2,bb:Byte;

  row,col:Integer;

begin

   if OpenPictureDialog1.Execute then

  begin

   bb=:70;

   bitmap=:TBitmap.Create;

   bitmap.LoadFromFile(OpenPictureDialog1.FileName);

   for row=:0 to bitmap.Width-1 do

   begin

    for col=:0 to bitmap.Height-1 do

    begin
```

```
      cl=:bitmap.Canvas.Pixels[row,col];
      cl1=:bitmap.Canvas.Pixels[row+1,col];
      cl2=:bitmap.Canvas.Pixels[row,col+1];
       r=:GetRValue(cl);
       g=:GetGValue(cl);
       b=:GetBValue(cl);
       r1=:GetRValue(cl1);
       g1=:GetGValue(cl1);
       b1=:GetBValue(cl1);
       r2=:GetRValue(cl2);
       g2=:GetGValue(cl2);
       b2=:GetBValue(cl2);
      if(sqrt((r-r1)*(r-r1)+(g-g1)*(g-g1)+(b-b1)*(b-b1))>=bb)or
     (sqrt((r-r2)*(r-r2)+(g-g2)*(g-g2)+(b-b2)*(b-b2))>=bb)then
      bitmap.Canvas.Pixels[row,col]=:RGB(255,255,255)
      else
      bitmap.Canvas.Pixels[row,col]=:RGB(0,0,0);
      end;
    end;
    bitmap.SaveToFile('c:\sample1.bmp');
    bitmap.Free;
  end;
end;
```

مقدار bb یک مقدار دلخواه (درصد تشابه)میباشد برای تشخیص لبه که انتخاب یک پیکسل به عنوان لبه



Picture 1 : *Edge detection Results*

این الگوریتم در جاهای مختلفی بکار برده شده و نتایج خوبی بدست امده است این الگوریتم قدری کند میباشد زیرا نیاز به دسترسی مکرر به حافظه برای هر پیکسل دارد زیرا اگر ما بخواهیم تعداد بیشتری پیکسل را مقایسه کنیم مثلا ۴ پیکسل یک چند جمله ای با پیچیدگی C*R*4 حاصل می شود که Rتعداد سطرها و C تعداد ستون ها می باشد حال اگر بخواهید تصوری یک Cam با سرعت ۳۰فریم در ثانیه را پردازش کنید نیاز به چه حجم حافظه خواهید داشت این الگوریتم هنوز برای سرعت قابل قبول نیست .توجه داشته باشید که حصول نتیجه بهتر به شفاف بودن عکس بستگی دارد اگر عکس دارای لبه های شفافی باشد این الگوریتم به کمال میرسد اما اگر عکس شما کدرباشد پس قبل از استفاده از این فیلتر عکس خود را با استفاده از فیلتر sharpness شفافیت انرا بالا ببرید .شما میتوانید هر پیکسل را با ۲ یا ۳ پیکسل هم جوار خود مقایسه کنید با این کار در لبه های کلفت تر نتیجه بهتری حاصل خواهد شد در قسمت بعدی از پیچیدگی ماتریکس برای استخراج رنگها استفاده میکنیم

color Extracting

از دیگر روشهای سریع برای مقایسه پیکسلی استفاده از تکنیک استخراج رنگ(Color Extraction)می باشد در این روش ما بجای مقایسه هر پیکسل با پیکسل مجاور خود هر پیکسل با مقدار C1مقایسه می شود همه بخشها یا اشیاء درون عکس که برابر با C1 باشد شناسایی می شود این تکنیک برای کارهایی همانند روباتیک بسیار سودمند می باشد این روش به شما امکان جستجو برای رنگ خاص درون عکس را میدهد

مثلا روبات شما توپ قرمز را یافته و به سمت ان حرکت میکند .پیاده سازی الگوریتم به صورت زیر می باشد ما در عکس به دنبال C0=(R,G,B) می باشیم

برای هر پیکسل (i,j) در یک Bitmap

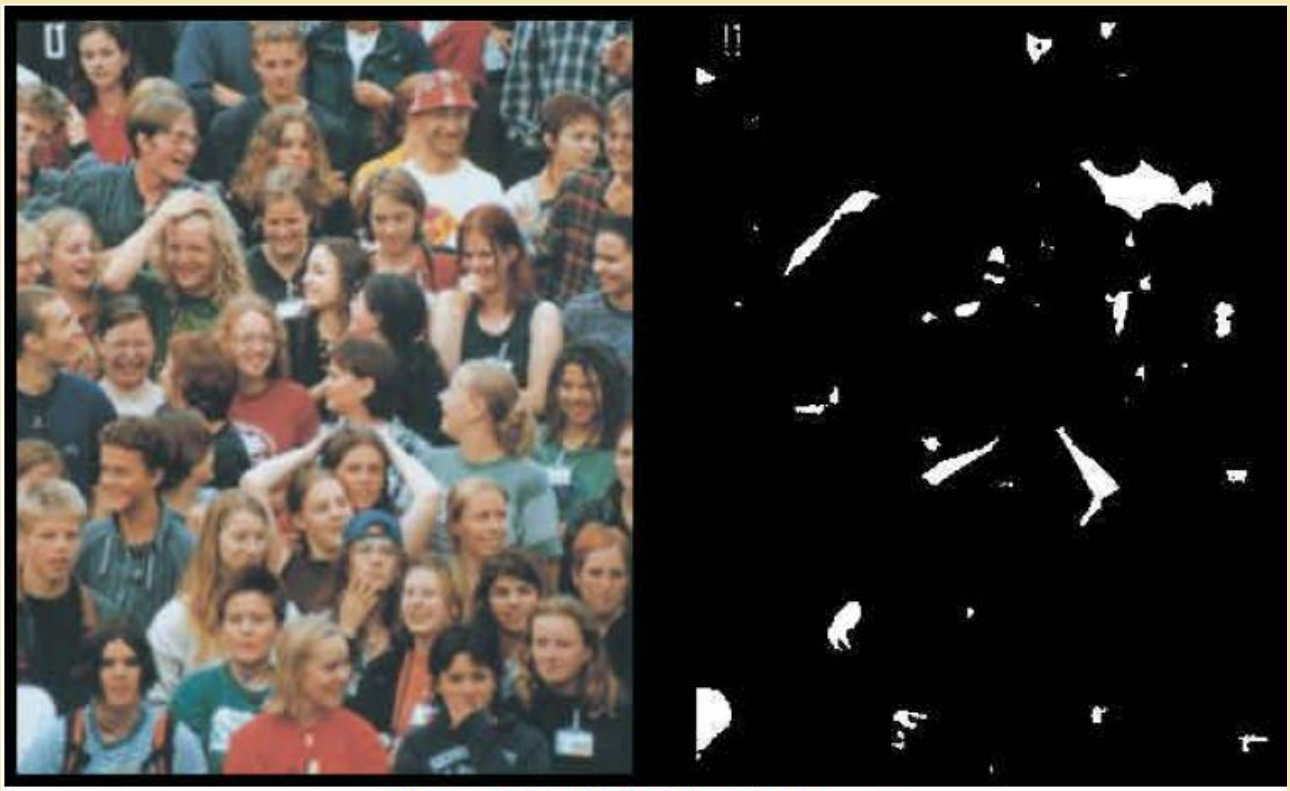
مقدار C=(R,G,B) را بدست می اوریم مقدار D(C,C0) را با استفاده از R1 محاسبه می کنیم

اگر مقدار D(C,C0) کمتر از مقدار مقدار پارامتر Kباشد رنگ پیکسل مورد نظر برابر با رنگی است که ما جستجو میکنیم.به طور مثال اگر مقدار برابر بود این پیکسل راسفید و در غیر این صورت این پیکسل را سیاه می گیریم

```
procedure TForm1.Button2Click(Sender :TObject);
var
  bitmap:TBitmap;
  cl,cl1:TColor;
  r,g,b,r1,g1,b1,bb:Byte;
  row,col:Integer;
begin
    if OpenPictureDialog1.Execute then
  begin
    bb=:70;
    cl1=:$AABBCCDD;
    r1=:GetRValue(cl1);
    g1=:GetGValue(cl1);
    b1=:GetBValue(cl1);
    bitmap=:TBitmap.Create;
    bitmap.LoadFromFile(OpenPictureDialog1.FileName);
    for row=:0 to bitmap.Width-1 do
    begin
      for col=:0 to bitmap.Height-1 do
      begin
      cl=:bitmap.Canvas.Pixels[row,col];
       r=:GetRValue(cl);
```

```
      g=:GetGValue(cl);
      b=:GetBValue(cl);
     if sqrt((r-r1)*(r-r1)+(g-g1)*(g-g1)+(b-b1)*(b-b1))<=bb then
     bitmap.Canvas.Pixels[row,col]=:RGB(255,255,255)
     else
     bitmap.Canvas.Pixels[row,col]=:RGB(0,0,0);
     end;
   end;
   bitmap.SaveToFile('c:\sample1.bmp');
   bitmap.Free;
  end;
end;
```

مقدار bb یک مقدار دلخواه میباشد(درصد تشابه). با برداشتن ریشه دوم باز هم سرعت هم سرعت الگوریتم تغییر خاصی پیدا نخواهد کرد زیرا پیکسلها در حلقه پردازش می شوند



Picture 2 : *White Extraction Results*

Color to GrayseCale Conversion

در فضای ۳ بعدی رنگها رنگ خاکستری مستقیما به صورت(N,N,N) ایجاد می شود که N یک مقدارعددی intger بین ۰ تا ۲۵۵ می باشد به طور مثال black (0,0,0), dark gray(32,32,32) ,

(255,255,255) white,(192,192,192) bright gray,(128,128,128) intermediate gray

ایده این الگوریتم پیدا کردن یک مقدار برای هدایت رنگ به بردار (1,1,1) می باشد

ما از یک مقدار عددی برای بدست اوردن ان استفاده میکنیم مقدار مورد نظر ما در بردارC=(R,G,B) در بردار (1,1,1) به صورت زیر محاسبه می شود

$$\overline{a} \bullet \overline{b} = ||\overline{a}|| \bullet ||\overline{b}|| \bullet \cos(\alpha)$$

(R2)

$$Proj(C/(1,1,1)) = ||\overline{b}|| \bullet \cos(\alpha) = \frac{1\times R + 1\times G + 1\times B}{\sqrt{3}}$$

(R3)

به هر حال مقدار مورد نظر ما می تواند تا مقدار ۴۴۱.۶۷ برسد که برای رنگ سفید می باشد (255,255,255) برای جلوگیری از بدست امدن مقدار بیشتر از محدوده ۲۵۵ ما مقدار بدست امده را در عدد ۴۴۱.۶۷/۲۵۵یا (۱/مجزور۳)ضرب میکنیم .رنگخاکستری از فرمول زیر بدست می اید

$$Grayscale(C) = \frac{255}{\sqrt{255^2 + 255^2 + 255^2}} \cdot Proj(C/(1,1,1)) = \frac{1}{\sqrt{3}} \frac{1\times R + 1\times G + 1\times B}{\sqrt{3}} = \frac{R+G+B}{3}$$

(R4)

پس در این صورت برای تبدیل یک رنگ به خاکستری معدل ۳ مقدار قرمز –سبز-ابی ان را محاسبه میکنیم شما میتوانید این فرمول را برای رنگهای دیگر نیز پیاده سازی کنید مثلا برای داشتن عکس قرمز میتوانید فقط مقدار رنگ قرمز را انتخاب کنید ((R=Red Scale C یا عکس زرد داشته باشید عکس زرد از تلفیق سبز و ابی بدست می اید Yellowscale(C)=(B+G)/sqrt(6) پیاده سازی این الگوریتم به صورت زیر می باشد.

برای هر پیکسل بکار رفته در bitmap (row,col)

مقدار c=(R,G,B) هر پیکسل را بدست می اوریم

مقدار Graysecale ان رنگ را محاسبه میکنیم

مقدار رنگ پیکسل (row,col) در bitmap خروجی رنگ Grayscalse محاسبه شده را قرار می دهیم

```
procedure TForm1.Button3Click(Sender :TObject);
var
  bitmap:TBitmap;
  cl:TColor;
  r,g,b,av:Byte;
  row,col:Integer;
begin
    if OpenPictureDialog1.Execute then
  begin
    bitmap=:TBitmap.Create;
    bitmap.LoadFromFile(OpenPictureDialog1.FileName);
    for row=:0 to bitmap.Width-1 do
    begin
      for col=:0 to bitmap.Height-1 do
      begin
      cl=:bitmap.Canvas.Pixels[row,col];
       r=:GetRValue(cl);
       g=:GetGValue(cl);
       b=:GetBValue(cl);
       av=:(r+g+b)div 3;
      bitmap.Canvas.Pixels[row,col]=:RGB(av,av,av);
      end;
    end;
    bitmap.SaveToFile('c:\sample1.bmp');
```

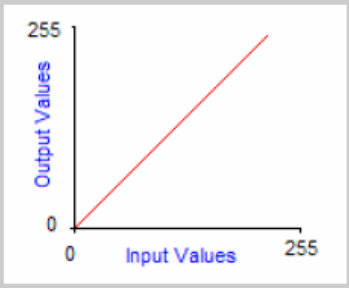
```
  bitmap.Free;
 end;
end;
```

این الگوریتم قابلیت ساده تر شدن را نیز دارد اما در پردازشهای لحضه ای پردازش تعداد Frame per sec*Row*Col مثلا ۳۵ فریم درثانیه باز قابل قبول نیست اگر ما در عکس رنگ قرمز خالص c=(255,0,0) یا سبز خالص c=(0,255,0) یا ابی خالص c=(0,0,255) داشته باشیم مقدار خاکستری این رنگها یکسان خواهد بود برابر ۳/۲۵۵ که در بعضی جاها قابل استناد نمی باشد برای رفع این مسئله در صورت نیاز میتوانید از مقدار Y در کد رنگ YUV که توضیحات ان در annexe وجود دارد استفاده کنید

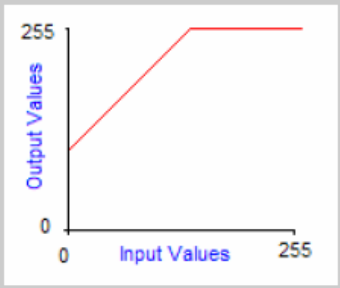
Picture 3 : *Grayscale conversion Results*

light and contrast :Grayscale transforms

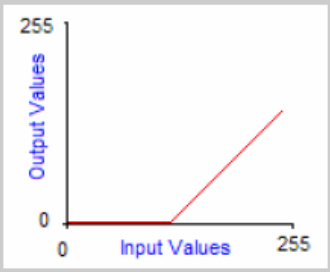
تبدیل به خاکستری یک مقدار integer از ۰-۲۵۵ تا ۰-۲۵۵ میباشد در این حالت ما از اعداد بین ۰تا ۲۵۵ می توانیم یک عدد دیگر برابر با ان بین ۰تا۲۵۵ بدست اوریم برای بدست اوردن رنگهای یک عکس با استفاده از مبدل سیاه سفید(Grayscale transform)ابتدا باید رنگهای قرمز-سبز-ابی از هم جدا شود و سپس به صورت یک رنگ واحد دراید نمودار مبدل سیاه سفید به نام up table-Outrput look یا منحنی گاما ((Gamma Curve نامیده می شود .در عمل شما می توانید با کنترل منحنی گاما کارت ویدئوی خود و تنظیم ان روی شفافیت یا کنتراست تا هر زمان پیکسلی به سوی صفحه فرستاده شد ابتدا انرا ازمبدل سیاه سفید عبور دهد .این یک افکت برای افزایش شفافیت و کنتراست به منحنی گاما متناظر با مبدل سیاه سفید می باشد



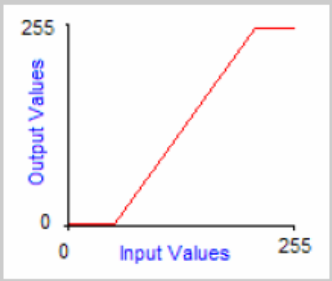
Normal grayscale transform



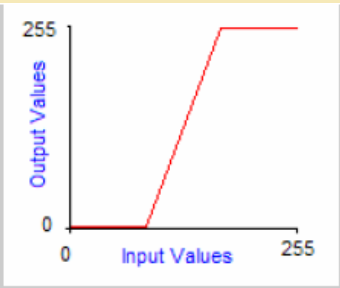
Increased brightness by 128



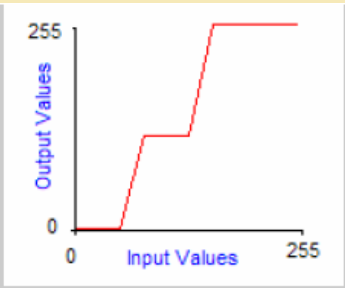
Decreased brightness by 128



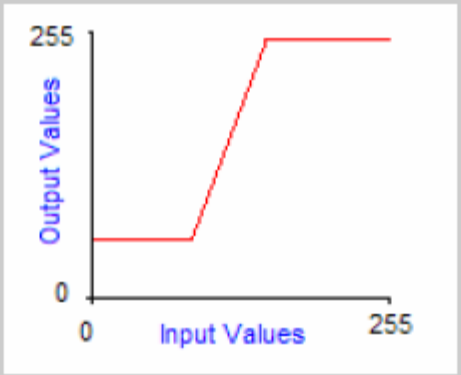
Increased contrast by 15°



Increased contrast by 30°



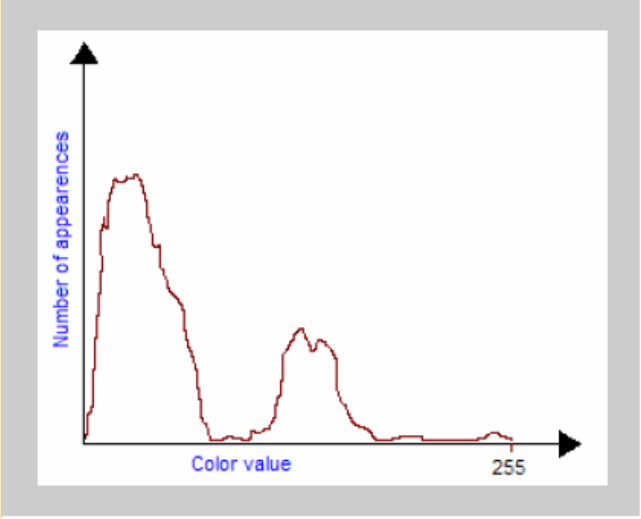
Increased contrast by 30° at 64 and 150



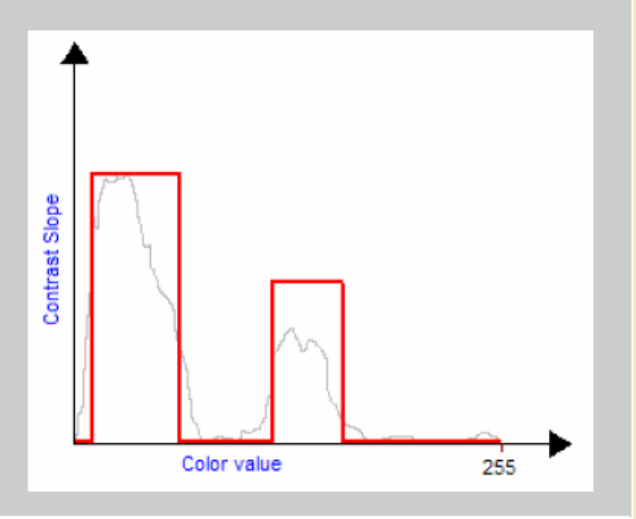
Increased brightness and contrast

توجه داشته باشید که اگر شما میخواهید یک مبدل شفافیت و کنتراست بسازید ابتدا باید مبدل کنتراست بسازید تا کیفیت عکس را بالا ببرید .ایجاد یک مبدل سیاه سفید برای بالا بردن قابلیت کیفیت عکس اسان می باشد .اگر شما یک عکس بسیار تاریکبا یک نقطه کوچک روشن در ان دارید شما نمی توانید شفافیت انرا

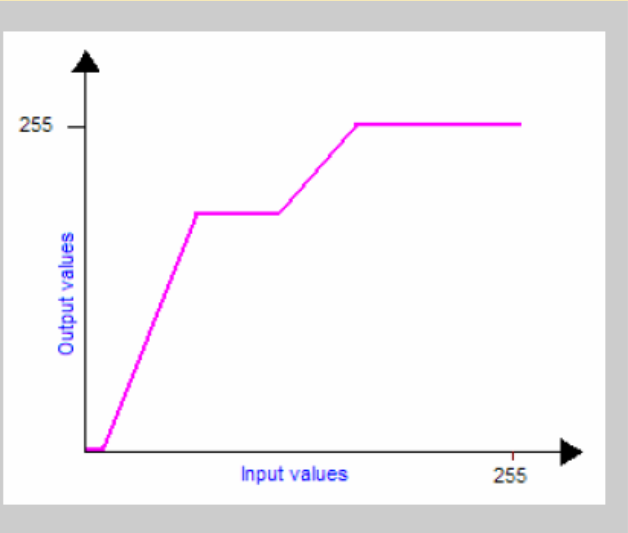
زیاد بالا ببرید زیرا ان نقطه کوچک روشن به زودی اشباح شده اما به راحتی میتوان مقدار کنتراست هر پیکسل را بالا برد این کار باعث پهن کردن رنگهای فاصله بین ۲ حالت (تاریکی و شفافیت) عکس شده و باعثمی شود تا عکس رنگهای بیشتری را در بر گیرد این کار visiblity عکس را بالا می برد برای اینکار ابتدا باید تراکم نمایش پیکسلها درون محدوده ۰تا۲۵۵ شمارش شود سپس کنتراست درونی که نیاز داریم را ایجاد کنیم واین اخرین تبدیلات را انجام میدهیم در این حالت مقدار انتگرال کنتراست یک مقدار نزولی می باشد (شیب ان به سمت پایین خواهد بود)که در نمودارهای زیر مشخص است



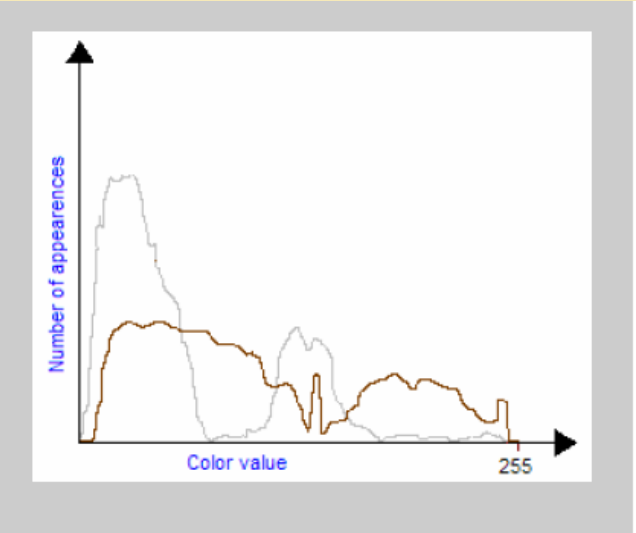
Bitmap input color histogram: two large amounts of pixels in thin color bandwidths.



The contrast slope must be big in those two areas to flatten the input histogram.



The contrast transform: It is obtained by integrating the previous plot.



The input bitmap color histogram before and after the contrast transform: Now pixels are averagely distributed on the whole color bandwidth.

Light and contrast

روشنایی و کنتراست

تبدیلات بلادرنگ کنتراست و روشنایی بسیار مهم میباشد جدول خروجی Table- look up یک بار برای همیشه در حافظه ذخیره میشود سپس هر پیکسل به کمک این جدول در Bitmap خروجی مشخص می شود .جدول up-look به اسانی همانند یک لیست قابل نمایش است ایندکس این لیست مقدار رنگ ورودی و کنتراست خروجی انرا مشخص میکند پس با فراخوانی لیست ۱۵۳ به ما عکسی با عکس سیاه سفید سطح ۱۵۳(level 53) تحویل خواهد داد

پیاده سازی الگوریتمهای تغییر کنتراست و شفافیت

ابتدا جدول up-look مورد نظر خود را ایجاد کنید.سپس برای هر ایندکس j از ۰تا۲۵۵ که در عکس قرار دارد بوسیله List-Transform[j]

برای هر پیکسل درون عکس (i,j)

مقدار R,G,B هر پیکسل را استخراج میکنیم

سپس رنگ پیکسل در خروجی را بوسیله ([R]transform_list ,[G]transform_list ,[R]transform_list)بدست می اوریم

در سورس کد پیاده سازی کنتراست یک مقدار از نوع float وجود دارد که مشخص کننده کمیت زاویه شیب کنتراست می باشد اگر کنتراست 4/pi تعریف شده باشد تغییری در عکس ایجاد نخواهد کرد زیرا این مقدار برابر با normal میباشد تغییرات کنتراست از مقدار 15^+4/contrast>piشروع می شود اگر مقدار کنتراست به ۰ برشد کل عکس سیاه سفید می شود زیرا تبدیل افقی برابر ۱۲۸ می شود .اگر مقدار کنتراست بیش از حد زیاد شود تمام رنگها به ۰ یا ۲۵۵ نزدیک می شوند در این حالت عکس خروجی داردی ۸ رنگ خواهد شد که از ۰ و ۲۵۵ تشکیل می شود(0,0,0).

(255,255,255),(0,255,255)-(255,0,255)-(255,255,0)-(255,0,0)-(0,0,255)-(0,255,0)

الگوریتم مربوط به کنتراست

با کم و زیاد کردن مقدار متغییر radius میتوانید کنتراست را کم یا زیاد کنید

```
Uses

Math

function lookupTable(loop:Integer):Integer;

var

  radius:Real;

begin
```

```
  radius=:0.15;
    if (loop<Round(128.0+128.0*tan(radius)))and(loop>Round(128.0-
128.0*tan(radius)))then
    Result=:Round((loop-128)/tan(radius)+128)
    else if loop>=(Round(128.0+128.0*tan(radius)))then
    Result=:255
    else
    Result=:0;
end;
procedure TForm1.Button4Click(Sender :TObject);
var
  bitmap:TBitmap;
  cl:TColor;
  r,g,b:Byte;
  row,col:Integer;
begin
    if OpenPictureDialog1.Execute then
  begin
    bitmap=:TBitmap.Create;
    bitmap.LoadFromFile(OpenPictureDialog1.FileName);
    for row=:0 to bitmap.Width-1 do
    begin
      for col=:0 to bitmap.Height-1 do
      begin
      cl=:bitmap.Canvas.Pixels[row,col];
```

```
      r=:GetRValue(cl);
      g=:GetGValue(cl);
      b=:GetBValue(cl);
bitmap.Canvas.Pixels[row,col]=:RGB(lookupTable(r),lookupTable(g),lookupTable(b));
     end;
   end;

bitmap.SaveToFile(ExtractFilePath(OpenPictureDialog1.FileName)+'sample1.bmp');
   bitmap.Free;
  end;
end;
```

الگوریتم مربوط به افزایش روشنایی

با کم یا زیاد کردن متغییر light میتوانید روشنایی تصویر را کم یا زیاد کنید

```
uses
math;
function brightness(value:byte):Byte;
var
  i:Integer;
  light:Byte;
begin
  light=:100;
  i=:i+light;
  if i>255 then
  Result=:255
  else if i<0 then
```

```
  Result=:0
  else
  Result=:i;
end;
procedure TForm1.Button5Click(Sender :TObject);
var
  bitmap:TBitmap;
  cl:TColor;
  r,g,b:Byte;
  row,col:Integer;
begin
    if OpenPictureDialog1.Execute then
  begin
    bitmap=:TBitmap.Create;
    bitmap.LoadFromFile(OpenPictureDialog1.FileName);
    for row=:0 to bitmap.Width-1 do
    begin
      for col=:0 to bitmap.Height-1 do
      begin
      cl=:bitmap.Canvas.Pixels[row,col];
       r=:GetRValue(cl);
       g=:GetGValue(cl);
       b=:GetBValue(cl);
      bitmap.Canvas.Pixels[row,col]=:RGB(brightness(r),brightness(g),brightness(b));
```

```
    end;
  end;

bitmap.SaveToFile(ExtractFilePath(OpenPictureDialog1.FileName)+'sample1.bmp');
  bitmap.Free;
 end;
end;
```

چند مثال از الگوریتمهای فوق

Picture 4 : *Brightness transform result*

Picture 5 : *Contrast transform result*

Resizing algorithm

تغییر اندازه عکس خیلی ساده می باشد .الگوریتمهای کمی وجود دارد که هر کدام نتیجه کم و بیش خوب در مورد بزرگ کردن یا کوچک کردن تصویر به ما میدهند .الگوریتم Liner Resizing الگوریتمی است که ما در مورد ان صحبت خواهیم کرد البته متد های مناسب دیگری نیز وجود دارد معمولا تغییر اندازه عکس به ندرت پیش میاید.در این الگوریتم کیفیت بیشتر از سرعت می باشد .مقدار E(x) جزء صحیح از X برابر E(14.3)=3 می باشد و F(x) جزء کسری ان F(14.3)=14.0 .در ابتدا ما باید بدانیم که میخواهیم تصویر را چه مقدار بکشیم یا کوچک کنیم این تکنیک برای درازا و پهنا یکی می باشد و شما میتوانید یک bitmap به اندازه های دلخواه ایجاد کنید .فرض کنید یک ردیف پیکسل خروجی برابر wout باشد و شماره هر پیکسل در این ردیف Xi باشد برای پیدا کردن هر رنگ در این پیکسل ما مقدار Xi را به مقدار معادل خود Ci در bitmap مبدا تبدیل میکنیم اگر یک ردیف پیکسل ورودی را win در نظر بگیریم رنگ هر پیکسل خروجی از فرمول زیر بدست می اید

$$\frac{xi}{wout} = \frac{ci}{win}$$

$$ci = xi \times \frac{win}{wout}$$

رنگ پیکسل xi در عکس خروجی برابر رنگ رنگ پیکسل متناظر با ان در Ci در عکس مبدا می باشد .مشکل اینجاست که مقدار Ciیک مقدار شناور(اعشاری)می باشد و نمیتوان به اندازه یک پیکسل اعشاری دست پیدا کرد دراین حالت Ci بین ۲ پیکسل Ci و Ci+1 قرار میگیرد که شما میتوانید یکی را انتخاب کنید با

گرد کردن ان به جزء صحیح بالایی یا پایینی .رنگ نهایی برای یکسلهای عکس خروجی در نهایت از فرمول زیر بدست می اید

$$Color(xi) = Color(E(ci)) \; + \; (F(ci)) \times \frac{Color(E(ci)+1) - Color(E(ci))}{(i+1)-(i)=1}$$

با فرمول زیر ساده شده فرمول بالایی می باشد

$$Color(xi) = (1-F(ci)) \cdot Color(E(ci)) \; + \; F(ci) \cdot Color(E(ci)+1)$$

هنگامی که ما یک عکس را میشیم(بزرگ میکنیم)این الگوریتم اطلاعات مربوط به پیکسلهای جدید را از طریق درون یابی خطی پیدامیکند.تفاوت میان درون یابی خطی برای محاسبه رنگ یا گرفتن رنگ از E(Ci) زیاد گیج کننده نیست اما دومی حالت نرم تری دارد و وقتی شما یک نسبت زیرگ نمایی زیادی دارید این روش کاملا بهتر از روش اول عمل خواهد کرد متد اولی معروف به neighbour-Nearest است و متد دومی معروف به Liner-bi.البته photoshop از متد دیگری بنام Cubic-Bi استفاده میکند

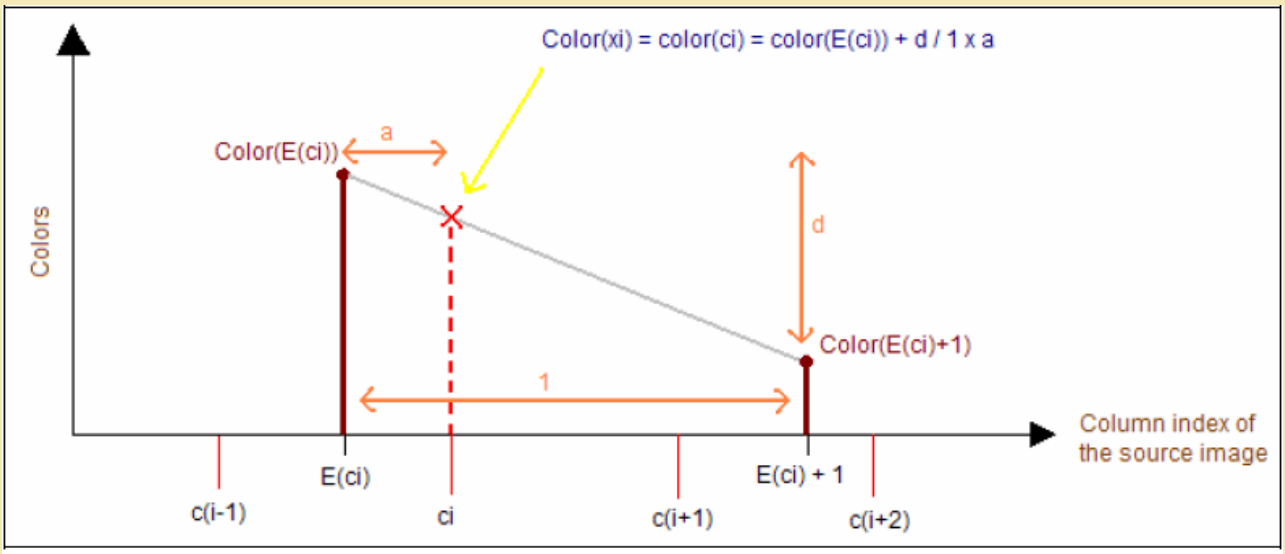


Figure 4: *Color interpolation in stretching image*

پیاده سازی الگوریتم

ابتداما عکس را از درازا کش میدهیم

برای هر پیکسل در bitmap ورودی (row,col)

ابتدا Ci را توسط فرمول محاسبه میکنیم

سپس (R,G,B) رنگ پیکسل اصلیE(Ci) و پیکسل بعدی E(Ci)+1را استخراج میکنیم

رنگ پیکسل bitmapخروجی(i,j) را از طریق فرمول بدست می اوریم

همین مراحل دقیقا برای برای کشیدن پهنا بکار می بریم

```
procedure TForm1.Button6Click(Sender :TObject);
var
  bitmap,bitmap2:TBitmap;
  cl,ct:TColor;
  r,g,b,r1,g1,b1:Byte;
  row,col,siz,ess,fss,rx,gx,bx:Integer;
  be,ass,css:Real;
begin
    if OpenPictureDialog1.Execute then
  begin
    bitmap=:TBitmap.Create;
    bitmap2=:TBitmap.Create;
    try
    bitmap.LoadFromFile(OpenPictureDialog1.FileName);
    siz=:StrToInt(Edit1.Text);
    bitmap2.Width=:siz;
    bitmap2.Height=:bitmap.Height;
    be=:bitmap.Width/siz;
```

```
    for row=:0 to siz-1 do
    begin
     for col=:0 to bitmap.Height-1 do
     begin
     ass=:be*row;
     ess=:Round(ass);
     fss=:ess+1;
     css=:frac(ass);
     cl=:bitmap.Canvas.Pixels[ess,col];
     ct=:bitmap.Canvas.Pixels[fss,col];
      r=:GetRValue(cl);
      g=:GetGValue(cl);
      b=:GetBValue(cl);
      r1=:GetRValue(ct);
      g1=:GetGValue(ct);
      b1=:GetBValue(ct);
      rx=:r-r1;
      gx=:g-g1;
      bx=:b-b1;
     bitmap2.Canvas.Pixels[row,col]=:RGB(Round(r+css*(r1-r)),Round(g+css*)g1-
g)),Round(b+css*(b1-b)));
     end;
    end;
bitmap2.SaveToFile(ExtractFilePath(OpenPictureDialog1.FileName)+'sample1.bmp');
    finally
```

```
  bitmap.Free;
  bitmap2.Free;
  end;
 end;
end;
```

rotate Algorithm

الگوریتم چرخش عکس

الگوریتمهای زیادی برای چرخش عکس وجود دارد.الگورتمهای سرعتر پیچیده تر هستند .ما یک متد پایه و کارا را برای این کار انتخاب کردیم .ابتدا مقداری ریاضیات برای درک الگوریتم فرض میکنیم که x و y مختصات یک پیکسل در یک bitmap باشد ما میتوانیم مختصات x' و y' را با مرکز چرخش x0 و y0 با زاویه مورد نظر ازفرمول زیر بدست اوریم

$$x' = (x - x0) \cdot \cos(\alpha) - (y - y0) \cdot \sin(\alpha) + x0$$
$$y' = (x - x0) \cdot \sin(\alpha) + (y - y0) \cdot \cos(\alpha) + y0$$

اگر ما مستقیما از این فرمول برای هر کدام از پیکسلهای عکس استفاده کنیم سوراخای زیادی در عکس ایجاد خواهد شد و این به خاطر ان است که مقدار x وy یک مقدار صحیح می باشند بعضی از x'وy' ها مقداری بدست نمی اید و باید به مقدار قبلی خود بازگردند .روشی که برای حل این مشکل می توان به کار برد انتخاب پیکسل نهایی و چرخش معکوس و بدست اوردن رنگ پیکسل مبدا می باشد .پس ما بااستفاده از x' و y' برای بدست اوردن xوy استفاده می کنیم

$$x = (x' - x0) \cdot \cos(-\alpha) - (y' - y0) \cdot \sin(-\alpha) + x0$$
$$y = (x' - x0) \cdot \sin(-\alpha) + (y' - y0) \cdot \cos(-\alpha) + y0$$

وقتی پیکسل اولیه مشخص شد رنگ پیکسل نهایی را برابر ان قرار میدهیم

پیاده سازی الگوریتم به صورت زیر می باشد

برای هر پیکسل در bitmap خروجی [row,col]=(x',y')مختصات پیکسل اولیه(x,y) را با استفاده از فرمول فوق حساب میکنیم

سپس رنگ پیکسل نهایی را برابر پیکسل اولیه که بدست اوردیم قرار میدهیم

```
procedure TForm1.Button7Click(Sender :TObject);
var
```

```
  bitmap,bitmap2:TBitmap;
  row,col,HalfWidth,HalfHwight,bs,cs,x,y:Integer;
  sinVal,CosVal,angle:real;
begin
   if OpenPictureDialog1.Execute then
  begin
   bitmap=:TBitmap.Create;
   bitmap2=:TBitmap.Create;
   try
   bitmap.LoadFromFile(OpenPictureDialog1.FileName);
   bitmap2.Height=:bitmap.Height;
   bitmap2.Width=:bitmap.Width;
   angle=:0.15;
   sinval=:Sin(angle);
   CosVal=:cos(cosval);
   HalfWidth=:bitmap.Width shr 1;
   HalfHwight=:bitmap.Height shr 1;
   for col=:0 to bitmap.Height do
   begin
    for row=:0 to bitmap.Width do
    begin
    bs=:row-HalfWidth;
    cs=:col-HalfHwight;
    x=:round(bs*sinval+cs*cosval+HalfWidth);
```

```
      y=:Round(bs*cosval-cs*sinval+HalfHwight);
      bitmap2.Canvas.Pixels[row,col]=:bitmap.Canvas.Pixels[x,y];
      end;
    end;
bitmap2.SaveToFile(ExtractFilePath(OpenPictureDialog1.FileName)+'sample1.bmp');
    finally
    bitmap.Free;
    bitmap2.Free;
    end;
  end;
end;
```

این الگوریتم نسبت به نمونه های مشابه از سرعت بالاتری برخوردار می باشد .پیچیدگی این الگوریتم col*row می باشد .البته راهی برای ارتقای این الگوریتم وجود دارد به جای اینکه ما پیکسل مقصد را بدیت اوریم و با معکوس زاویه پیکسل مبدا را حساب کنیم ما ای کار را برای خطوط انجام میدهیم برای هر خط افقی در عکس همانند قبل معکوس زاویه را برای نقاط حساب میکنیم ما رنگ تمام پیکسلهای این خط افقی را میدانیم زیرا ما تمام رنگهای bitmap مبدا را داریم بدین ترتیب مامحاسباتی را که برای هر پیکسل انجام میدادیم برای هر خط انجام میدهیم

پیاده سازی الگوریتم

برای هر سطر پیکسل در bitmap خروجی

ما اولین پیکسل(x1,y1) و اخرین پیکسل (x2,y2)را محاسبه میکنیم

برای هر پیکسل ما بین (x1,y1) و(x2,y2) nvدر bitmap ورودی ما رنگ ان پیکسل را برای خط افقی در bitmap خروجی قرار میدهیم

Blending

Average

ترکیب به زبان ساده از میانگین مقدار ۲ پیکسل برای ایجاد افکت شفاف می باشد.مقدار a بین ۲ رنگ ترکیب می شود مقدارمیانگین حاصله میتواند تعادل بیشتر یا کمتر ایجاد کند اگر مقدار عمق رنگ ۲ پیکسل کم و زیاد شود.بعنوان مثال اگر شما بخواهید بین ۲ رنگ c1و c2 تعادل ایجاد کنید میتوانید از فرمول زیر استفاده کنید

$$FinalColor = \frac{\alpha \cdot C1 + \beta \cdot C2}{\alpha + \beta}$$

از این تکنیک دربازیهای ۳ بعدی یا دمو ها برای ایجاد افکت شفاف Transparency یا transition effect در فیلمها استفاده می شود ایجاد تعادل معمولا بین ۲ عکس یا یک عکس با یک رنگ ایجاد می شود شما حتی میتوانید یک عکس را با خودش متعادل کنید .در اینجا ما ۲ عکس b1 و b2 داریم که میخواهیم انها را ترکیب کنیم این عکسها معمولا به طور کامل لبه دار و تمیزو هم اندازه نیستند ما عکس بزرگتر را b1 و عکس کوچکتر را b2 در نظر میگیریم اگر مقدار FinalColor دارای مقدار بود انرا به عنوان رنگ خروجی قرار میدهیم در غیر این صورت رنگ پیکسل b1 بعنوان رنگ خروجی در نظر گرفته می شود

Other blend remark

ترکیب blending به شما امکان ایجاد افکتهای زیادی را میدهد.شما با نوسان A یا B میتوانید ترکیبات گرادیانت gradiant belending ایجاد کنید-افکت روشنایی-افکت گداختن-افکت شفافیت-افکت اب-افکت باد و بسیاری از افکتهای جالب فوتوشاپ از این روش استفاده میکنند

توجه داشته باشید با استفاده از Lookup table شما به سرعت میتوانید به تکنیک ترکیب دست پیدا کنید بیایید یک جدول ۲۵۵*۲۵۵ ایجاد کنیم هر کدام از سطرها و ستونها میتواند مقداری بین ۰-۲۵۵ داشته باشد هنگامی که ما مقدار یک سلول از سطر و ستون خاصی را میابیم مقدار ان برابر ترکیب ۲ رنگ مورد نظر می باشد پس بنابر این ما جدولی خواهیم داشت که سلول [I,j] آن نتیجه ترکیب ۲ رنگ بین ۲ رنگ i و j با استفاده از فرمول (i+j)/2 می باشد. با این کار نمیمی از حافظه بیخودی اشغال می شود اما در عوض زمان انجام محاسبات به مقدار قابل ملاحظه ای کاهش می یابد زیرا پردازش اضافه ای در حلقه قرار نمیگیرد تنها جدول رنگهاست اگر شما این کار را با یک رنگ ثابت مثلا قرمز انجام دهید جدول شما ۲۵۵*۱ خواهد بود زیرا تنها ۱ امکان برای رنگ نهای وجود خواهد داشت معمولا شما از این جدول برای ۳ جزء رنگ به صورت جداگانه استفاده میکنید و نتیجه انرا بعنوان رنگ نهایی نمایش خواهید داد

```
var

  Form1: TForm1;

   b1,b2:TBitmap;

implementation


uses Math;


{$R *.dfm}


procedure TForm1.Button1Click(Sender: TObject);

var s:String;

begin


  if OpenDialog1.Execute then

  begin

    b1.Free;

    b1:=TBitmap.Create;

    s:=OpenDialog1.FileName;

    b1.LoadFromFile(s);
```

```
  end;
end;

procedure TForm1.Button2Click(Sender: TObject);
var s:string;
begin
  if OpenDialog1.Execute then
  begin
    b2.Free;
    b2:=TBitmap.Create;
    s:=OpenDialog1.FileName;
    b2.LoadFromFile(s);
  end;
end;

procedure TForm1.Button3Click(Sender: TObject);
var x,y : integer;
    p1:PByteArray;p2:PByteArray;
begin
  Image1.Height:=b1.Height;
  image1.Width:=b1.Width;
  for y:=0 to Min(b1.Height-1,b2.Height-1) do
  begin
    p1:=b1.ScanLine[y];
    p2:=b2.ScanLine[y];
```

```
    for x:=0 to Min(b1.Width-1,b2.Width-1) do
    begin
      Image1.Canvas.Pixels[x,y]:=
        rgb(((p1[x*3]-4)+(p2[x*3]-4))div 2,
           ((p1[x*3]-2)+(p2[x*3]-2))div 2,
           ((p1[x*3]-3)+(p2[x*3]-3))div 2);
    end;
  end;
end;
```

در اینجا قسمت اول اموزشها به پایان رسید در قمت بعدی به استفاده از ماتریسها در پردازش تصاویر و تشخیص خواهیم پرداخت

Ghoghnoose.dsana@gmail.com

WwW.xexample.com