# Chapter 3

### Image Manipulation and Enhancement

* Image quality can be improved by simple operations on individual pixels.
* These usually include manipulating pixels data such as adding and subtracting, and changing the pixel value, e.g. brightness, color, etc.

**3.2 Arithmetic Operations**

We have several simple arithmetic operations. Adding or subtracting a value to every pixel, lightens or darkens and image. The picture blonde is darkened by 50 for every pixel, as shown below:

*>> bld=imread('blonde.bmp');*

*>> imshow(bld-50)*

Blonde.bmp Blonde.bmp darkened by 50

Fig. 1 Effect of changing graylevel values

Note that the operation of adding and subtracting clips, i.e. values above 255 are set to 255, and negative values are set to zero.

We can also multiply and divide each pixel value by a constant, e.g.

*>> bld2=bld\*2; bld3=bld/3;*

**Reflection**: This operation can be performed in place by reversing the ordering of pixels in the row or column. The reflection of the image f(x, y) about x = width/2 axis, and y=height/2 result in images as follows

Original image Reflection about x axis Reflection about y axis

Fig. 2 Effect of reflections

**Rotation**: This is a simple operation if the angle of rotation is a multiple of 90 degrees and the rotation is about the image center. Rotation of 90 deg or 270 deg requires creation of a new image with dimensions interchanged.

**Addition and averaging**: Like other simple operations, this operation is on pixel by pixel basis. Thus if the sizes of two images are different, then the resulting image will have the width equal to the minimum of the widths of the two images, and the same applies to the height.

* If we add two 8-bit images, the resulting pixel values could range from 0 to 510. If the resulting image is to be 8-bit then we must divide every pixel value by 2, which is an averaging operation.
* To put more emphasis on one image relative to the other we perform “alpha blending”, i.e.

 (3.1)

where  for simple averaging.

* Averaging can be used for noise reduction, if the scene is static. In this case, if several images are taken, and are then averaged, the image noise is reduced since in most cases noise has zero mean.

**Subtraction**: Subtracting two b-bit grayscale images can produce values between . One way to deal with negative values is to rescale the range between 0 and , or between 0 and . The main application of the image subtraction is for detection of changes in the scene.

**2. Gray Level Linear Mapping (Contrast Stretching)**

***Let f(x, y)*** be input image and ***g(x, y)*** be processed (output) image.

Mapping

f(x,y) g(x,y)

* Changing the pixel values of the input image to produce the output image is called gray level mapping.
* Linear mapping

 (3.2)

***a*** = slope or gain,

***b*** = y-intercept or bias

We often want to map a certain gray level range, say  onto a new range 

 (3.3)

***g g***

slope = ***a*** 255

***b***

***f*** 0255  ***f***

1. (b)

Fig. 3. (a) Linear mapping, (b) Linear mapping to invert

Special cases of linear mapping:

Negatization (inversion):

***a*** = –1, and bias of  ***b*** = 255 in Matlab:

*>>bldNeg=255 – bld;*

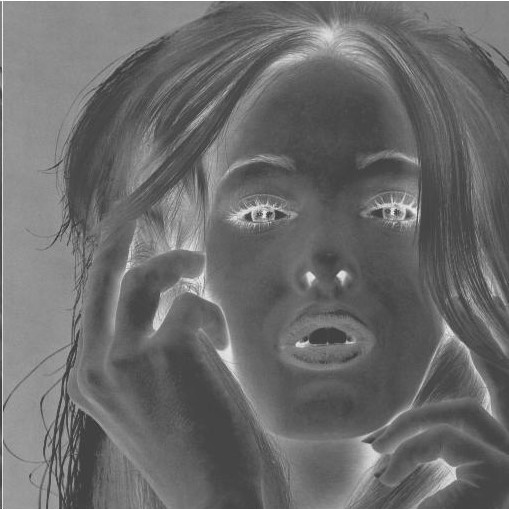
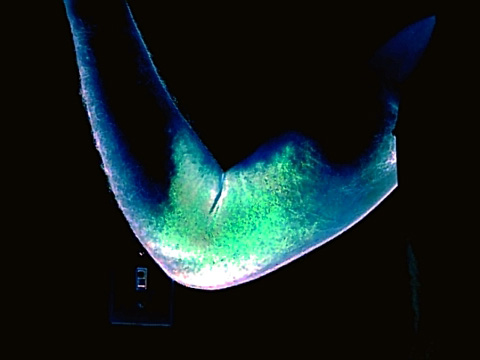
****

Fig.4 Negatization of Blonde.bmp

Note that negatizing (inverting) a color image can produce unexpected results as show below. Inversion in color images- each color band is inverted

As an exercise explain the colors in the image and its negatization below.

1. (b)

Fig. 5. (a) Original Image and (b) Inverted (negatized) image

###### **Thresholding**

***g***

Hi

Lo

0 ***T f***

Fig. 6 Thresholding function

>> Blond.Thresh=(bld<128);

1. (b)

Fig. 7 (a) Original image and (b) thresholded image

###### **Contrast Stretching**

This mapping is piecewise linear. Usually used to improved the contrast, and is usually called contrast stretching.

g

b4

b3

b2

b1

a1 a2 a3 a4 f

Fig. 8 Contrast stretching function

The syntax in Matlab is a=[a1 a2 a3 …an], b=[b1 b2 b3 …bn], and we use *interp1* to join line segments, i.e. *lin=interp1(a,b,0:255, ‘linear’)* .

It can then be applied to image1 to get *image2=unit8(lin(image1+1))* .

For example

*>> a =[0 100 150 255]*

*>> b=[40 100 220 225]*

*>> lin=interp1 (a,b, 0:255, ‘linear’);*

*>> bld2=unit8(lin(bld+1))*

**Nonlinear mapping**

A nonlinear mapping has the characteristic that the slope is variable, i.e.

 < 

g

255





  f

Fig. 9 A nonlinear mapping

The function is imadjust(im, [a ,b], [c,d], gam) is used to achieve a particular nonlinear function. When the so called gamma parameter , we get the mapping below.

g

1

d

c

*f*

0 **a b 1**

Fig. 10 Adjusting the shape of gamma mapping with

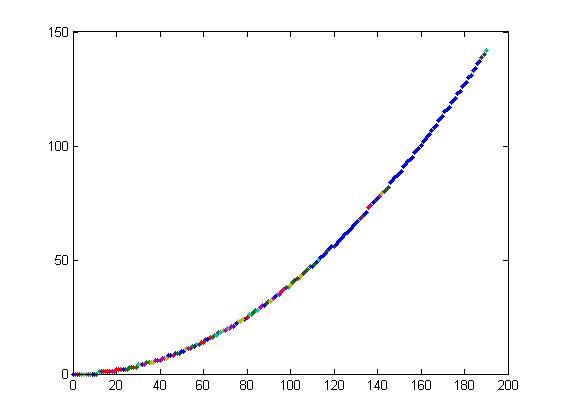
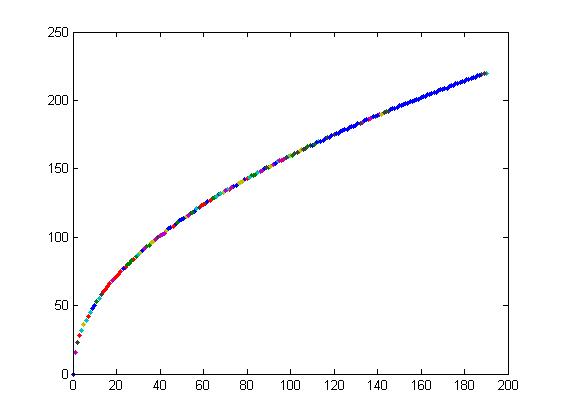
To make a curve instead of middle line in the above figure, we use the function

Values of make the image darker, and values make the image brighter.

*curve1=imadjust(bld,[a,b], [c,d], gamma);*

*>> curve1=imadjust(bld,[0,1], [0,1], 0.5);*

*>> plot(bld,curve1,'.')*



1. (b)

Fig 11. Gamma mapping, (a) , (b)

**Implementations of mapping**

Approach One

The algorithm below shows one of two possible algorithms to implement the square root mapping, assuming a square image where w = h = n.



for (y = 0; y < n; ++y)

for (x = 0; x < n; ++x)

;

The above algorithm requires  square root calculations that can be costly.

**Approach Two**

Use a look-up table (LUT) as follows



//create an array (table) with space for 256 gray level values

for (i = 0; i < 256; ++i)

table[i] = ;

for (y = 0; y < n; ++y)

for (x = 0; x < n; ++x)

g(x,y) = table[f(x,y)];

Note that in the look-up table (LUT) implementation, we require only 256 square root evaluations, and assignments. This takes less computing time than  square root calculations required by the first algorithm. In general LUT is preferable if the number of gray levels is (why?)

**3. Image Histogram**

* The histogram of an image is the distribution of gray levels in that image.

Histogram array: H[*l*], *l* = 0, 1, 2, …, L

H[*l*] = number of pixels that have gray level equal to *l*.

H(*l*)

0 1 2 … L *l*

Fig. 12 An example of a histogram

* The histogram of an image provides a useful indication of the relative importance of different gray levels.
* Using a histogram, it is possible to determine if the brightness or contrast adjustment is necessary. For example, a low contrast image has few bright and few dark pixels.

Normalized Histogram:



*where*and ***h[l]*** = probability that a randomly chosen pixel has gray level ***l***.

* Gray level mapping operations affect the histogram. Multiplication of gray levels by a constant gain ***a*** will spread out the histogram if ***a*** > 1, or compresses it if ***a*** < 1.

The algorithm for determining the histogram

//Create an array histogram with L+1 elements

for (i = 0; i <= L, i++)

histogram[i] = 0;

for( y = 0; y < h; y++)

for(x = 0; x < w; x++)

histogram[f(x,y)] = histogram[f(x,y)] + 1;

Note: Two completely different images can have exactly the same histogram.

In Matlab imhist(ImageName) is the function to obtain the histogram of the woman.

>>wm=imread(‘woman’);

>>imhist(wm)

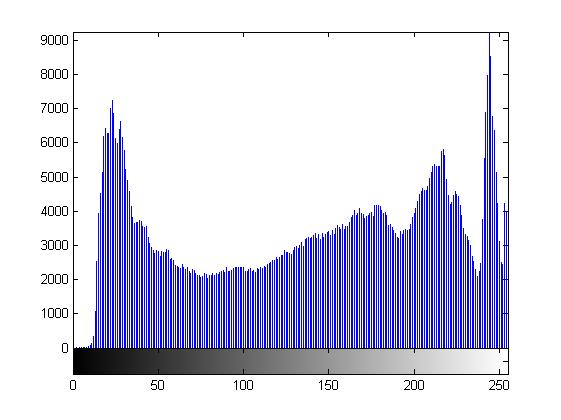


Fig. 13 Image and its histogram

The following show the histogram of several images.

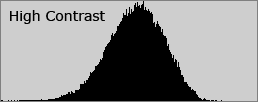
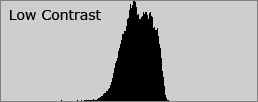
 

Fig. 14 Histograms of low and high contrast images

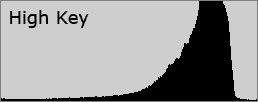
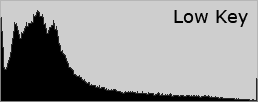
 

Fig. 15 High and low key histograms

**Histogram Equalization**

Perfectly equalized histogram: if

, i =0,1, 2, …, L

or for normalized case

, i =0,1, 2, …, L

* In this image there are equal number of pixels of different gray levels.
* A equalized image has good contrast .
* A low contrast image has most pixels in a certain gray level range, and to improve the contrast one must spread out the gray levels such that the resulting histogram is approximately equalized.

Cumulative histogram plays an important role in achieving equalization,

 ; i = 0, 1, 2, …, L

* A perfectly equalized histogram image has a flat and has a linearly increasing cumulative histogram since

 i = 0, 1, 2, …, L

***h*** Hc

1

1/L

0 L *l*  0 L *l*

Fig. 16 Normalized histogram, and cumulative histogram

* It can be shown that the mapping needed to achieve the equalization is

This means the gray level ***i (i= 0, 1,2, …L)*** in the input image ***f(x, y)*** must map into the gray level in the output image ***g(x,y).***

* Note that the above achieves perfect equalization if the gray levels values are continuous, i.e. if there are infinite number of gray levels, .
* In most cases, however, L is finite and small (e.g. L = 255), and therefore the histogram of the equalized image is not perfect.

Algorithm implements histogram equalization:

Compute normalization factor a=(L/w\*h)

Calculate histogram using the above algorithm

Hc[0] = a\*H[0] //

for (***l*** = 1; ***l*** <= L; ***l*** ++)

Hc[***l***] = Hc[***l*** -1] + a \* H[i];

for (y = 0; y < h; y++)

for (x= 1; x < w; x++)

g(x,y) = Hc[f(x,y)];

In Matlab:

*>> histeq(bld);*

(a) (b)

Fig. 17 (a) Original image and (b) Equalized image.

* Note that the histogram of the original image has a high concentration of bright and medium values with few darker pixels. The equalized image has a more uniform distribution of the gray levels.
* Histogram equalization is widely used in image processing, and it can improve the quality of an image.
* However, this operation is not always beneficial since the improvement in contrast is optimal statistically rather than perceptually.
* In images with narrow histograms and relatively few gray levels, excessive increase in contrast due to histogram equalization can have the adverse effect of reducing the perceived image quality, especially if noise is present in the image.
* However, this operation is not always beneficial since the improvement in contrast is optimal statistically rather than perceptually (why?). In images with narrow histograms and relatively few gray levels, excessive increase in contrast due to histogram equalization can have the adverse effect of reducing the perceived image quality, especially if noise is present in the image.

**Color Images and Contrast Enhancement**

* To find the histogram of a color image, we must provide a Histogram class that gives us three separate components, one for each of R, G, B. The histogram array is now histogram[r][g][b], and can be implemented by the following algorithm.

Create a 3D array of histogram of dimension (L+1)(L+1) (L+1)

for (r = 0; r <= L, r++)

for (g = 0; g <= L, g++)

for (b = 0; b <= L, b++)

histogram[r][g][b] = 0;

for (y = 0; y< h, y++)

for (x = 0; x< h, x++)

find red component of f(x,y)

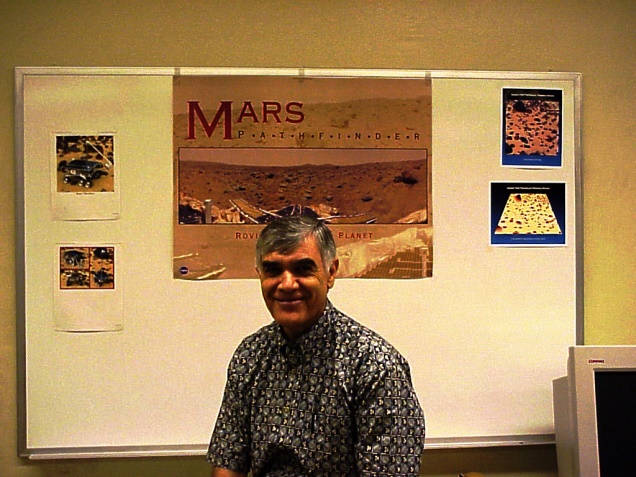
find green component of f(x,y)

find blue component of f(x,y)

increment histogram[r][g][b] by 1;

* A histogram for color images can be plotted either as three 1D-curves (one for each color) on the same graph, as three 2-Ddimensional surfaces, or as one 3-D volume. The 3-D volume representation is used in the above algorithm.
* If L = 255, then an array of dimension 256256256 is needed giving over 16 million values. If a 32 bit integer is used for pixel counts, then the total storage requirement for a single histogram will be 64 MB ! This type of histogram will be very sparsely populated, and we should use a data structure for the histogram (e.g. a hash table) that will use the sparseness property.
* If we want to enhance the image, say by histogram equalization on each of the three components, separately, then the intensity distribution of each component is changed in a different way with the result that both contrast and color are changed.
* The above problem arises because each component of the RGB model contains both color and intensity information. If we wish to manipulate color and intensity separately, we must use the HSI model, where intensity (I), hue (H) and saturation (S) can be adjusted independently.





Top: original image; Bottom: equalized image