Chapter 2
Point Processing and Histogram

Image Processing and Computer Vision

LE Thanh Sach
Faculty of Computer Science and Engineering
Ho Chi Minh University of Technology, VNU-HCM

Overview

1. **Point processing**
   - Linear transformation
   - Logarithmic transformation
   - Exponential transformation
   - Power-law transformation
2. **Histogram**
   - Definition
   - Computation
   - Equalization
   - Specification

Point processing

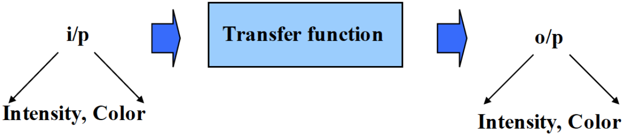
Concepts

- Image: $f(x, y)$
  1. x: discrete variable, in $[0, 1, .., N]$
  2. y: discrete variable, in $[0, 1, .., M]$
  3. $f(x, y)$: discrete value, in $[0, 1, .., 255]$
- point $\equiv$ pixel
- $f(x, y)$: has $M \times N$ points (or pixels)

Point processing

Definition and Notation

- **Point processing**: Process each point by a function that depends ONLY the pixel's value and that does not depend on the point's neighbors.
- Pixel processing function:
    - referred to as **transfer function**
    - denoted as $T[.]$



- $f(x, y)$: input image
- $g(x, y) = T[f(x, y)]$: output image

Point processing

Linear transformation: Math

- $g(x, y) = a \times f(x, y) + b$
- Where, a and b: pre-defined parameters

Linear transformation: Applications

- General: Change the image's intensity, cause the input bighter or darker
  - Users have to choose appropriate a and b (manually)
- Specific:
  - Create negative images
  - Convert to back-white image

Point processing

Linear transformation: with Matlab

```
%read input image
im = imread('cameraman.tif');
%set parameters
a1 = 1.5; b1 = 20;
a2 = 1.5; b2 = 50;
a3 = 0.5; b3 = 0;
%transform the input
im1 = a1 * im  + b1; %clipped to [0,255] auto
im2 = a2 * im  + b2; %clipped to [0,255] auto
im3 = a3 * im  + b3; %clipped to [0,255] auto
```

Point processing

Linear transformation: with Matlab(continued)

```
%show output image
figure;
subplot(2,2,1); imshow(im); title('input image')
subplot(2,2,2); imshow(im1);
t1 = sprintf('output image [a =%5.2f, b = %5.2f]',
             a1, b1);
title(t1)
subplot(2,2,3); imshow(im2);
t2 = sprintf('output image [a =%5.2f, b = %5.2f]',
             a2, b2);
title(t2)
subplot(2,2,4); imshow(im3);
t3 = sprintf('output image [a =%5.2f, b = %5.2f]',
             a3, b3);
title(t3)
suptitle('Linear Transformation: output =  a*input + b');
```

Point processing

Linear transformation: Illustration

Input image

Output Image $[a = 1.5, b = 20]$

Output Image $[a = 1.5, b = 50]$

Output Image $[a = 0.5, b = 0]$

Table: Linear Transformation: **output** = a * **input** + b

Point processing

Linear transformation: Illustration

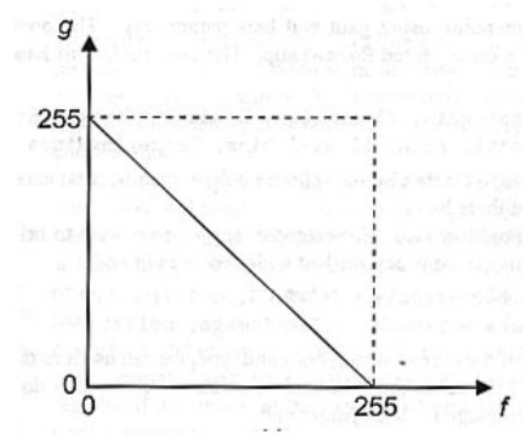


Table: Creating negative image: **output** = a * **input** + b; where, $\mathbf{a = -1, b = 255}$

Point processing

Linear transformation: Illustration

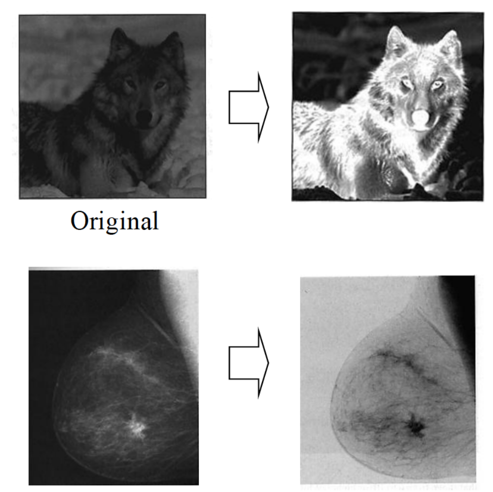


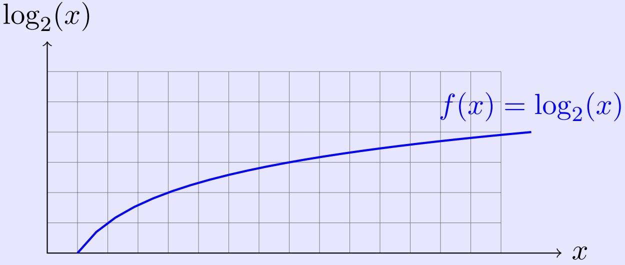
Table: Creating negative image: **output** = a * **input** + b; where, $\mathbf{a = -1, b = 255}$

Point processing

Logarithmic transformation: Math

- $g(x, y) = a \times \log[1 + f(x, y)] + b$
- Where, a: a pre-defined parameter

Logarithm function



- a large value (x) $\rightarrow$ mapped to a smaller value $(\log_2 x)$
- a large distance between two intensities in the input image $\rightarrow$ mapped to a smaller distance in the output

Point processing

Logarithmic transformation: Applications

- Input image:
  - have some regions with too dark intensities (**near 0**)
  - have some regions with too bright intensities (**near 255**)
- Requirement:
  - Make dark regions brighter while keeping the intensity in bright regions lower than 255.
  - Or, make bright regions darker while keeping the intensity in dark regions lower than 0.
- Solution: use logarithmic transformation
- Often case:
  - Should use logarithmic transformation to enhance the result of FFT before displaying

Point processing

Logarithmic transformation: Illustration

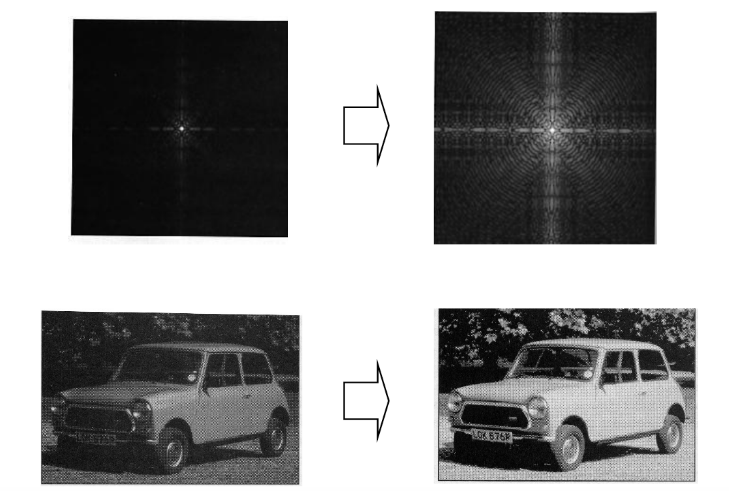


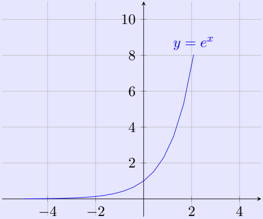
Table: Samples

Point processing

(Inverse-log) Exponential transformation: Math

- $g(x, y) = a \times e^{f(x,y)} + b$
- Where, a and b: pre-defined parameters

Exponential function



- a small value $(x) \rightarrow$ mapped to a larger value e^x
- a small distance between two intensities in the input $\rightarrow$ mapped to a larger distance in the output image

Point processing

Exponential transformation: Applications

- map small differences between two intensities in the input to a larger differences in the output image (**contrast input images**)

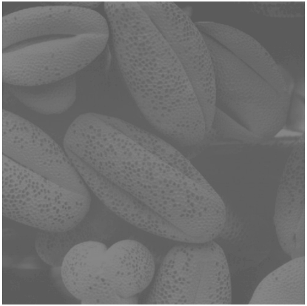
Exponential transformation: with Matlab

```
%read input image
path = 'coffebean.png'
im = imread(path); %range [0->255]
%convert double image: [0->1]
im = im2double(im);
%set parameters
a = 1; b = -1;
%transform input image
im1 = a * exp(im) + b;
%normalize the output, to [0->1]
im1 = im1./max(im1(:));
```

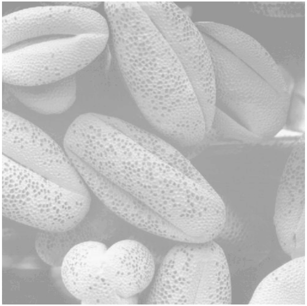
Point processing

Exponential transformation: with Matlab (continued)

```
%show image
figure; imshow(im); title('input image');
figure; imshow(im1); title('output image');
```



Input image



output image

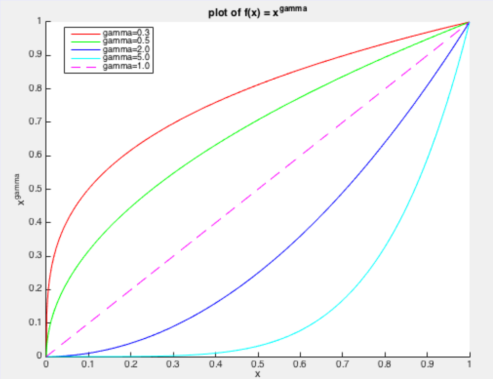
Table: $g(x,y) = a \times e^{f(x,y)} + b$; where, $a = 1, b = -1$

Point processing

Power-law transformation: Math

- $g(x, y) = a \times f(x, y)^{\gamma} + b$
- Where, a and b: pre-defined parameters

Power function



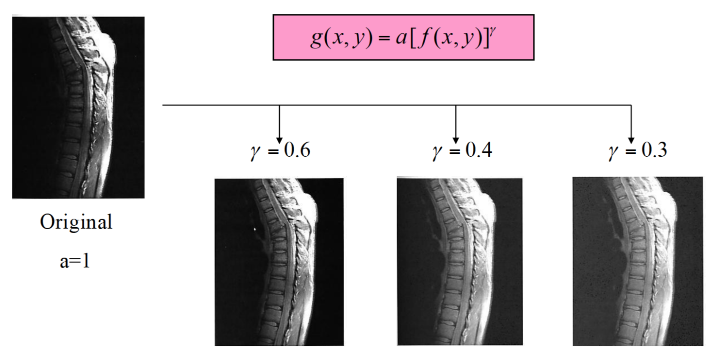
Point processing

Power-law function: characteristics

- Depend on γ, power-law transformation can be either
  1. **linear transformation**: for $\gamma = 1$
  2. **log transformation**: for $\gamma < 1$
  3. **inverse-log transformation**: for $\gamma > 1$

Point processing

Power-law transformation: Illustration



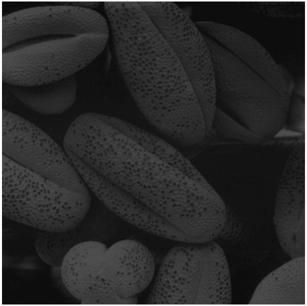
Histogram

Definition

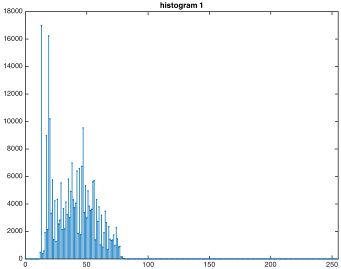
Histogram computed for image I is a **statistical quantity**, contains the following information

- number of pixels in I for each intensity (for each value in [0,255]). This kind of histogram is referred to as **unnormalized histogram**.
  - Sum of all values in this kind of histogram = total number of pixels in I
- probability distribution of intensities in image I. This kind of histogram is referred to as **normalized histogram**.
  - Sum of all values in this kind of histogram = 1

Histogram



(a) An image

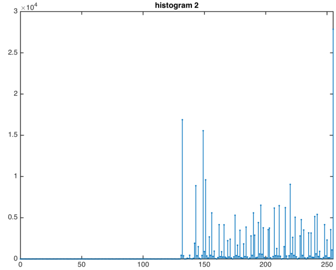


(b) Its Histogram

- From histogram (b): almost of pixels in the image have the intensity near 0. Therefore, the image is too dark, confirmed by image (a).

Histogram

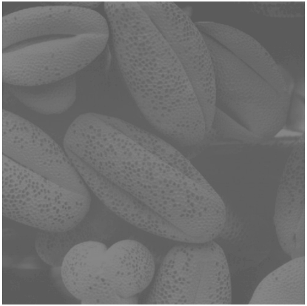
(a) An image



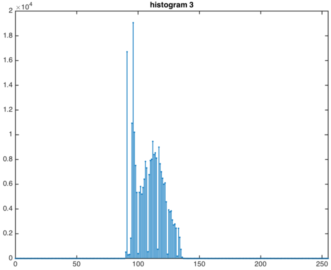
(b) Its Histogram

- From histogram (b): almost of pixels in the image have the intensity near 255. Therefore, the image is too bright, confirmed by image (a).

Histogram



(a) An image

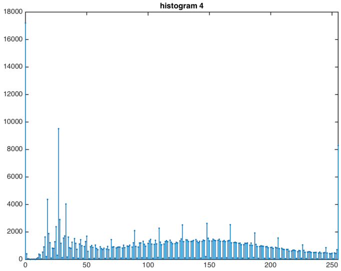


(b) Its Histogram

- From histogram (b): almost of pixels in the image have the intensity compacted in short range $[100, 130]$. Therefore, the image has very low contrast, confirmed by image (a).

Histogram

(a) An image



(b) Its Histogram

- From histogram (b): the pixels in the image are distributed in a full range $[0, 255]$. Therefore, the image is balanced (not to dark, not to bright) and has a high contrast (**preferred**).

Histogram

Histogram: with C/C++/Java

Excercise:

- Write a function for computing the histogram (unnormalized and normalized) of the input image passed to the function.

Histogram

Histogram: with Matlab

- use functions: **imhist**

```
%read input images
path1 = 'Fig3.15(a)1.jpg';
path2 = 'Fig3.15(a)2.jpg';
path3 = 'Fig3.15(a)3.jpg';
path4 = 'Fig3.15(a)4.jpg';
im1 = imread(path1);
im2 = imread(path2);
im3 = imread(path3);
im4 = imread(path4);

%Compute histogram
[count1, x1] = imhist(im1);
[count2, x2] = imhist(im2);
[count3, x3] = imhist(im3);
[count4, x4] = imhist(im4);
```

Histogram

Histogram: with Matlab (continued)

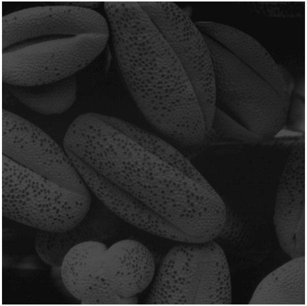
```
%show input images
figure; imshow(im1); title('input␣image␣1');
figure; imshow(im2); title('input␣image␣2');
figure; imshow(im3); title('input␣image␣3');
figure; imshow(im4); title('input␣image␣4');

%draw histograms
figure; stem(x1, count1, '.');
xlim([0,255]); title('histogram␣1');
figure; stem(x2, count2, '.');
xlim([0,255]); title('histogram␣2');
figure; stem(x3, count3, '.');
xlim([0,255]); title('histogram␣3');
figure; stem(x4, count4, '.');
xlim([0,255]); title('histogram␣4');
```

Histogram: Equalization

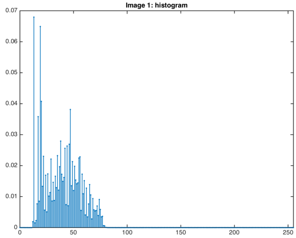
Histogram Equalization: What?

Histogram Equalization is an operation to create an image (from a given image) whose pixels's value distributed uniformly in $[0, 255]$

- Purpose of histogram equalization: to create image not too dark, not too bright, and high contrast



(a) An image (not preferred)



(b) Its normalized histogram (not preferred)

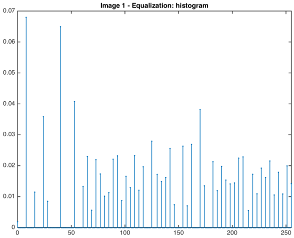
Histogram: Equalization

Histogram Equalization: What?

Histogram Equalization is an operation to create an image (from a given image) whose pixels's value distributed uniformly in $[0, 255]$

- Purpose of histogram equalization: to create image not too dark, not too bright, and high contrast

(a) Image, preferred (after equalization)



(b) Histogram, preferred (after equalization)

Histogram: Equalization

Histogram Equalization: How?

Input:

1. Input image I
   - So, compute the **normalized-histogram** of image I, to obtain: $p_x(x)$:
2. Expected **normalized-histogram** of the output image: $p_z(z)$
   - with histogram equalization: $p_z(z)$ is a uniform distribution over $[0, 255]$
   - with histogram specification: $p_z(z)$ is specified by users (interactively)

Histogram: Equalization

Histogram Equalization: How?

We can determine z (e.g., pixel value of the output image), if we know:

1. x: pixel value of the input image
2. $p_x(x)$: pdf of x
3. $p_z(z)$: pdf of z

Assume that $z = f(x)$ has been determined, the output image $g(u, v)$ can be produced from the input $f(u, v)$ as follows:

- **foreach** point (u, v) in $f(u, v)$:
    - $x = f(u, v)$ (get the value of the input pixel)
    - $z = f(x)$ (map to the output value)
    - $g(u, v) = z$ (assign to the output pixel)

Histogram: Equalization

Determination of $z = f(x)$

Let $c_x(x)$ and $c_z(z)$ be cumulative density functions of x and z.

$$c_x(x) = \sum_{i=0}^{x} p_x(i)$$

$$c_z(z) = \sum_{i=0}^{z} p_z(i)$$

Histogram: Equalization

$z = f(x)$ be a non-decreasing function that map one-to-one between x and z.

Determination of $z = f(x)$

- Assume that we have $z_1 = f(x_1)$
- Then, $c_z(z_1) = c_x(x_1)$. This is because of one-one mapping
  - A value $x < x_1$ will be mapped into $z < z_1$
- Define $w \equiv c_x(x_1)$, i.e., $w \equiv c_z(z_1) \equiv c_x(x_1)$

Therefore,

- $z_1 = c_z^{-1}(w) \equiv c_z^{-1}(c_x(x_1))$

Mapping function $f(x)$

$$z = f(x) = c_z^{-1}(c_x(x))$$

Histogram: Equalization

Implemtation of $z = f(x) = c_z^{-1}(c_x(x))$

1. If we have closed-form of $z = c_z^{-1}(w)$ then can use this function to determine z.
2. If we do not have closed-form of $c_z^{-1}(z)$
   - Create a lookup table at the beginning for mapping $z \rightarrow c_z^{-1}(p)$, for discrete values p. We can do this because we know $c_z(z)$ in advance. This task is equal to rasterize $p = c_z(z)$ and store pairs into lookup table
   - Determine z according to the lookup table.