Digital Image Processing
COSC 6380/4393

Lecture – 7
Feb 5th, 2018
Pranav Mantini

Slides from Dr. Shishir K Shah and Frank (Qingzhong) Liu
Today

• Review
• Binary Image Processing
  – Opening and Closing
  – Skeletonization
  – Compression:
    • Run-length encoding
    • Chain Coding
• Point Operations
  – Linear Point Operations
    • Offset
    • Scaling
  – Non-Linear Point Operations
Review: Creating Binary Images

- Determine peaks
- Choose $T$ between peaks (say average)
Review: Processing Binary Images

- A simple technique for **region classification** and **correction**
- **Motivation:** Gray-level image thresholding **usually** produces an imperfect binary image:
  - Extraneous blobs or holes due to noise
  - Extraneous blobs from thresholded objects of little interest
  - Nonuniform object/background surface reflectances
Review: BLOB COLORING

• It is usually desired to **extract a small number of objects** or even a **single object** by thresholding.

• Blob coloring is a very simple technique for **listing** all of the blobs or objects in a binary image.
BLOB COLORING
BLOB COLORING
BLOB COLORING

- If top and left pixels belong to different regions, make them same.
- Assign same region as left.
- Assign same region as top.
- New region.
Review: Blob Coloring

- Using blob coloring

"Color" of largest blob: 2
Review: Blob Coloring
Review: BINARY MORPHOLOGY

- The most powerful class of binary image operators
- A general framework known as **mathematical morphology**
  \[ \text{morphology} = \text{shape} \]
- **Morphological operations** affect the **shapes** of **objects** and **regions** in binary images
- All processing is done on a **local basis** - region or blob shapes are affected in a local manner
- Morphological operators
  - Expand (dilate) objects
  - Shrink (erode) objects
  - Smooth object boundaries and eliminate small regions or holes
  - Fill gaps and eliminate ’peninsulas’
- All is accomplished using **local logical operations**
Morphological Operations

Input binary image

Output/Filtered binary image
Morphological Operations

\[ 1 \lor 0 \lor 0 = 1 \]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Input binary image

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

Output/Filtered binary image
Morphological Operations

\[ 0 \lor 0 \lor 1 = 1 \]

Input binary image

Output/Filtered binary image
Morphological Operations

\[0 \lor 1 \lor 0 = 1\]

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Input binary image

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Output/Filtered binary image
QUALITATIVE PROPERTIES OF DILATION

- Dilation removes object holes of too-small size:

- Dilation also removes gaps or bays of too-narrow width:
QUALITATIVE PROPERTIES OF EROSION

- Erosion removes objects of too-small size:

- Erosion also removes peninsulas of too-narrow width:
QUALITATIVE PROPERTIES OF MEDIAN

• Median removes both objects and holes of too-small size, as well as both gaps (bays) and peninsulas of too-narrow width.
OPENing

• We can define **new** morphological operations by performing the basic ones in sequence
• Given an image $I$ and window $B$, define
  \[
  \text{OPEN}(I, B) = \text{DILATE} \left[ \text{ERODE}(I, B), B \right]
  \]
• In other words,
  \[
  \text{OPEN} = \text{erosion (by } B\text{) followed by dilation (by } B)\]

EXAMPLES
OPENing and CLOSing

- We can define **new** morphological operations by performing the basic ones in sequence.
- Given an image $I$ and window $B$, define
  
  OPEN($I$, $B$) = DILATE [ERODE($I$, $B$), $B$]
  
  CLOSE($I$, $B$) = ERODE [DILATE($I$, $B$), $B$]

- In other words,
- OPEN = erosion (by $B$) followed by dilation (by $B$)
- CLOSE = dilation (by $B$) followed by erosion (by $B$)
EXAMPLES
OPENing and CLOSing

• OPEN and CLOSE are very similar to MEDIAN:
• OPEN removes too-small objects/fingers (more effectively than MEDIAN), but not holes, gaps, or bays
• CLOSE removes too-small holes/gaps (more effectively than MEDIAN) but not objects or peninsulas
• OPEN and CLOSE generally do not affect object size
• OPEN and CLOSE are used when too-small BLACK and WHITE objects (respectively) are to be removed
• Thus OPEN and CLOSE are more specialized smoothers
Open-Close and Close-Open

- Very effective smoothers can be obtained by sequencing the OPEN and CLOSE operators:
- For an image $I$ and structuring element $B$, define
  \[
  \text{OPEN-CLOS}(I, B) = \text{OPEN} [\text{CLOSE} (I, B), B]
  \]
  \[
  \text{CLOS-OPEN}(I, B) = \text{CLOSE} [\text{OPEN} (I, B), B]
  \]
- These operations are quite similar (not mathematically identical)
Open-Close and Close-Open

- Both remove too-small structures without affecting size much
- Both are similar to the median filter except they smooth more (for a given structuring element B)
- One notable difference between OPEN-CLOS and CLOS-OPEN:
  - OPEN-CLOS tends to link neighboring objects together
  - CLOS-OPEN tends to link neighboring holes together
EXAMPLES
SKELETONIZATION

• A way of obtaining an image’s medial axis or skeleton
EXAMPLE

• Image $I_0$:

• Structuring Element $B$:

• $SKEL(I_0, B)$:
THE STEPS

\[
\begin{align*}
I_n & \quad \text{NOT}[\text{OPEN}(I_n, B)] & \quad S_n \\
\text{SKEL}(I_0, B)
\end{align*}
\]
SKELETONIZATION

• A way of obtaining an image’s **medial axis** or skeleton

• Given an image \( I_0 \) and window \( B \), the skeleton is \( \text{SKEL}(I_0, B) \)

• Obtaining the skeleton requires a fairly complex iteration:

• Define \( I_n = \text{ERODE} \ [ \cdot \cdot \cdot \text{ERODE} \ [\text{ERODE}(I_0, B), B], \cdot \cdot B ] \) (n consecutive EROSIONS of \( I_0 \) by \( B \))

• \( N = \max \{ n : I_n \cdot \emptyset \} \emptyset = \text{empty set} \)

• (the largest number of erosions before \( I_n \) "disappears")

• \( S_n = I_n \land \text{NOT}[\text{OPEN}(I_n, B)] \)

• Then \( \text{SKEL}(I_0, B) = S_1 \lor S_2 \lor \cdot \cdot \cdot \lor S_N \)
EXAMPLE

binary image

skeleton (of background)
APPLICATION EXAMPLE

• **Simple Task: Measuring Cell Area**
  
  Simple processing steps:
  
  – (i) Find general cell region by **simple thresholding**
  – (ii) Apply region correction techniques:
    – Blob coloring
    – Minor region removal
  – CLOS-OPEN
    – (iii) Display cell boundary for operator verification
    – (iv) Compute image cell area by counting pixels
    – (v) Compute actual cell area using perspective projection
COMMENTS

• Previous manual measurement techniques required > 1 hour per cell image to analyze
• Algorithm runs in less than a second. Has been applied to > 50,000 cell images over the last several years
• Published in CRC Press’ s *Image Analysis in Biology* as the standard for "Automated Area Measurement.”
Compression: RUN LENGTH CODING

• The number of bits required to store an N x N binary image is \( N^2 \)
• This can be significantly reduced in many cases.
• Run-length coding works well if the WHITE and BLACK regions are generally not small.
EXAMPLE

what's stored: row m

[Diagram showing row m with some elements highlighted]
EXAMPLE

what's stored: '1' 7 5 8 3 1
row m
HOW DOES IT WORK?

• Binary images are stored (or transmitted) on a line-by-line (row-by-row) basis

• For each image row numbered \( m \):
  – Store the first pixel value ('0' or '1') in row \( m \) as a reference
  – Set run counter \( c = 1 \)
  – For each pixel in the row:
    • Examine the next pixel to the right
    • If same as current pixel, set \( c = c + 1 \)
    • If different from current pixel, store \( c \) and set \( c = 1 \)
    • Continue until end of row is reached

• Each run-length is stored using \( b \) bits.
COMMENTS

• Can yield excellent **lossless compressions** on some images.
• This will happen if the image contains lots of runs of 1's and 0's.
• If the image contains only very short runs, then run-length coding can actually **increase** the required storage.
WORST CASE

- In this worst-case example the storage increases b-fold!
- Rule of thumb: the average run-length L should satisfy: 
  \[ L > b. \]
CONTOUR REPRESENTATION & CHAIN CODING

• We can distinguish between two general types of binary image: region images and contour images.

![Region Image](image1.png) ![Contour Image](image2.png)

• We will require contour images to be special:
  • Each BLACK pixel in a contour image must have **at most** two BLACK 8-neighbors
  • a BLACK pixel and its 8-neighbors –

• **Contour images** are composed only of **single-pixel width** contours (straight or curved) and single points.
CHAIN CODE

• The chain code is a highly efficient method for coding contours

• Observe that if the initial \((i, j)\) coordinate of an 8-connected contour is known, then the rest of the contour can be coded by giving the directions along which the contour propagates
CHAIN CODE

• We use the following 8-neighbor direction codes:

• Since the numbers 0, 1, 2, 3, 4, 5, 6, 7 can be coded by their 3-bit binary equivalents:

  000, 001, 010, 011, 100, 101, 110, 111

the location of each point on the contour after the initial point can be coded by 3 bits.
EXAMPLE

- Its chain code: (after recording the initial coordinate \((i_0, j_0)\)

  \[
  1, 0, 1, 1, 1, 3, 3, 3, 4, 4, 5, 4
  \]

  \[
  = \quad 001, 000, 001, 001, 001, 001, 011, 011, 011, 100, 100, 101, 100
  \]
COMMENTS

• The compression obtained can be quite significant: coding the contour by M-bit coordinates (M = 9 for 512 x 512 images) requires 6 times as much storage.

• The technique is effective in many computer vision and pattern recognition applications, e.g. character recognition.

• For closed contours, the initial coordinate can be chosen arbitrarily. If the contour is open, then it is usually an end point (one 8-neighbor).
SIMPLE HISTOGRAM OPERATIONS

• Recall: the gray-level histogram $H_I$ of an image $I$ is a graph of the frequency of occurrence of each gray level in $I$

• $H_I$ is a one-dimensional function with domain $0, \ldots, K-1$:

• $H_I(k) = n$ if gray-level $k$ occurs (exactly) $n$ times in $I$, for each $k = 0, \ldots, K-1$
SIMPLE HISTOGRAM OPERATIONS

• The histogram $H_i$ contains **no spatial information** about $I$ - only information about the relative frequency of intensities

• Nevertheless
  
  — Useful information can be obtained from the histogram
  
  — Image quality is effected (enhanced, modified) by altering the histogram
Average Optical Density

- A measure of the average intensity of the image $I$:

$$AOD(I) = \left( \frac{1}{N^2} \right) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} I(i, j) = \left( \frac{1}{N^2} \right) \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} I(i, j)$$

- Can compute it from the histogram as well:
Average Optical Density

• A measure of the average intensity of the image $I$:

$$AOD(I) = \left(\frac{1}{N^2}\right) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} I(i, j) = \left(\frac{1}{N^2}\right) \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} I(i, j)$$

• Can compute it from the histogram as well:

$$\left(\frac{1}{N^2}\right) \sum_{k=0}^{K-1} kH_i(k)$$

• $k^{th}$ term = (brightness level $k$) x (# occurrences of $k$)
Average Optical Density

- Examining the histogram can reveal possible errors in the imaging process:
  - Methods for correcting such errors utilize the histogram.
  - The histogram will arise throughout this lecture.
POINT OPERATIONS

- A **point operation** on an image $I$ is a **function** $f$ that maps $I$ to another image $J$ by operating on **individual pixels** in $I$:
  \[ J(i, j) = f[I(i, j)], \quad 0 \leq i, j \leq N-1 \]
- The same function $f$ is applied at every image coordinate.
- This is different from **local operations** such as OPEN, CLOSE, etc., since those are functions of both $I(i, j)$ and its neighbors.
LINEAR POINT OPERATIONS

• Point operations do not modify spatial relationships between pixels
• They do modify the image histogram, and therefore the overall appearance of the image
• Linear point operations are the simplest class of point operations
LINEAR POINT OPERATIONS

• Point operations **do not** modify **spatial relationships** between pixels

• They **do** modify the **image histogram**, and therefore the overall appearance of the image

• **Linear point operations** are the simplest class of point operations

\[ F(X) = P \cdot X + L \]
Image Offset

• Suppose $L$ falls in the range $-(K-1) \leq L \leq K-1$ ($\pm$ the nominal gray scale)

• An **additive image offset** is defined by the function

  $$J(i, j) = I(i, j) + L, \text{ for } 0 \leq i, j \leq N-1$$

• Thus, the same constant $L$ is added to every image pixel value

• If $L > 0$, $J$ will be a **brightened** version of the image $I$

• Otherwise its appearance will be essentially the same
Image Offset

• If $L < 0$, $J$ will be a **dimmed** version of the image $I$
• Adding offset $L$ **shifts** the histogram by amount $L$ to left or right:

![Histograms of additive image offsets]

• The input and output histograms are related by:

$$H_J(k) = H_I(k-L)$$
Suppose it is desired to compare multiple images $I_1, I_2, ..., I_n$ of the same scene.

However, the images were taken with a variety of different exposures or lighting conditions.

One solution: equalize the AOD's of the images.

If the gray-scale range of the images is $0, ..., K-1$, a reasonable AOD is $K/2$.

Let $L_m = \text{AOD}(I_m)$, for $m = 1, ..., n$.

Then define "AOD-equalized" images $J_1, J_2, ..., J_n$ according to $J_m(i, j) = I_m(i, j) - L_m + K/2$, for $0 \leq i, j \leq N-1$. 

Image Offset Example
Image Offset Example

• The effect:

\[ H_1(k) \]

\[ H_2(k) \]

\[ \ldots \]

\[ \text{etc.} \]
Image Scaling

- Suppose \( P > 0 \) (not necessarily an integer)
- **Image scaling** is defined by the function
  \[
  J(i, j) = P \cdot I(i, j) \text{, for } 0 \leq i, j \leq N-1
  \]
- Thus, \( P \) multiplies every image pixel value
- In practice:
  \[
  J(i, j) = \text{INT} [ P \cdot I(i, j) + 0.5 ] \text{, for } 0 \leq i, j \leq N-1
  \]
  where \( \text{INT} [ R ] = \text{nearest integer that is } \leq R \)
- If \( P > 1 \), \( J \) will have a **broader grey level range** than image \( I \)
Image Scaling

- If $P < 1$, $J$ will have a **narrower grey-level range** than $I$.
- Multiplying by a constant $P$ **stretches or compresses** the "width" of the image histogram by a factor $P$: 
Comments

- An image with a compressed gray level range generally has a **reduced visual contrast**
- Such an image may have a **washed-out** appearance
- An image with a wide range of gray levels generally has an **increased visual contrast**
- Such an image may have a more striking, viewable appearance
Linear Point Operations: Offset & Scaling

• Suppose L and P are real numbers (not necessarily integers)
• A **linear point operation** on I is defined by the function
  \[ J(i, j) = P \cdot I(i, j) + L, \text{ for } 0 \leq i, j \leq N-1 \]
• In practice:
  \[ J(i, j) = \text{INT}[ P \cdot I(i, j) + L + 0.5 ] , \text{ for } 0 \leq i, j \leq N-1 \]
• The image J is a version of I that has been scaled and given an additive offset
Linear Point Operations: Offset & Scaling

- If $P < 0$, the histogram is reversed, creating a negative image.
- By far the most common use is $P = -1$ and $L = K - 1$:
  $$J(i, j) = (K - 1) - I(i, j), \text{ for } 0 \leq i, j \leq N - 1$$

- Hereafter we assume $P > 0$
Caveat

• Generally, the available gray-scale of the transformed image $J$ is the same as that of the original image $I$: \{0 ,..., K-1\}

• When making the transformation

\[ J(i, j) = P \cdot I(i, j) + L, \text{ for } 0 \leq i, j \leq N-1 \]

• care must be taken that the maximum and minimum values $J_{\text{max}}$ and $J_{\text{min}}$ satisfy

\[ J_{\text{max}} \leq K-1 \text{ and } J_{\text{min}} \geq 0 \]

• **At best**, values outside these ranges will be "clipped"

• At worst, an overflow or sign-error condition may occur

• In that instance, the gray-scale value assigned to an error pixel will be highly unpredictable
Full-Scale Contrast Stretch

- The **most common** linear point operation. Suppose $I$ has a compressed histogram:

![histogram diagram]

- Let $A$ and $B$ be the min and max gray levels in $I$
- Define
  $$J(i, j) = P \cdot I(i, j) + L$$
- such that $P \cdot A + L = 0$ and $P \cdot B + L = (K-1)$
Full-Scale Contrast Stretch

• The result of solving these 2 equations in 2 unknowns \((P, L)\) is an image \(J\) with a full-range histogram:

• The solution to the above equations is

\[
P = \left\lfloor \frac{K-1}{B-A} \right\rfloor \quad \text{and} \quad L = -A \left\lfloor \frac{K-1}{B-A} \right\rfloor
\]

or

\[
J(i, j) = \left\lfloor \frac{K-1}{B-A} \right\rfloor \left[ I(i, j) - A \right]
\]
NONLINEAR POINT OPERATIONS

• A nonlinear point operation on I is a pointwise function f mapping I to J:
  \[ J(i, j) = f[I(i, j)] \text{ for } 0 \leq i, j \leq N-1 \]

• where f is a nonlinear function.

• This is of course a very broad class of functions

• However, only a few are used much:
  – \( J(i, j) = |I(i, j)| \) (absolute value or magnitude)
  – \( J(i, j) = [I(i, j)]^2 \) (square-law)
  – \( J(i, j) = I(i, j)^{1/2} \) (square root)
  – \( J(i, j) = \log[1+I(i, j)] \) (logarithm)
  – \( J(i, j) = \exp[I(i, j)] = e^{I(i,j)} \) (exponential)
Logarithmic Range Compression

- **Motivation**: An image may contain information-rich, smoothly-changing low intensities - and small very bright regions
- Useful for detecting faint objects
- The bright pixels will dominate our visual perception of the image
- A typical histogram:
Logarithmic Range Compression

- **Logarithmic transformation** \( J(i, j) = \log[1+I(i, j)] \) non-linearly compresses and equalizes the gray-scales.
- Bright intensities are compressed much more heavily - thus **faint details** emerge.
- A full-scale contrast stretch then utilizes the full gray-scale range: