HW 2

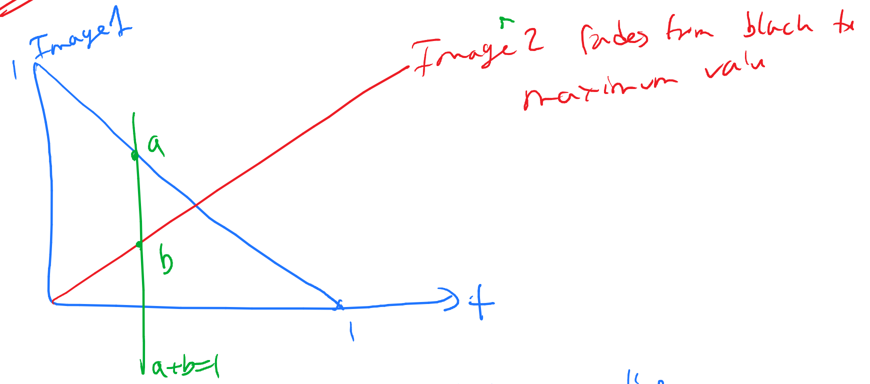
Arithmetic/Logic Operations

Prof. George Wolberg
Dept. of Computer Science
City College of New York

Objectives

- In this lecture we describe arithmetic and logic operations commonly used in image processing.
- Arithmetic ops:
  - Addition, subtraction, multiplication, division
  - Hybrid: cross-dissolves
- Logic ops:
  - AND, OR, XOR, BIC, …

cross dissolve → fade out of one
fade in of another.



at any moment in time. the scale factor applied to both =1

image 1 fades over time

Arithmetic/Logic Operations

- Arithmetic/Logic operations are performed on a pixel-by-pixel basis between two images.
- Logic NOT operation performs only on a single image.
  - It is equivalent to a negative transformation.
- Logic operations treat pixels as binary numbers:
  - 158 & 235 = 10011110 & 11101011 = 10001010
- Use of LUTs requires 16-bit rather than 8-bit indices:
  - Concatenate two 8-bit input pixels to form a 16-bit index into a 64K-entry LUT. Not commonly done.

cross dissolve → fade out of one
fade in of another.

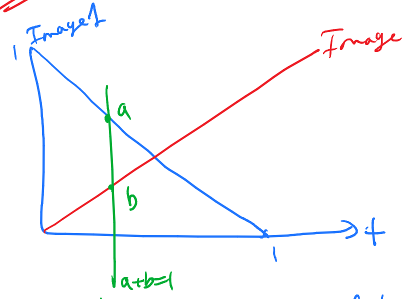
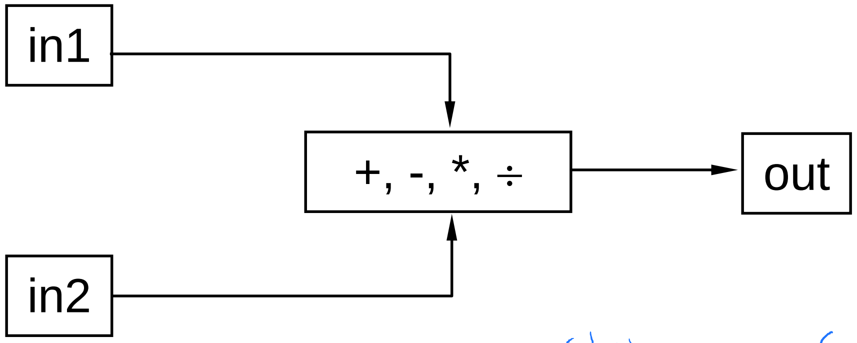


Image 2 fades from black to maximum value

at any moment in time.
The scale factor applied to both =1

image 1 fades over time

Addition / Subtraction



Addition:

take min to not get overflow
limit to 255

```
for(i=0; i<total; i++)
        out[i] = MIN(((int)in1[i]+in2[i]), 255);
```

casting bec in1 and in2 are unsigned char → so promote one of them to a higher datatype
so will do [int] + [char] → compiler will promote [char] → to higher datatype [int]

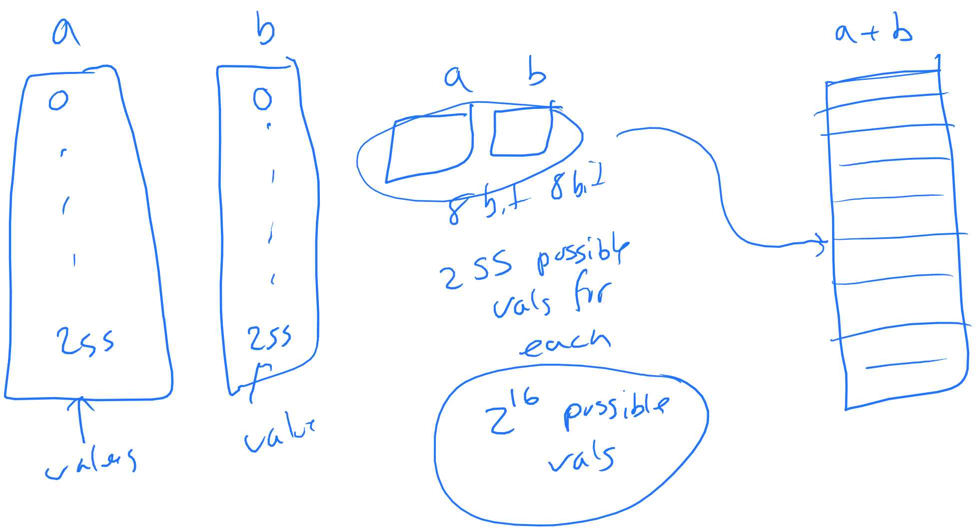
Avoid overflow: clip result

Subtraction:

```
for(i=0; i<total; i++)
        out[i] = MAX(((int)in1[i]-in2[i]), 0);
```

Avoid underflow: clip result

no negative intensity light
so limit to 0



always know these

$$2^{16} = 2^{10} + 2^{6} = 1024 + 64 = 1088$$

$$\boxed{\begin{array}{l} 2^{10} = 1024 \\ 2^{8} = 256 \end{array}}$$ or 1K

char
+ 0
− 1

1|111 1111 = −1 ← smallest neg

1000 0000 = −128 = -2^7 ← largest neg

Overflow / Underflow

why not use char
· We want the signed bit.
· don't want to waste it.

0|111 1111 = 127 = 2^7-1
largest (+)

$-128 \le a \le 127$
range of signed char

$0 \le 255$
1111 1111 = 2^8-1 = 255
range of unsigned char

- Default datatype for pixel is `unsigned char`.
- It is 1 byte that accounts for nonnegative range [0,255].
- Addition of two such quantities may exceed 255 (overflow).
- This will cause wrap-around effect:
  - 254: 11111110
  - 255: 11111111
  - 256: 100000000
  - 257: 100000001

overflow causes pic to go dark
so need clipping at 255 and 0
[0,255].

- Notice that low-order byte reverts to 0, 1, … when we exceed 255.
- Clipping is performed to prevent wrap-around.
- Same comments apply to underflow (result < 0).

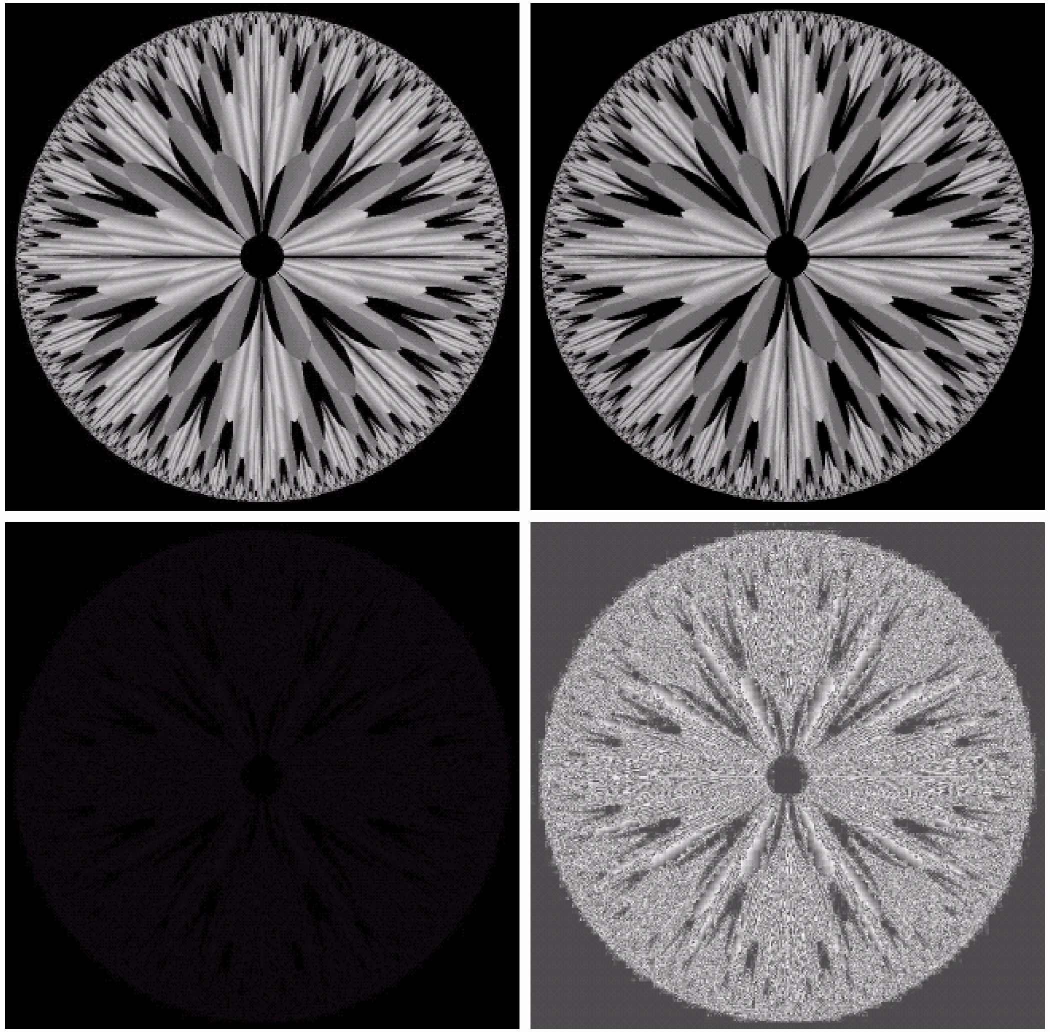
Scaling Methods

Implementation Issues

- The values of a subtraction operation may lie between -255 and 255. Addition: [0,510].
- Clipping prevents over/underflow. ← 1 way to do it is to clip for [0,255]
- Alternative: scale results in one of two ways: ← other ways

1. Add 255 to every pixel and then divide by 2.
   - Values may not cover full [0,255] range
   - Requires **`short`** (16 bits) intermediate image
   - Fast and simple to implement
2. Add negative of min difference (shift min to 0). Then, multiply all pixels by 255/(max difference) to scale range to [0,255] interval.
   - Full utilization of [0,255] range
   - Requires **`short`** intermediate image
   - More complex and difficult to implement

Example of Subtraction Operation



a b
c d

FIGURE 3.28 (a) Original fractal image. (b) Result of setting the four lower-order bit planes to zero. (c) Difference between (a) and (b). (d) Histogram-equalized difference image. (Original image courtesy of Ms. Melissa D. Binde, Swarthmore College, Swarthmore, PA).

Example: Motion Detection

- Use **frame differencing** to compare successive video frames
- Insight: since camera is stationary, the background will not change much
- The moving foreground will change considerably
- Basic approach: use two successive frames to compute a **difference image**
- This produces a binary image from the threshold of | I1-I2 |
- Unfortunately this two-frame approach suffers from the double-image problem, which will display foreground pixels in both current and adjacent frame
- Solution: use double difference image (or three-frame difference)

rec 1:51pm start

dim 2:34 pm

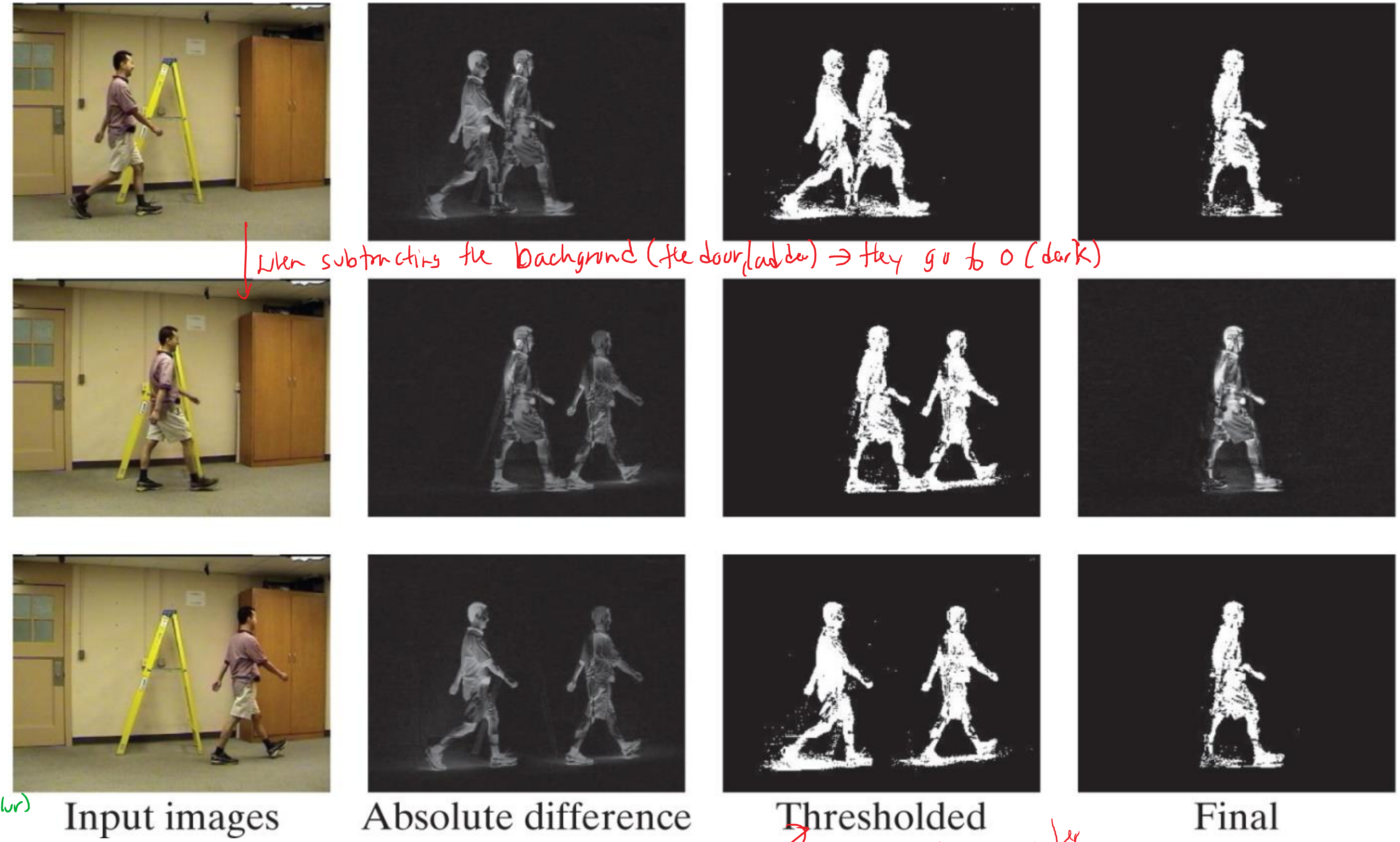
We want to detect who is moving.

Frame Differencing

· background turns black because it dont change (bla-bla=0 ⇒ dark)

· the person moves so his pixels arnt turned to 0

Figure 3.27 Detecting a moving object by frame differencing. LEFT COLUMN: Three image frames from a video sequence. SECOND COLUMN: The absolute difference between pairs of frames. THIRD COLUMN: Thresholded absolute difference. RIGHT COLUMN: Final result using double difference (top), triple difference (middle), and thresholded triple difference (bottom) methods.



Stan Birchfield

when subtracting the background (the door/ladder) ⇒ they go to 0 (dark)

wherever it is greater that a threshold ⇒ white

Real life ex

Detect if a nail gun misfires
focus camera on gun and background

· If the nail gun misfires, background wont change (need slowmo camera to reduce blur)

· If fires, background changes bec there is a difference in the frames

Pseudocode: Double Differencing

Simple

ALGORITHM 3.13 Compute the double difference between three consecutive image frames

FrameDifferenceDouble $(I_{t-1}, I_t, I_{t+1}, \tau)$

previous image middle img 3rd image more of the.

Input: successive images I_{t-1}, I_t, and I_{t+1}, and threshold τ

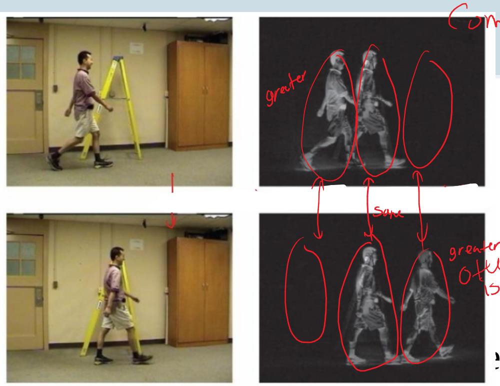
Output: binary image indicating the moving regions

```
for (x, y) ∈ I_t do
    d_1 ← |I_{t-1}(x, y) − I_t(x, y)|
    d_2 ← |I_{t+1}(x, y) − I_t(x, y)|
    I′(x, y) ← 1 if d_1 > τ AND d_2 > τ else 0
return I′
```

absolute value of the intensity value differences

If the 2 differences are greater than the treshold.

Compute pixels



return 1?

Pseudocode: Triple Differencing

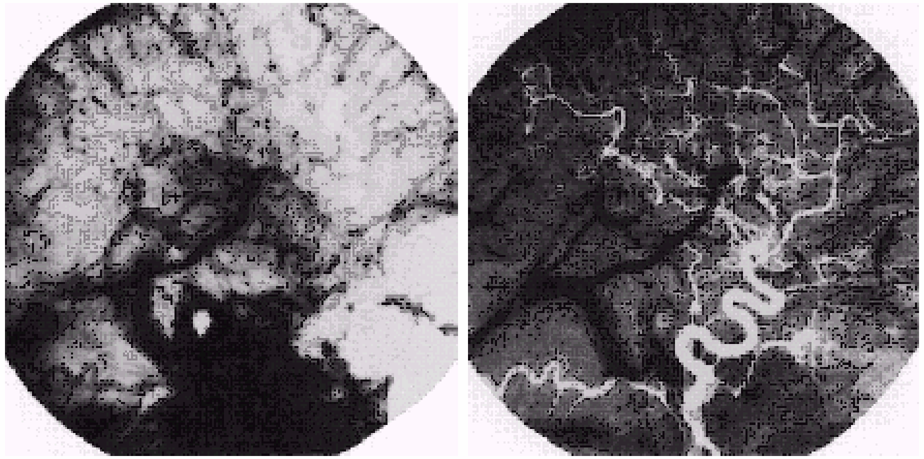
ALGORITHM 3.14 Compute the triple difference between three consecutive image frames

FRAMEDIFFERENCETRIPLE $(I_{t-1}, I_t, I_{t+1}, \tau)$

Input: successive images I_{t-1}, I_t, and I_{t+1}, and threshold τ
Output: binary image indicating the moving regions

for $(x, y) \in I_t$ **do**
  $d_1 \leftarrow |I_{t-1}(x, y) - I_t(x, y)|$
  $d_2 \leftarrow |I_{t+1}(x, y) - I_t(x, y)|$
  $d_3 \leftarrow |I_{t+1}(x, y) - I_{t-1}(x, y)|$
  $I'(x, y) \leftarrow 1$ **if** $d_1 + d_2 - d_3 > \tau$ **else** 0
return I'

Example: Mask Mode Radiography



mask image h(x,y)

image f(x,y) taken after injection of a contrast medium (iodine) into the bloodstream, with mask subtracted out.

Note:

• the background is dark because it doesn't change much in both images.

• the difference area is bright because it has a big change

- h(x,y) is the mask, an X-ray image of a region of a patient's body captured by an intensified TV camera (instead of traditional X-ray film) located opposite an X-ray source
- f(x,y) is an X-ray image taken after injection a contrast medium into the patient's bloodstream
- images are captured at TV rates, so the doctor can see how the medium propagates through the various arteries in an animation of f(x,y)-h(x,y).

Arithmetic Operations: Cross-Dissolve

· Terminator 2: used this tech

· Michael Jackson funded some research to have face morph in one of his vids ("black and white")

- Linearly interpolate between two images.
- Used to perform a fade from one image to another.
- Morphing can improve upon the results shown below.

no need lookup table

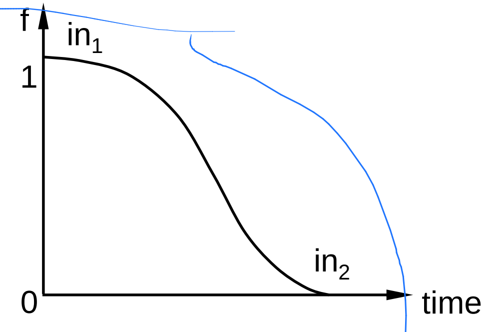
```
for(i=0; i<total; i++)
        out[i] = in1[i]*f + in2[i]*(1-f);
```

8 bit

floating point to

the output pixel is the corresponding pixels of in1 · f and in2 · (1-f)

Scale factor



in1 in2

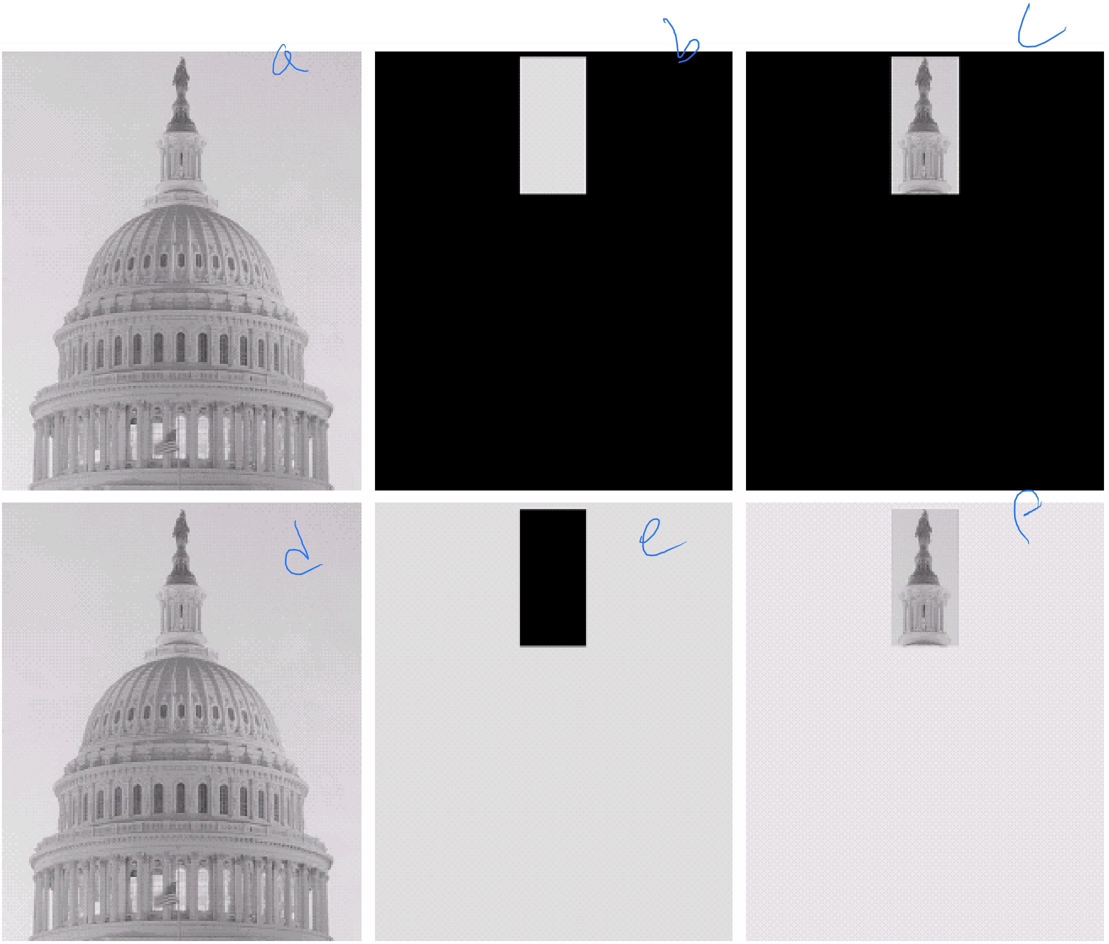
in2 is to 0

Masking

can use and/or operations

- Used for selecting subimages.
- Also referred to as region of interest (ROI) processing.
- In enhancement, masking is used primarily to isolate an area for processing.
- AND and OR operations are used for masking.

Example of AND/OR Operation



a b c
d e f

FIGURE 3.27 (a) Original image. (b) AND image mask. (c) Result of the AND operation on images (a) and (b). (d) Original image. (e) OR image mask. (f) Result of operation OR on images (d) and (e).