Bitmapped Images

Multimedia Techniques & Applications

Yu-Ting Wu

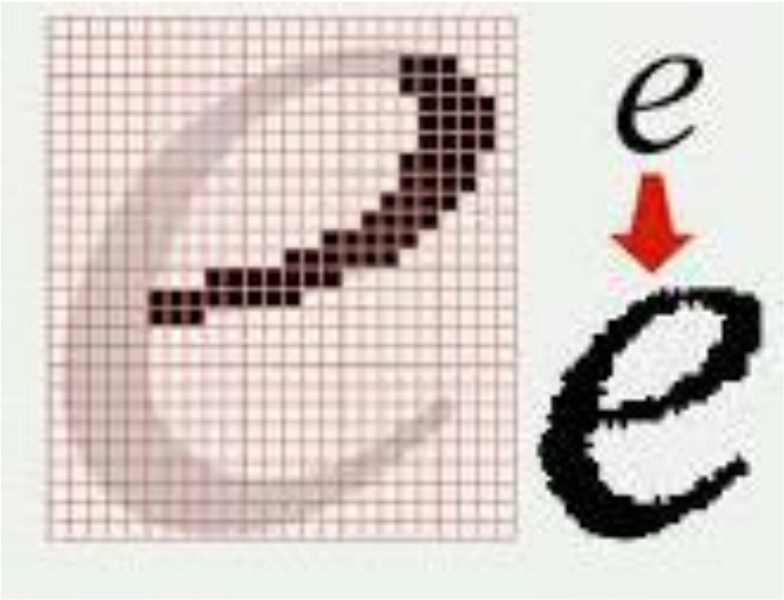
Outline

- Overview
- Image compression
- Image manipulation
- Image scaling

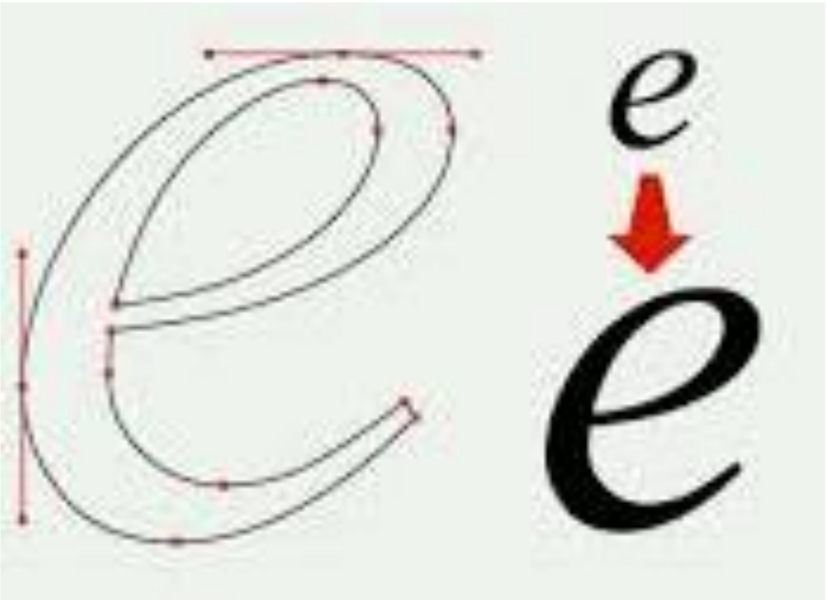
Outline

- **Overview**
- Image compression
- Image manipulation
- Image scaling

Recap: Two Approaches for Graphical Modeling



bitmapped graphics



vector graphics

Overview

- Record the value of every pixel in the image
- **Image size** is the main cost for the simplicity

- Images created from external devices are usually in a bitmapped fashion
  - Digital cameras
  - Scanners

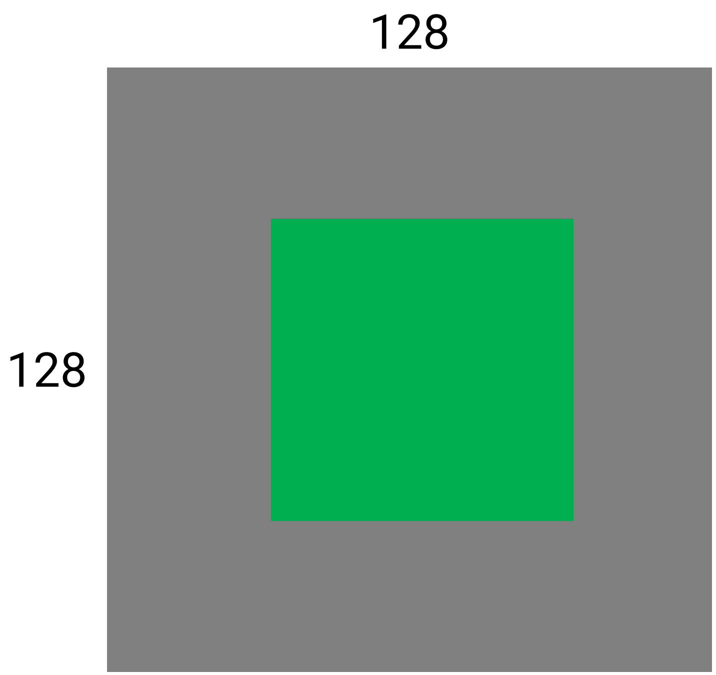
digital camera

scanner

Resolution

- A measure of **how finely** a device approximates continuous images using finite pixels
  - Closely related to sampling rates
- Two ways of specifying resolution
  - **Printers and scanners: number of dots per unit**
    - Dots per inch (dpi)
    - E.g., consumer printer (600 dpi), book production (1200 - 2700 dpi), scanners (300 dpi - 3600 dpi)
  - **Video: the size of a frame measured in pixels**
    - E.g., 640 x 480, 768 x 576
    - Can translate into the form of dpi if you know the physical dimension of the display device

Raw Size of Bitmapped Images



128 128 128 **setrgbcolor**

0 0 128 128 **rectfill**

0 255 0 **setrgbcolor**

32 32 64 64 **rectfill**

Four lines for the vector graphics format

128 x 128 x 3 = 49152 bytes for the bitmapped image format

File Formats of Bitmapped Images

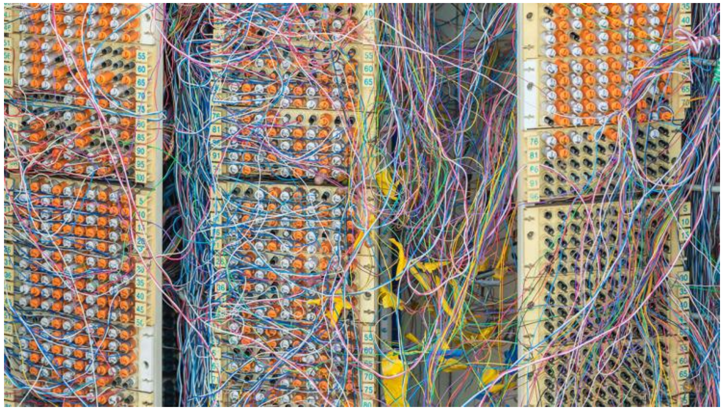
- Related to the way of compressing data
  - **Lossless compression**
    - GIF (Graphics Interchange Format)
    - PNG (Portable Network Graphics)
    - BMP (Windows Bitmap)
    - TIFF (Tagged Image File Format)
    - TGA (Truevision TGA, TARGA)
  - **Lossy compression**
    - JPEG (Joint Photographic Experts Group)
    - TIFF (Tagged Image File Format)

Outline

- Overview
- **Image compression**
- Image manipulation
- Image scaling

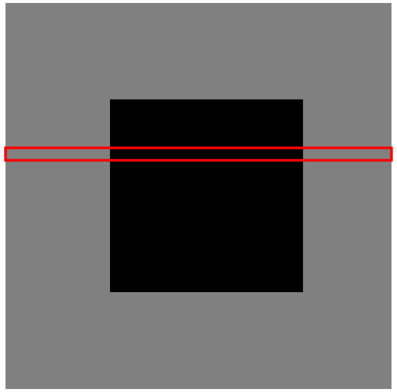
Image Compression

- **Motivation**
  - Faithfully storing all pixel values of an image takes lots of memory space
  - Human eyes can tolerate some minor errors in images
- **Observation**
  - Images are usually **smooth** and have some **spatial coherence**

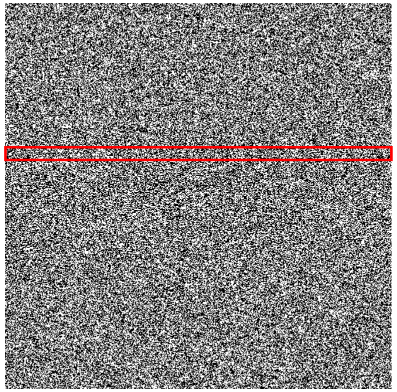


Compression Methods

- Spend some computation efforts to earn saving in space
- The effectiveness depends on the content of the compressed image
  - **Image size can become bigger after applying compression**
  - Definitely true, otherwise, any data can be compressed into one byte



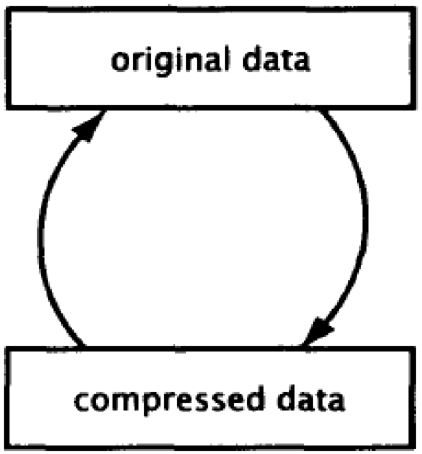
128 bytes ➔ 6 bytes for a row (RLE)



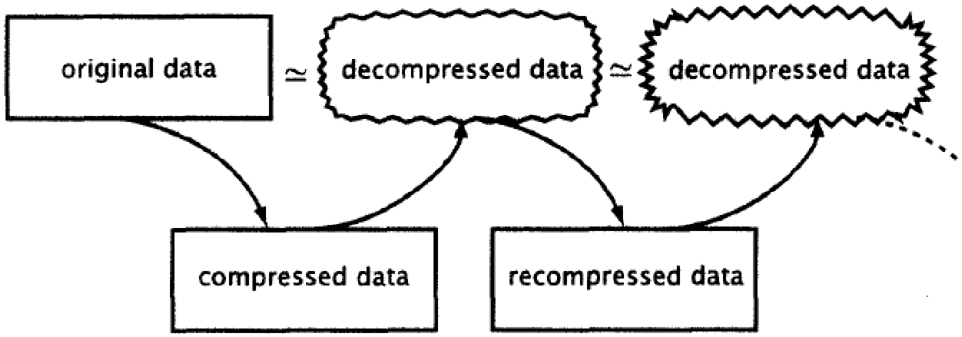
128 bytes ➔256 bytes for a row (RLE)

Compression Methods (cont.)

- **Lossless compression**
  - No information will lose during a compression/decompression cycle
  - E.g., run-length encoding (RLE), variable-length coding

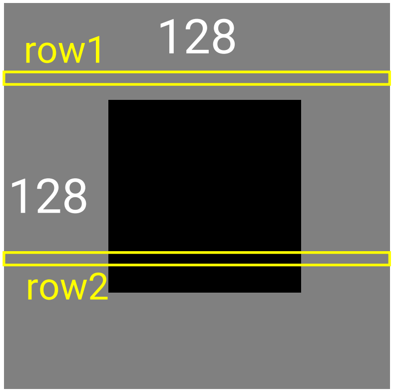


- **Lossy compression**
  - Discard some information during the compression process and the information can **never** be recovered
  - E.g., JPEG



Run-Length Encoding (RLE)

- The simplest compression technique
- Each stored value is followed by a **count** to indicate the number of consecutive pixels of that value
- Example
  - RLE for row1: *128 128*
    - *Originally, we need 128 bytes assuming 8-bit intensity is used*
  - RLE for row2: *128 32 0 64 128 32*



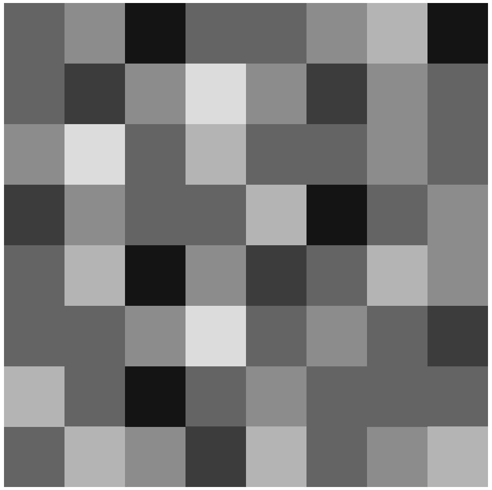
Huffman Coding

- The best-known variable length coding
- **Lossless** compression
- Example:

  Assume an 8 x 8 image containing 6 different pixel intensities

  We can count their occurrence:

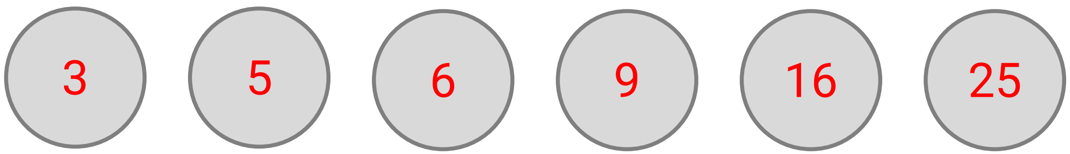
naïve encoding	000	001	010	011	100	101
intensity	20	60	100	140	180	220
occurrence	5	6	25	16	9	3



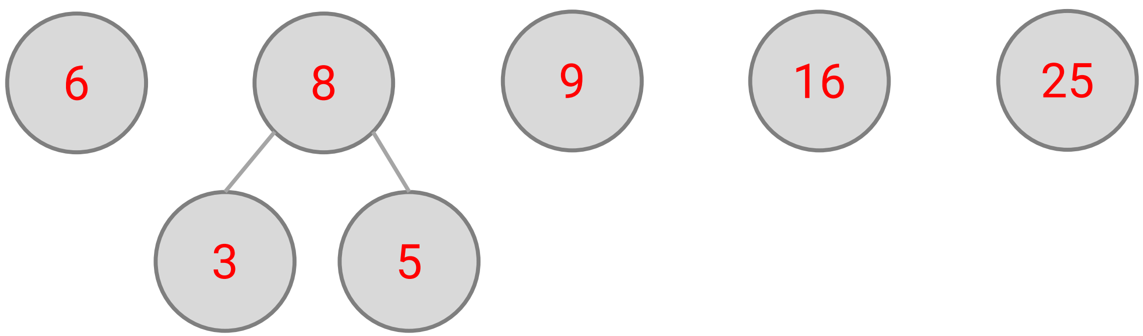
Huffman Coding (cont.)

20	60	100	140	180	220
5	6	25	16	9	3

- **Algorithm**
  - Build a **Huffman tree**
    - Sort the occurrence of intensity

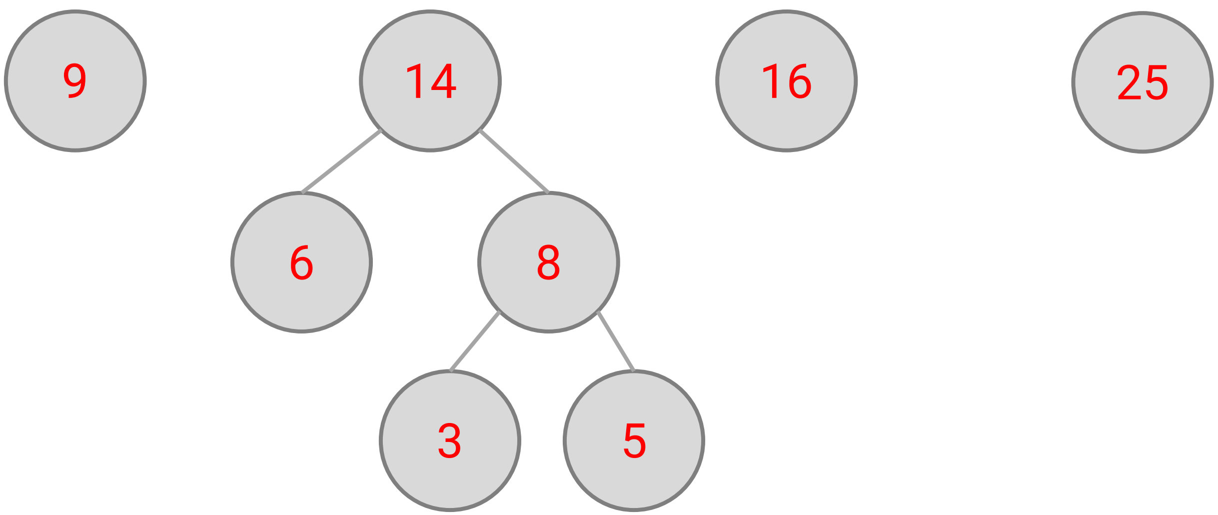


    - Merge the two with the smallest occurrence, and sort again



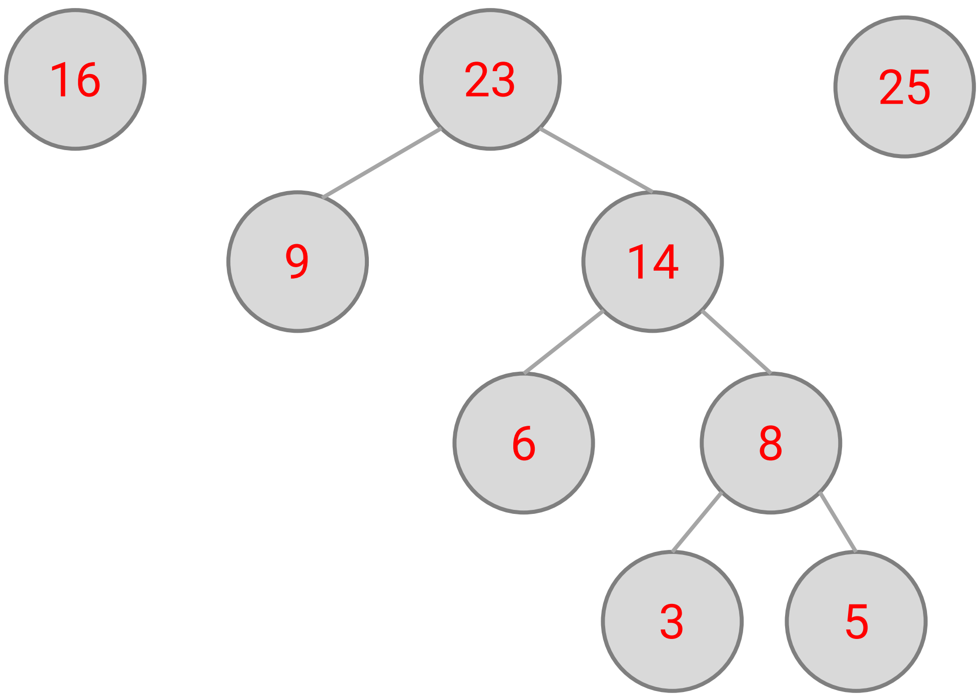
Huffman Coding (cont.)

- **Algorithm**
  - Build a **Huffman tree**
    - Keep doing …



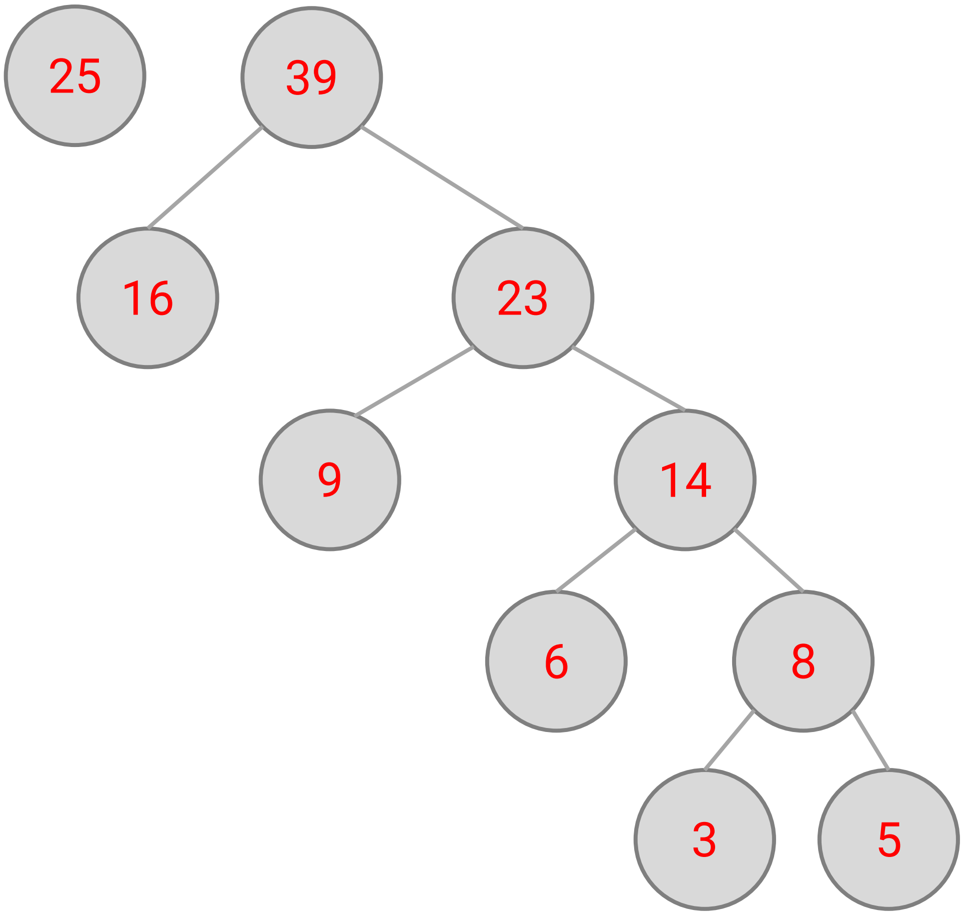
Huffman Coding (cont.)

- **Algorithm**
  - Build a **Huffman tree**
    - Keep doing …



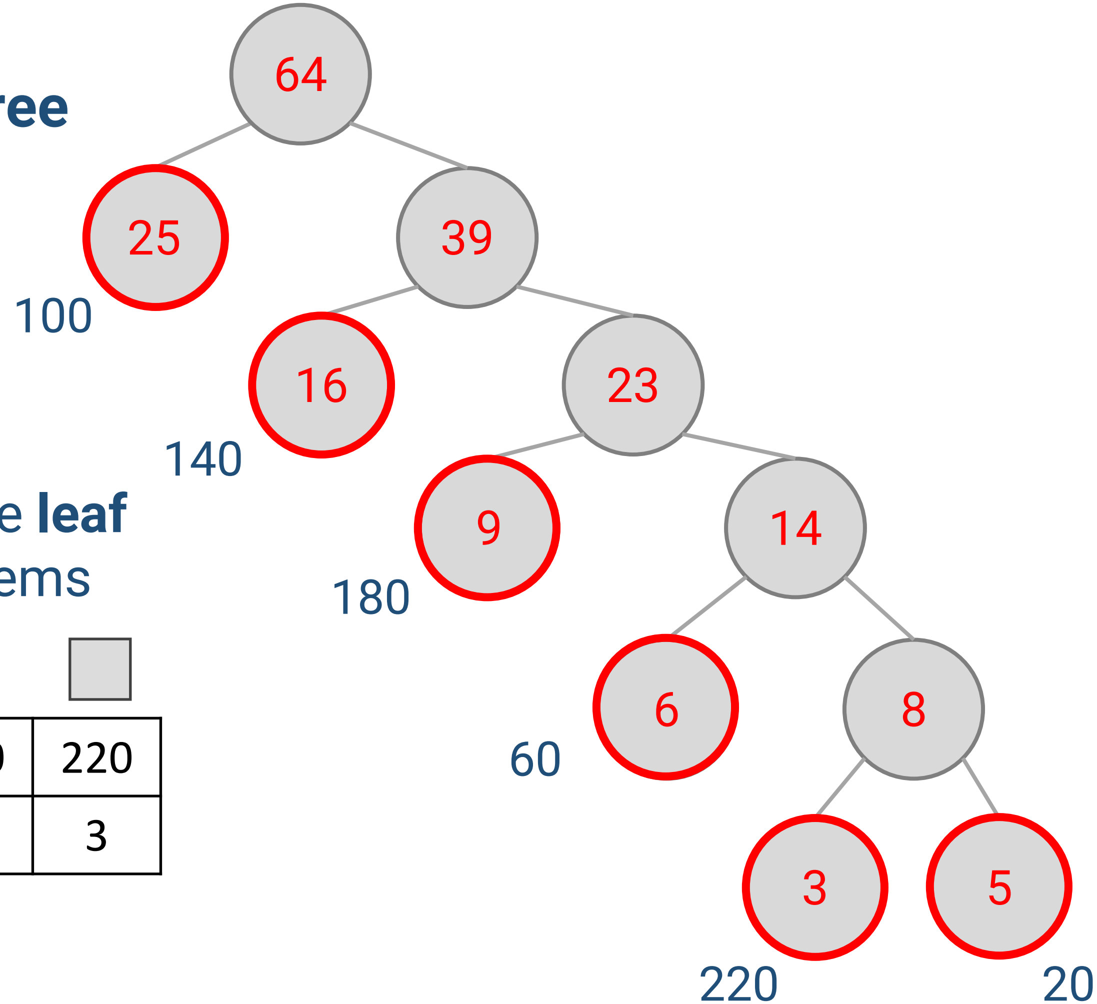
Huffman Coding (cont.)

- **Algorithm**
  - Build a **Huffman tree**
    - Keep doing …



Huffman Coding (cont.)

- **Algorithm**
  - Build a **Huffman tree**
    - Keep doing …

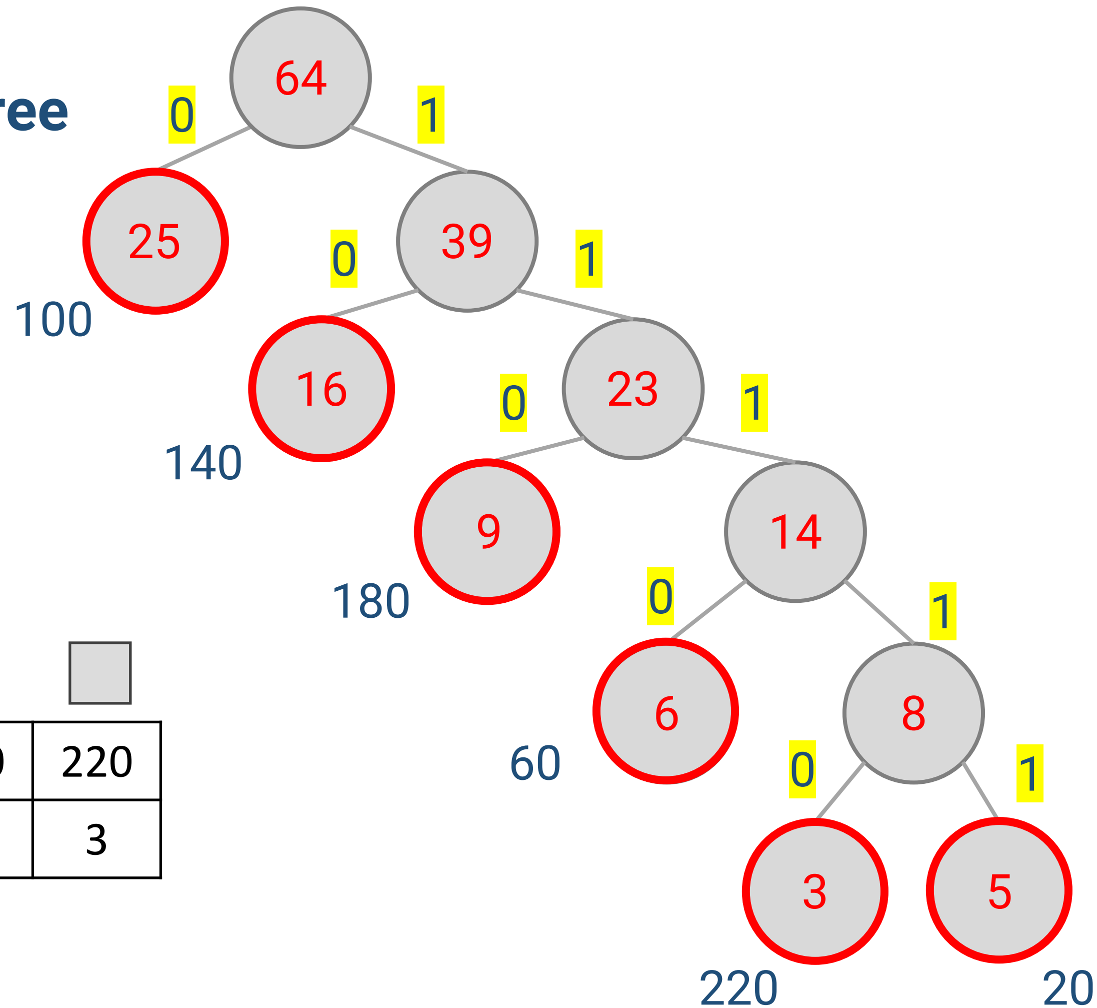


Once we have done this, the **leaf nodes** are the initial data items

20	60	100	140	180	220
5	6	25	16	9	3

Huffman Coding (cont.)

- **Algorithm**
  - Build a **Huffman tree**
  - Label the code (from root)

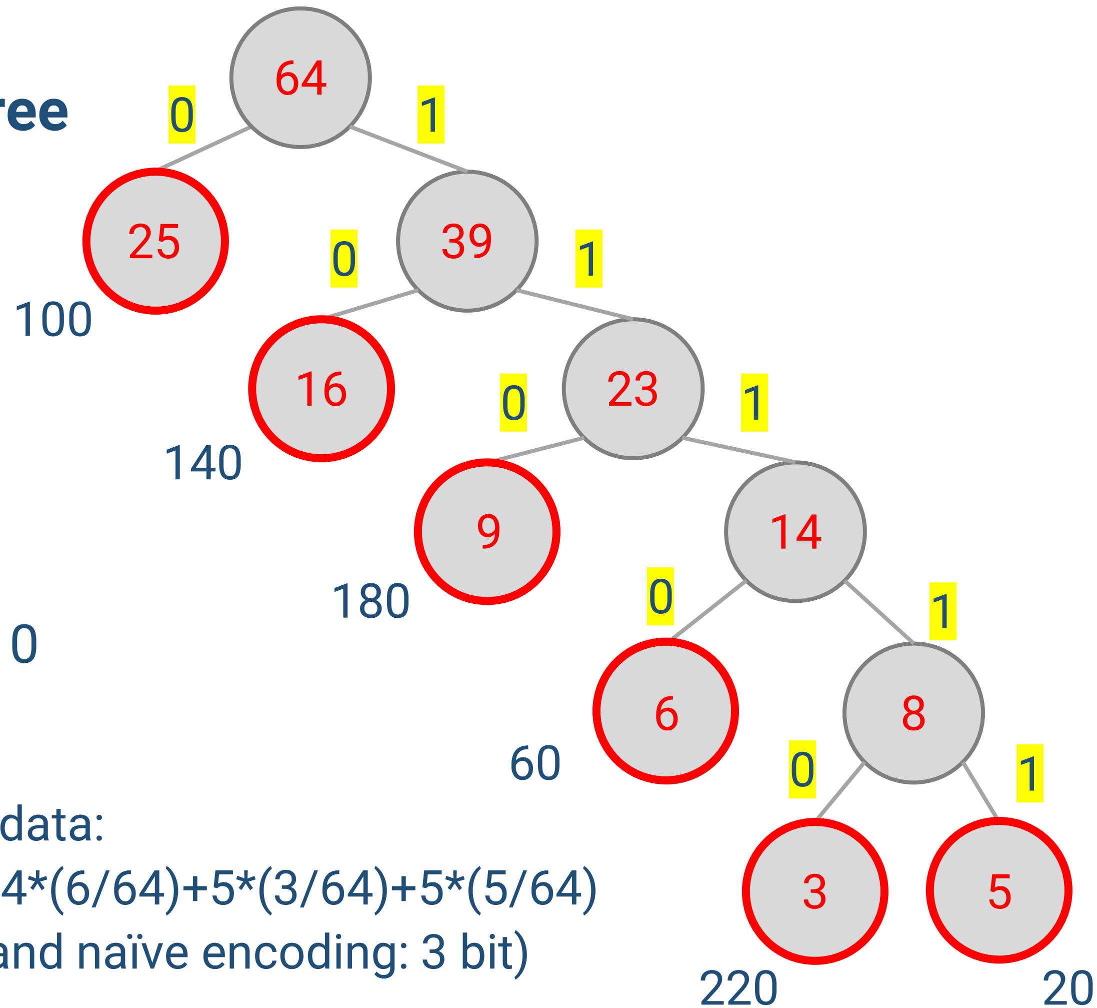


20	60	100	140	180	220
5	6	25	16	9	3

Huffman Coding (cont.)

- **Algorithm**
  - Build a **Huffman tree**
  - Label the code (from root)
  - We will obtain

20: 11111
100: 0
180: 110
60: 1110
140: 10
220: 11110



The average number of bits per data:
1*(25/64)+2*(16/64)+3*(9/64)+4*(6/64)+5*(3/64)+5*(5/64)
= **2.31** (compared to raw: 8 bit, and naïve encoding: 3 bit)

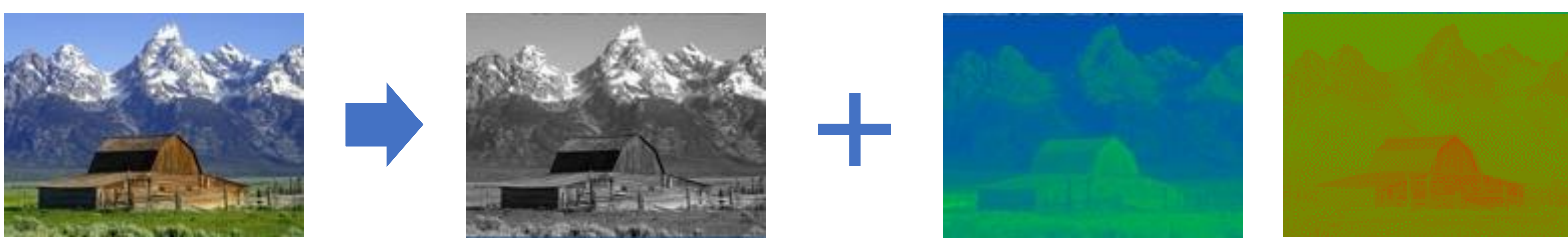
JPEG Compression

- JPEG is the most important **lossy** compression technique, which stands for ***Joint Photographic Experts Group***
  - Related file formats: *.jpg / *.jpeg / *.jpe / *.jfif / *.jfi / *.jif
- It works because image data can tolerate a certain amount of data loss

JPEG Compression (cont.)

RGB ➔ YCbCr → divide into 8x8 blocks → DCT → Quanti-zation → zigzag ordering → RLE / Huffman

- **RGB ➔ YCbCr**
  - People are more sensitive to intensity (Y) and less sensitive to color (Cb, Cr)
  - Cb and Cr are lower frequency and have more spatial coherence
  - Compress Cb and Cr; while keeping Y as it is

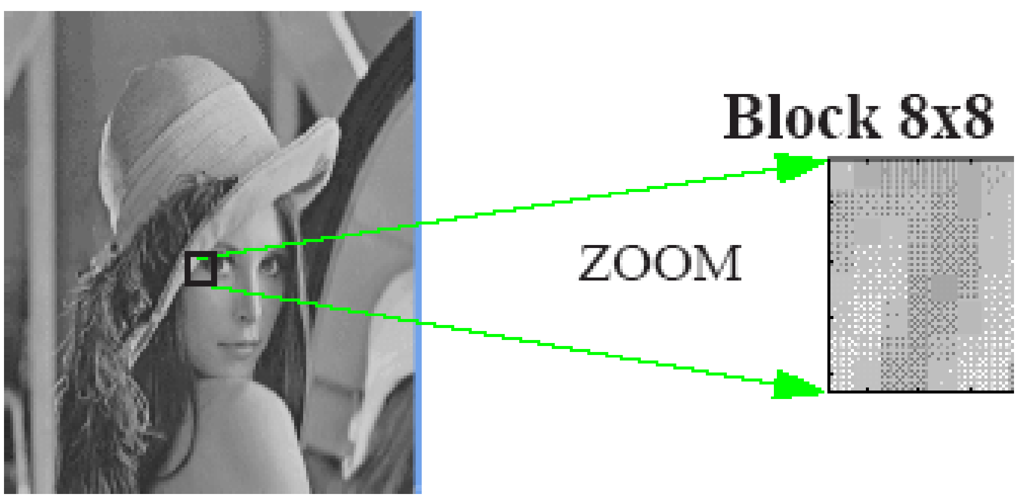


RGB Y Cb Cr

JPEG Compression (cont.)

RGB ➔ YCbCr → **divide into 8x8 blocks** → DCT → Quanti-zation → zigzag ordering → RLE / Huffman

- **Divide into 8x8 blocks (for Cb & Cr)**
  - The entire image is too difficult to compress
  - Small image block has higher coherence

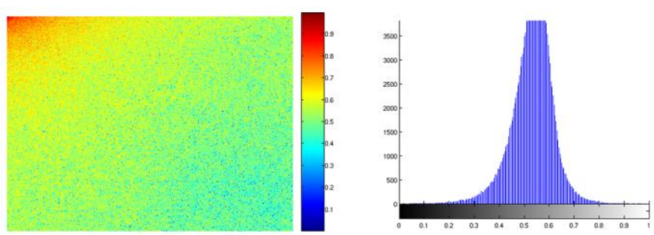


JPEG Compression (cont.)

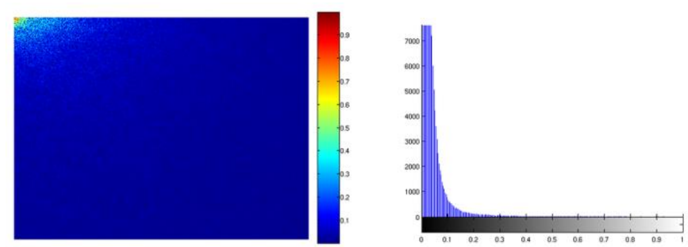
RGB → YCbCr → divide into 8x8 blocks → **DCT** → Quanti-zation → zigzag ordering → RLE / Huffman

- **Discrete Cosine Transform (DCT)**
  - A method for transforming an image into its frequency domain
  - The DCT of an image block is the coefficients of different cosines of the image block

image



DFT result

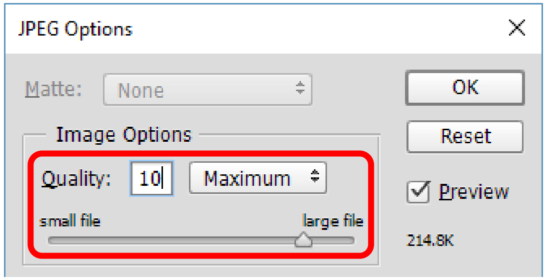


DCT result

JPEG Compression (cont.)

RGB → YCbCr → divide into 8x8 blocks → DCT → **Quanti-zation** → zigzag ordering → RLE / Huffman

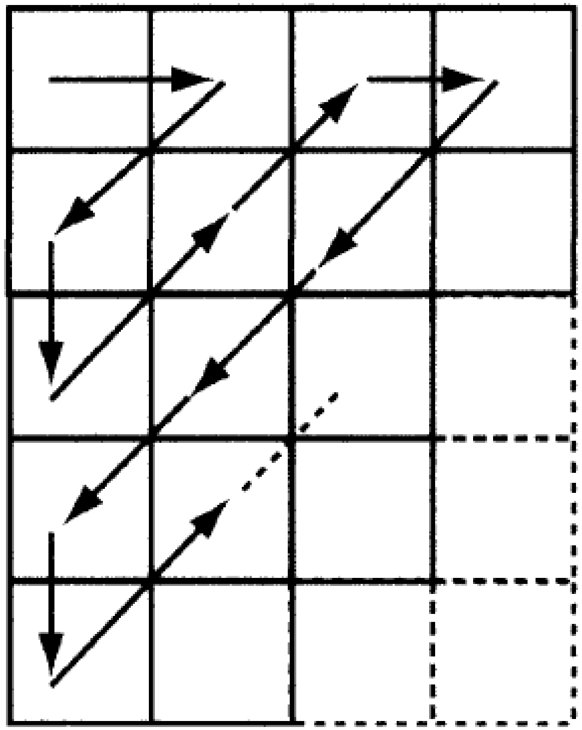
- **Quantization**
  - Humans are less sensitive to the high-frequency signal
  - Use fewer bits for high-frequency signals in the DCT result and vice versa
  - **This step is the reason for lossy compression**
  - After this step, many components will end up with zero coefficients



JPEG Compression (cont.)

RGB ➔ YCbCr → divide into 8x8 blocks → DCT → Quanti-zation → **zigzag ordering** → RLE / Huffman

- **Zigzag ordering**
  - For later Huffman encoding
  - Result in a longer zero sequence

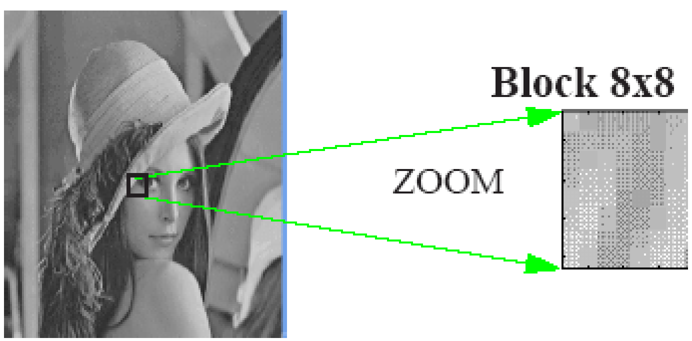


JPEG Compression (cont.)

RGB ➔ YCbCr → divide into 8x8 blocks → DCT → Quanti-zation → zigzag ordering → **RLE / Huffman**

- **RLE / Huffman encoding**
  - Different strategy for DC and AC term
  - The DC components of different blocks are encoded using Huffman algorithm
  - The AC components within a block are encoded using RLE

DC



JPEG Compression (cont.)

- The decompression of JPEG data is done by reversing the compression process
- We can control the degree of compression by altering the amount of quantization
- JPEG compression usually achieves very high compression rate for natural images (5% of the original size)

JPEG Compression (cont.)

PNG stands for Portable Network Graphics (lossless compression)

Outline

- Overview
- Image compression
- **Image manipulation**
- Image scaling

Image Manipulation

- **Motivations**
  - Correct deficiencies in an image (e.g., noise, red-eye)
  - Create images that are difficult or impossible to make naturally (e.g., glow)

- Type of image manipulations
  - Pixel **point** processing
  - Pixel **group** processing

Pixel Point Processing

- Compute a pixel's new value solely on the basis of its old value

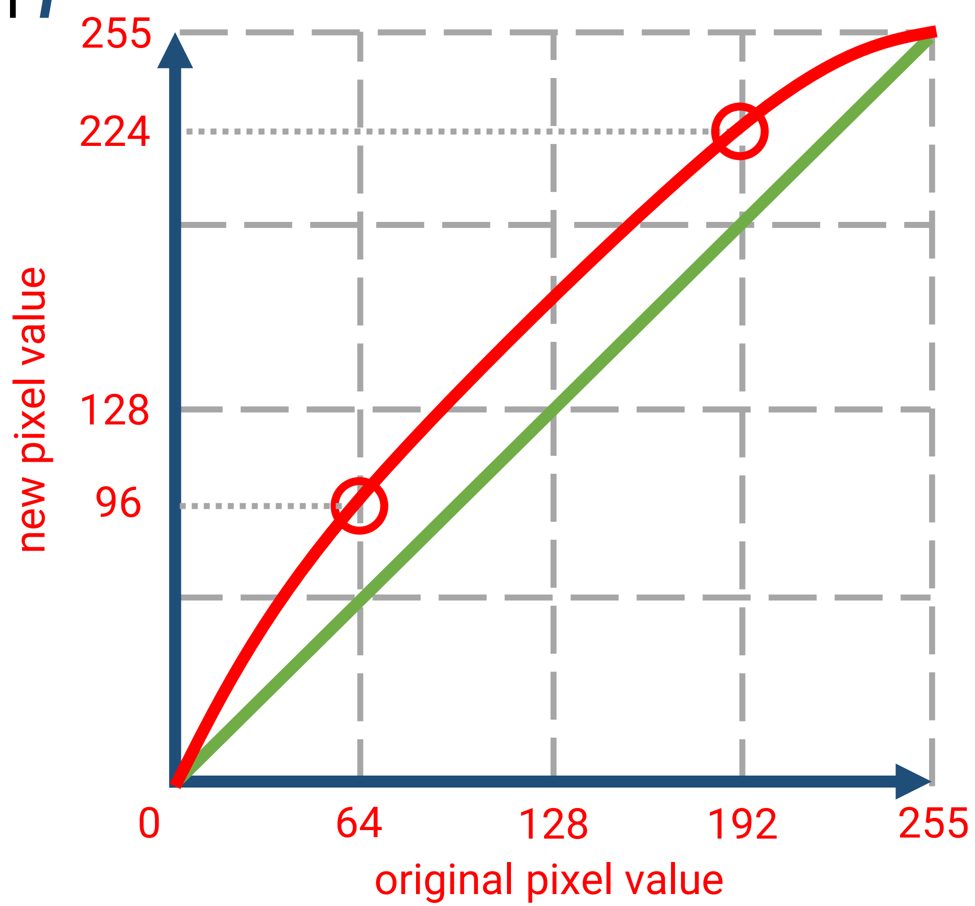
mapping function

$$p' = \boxed{f}(p)$$

- Some examples
  - Adjustment of brightness
  - Adjustment of contrast
  - Change the black/white levels

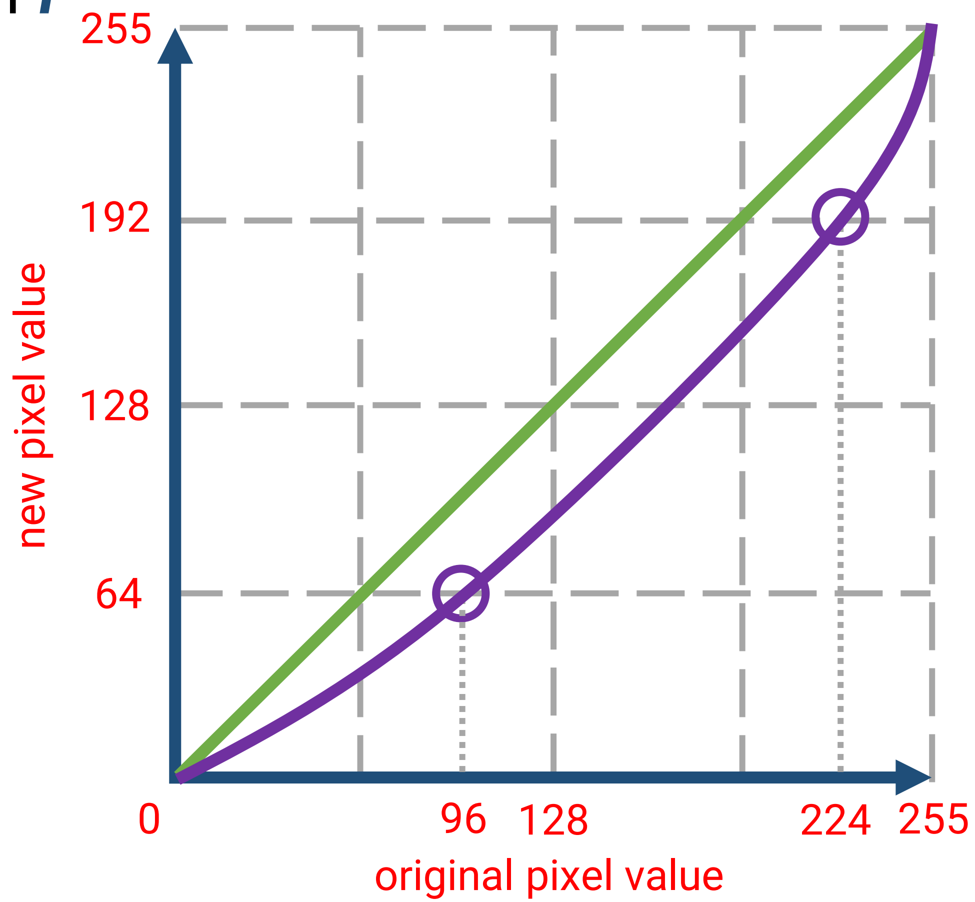
Color Curve

- The operations can be considered generally as altering the mapping function ***f***
- **Color curve**



Color Curve (cont.)

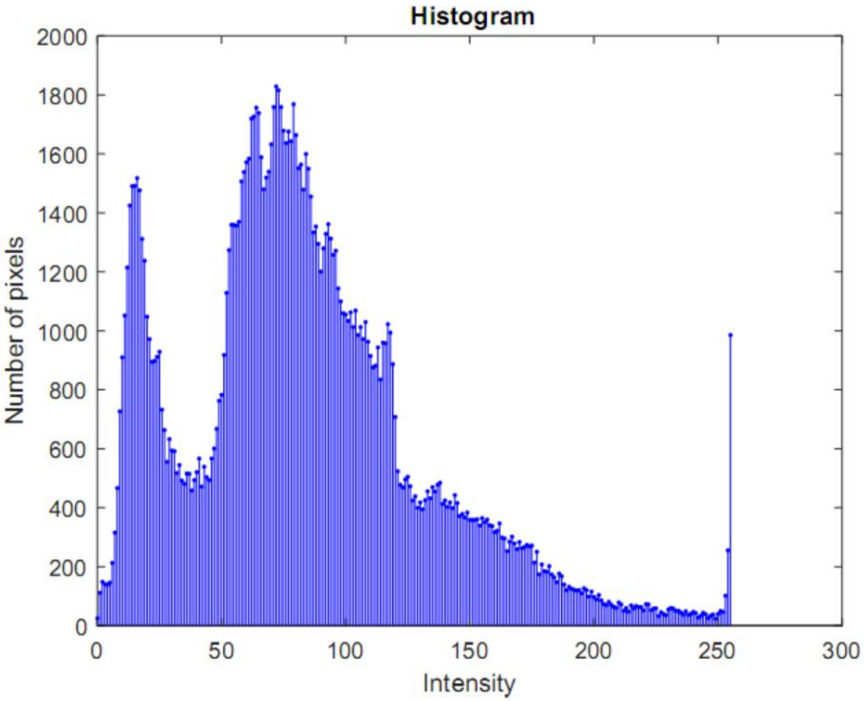
- The operations can be considered generally as altering the mapping function ***f***
- **Color curve**



Histogram

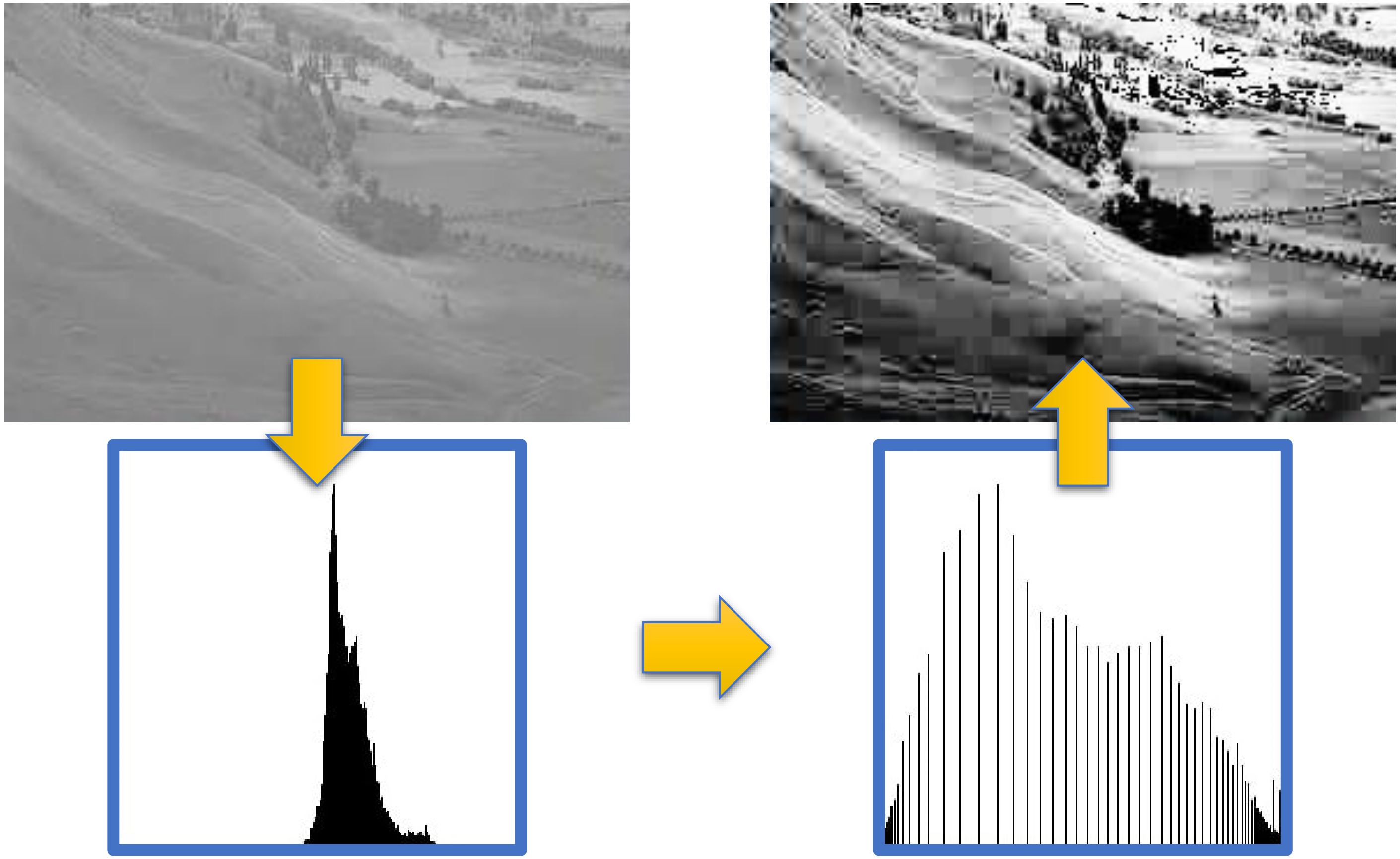
- An approximate representation of the **distribution** of numerical data

Intensity Image



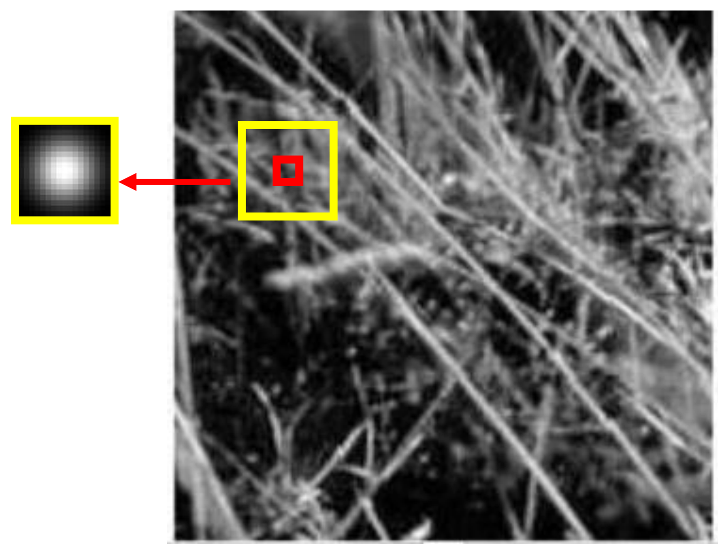
Histogram Equalization

- Remap pixel values to produce a new distribution with higher contrast



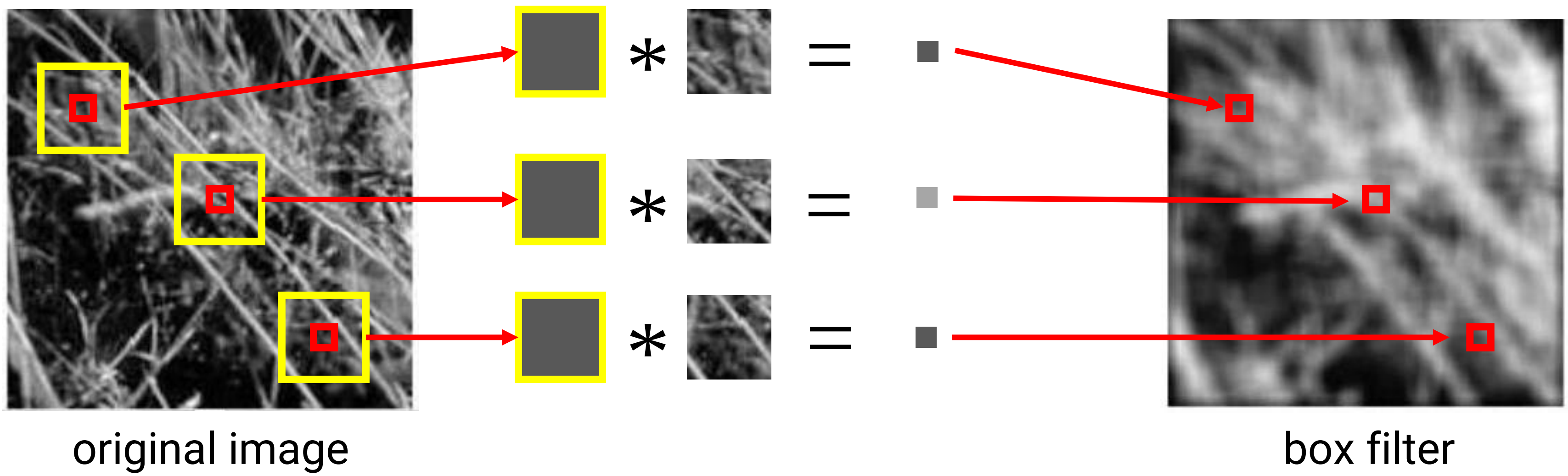
Pixel Group Processing

- Compute each pixel's new value as a function not just of its old value, but also of **the values of neighboring pixels**
- Usually related to **filtering**
  - For a pixel of an image, specify a two-dimensional array of weights of its neighbors
  - Several types of filters
    - Smoothing
    - Sharpening
    - Detecting edge



Box Filter

- Each neighbor has the same weight



Box Filter (cont.)

(1/9*0+1/9*0+1/9*0+
1/9*0+1/9*0+1/9*0+
1/9*9+1/9*9+1/9*9)

convolution mask
convolution kernel

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

filter

*

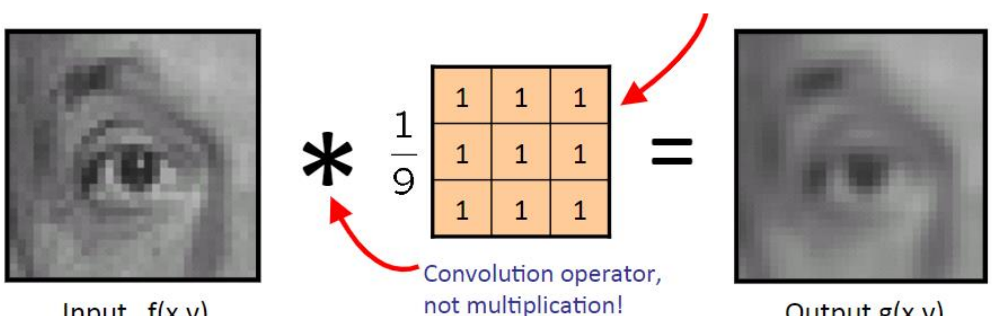
0	0	9	9
0	0	9	9
0	0	9	9
0	0	9	9

image (signal)

=

0	3	6	9
0	3	6	9
0	3	6	9
0	3	6	9

filtered image (signal)

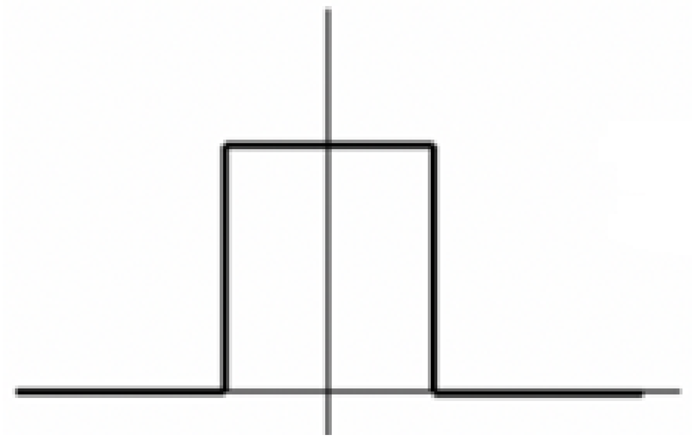


Input f(x,y)

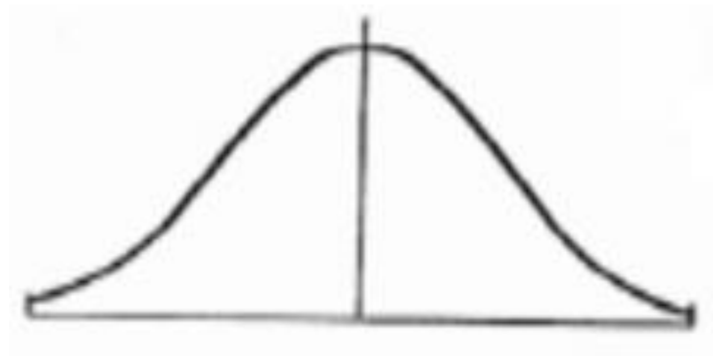
Output g(x,y)

Box Filter (cont.)

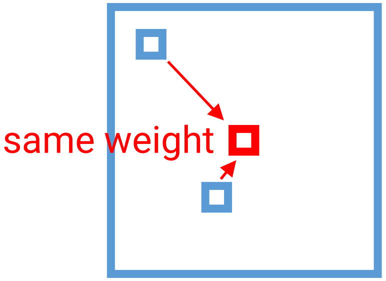
- Visualization of kernel weight

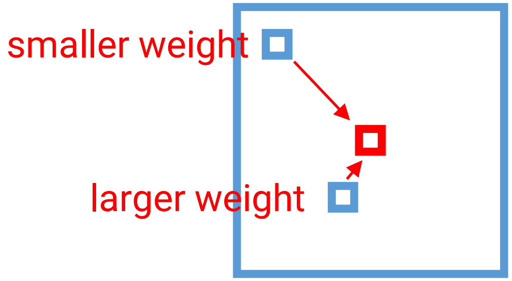


box filter



gaussian filter



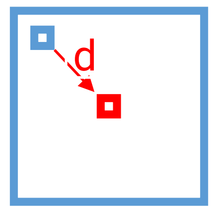


Gaussian Filter

- The weight of neighbor falls exponentially with its distance to the filtered pixel
  - Standard deviation α controls the speed of decreasing

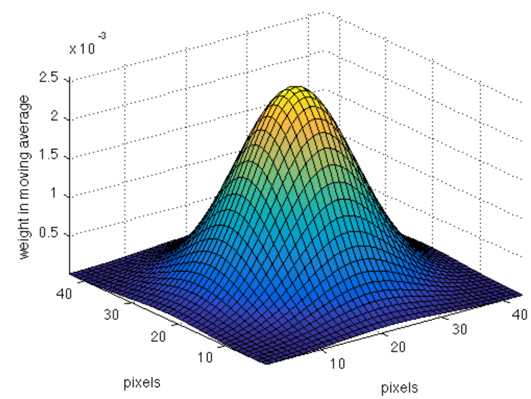
distance

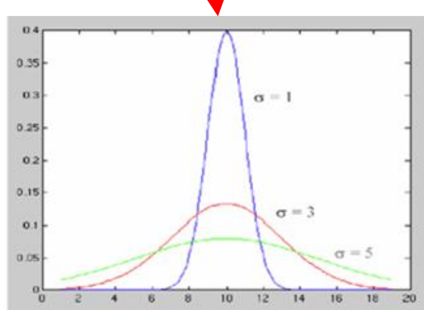
$$g(x, y) = \boxed{\frac{1}{2\pi\sigma^2}} e^{-\frac{\boxed{x^2+y^2}}{2\sigma^2}}$$



normalization
(sum of weight = 1)

standard
deviation

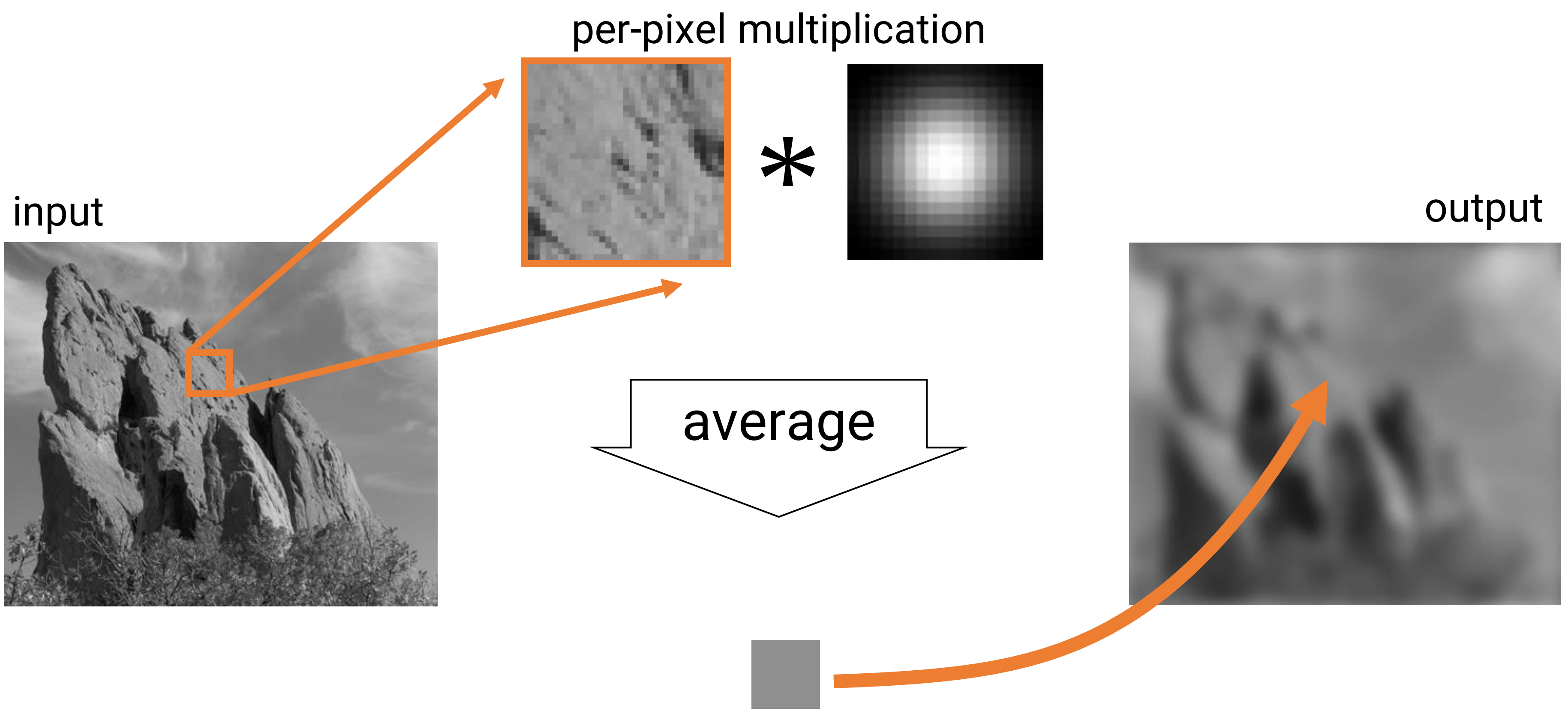




$\frac{1}{273}$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

Gaussian Filter (cont.)

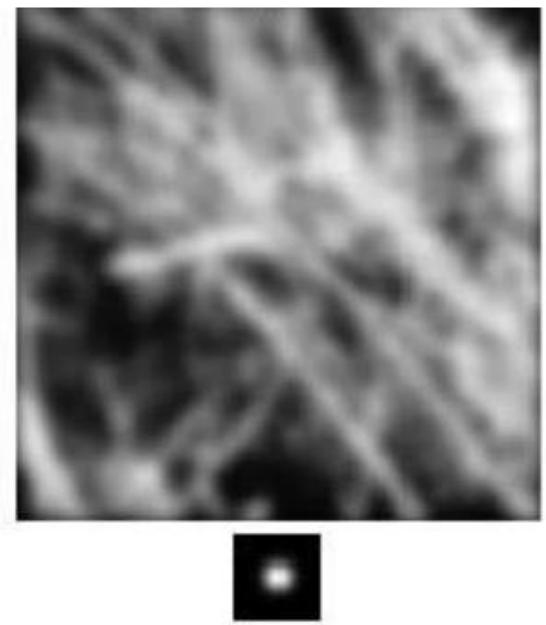


Box Filter v.s. Gaussian Filter

original image

box filter

gaussian filter



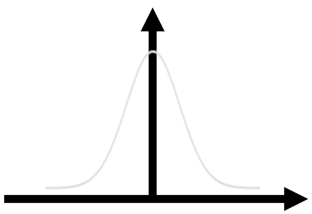
Bilateral Filter

- Properties of Gaussian filter

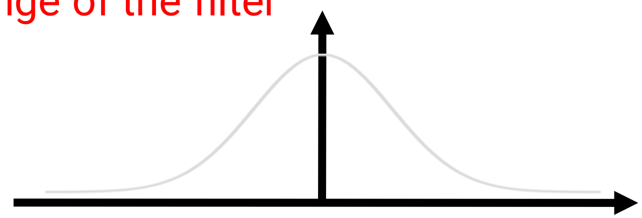
input

$$g(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

range of the filter

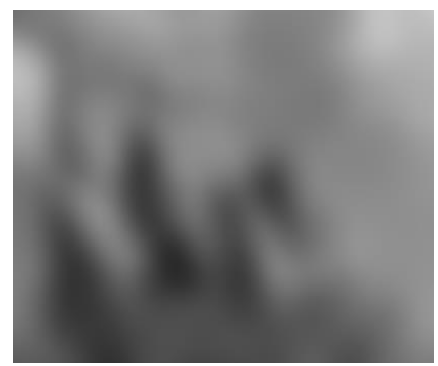


small σ



large σ

limited smoothing



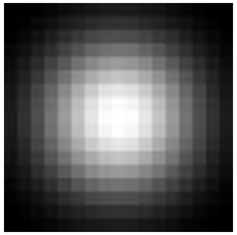
strong smoothing

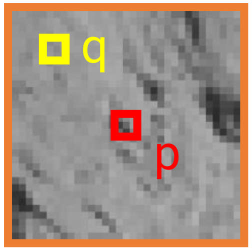
Bilateral Filter (cont.)

- Problems of Gaussian filter
  - Does smooth images
  - But smoothes too much: **edges are blurred**
  - Only spatial distance matters
  - No edge term

spatial distance

$$GB[I]_{\mathbf{p}} = \sum_{\mathbf{q}\in S} \boxed{G_\sigma(\|\mathbf{p}-\mathbf{q}\|)}\, I_{\mathbf{q}}$$





S

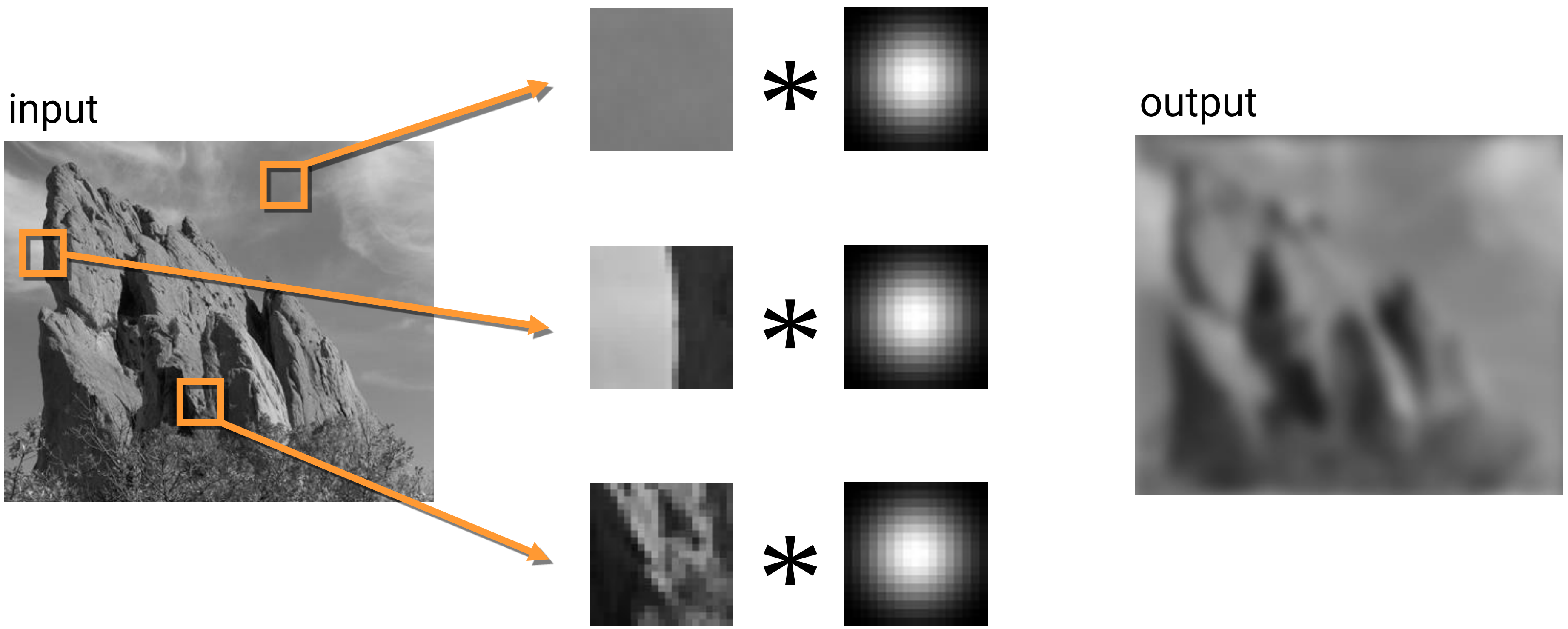
$$g(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

input

output

Bilateral Filter (cont.)

- Problems of Gaussian filter
  - Same Gaussian kernel everywhere



Bilateral Filter (cont.)

- Combine another Gaussian weight computed by **intensity difference** (**edge preserving**)

spatial distance

$$GB[I]_{\mathbf{p}} = \sum_{\mathbf{q} \in S} G_{\sigma}(\| \mathbf{p} - \mathbf{q} \|) I_{\mathbf{q}}$$

$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in S} G_{\sigma_s}(\| \mathbf{p} - \mathbf{q} \|) G_{\sigma_r}(| I_{\mathbf{p}} - I_{\mathbf{q}} |) I_{\mathbf{q}}$$

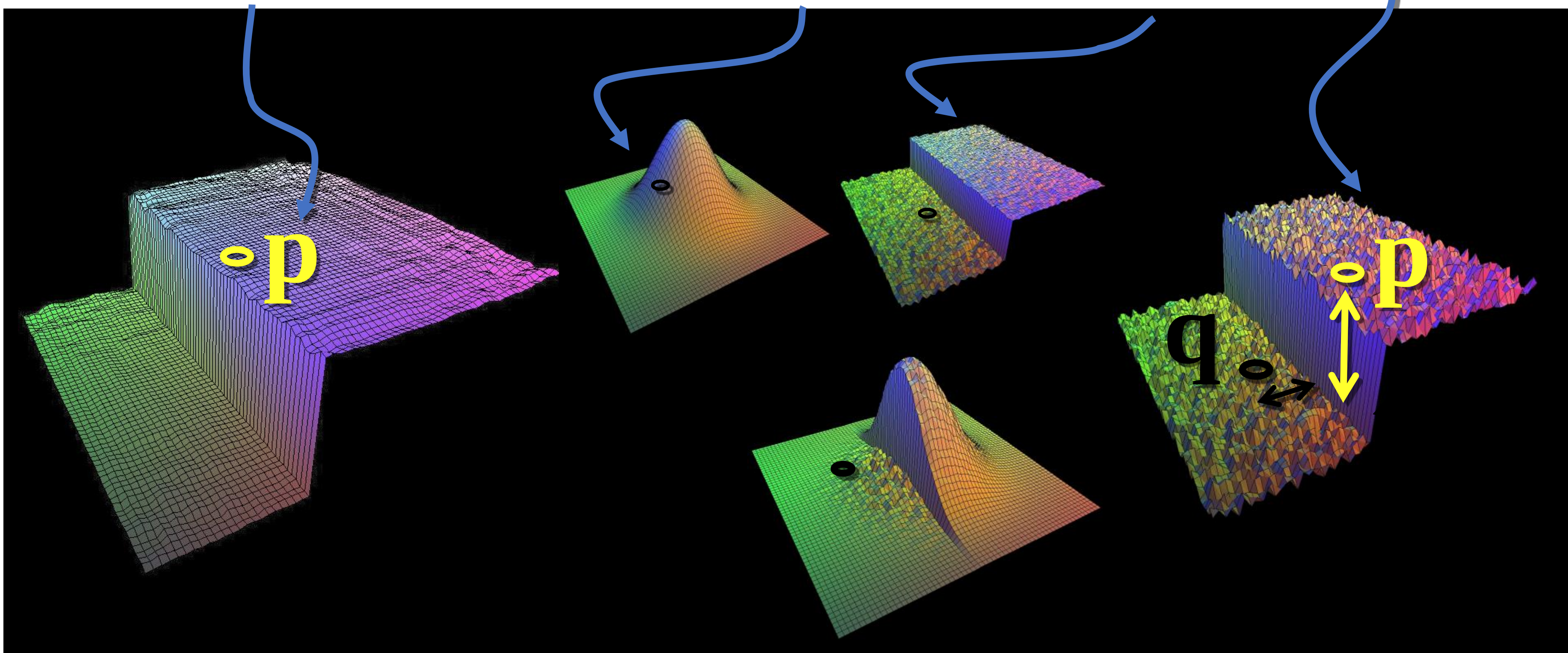
spatial distance　range (intensity) distance

NEW!

Bilateral Filter (cont.)

- Visualization

$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q}\in S} G_{\sigma_{\mathrm{s}}}(\|\mathbf{p}-\mathbf{q}\|)\; G_{\sigma_{\mathrm{r}}}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)\; I_{\mathbf{q}}$$



Bilateral Filter (cont.)

- Parameters

$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in S} G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|)\, G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)\, I_{\mathbf{q}}$$

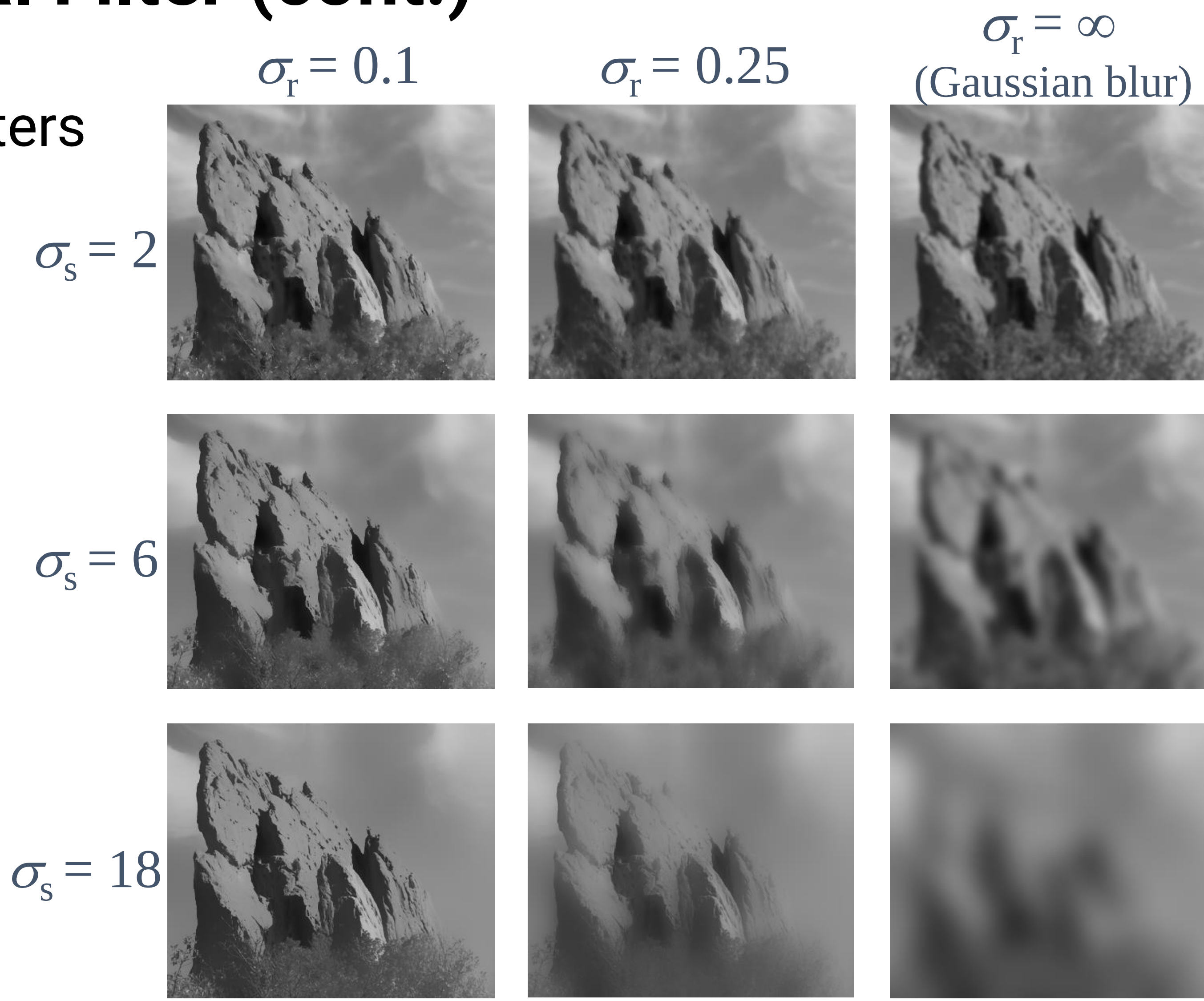
- **Spatial sigma σ_s :** the spatial extent of the kernel, size of the considered neighborhood

- **Range sigma σ_r :** how the weight decreases with respect to the range difference

Bilateral Filter (cont.)

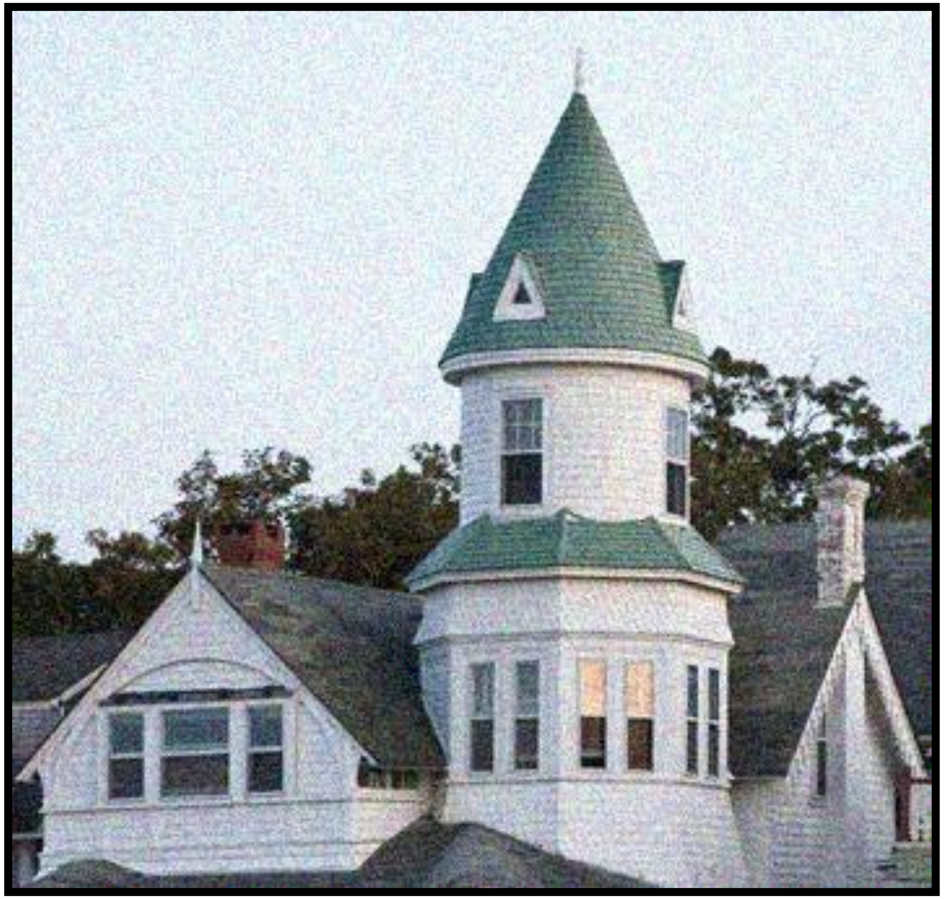
- Parameters

input



Bilateral Filter Application (cont.)

- Denoising

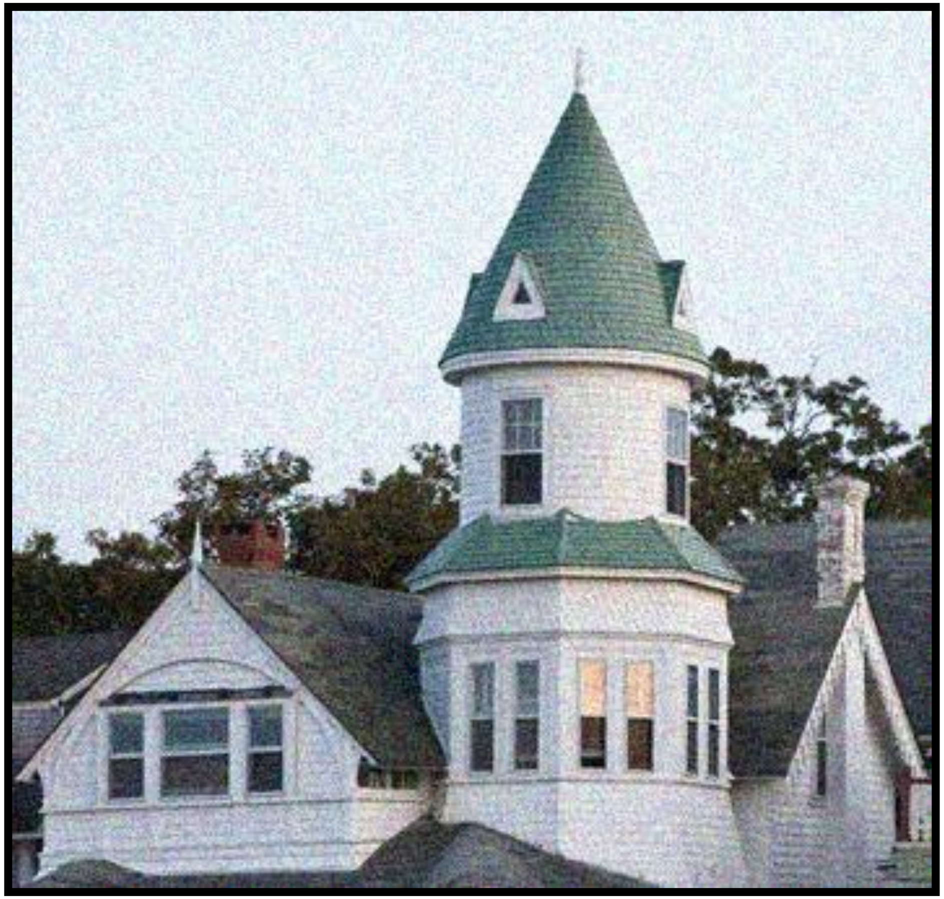


noisy input

Median Filter 5x5

Bilateral Filter Application (cont.)

- Denoising



noisy input

Bilateral Filter 7x7

Bilateral Filter Application (cont.)

- Denoising


noisy input


Gaussian Filter


Bilateral Filter

Sobel Filter

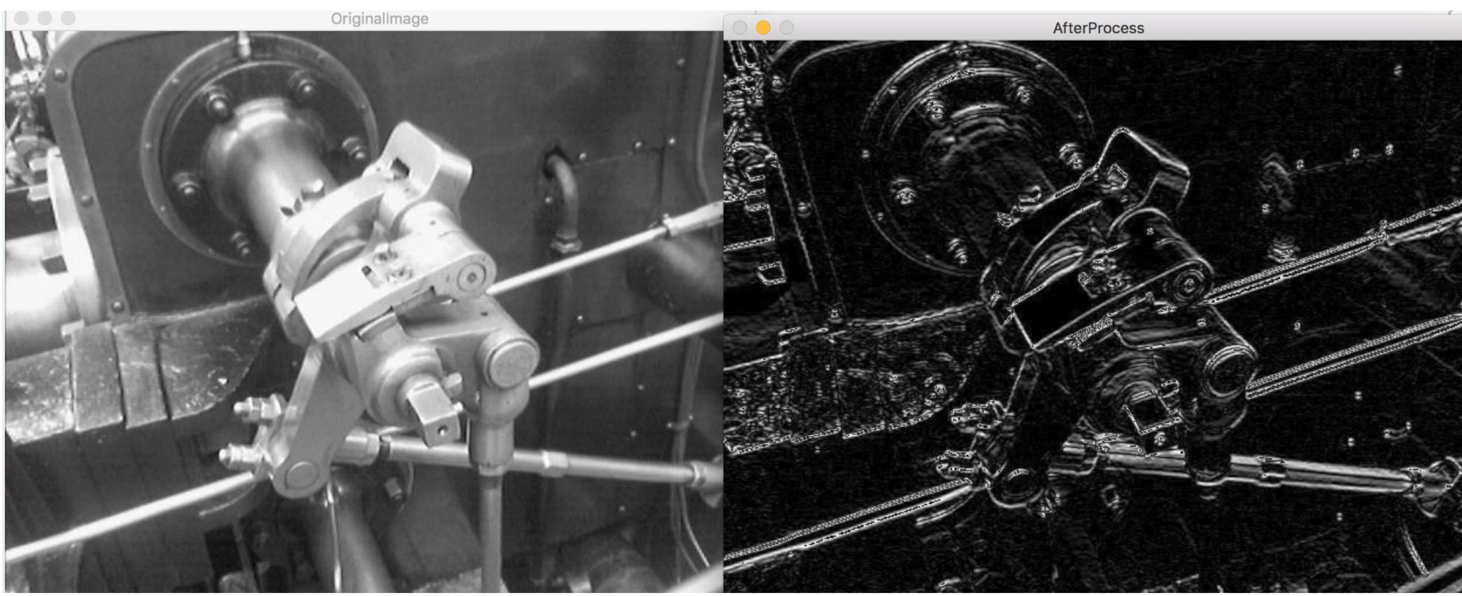
- Negative weights are commonly used for edge detection

-1	0	+1
-2	0	+2
-1	0	+1

Gx

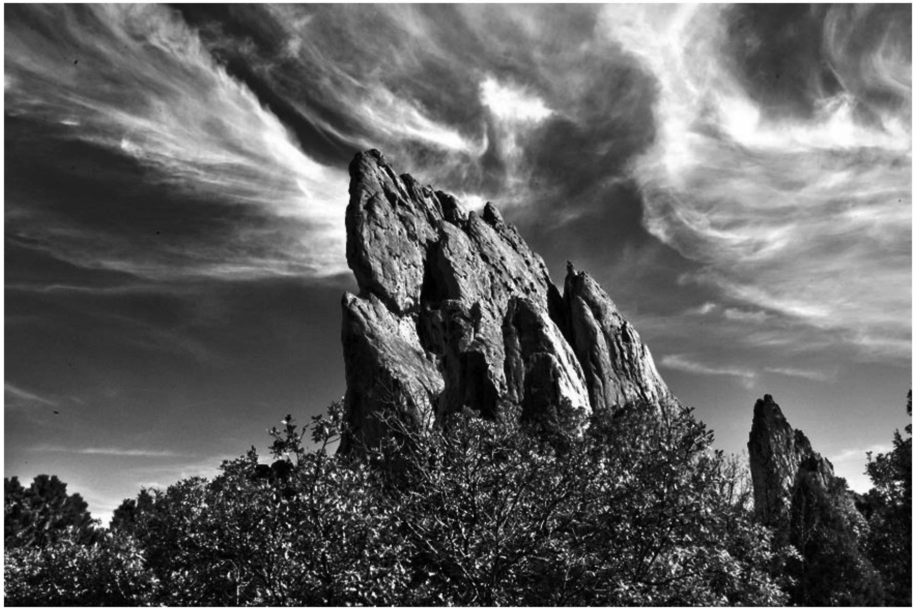
+1	+2	+1
0	0	0
-1	-2	-1

Gy



Photographic Style Transfer

- Two-scale Tone Management for Photographic Look, Bae et al. SIGGRAPH 2006



Photographic Style Transfer (cont.)

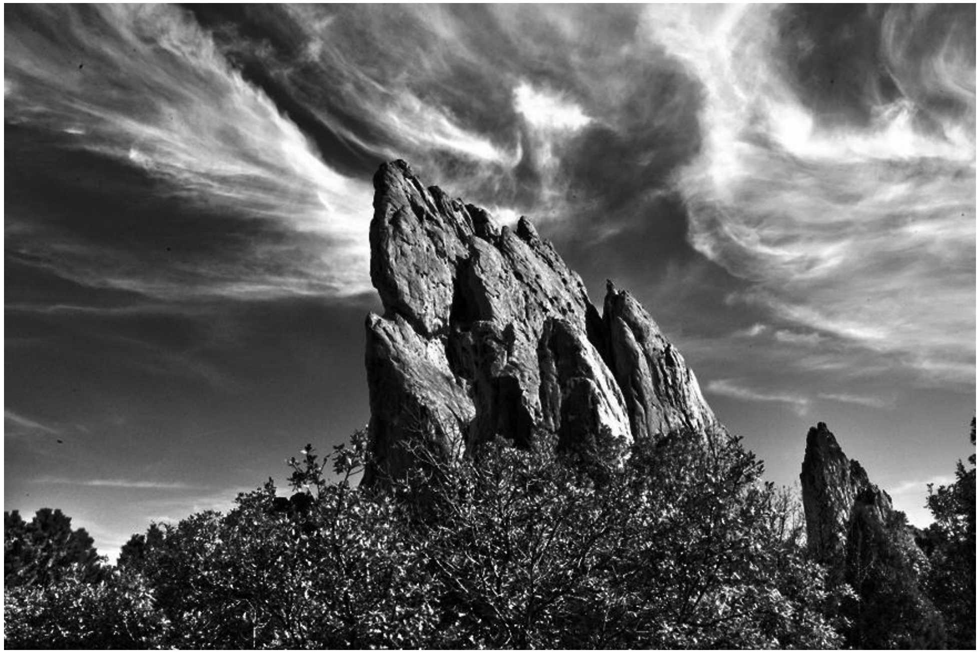
- **Motivation**

Ansel Adams
Clearing Winter Storm

an amateur photographer

Photographic Style Transfer (cont.)

- **Motivation**

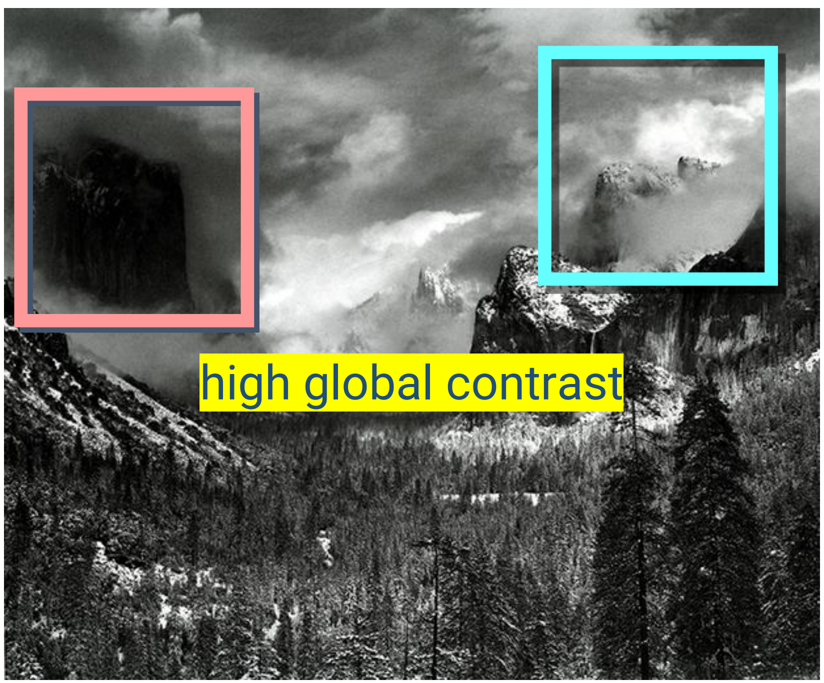


Ansel Adams
Clearing Winter Storm

style transferred result
(an imitation of Ansel Adams)

Photographic Style Transfer (cont.)

- **Observation**
  - Different photographers have different tonal styles
    - Global contrast

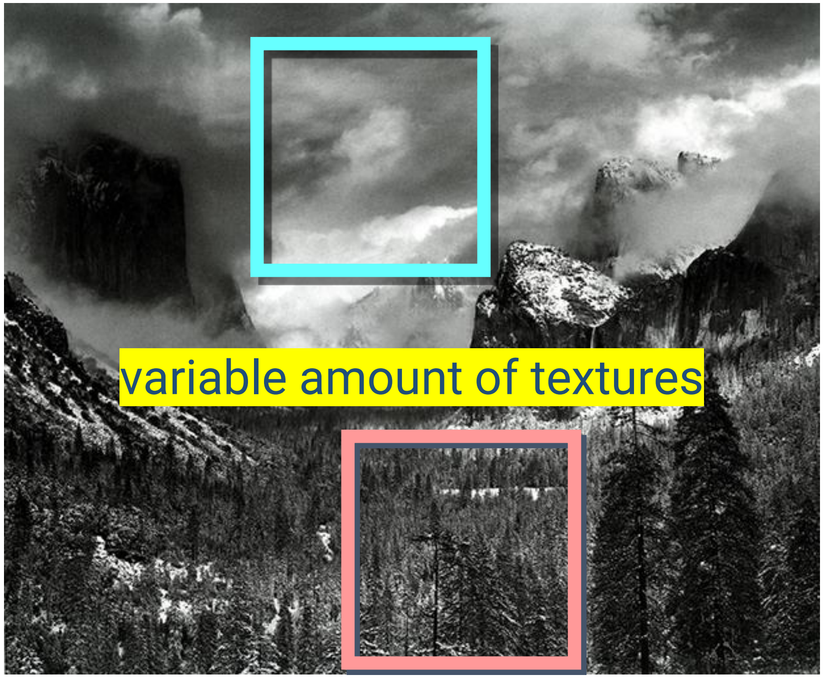


Ansel Adams

Kenro Izu

Photographic Style Transfer (cont.)

- **Observation**
  - Different photographers have different tonal styles
    - Global contrast
    - Local contrast



Ansel Adams

Kenro Izu

Photographic Style Transfer (cont.)

- **Goal**
  - Transfer look between photographers

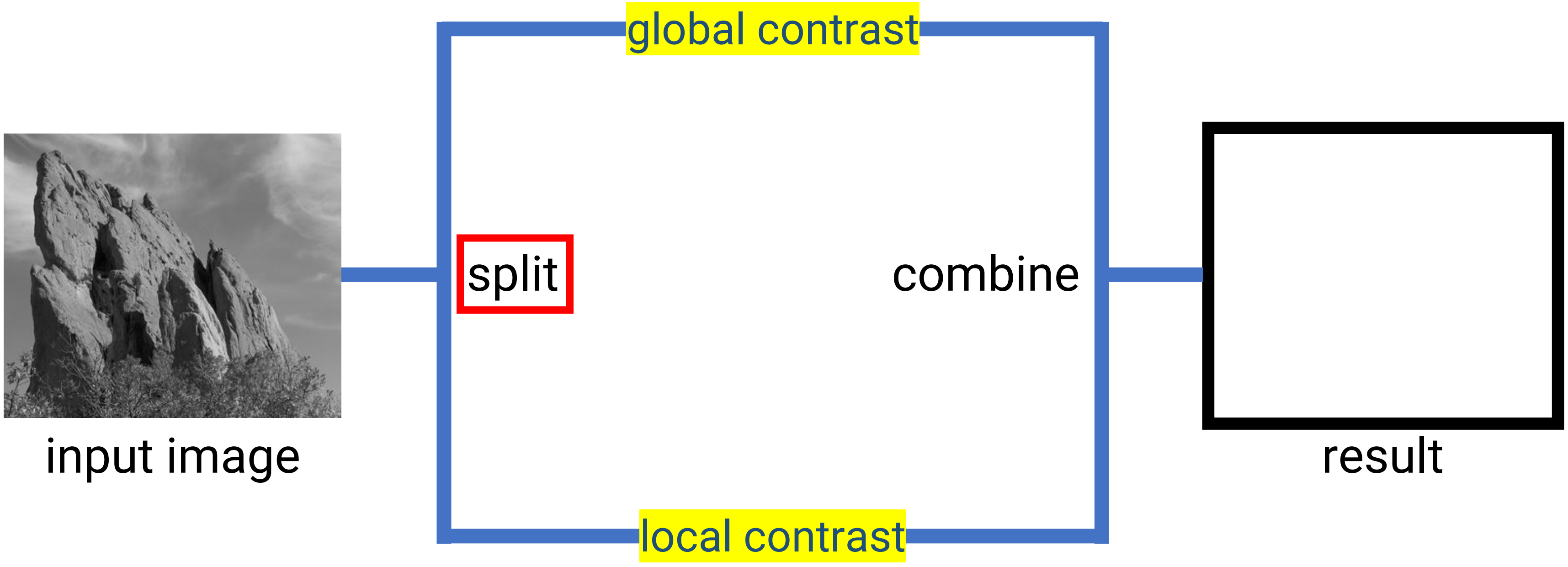
model

input image

result

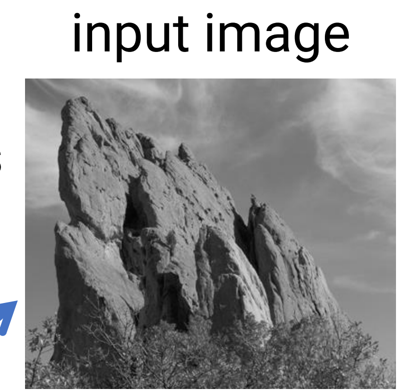
Photographic Style Transfer (cont.)

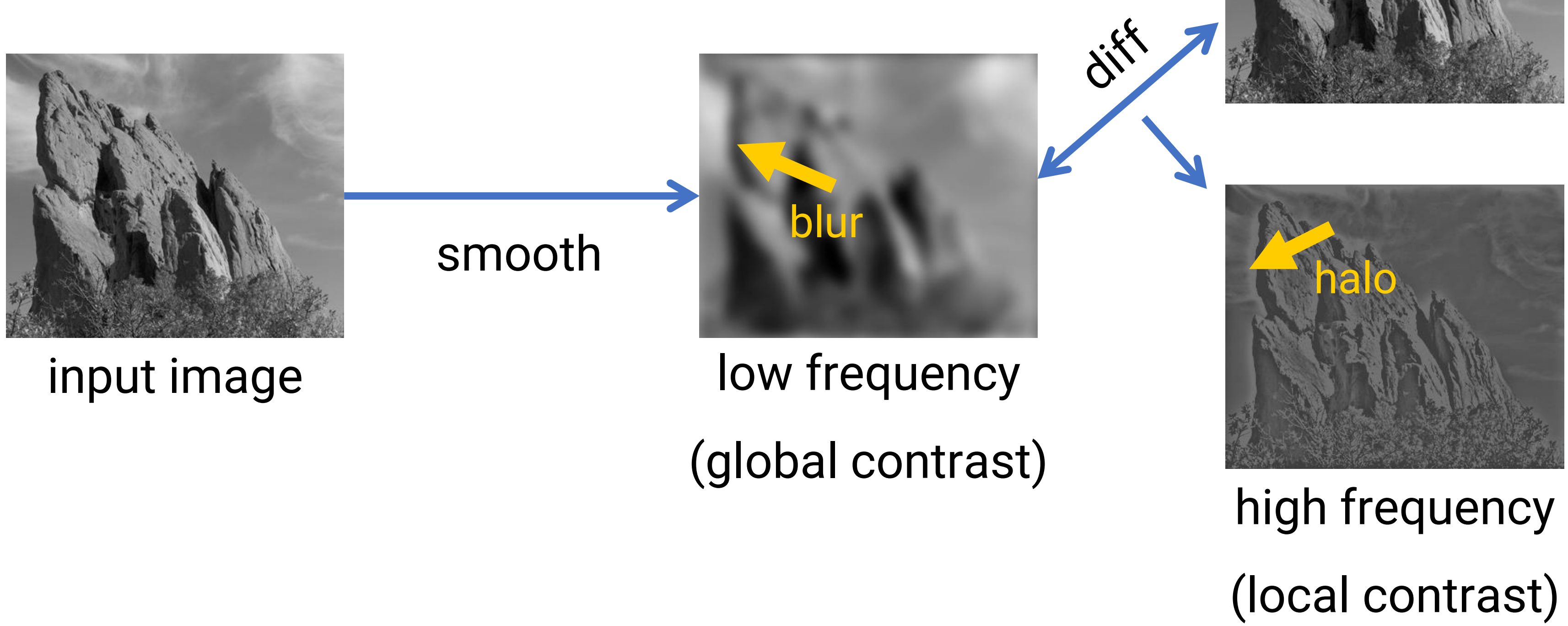
- **Algorithm**



Photographic Style Transfer (cont.)

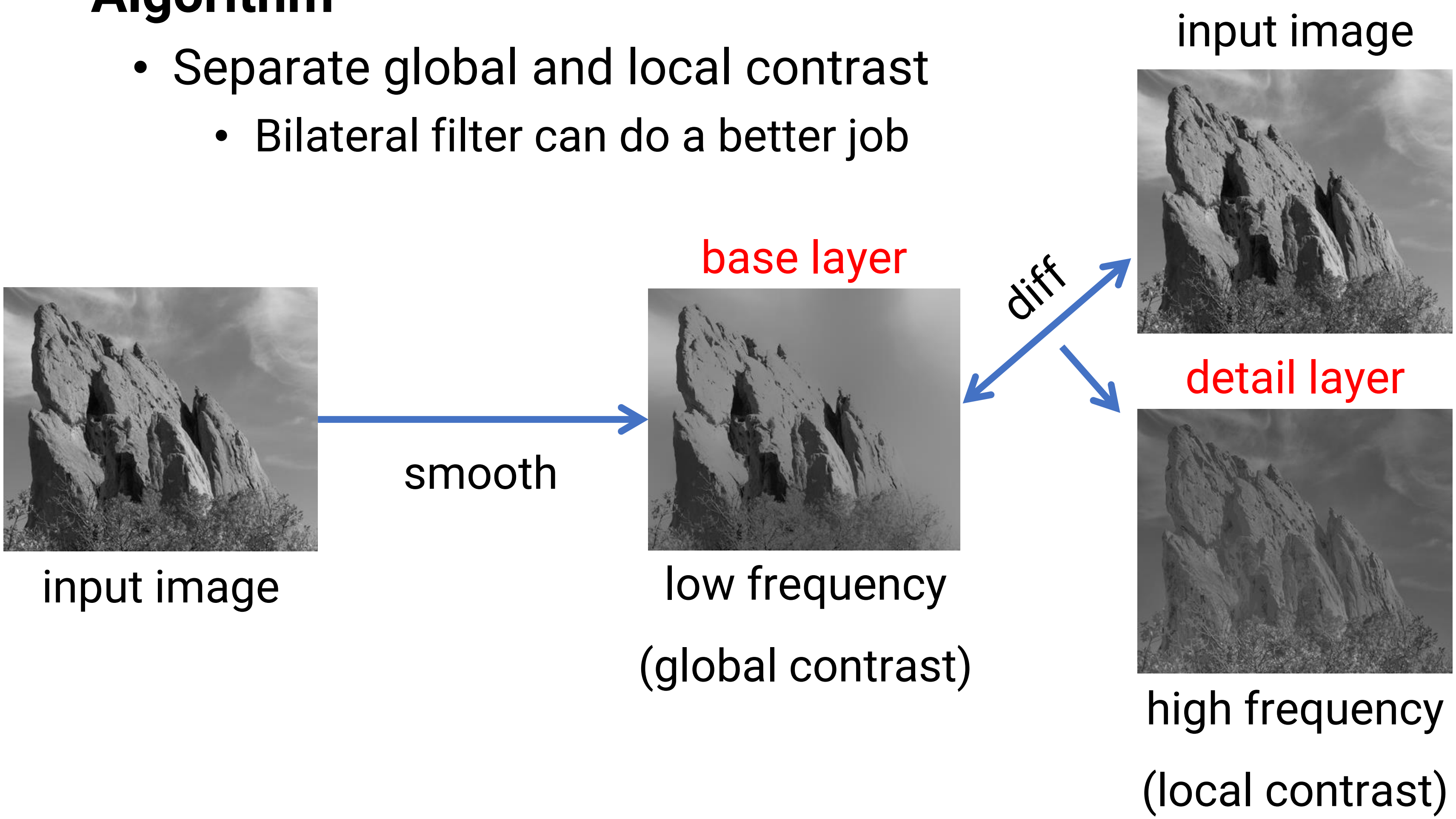
- **Algorithm**
  - Separate global and local contrast
    - Gaussian filter produces blurring and halos





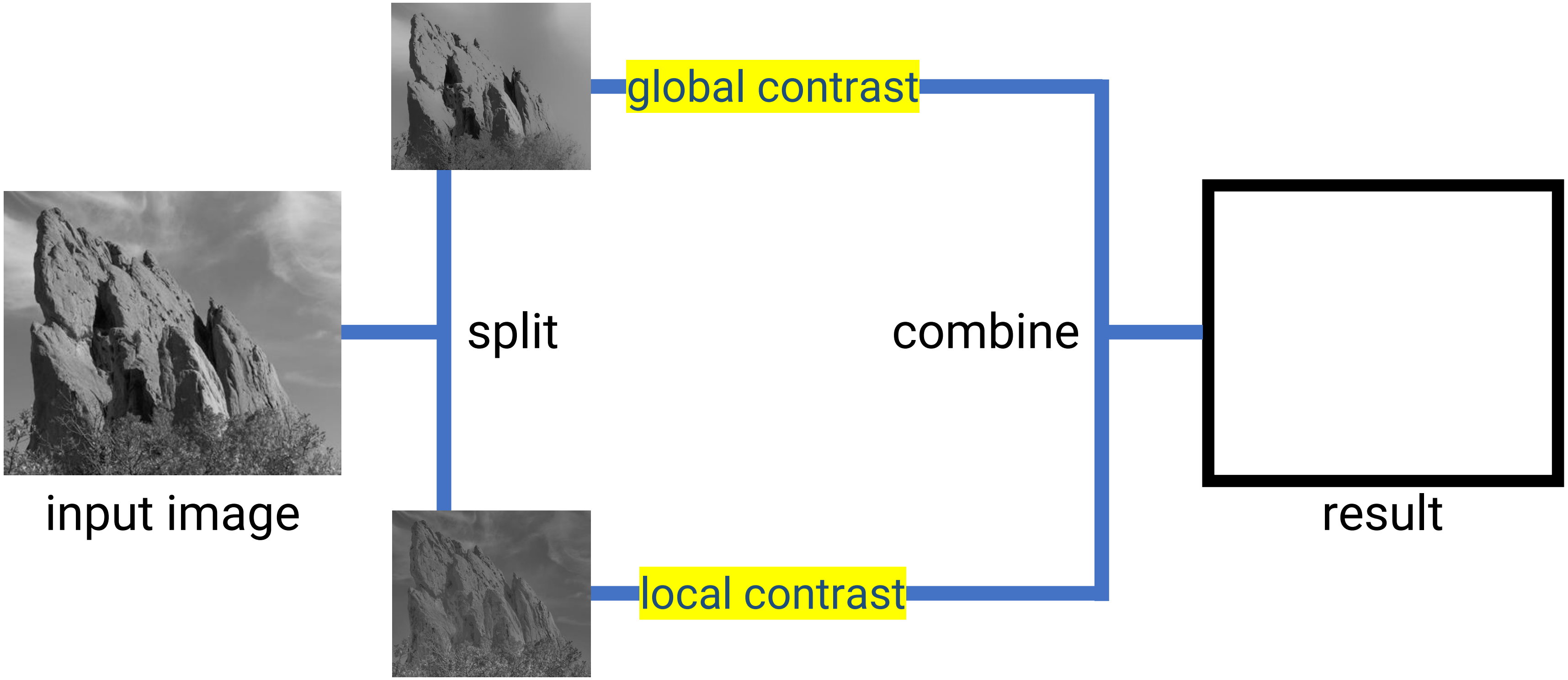
Photographic Style Transfer (cont.)

- **Algorithm**
  - Separate global and local contrast
    - Bilateral filter can do a better job



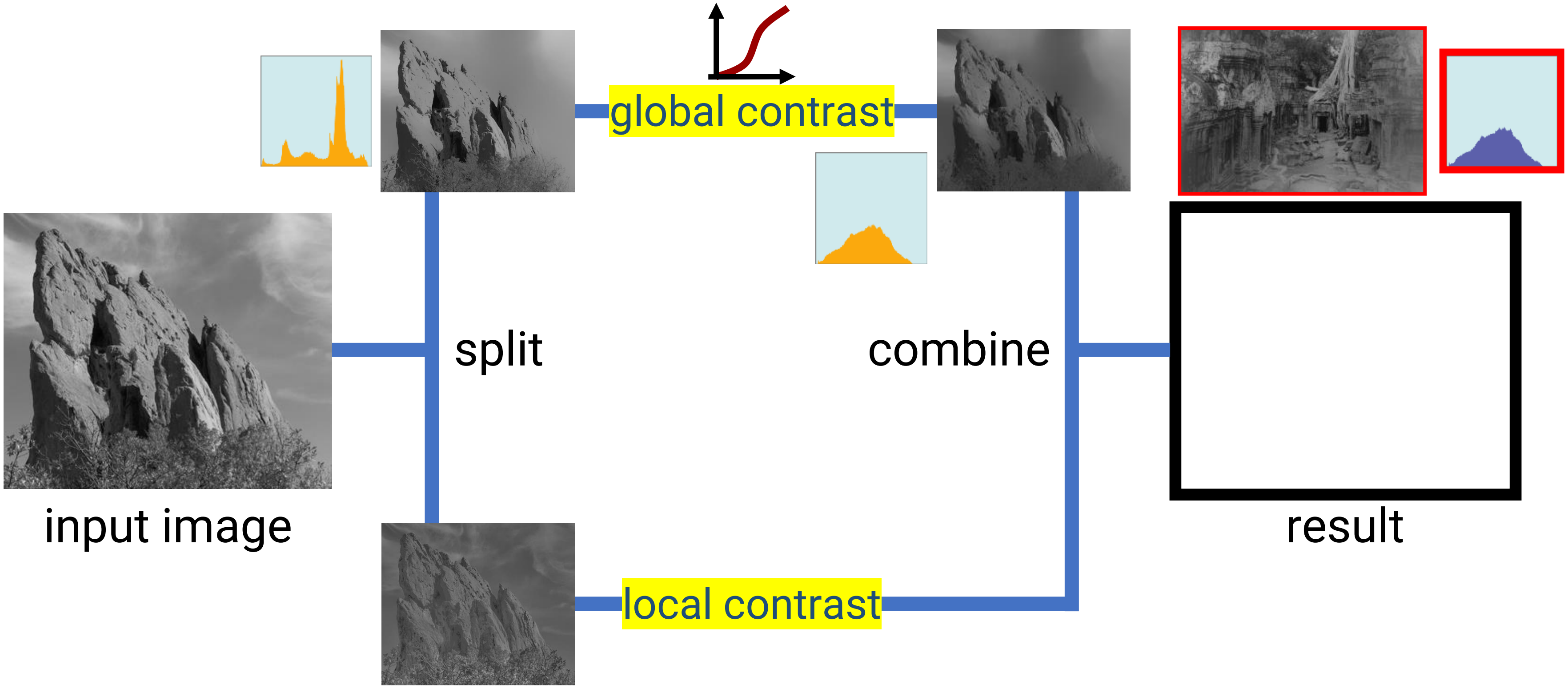
Photographic Style Transfer (cont.)

- **Algorithm**
  - Separate global and local contrast



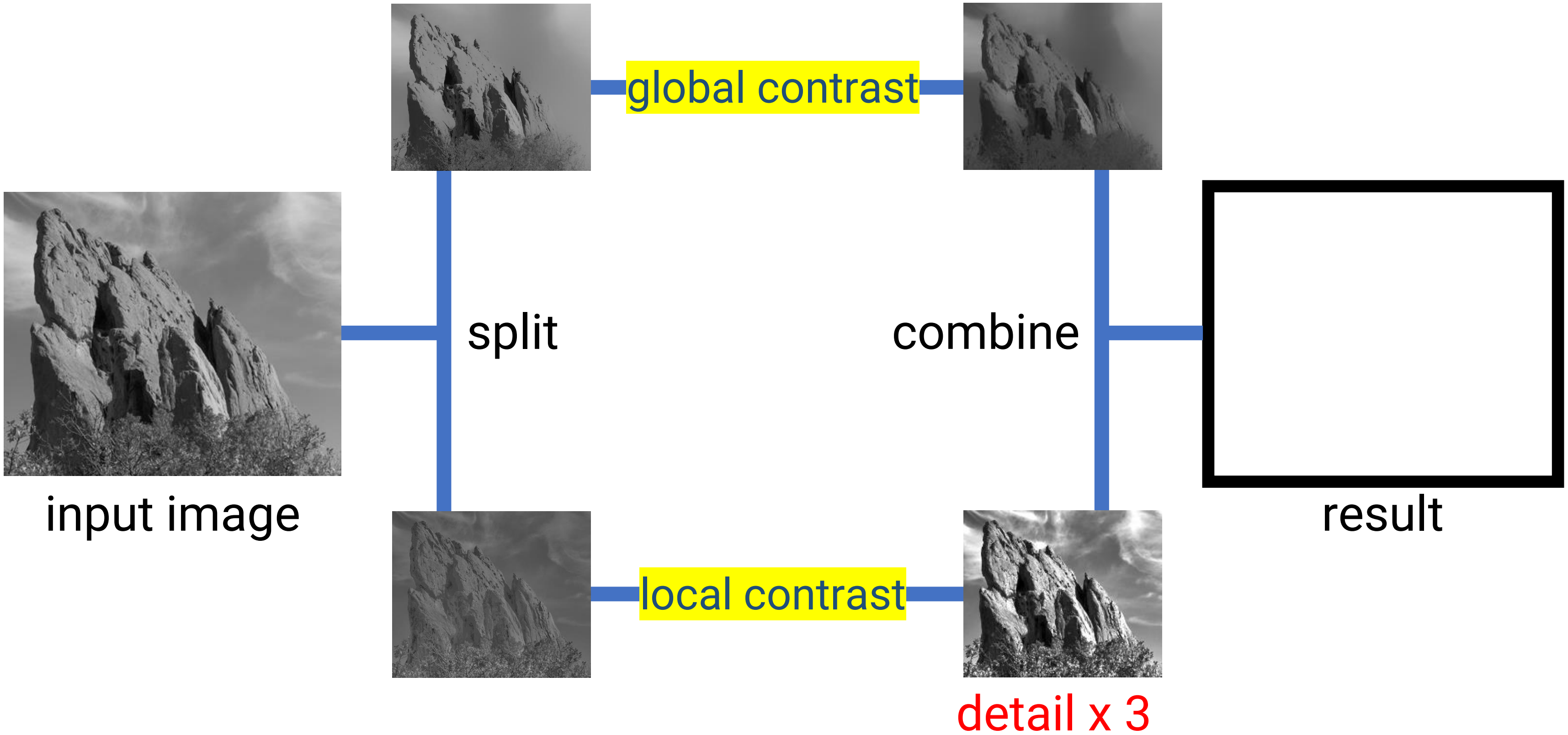
Photographic Style Transfer (cont.)

- **Algorithm**
  - Adjust global contrast by histogram matching



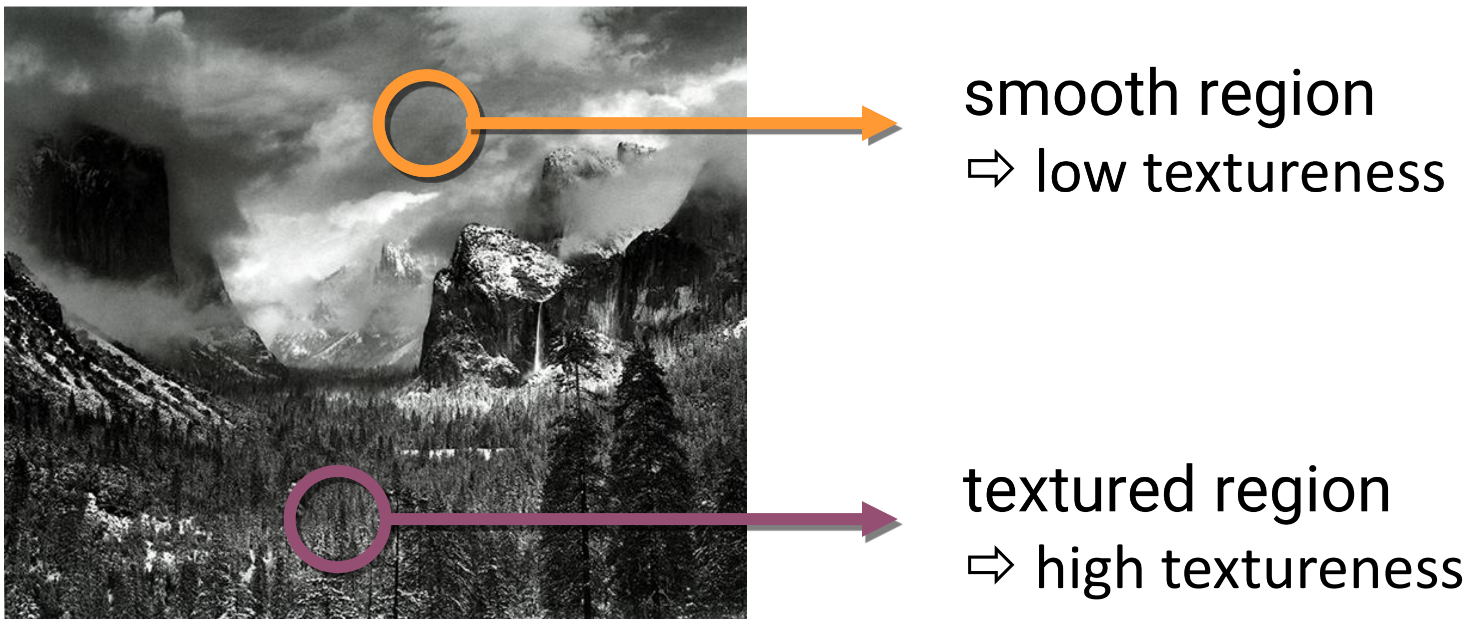
Photographic Style Transfer (cont.)

- **Algorithm**
  - Adjust local contrast by uniformly scaling



Photographic Style Transfer (cont.)

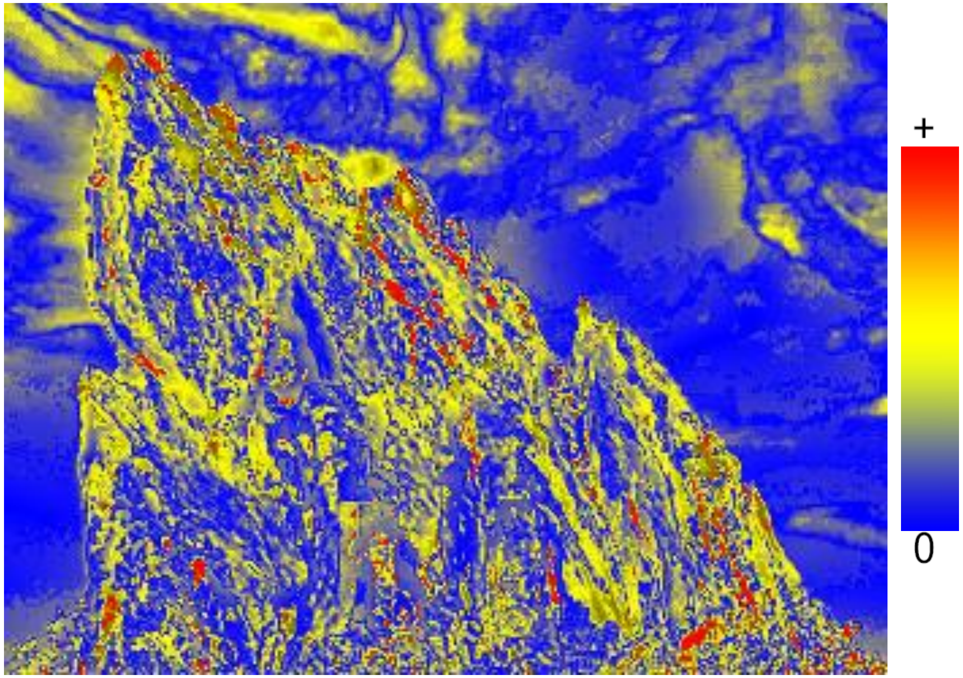
- **Algorithm**
  - Sometimes the local contrast is not uniform



Photographic Style Transfer (cont.)

- **Algorithm**
  - Textureness computation

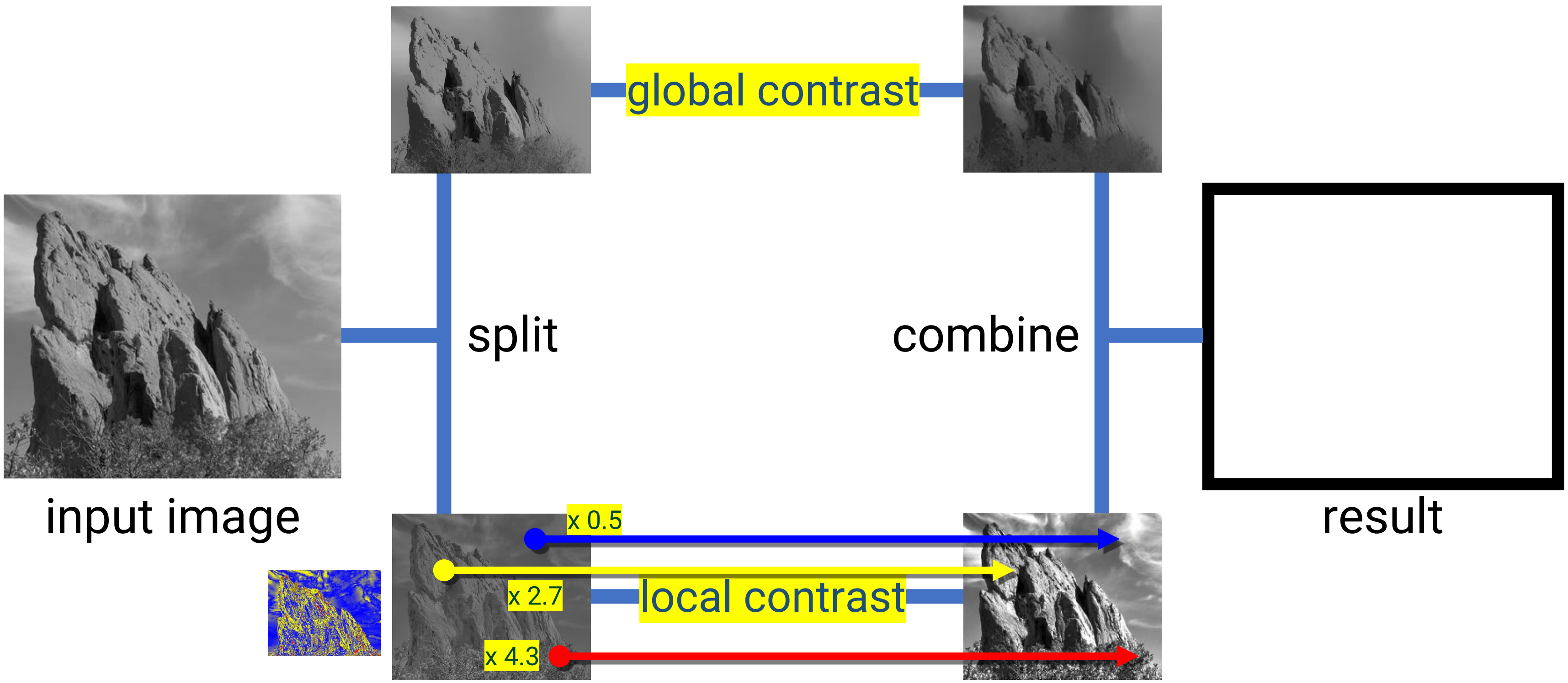
input



textureness

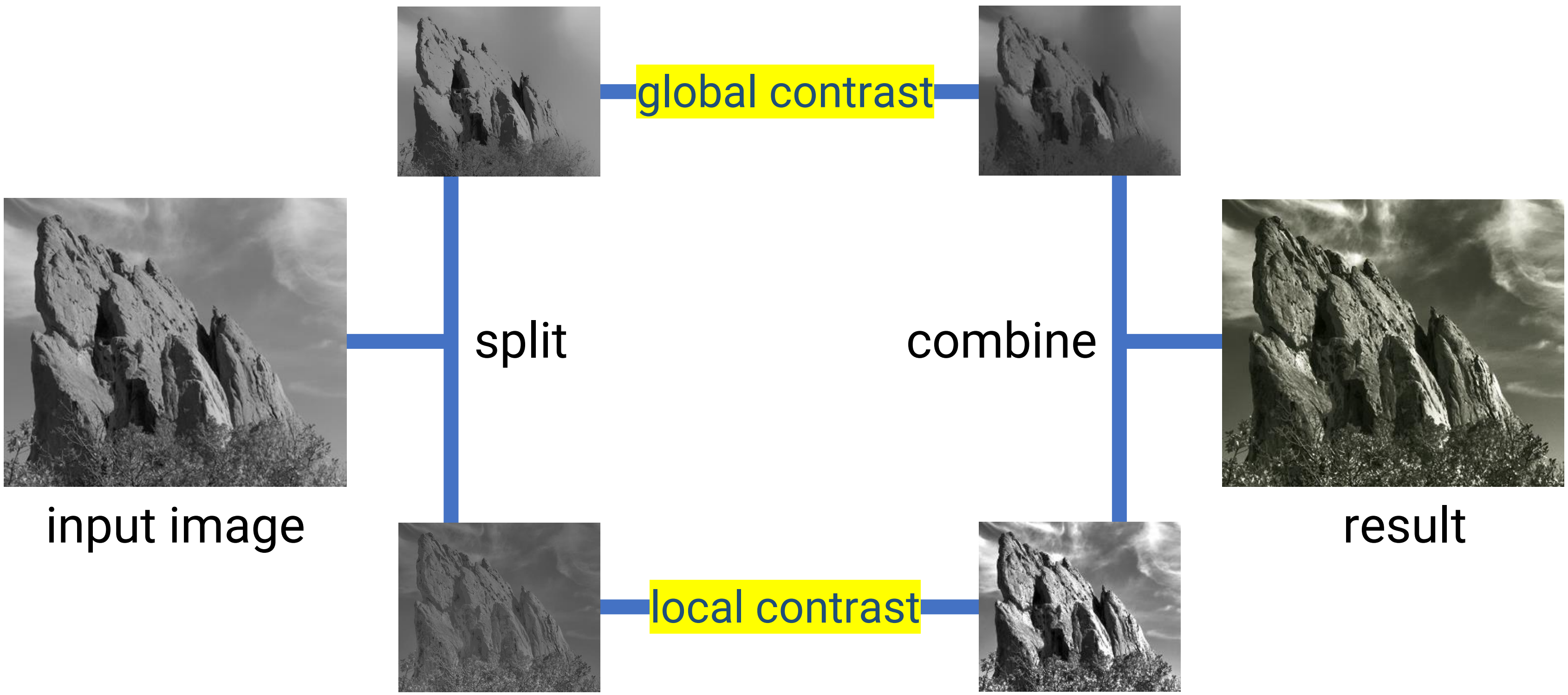
Photographic Style Transfer (cont.)

- **Algorithm**
  - Non-uniformly increase local contrast based on textureness



Photographic Style Transfer (cont.)

- **Algorithm**
  - Combine global and local contrast

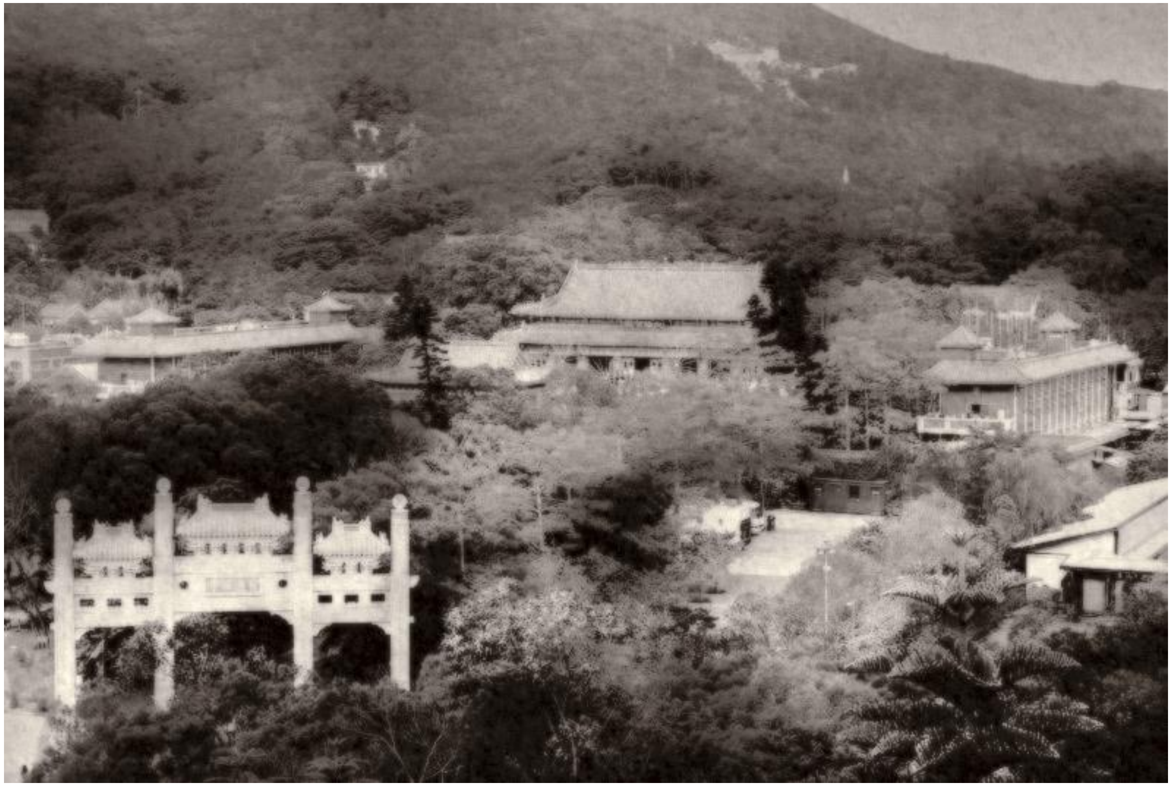


Photographic Style Transfer (cont.)

- **Results**

Photographic Style Transfer (cont.)

- **Results**

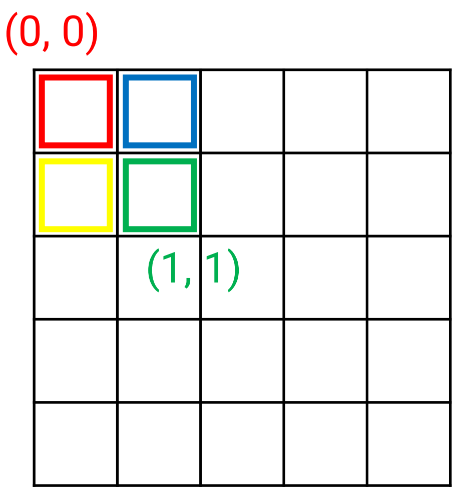


Outline

- Overview
- Image compression
- Image manipulation
- **Image scaling**

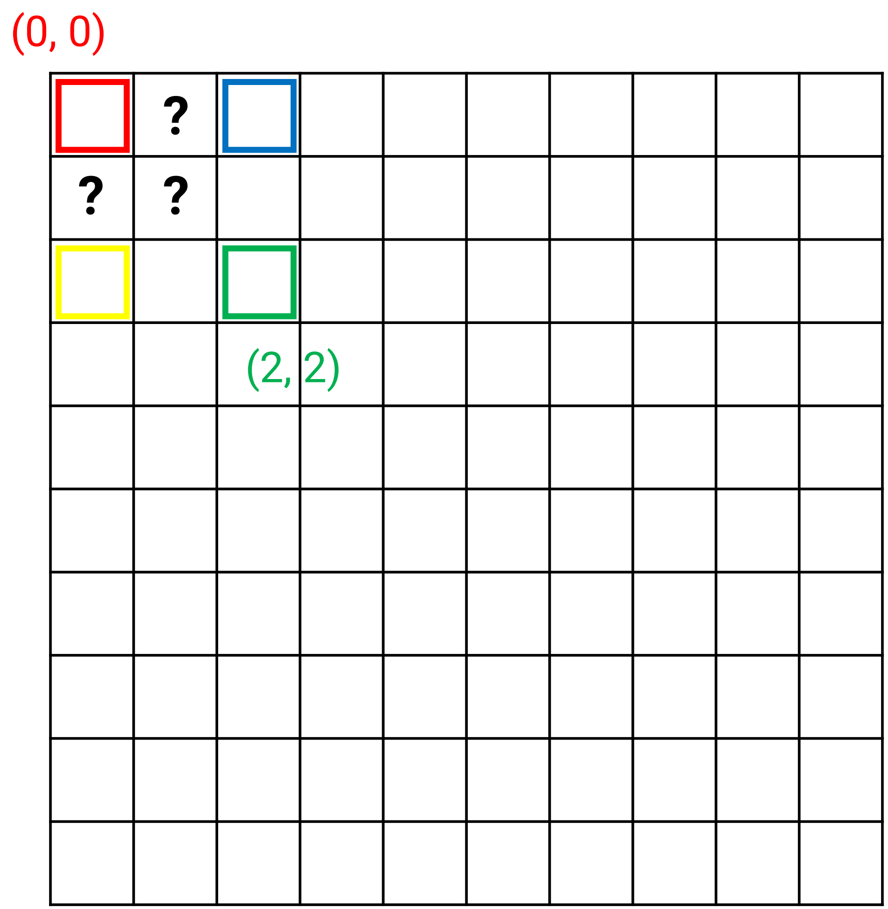
Forward Mapping

5 x 5



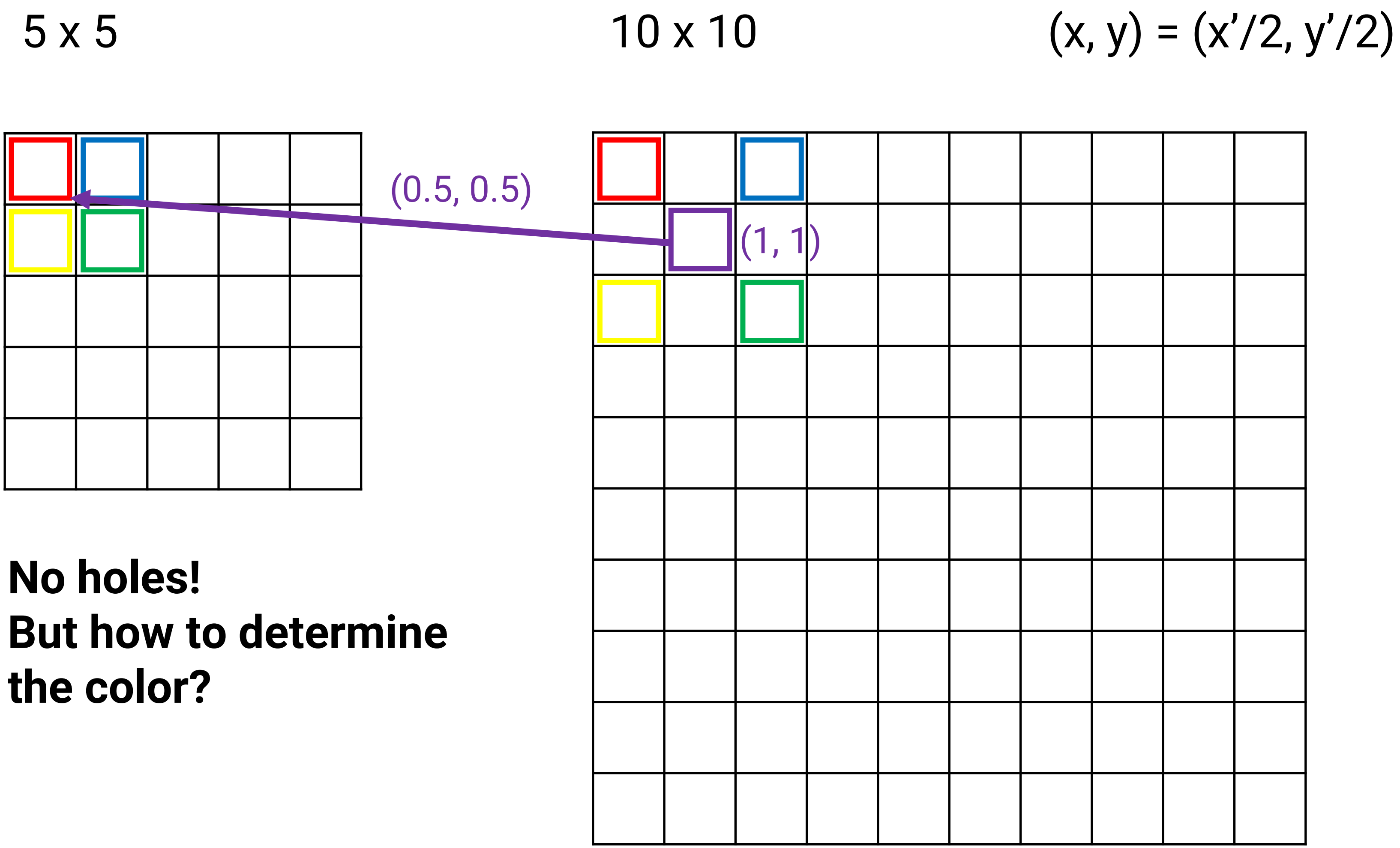
10 x 10

$(x', y') = (2x, 2y)$



Holes!

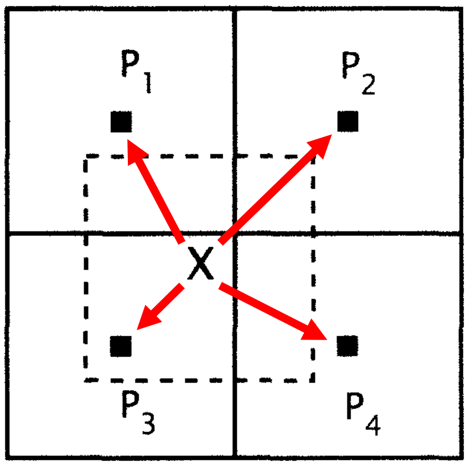
Inverse Mapping



No holes!
But how to determine the color?

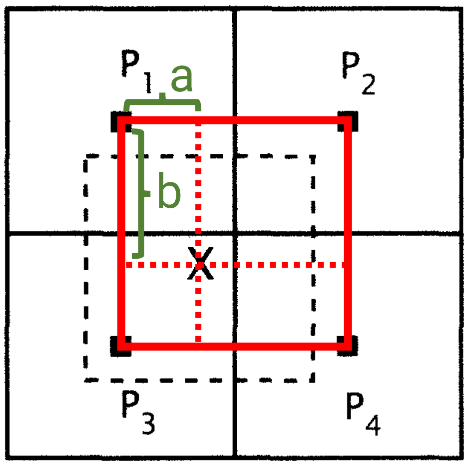
Image Scaling (cont.)

- Three strategies to obtain an estimation of X



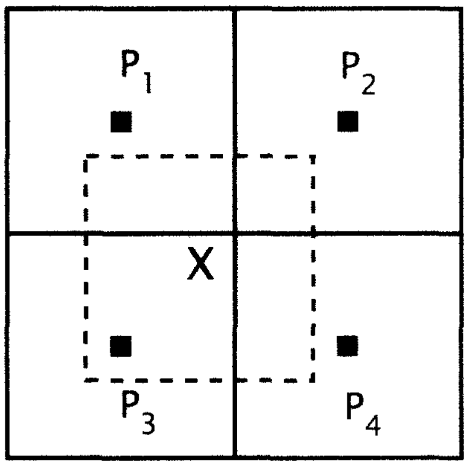
nearest neighbor

P_3 is closest
Use P_3's pixel value



bilinear interpolation

$(1\text{-}a)(1\text{-}b)P_1 + (a)(1\text{-}b)P_2 + (1\text{-}a)(b)P_3 + (a)(b)P_4$



bicubic interpolation

using curves to compute the weight (nonlinear)

Image Scaling (cont.)

- Three strategies to obtain an estimation of X

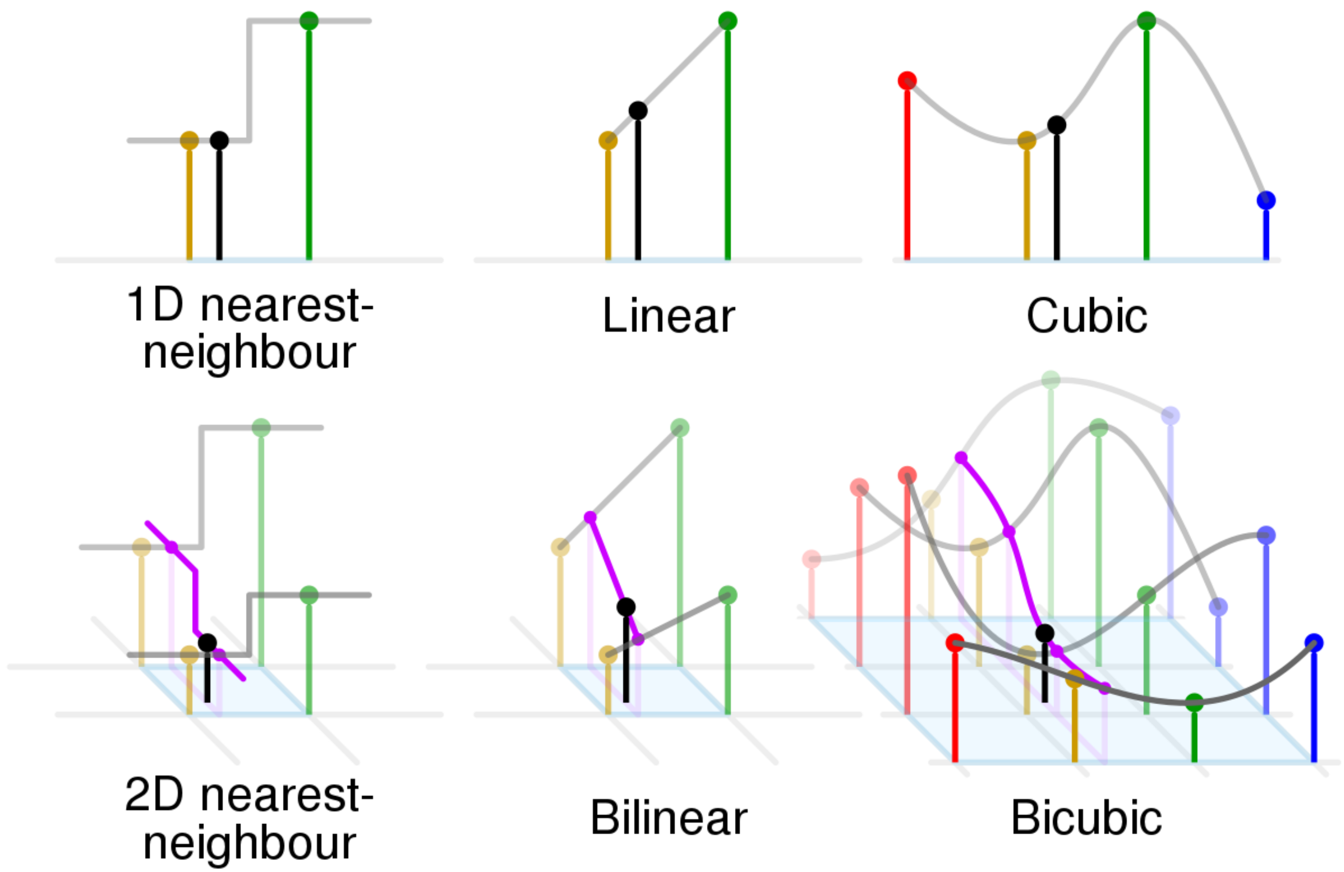
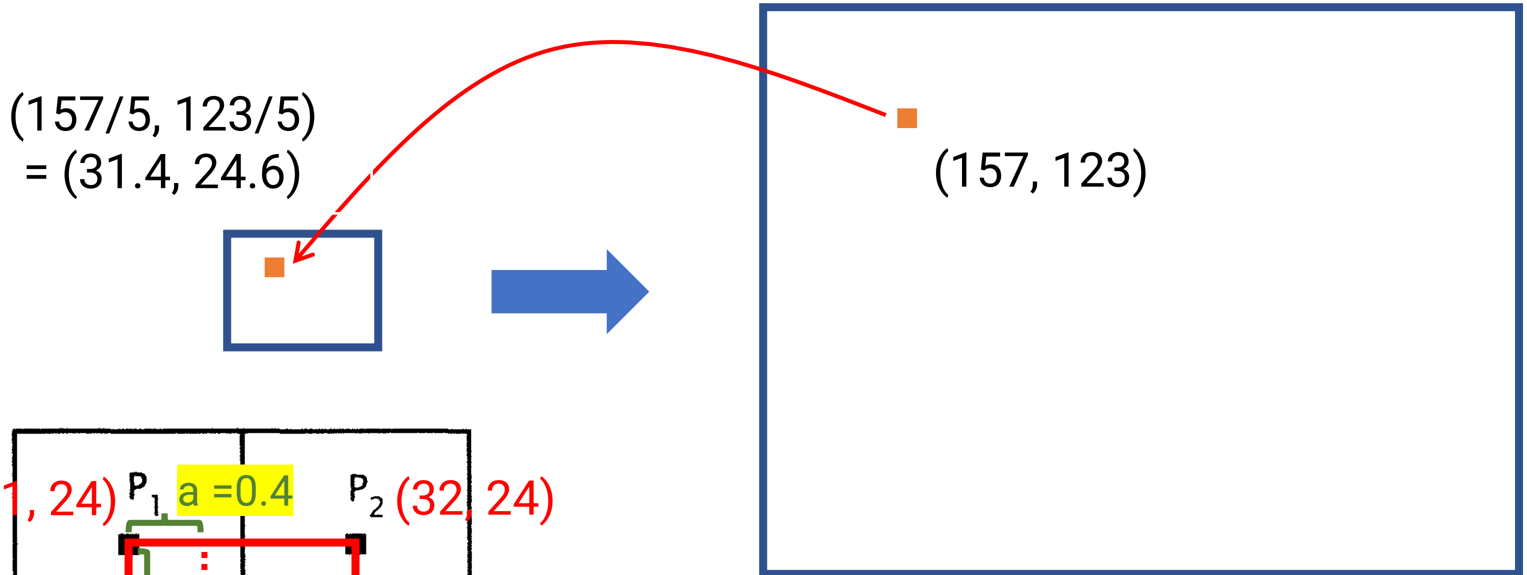
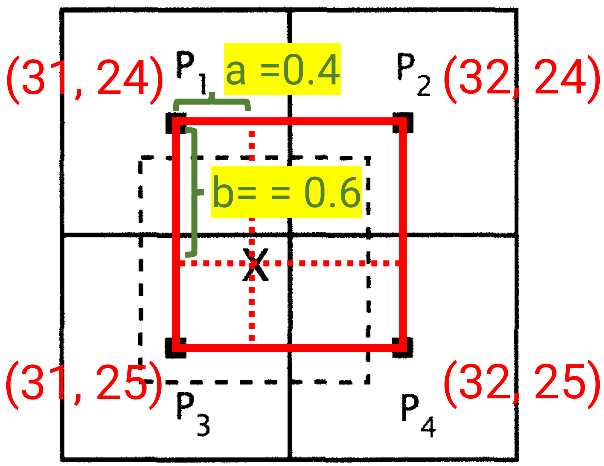


Image Scaling (cont.)

- Example: scale an image from 160 x 120 to 800 x 600





nearest neighbor: use color of (31,25)

bilinear: compute

$$(1\text{-}a)(1\text{-}b)P_1 + (a)(1\text{-}b)P_2 + (1\text{-}a)(b)P_3 + (a)(b)P_4$$

0.24 0.16 0.36 0.24

Image Scaling (cont.)

- Example

nearest neighbor

bilinear interpolation

bicubic interpolation

Image Scaling (cont.)

- Example

nearest neighbor

bilinear interpolation

bicubic interpolation

original  × 10

Image Scaling (cont.)

- Example

original

nearest neighbor

bilinear interpolation

bicubic interpolation


END OF PRESENTATION
ANY QUESTIONS?