Advanced Graphics & Image Processing

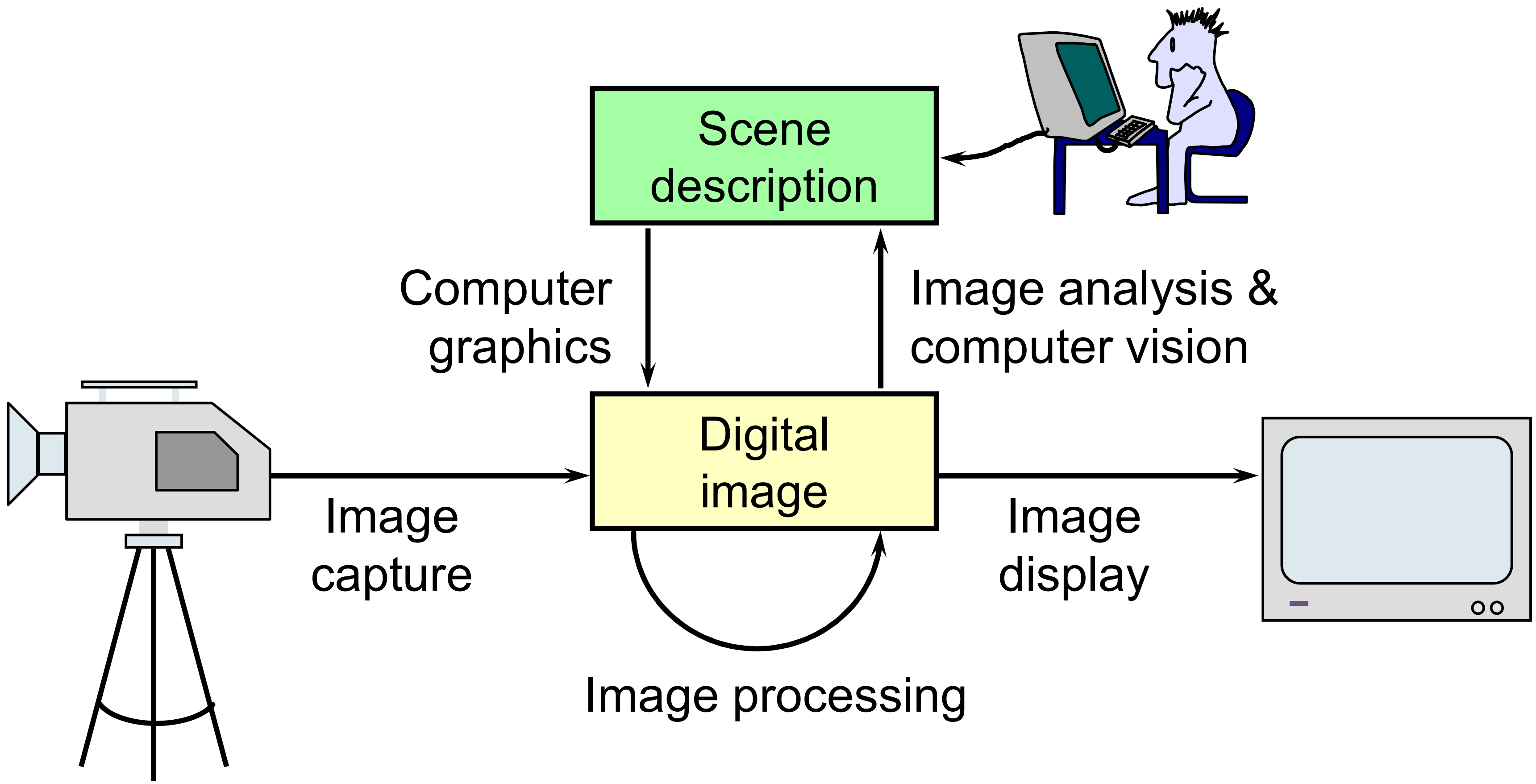
Introduction to Image Processing

Part 1/2 – Images, pixels and sampling

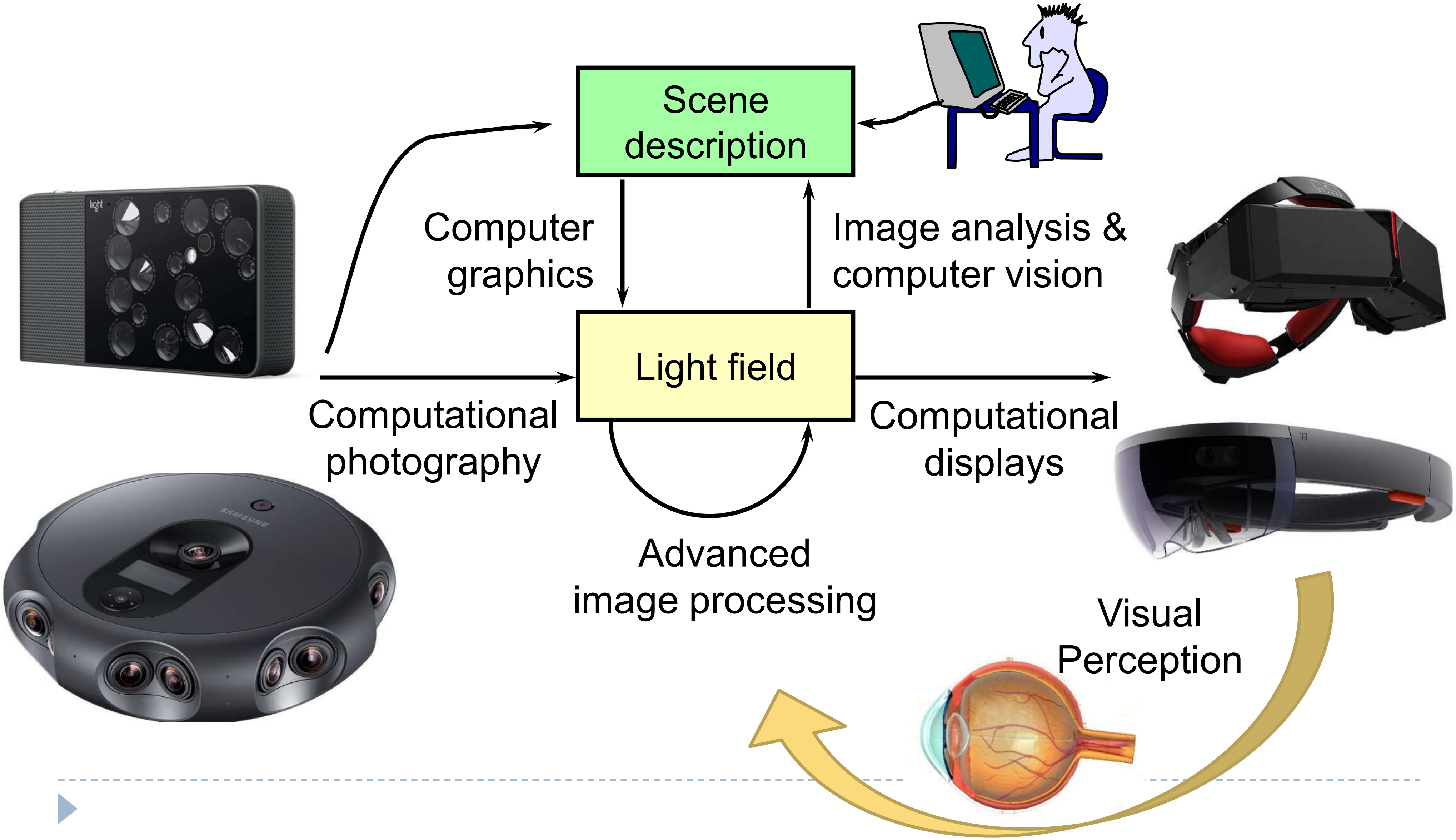
Rafał Mantiuk

Computer Laboratory, University of Cambridge

What are Computer Graphics & Image Processing?



Where are graphics and image processing heading?



What is a (computer) image?

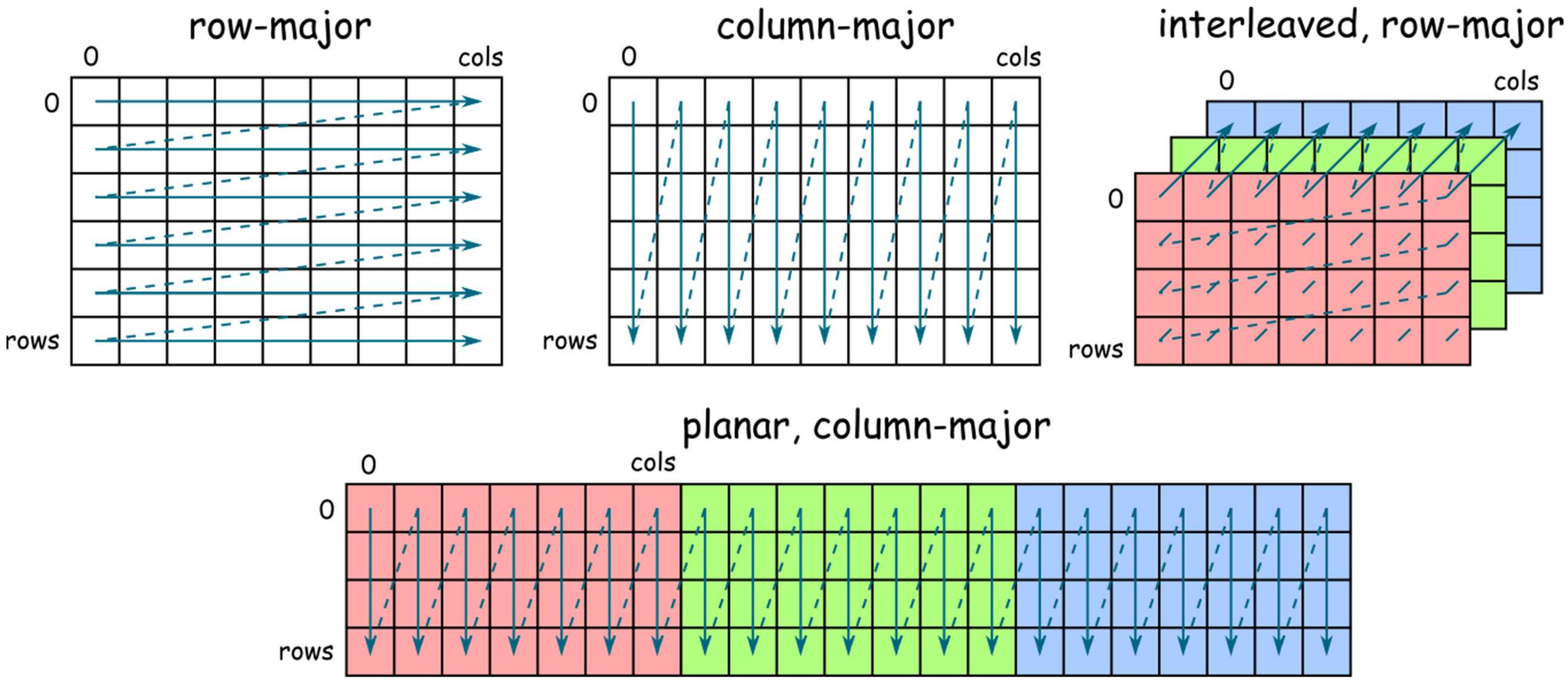
- A digital photograph? ("JPEG")
- A snapshot of real-world lighting?

Image

From computing perspective (discrete)	From mathematical perspective (continuous)
2D array of pixels	2D function
•To represent images in memory	•To express image processing as a mathematical problem
•To create image processing software	•To develop (and understand) algorithms

Image

- 2D array of pixels
- In most cases, each pixel takes 3 bytes: one for each red, green and blue
- But how to store a 2D array in memory?



Stride

- Calculating the pixel component index in memory
  - For row-major order (grayscale)

$$i(x, y) = x + y \cdot n_{cols}$$

  - For column-major order (grayscale)

$$i(x, y) = x \cdot n_{rows} + y$$

  - For interleaved row-major (colour)

$$i(x, y, c) = x \cdot 3 + y \cdot 3 \cdot n_{cols} + c$$

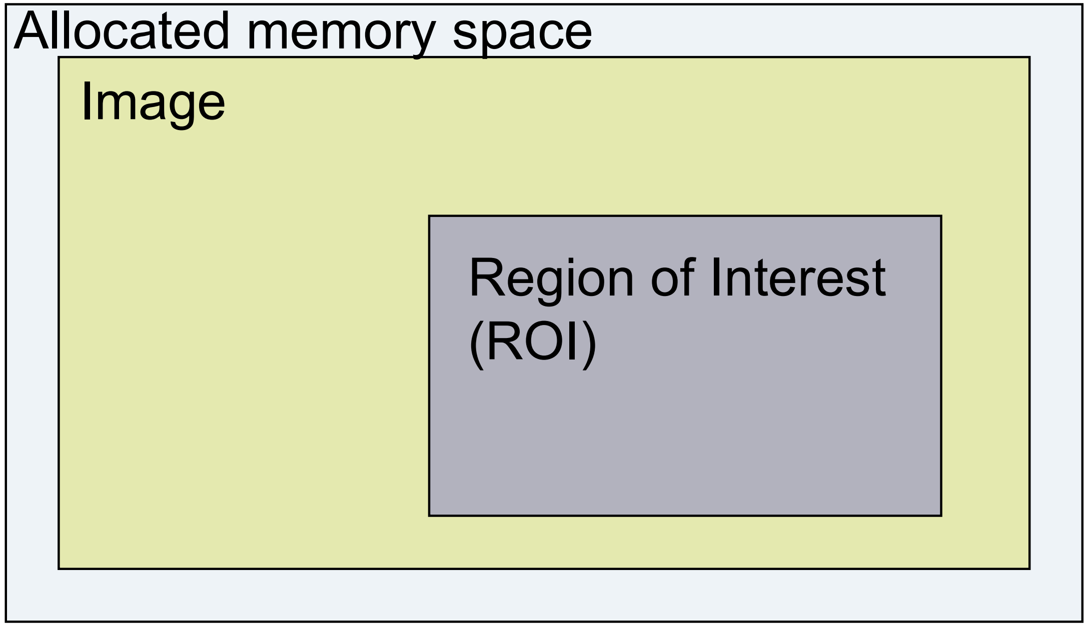
  - General case

$$i(x, y, c) = x \cdot s_x + y \cdot s_y + c \cdot s_c$$

where s_x, s_y and s_c are the strides for the x, y and colour dimensions

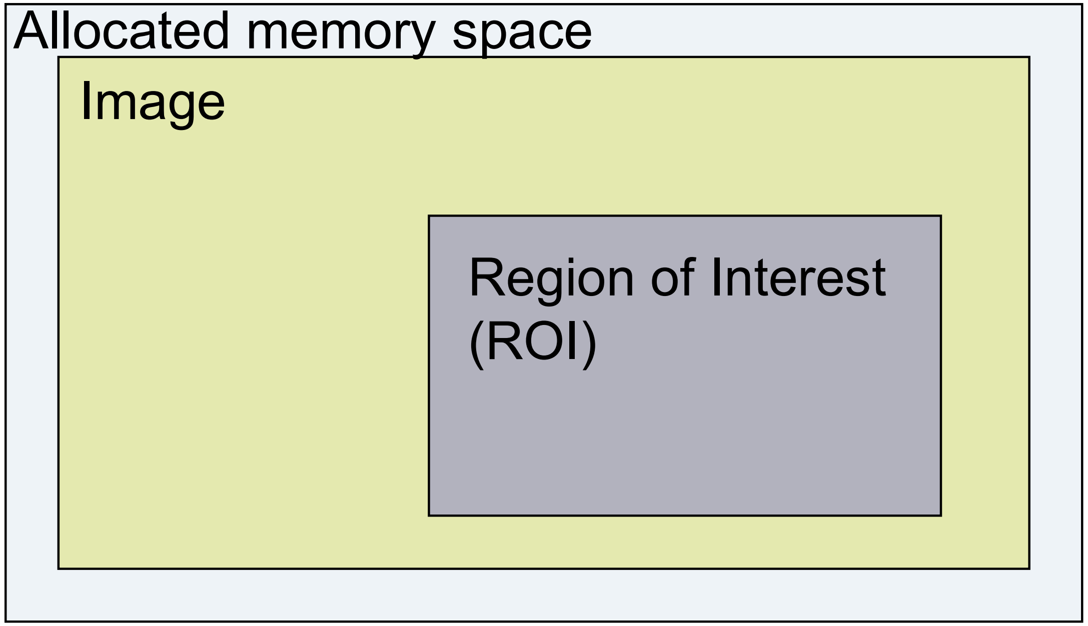
Padded images and stride

- Sometimes it is desirable to "pad" image with extra pixels
  - for example when using operators that need to access pixels outside the image border
- Or to define a region of interest (ROI)



- How to address pixels for such an image and the ROI?

Padded images and stride



$$i(x, y, c) = i_{first} + x \cdot s_x + y \cdot s_y + c \cdot s_c$$

- For row-major, interleaved
  - $s_x = ?$
  - $s_y = ?$
  - $s_c = ?$

Pixel (PIcture ELement)

- Each pixel (usually) consist of three values describing the color

(red, green, blue)

- For example
  - (255, 255, 255) for white
  - (0, 0, 0) for black
  - (255, 0, 0) for red
- Why are the values in the 0-255 range?
- Why red, green and blue? (and not cyan, magenta, yellow)
- How many bytes are needed to store 5MPixel image? (uncompressed)

Pixel formats, bits per pixel, bit-depth

- Grayscale – single **color channel**, 8 bits (1 byte)
- Highcolor – 2^{16}=65,536 colors (2 bytes)

Sample Length:	5					6						5				
Channel Membership:	**Red**					**Green**						**Blue**				
Bit Number:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RGBAX	**R. G. B. A. X**															
Sample Length Notation:	**5.6.5.0.0**															

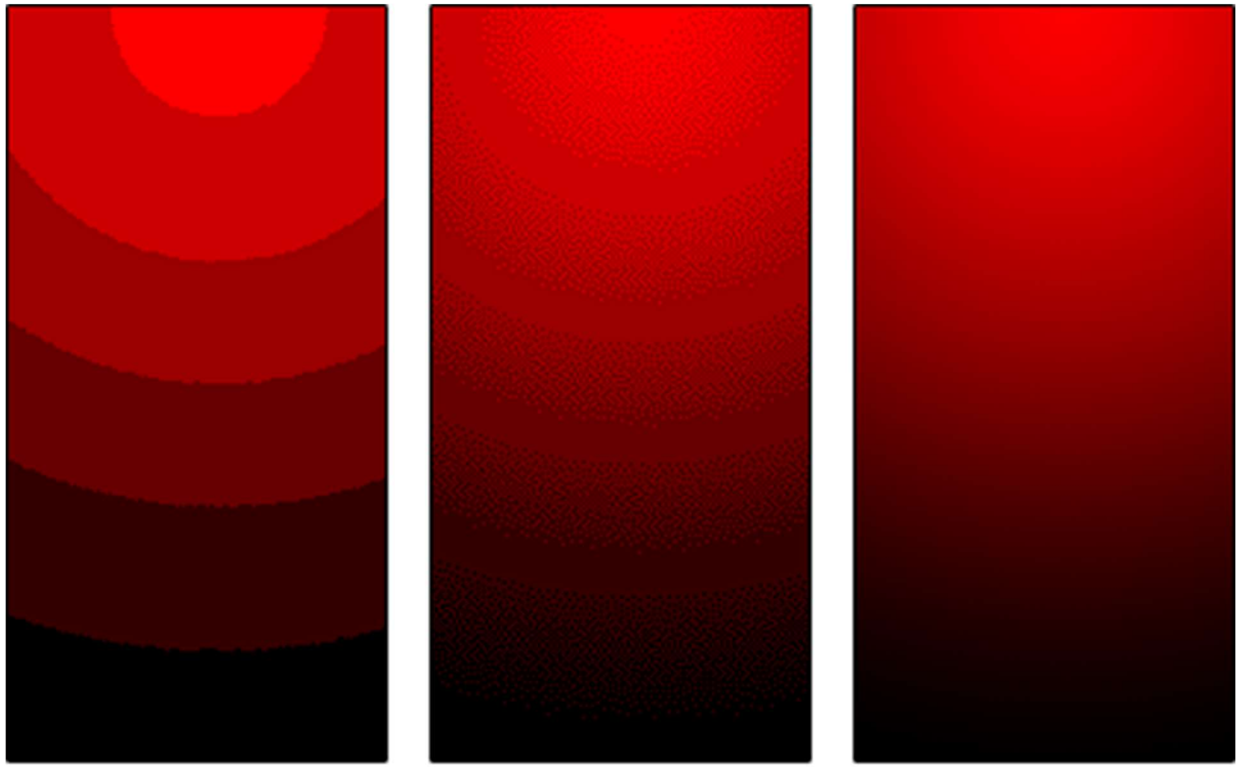
- Truecolor – 2^{24} = 16,8 million colors (3 bytes)
- Deepcolor – even more colors (>= 4 bytes)

Sample Length:	2		10										10										10									
Channel Membership:	**None**		**Red**										**Green**										**Blue**									
Bit Number:	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RGBAX	**R . G . B . A . X**																															
Sample Length Notation:	**10.10.10.0.2**																															

- But why?

Color banding

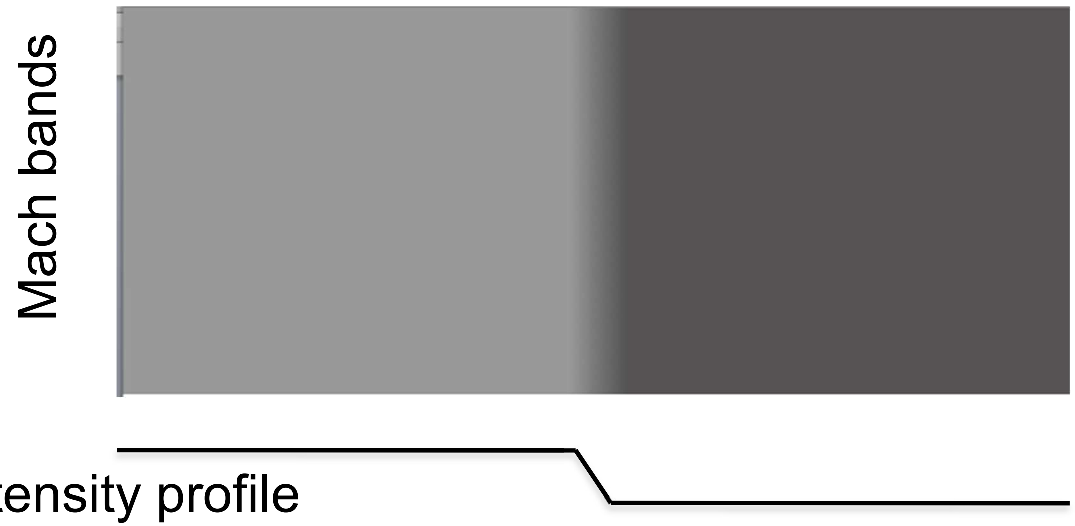
- If there are not enough bits to represent color
- Looks worse because of the **Mach band** illusion
- Dithering (added noise) can reduce banding
  - Printers
  - Many LCD displays do it too



8-bit gradient

8-bit gradient, dithered

24-bit gradient



What is a (computer) image?

- A digital photograph? ("JPEG")
- A snapshot of real-world lighting?

Image

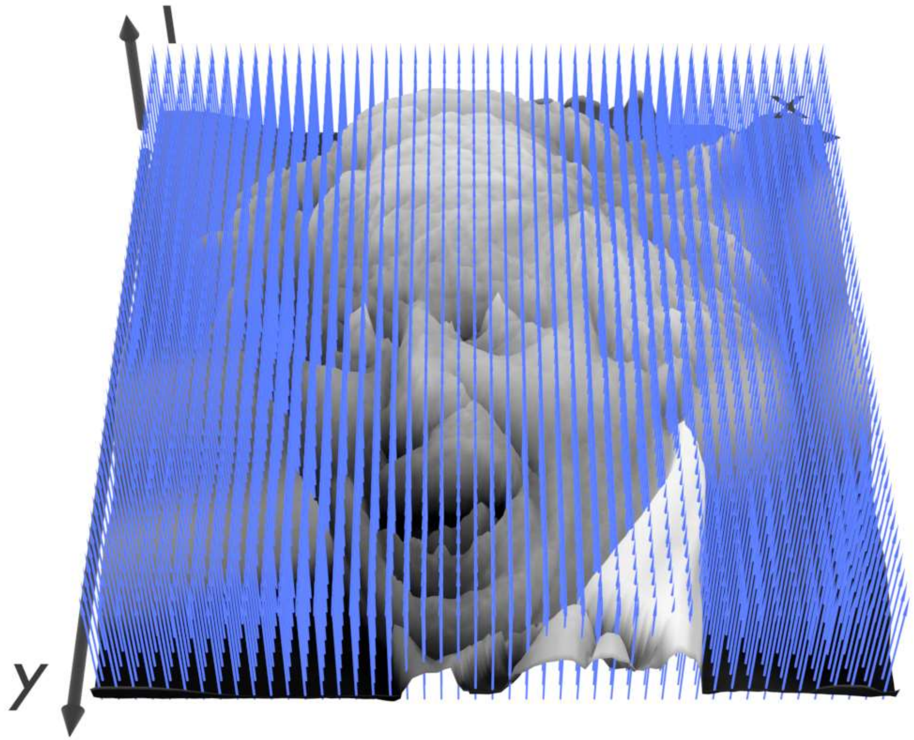
From computing perspective (discrete)	From mathematical perspective (continuous)
2D array of pixels	2D function
•To represent images in memory	•To express image processing as a mathematical problem
•To create image processing software	•To develop (and understand) algorithms

Image – 2D function

- Image can be seen as a function I(x,y), that gives intensity value for any given coordinate (x,y)

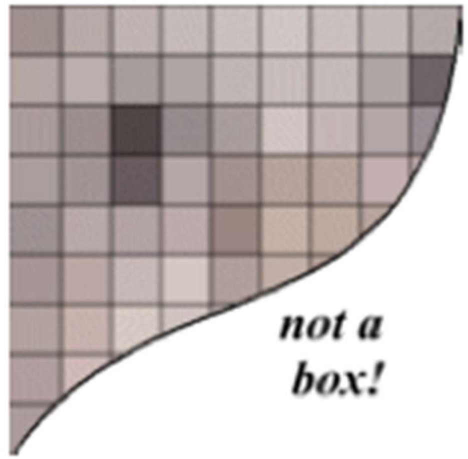
Sampling an image

- The image can be sampled on a rectangular sampling grid to yield a set of samples. These samples are pixels.

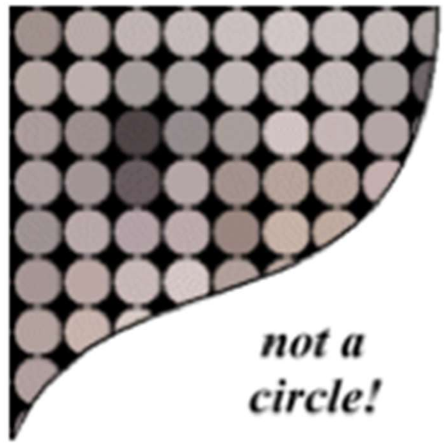


What is a pixel?

- A pixel is not
  - a box
  - a disk
  - a teeny light



- A pixel is a point
  - it has no dimension
  - it occupies no area
  - it cannot be seen
  - it has coordinates

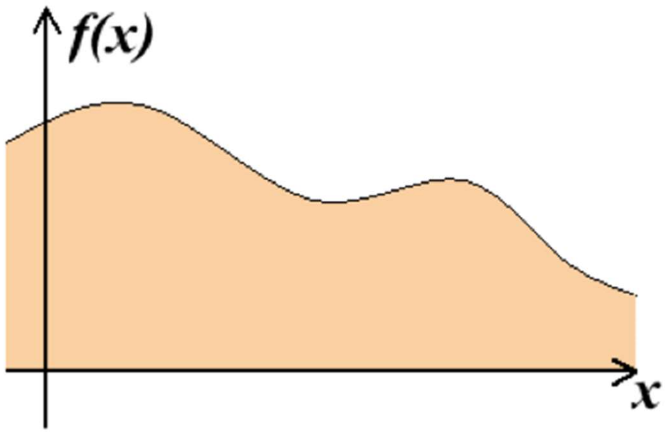


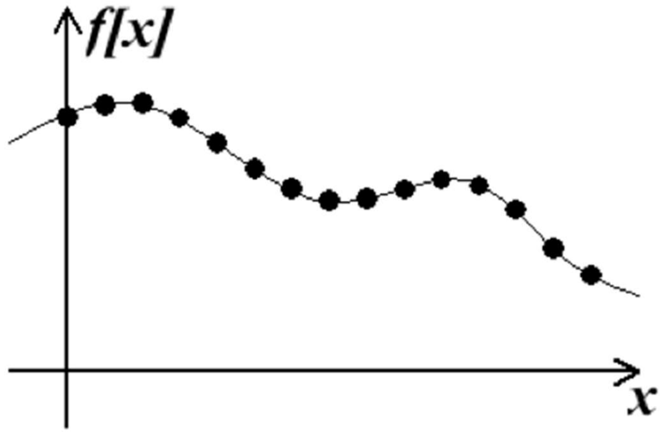
- A pixel is a **sample**

From: http://groups.csail.mit.edu/graphics/classes/6.837/F01/Lecture05/lecture05.pdf

Sampling and quantization

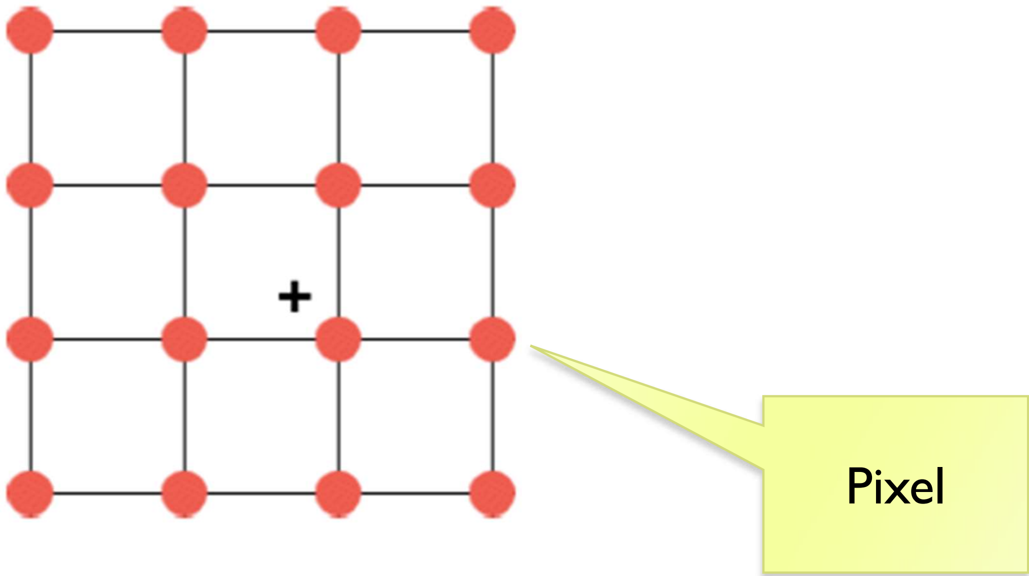
- The physical world is described in terms of continuous quantities
- But computers work only with discrete numbers
- Sampling – process of mapping continuous function to a discrete one
- Quantization – process of mapping continuous variable to a discrete one





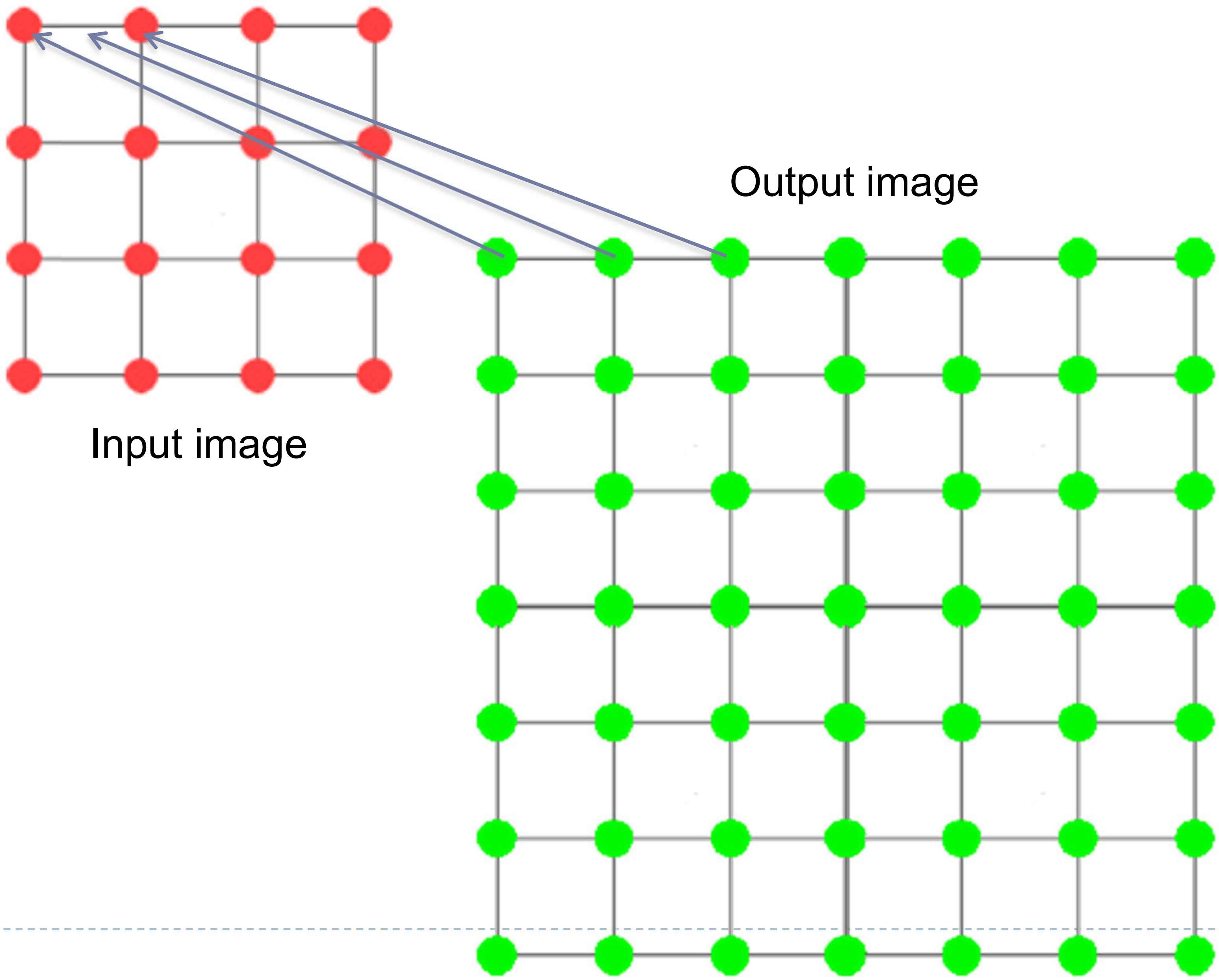
Resampling

- Some image processing operations require to know the colors that are in-between the original pixels

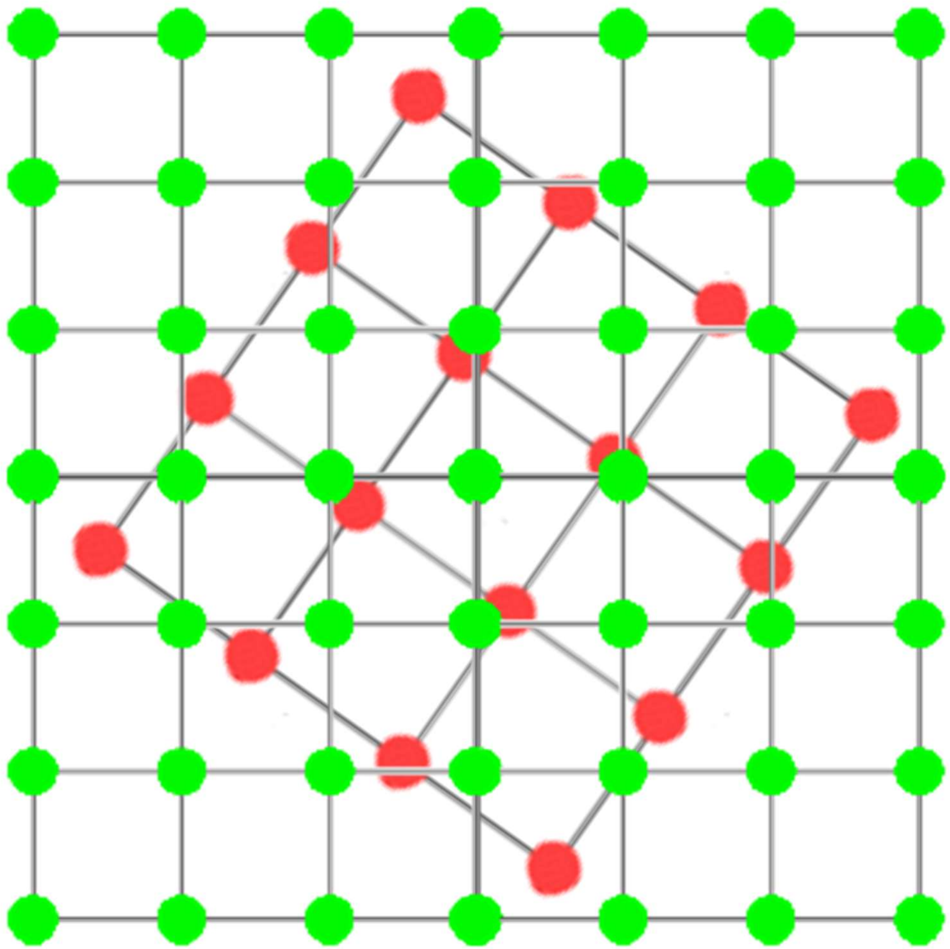


- What are those operations?
- How to find these *resampled* pixel values?

Example of resampling: magnification

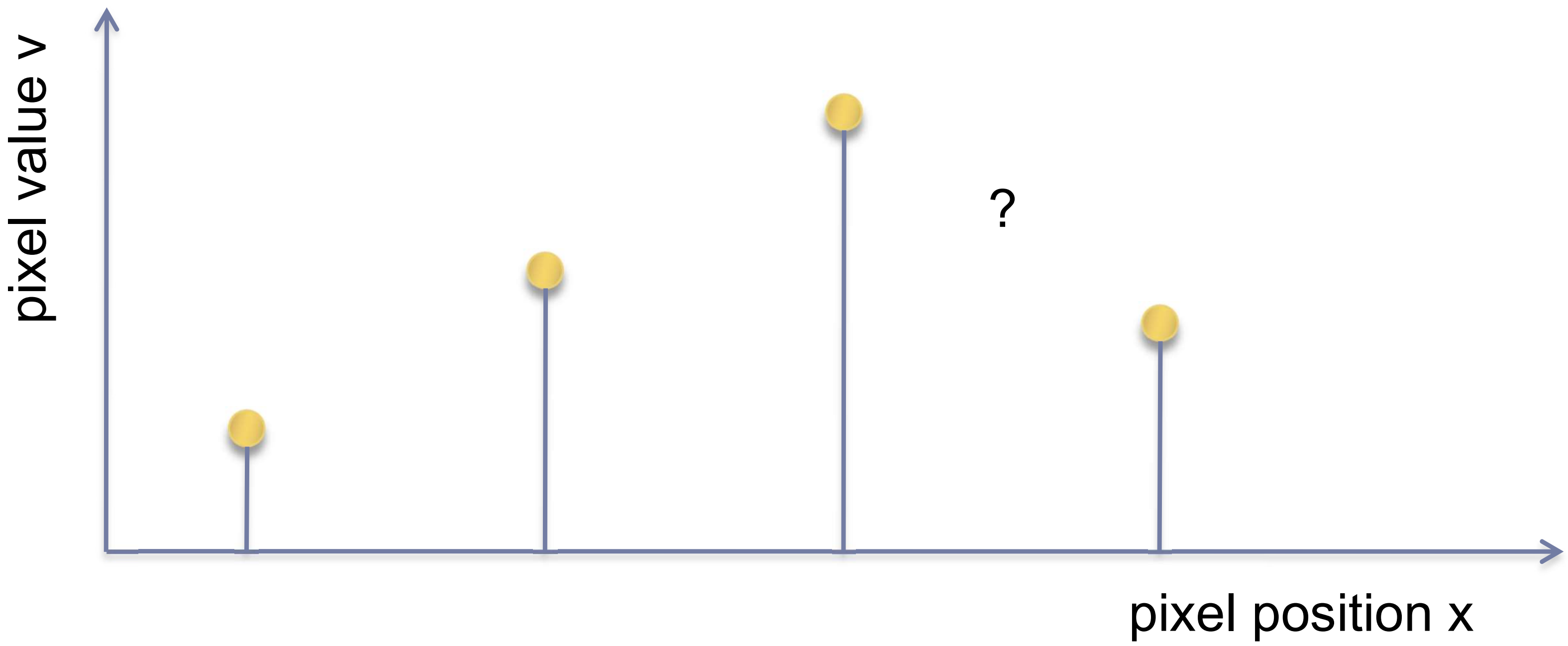


Example of resampling: scaling and rotation



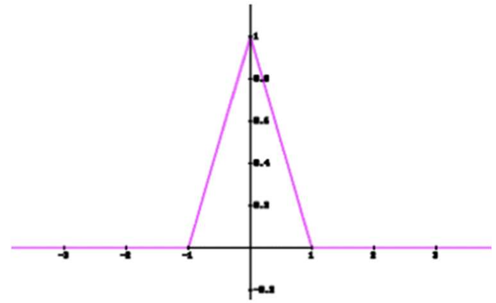
How to resample?

- In 1D: how to find the most likely resampled pixel value knowing its two neighbors?

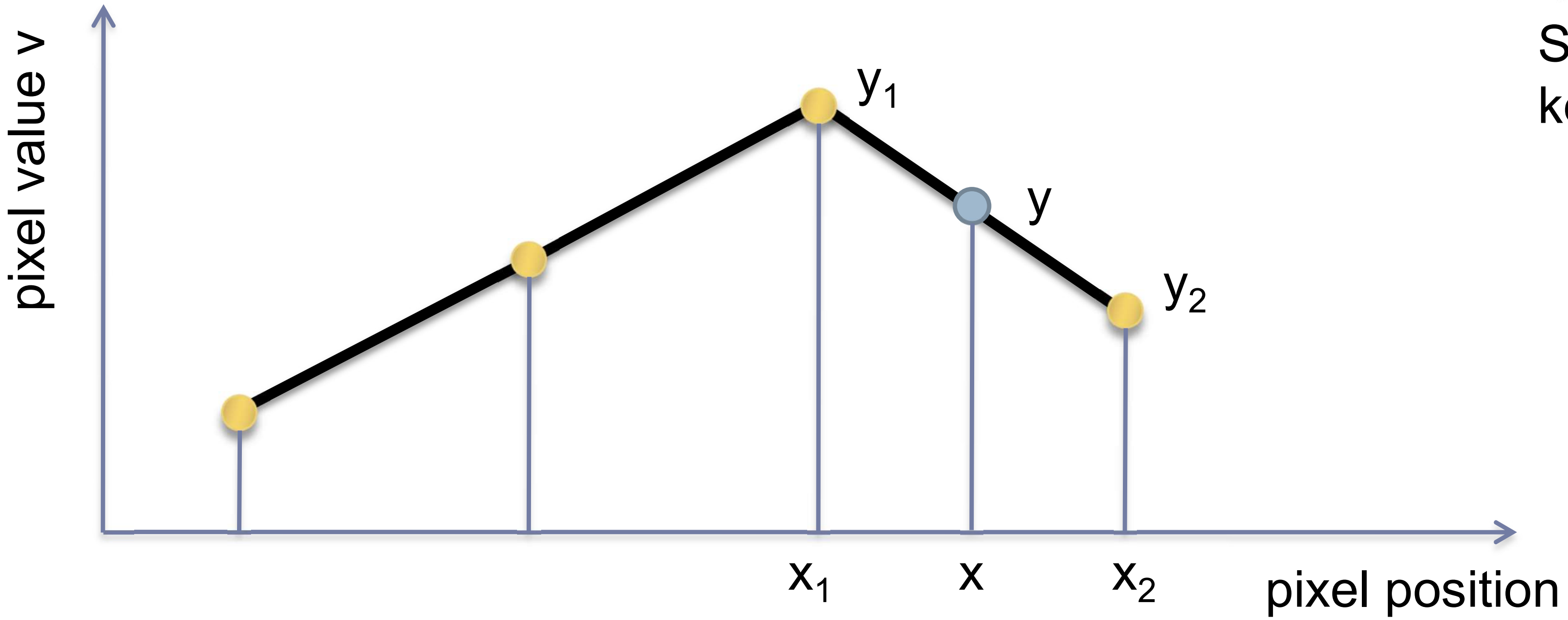


(Bi)Linear interpolation (resampling)

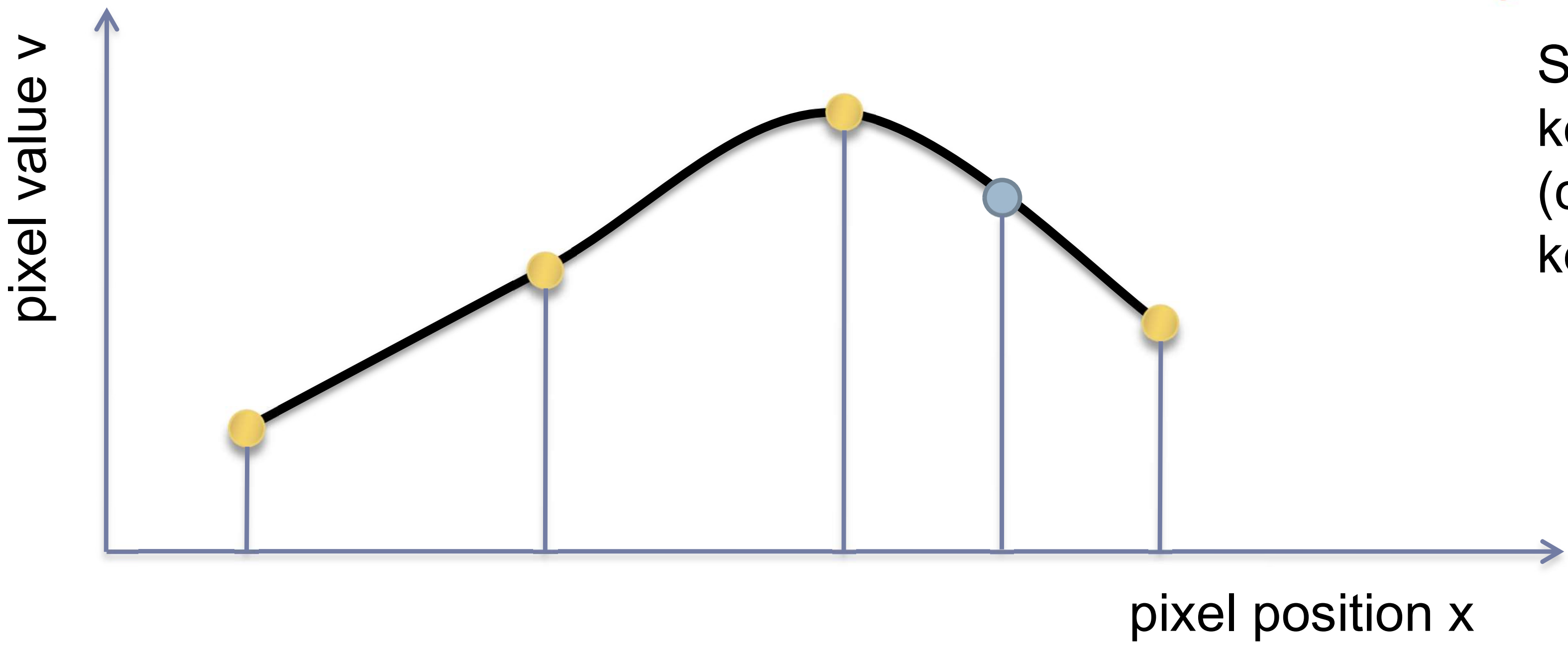
- Linear – 1D
- Bilinear – 2D

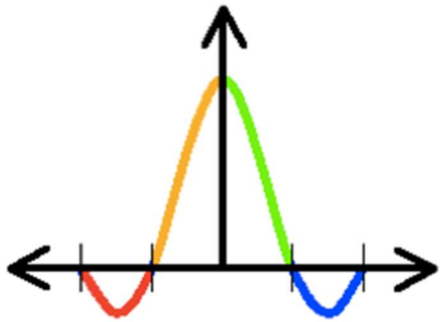


Sampling kernel



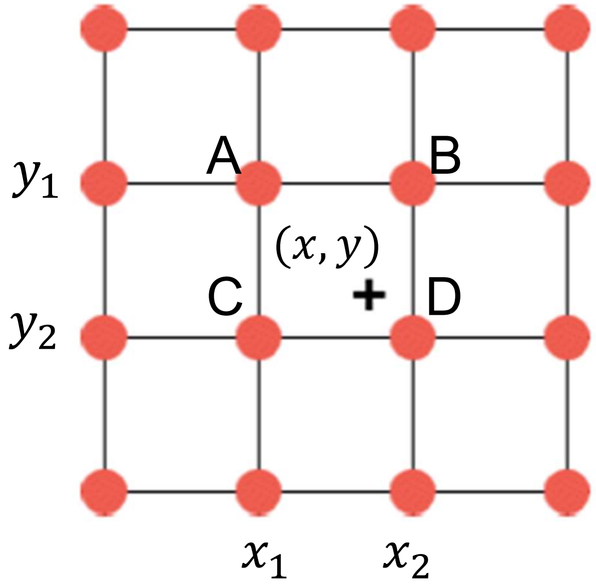
(Bi)cubic interpolation (resampling)





Sampling kernel (convolution kernel)

Bi-linear interpolation



Given the pixel values:

$$I(x_1, y_1) = A$$

$$I(x_2, y_1) = B$$

$$I(x_1, y_2) = C$$

$$I(x_2, y_2) = D$$

Calculate the value of a pixel $I(x, y) = ?$ using bi-linear interpolation.

Hint: Interpolate first between A and B, and between C and D, then interpolate between these two computed values.

Advanced Graphics & Image Processing

Introduction to Image Processing

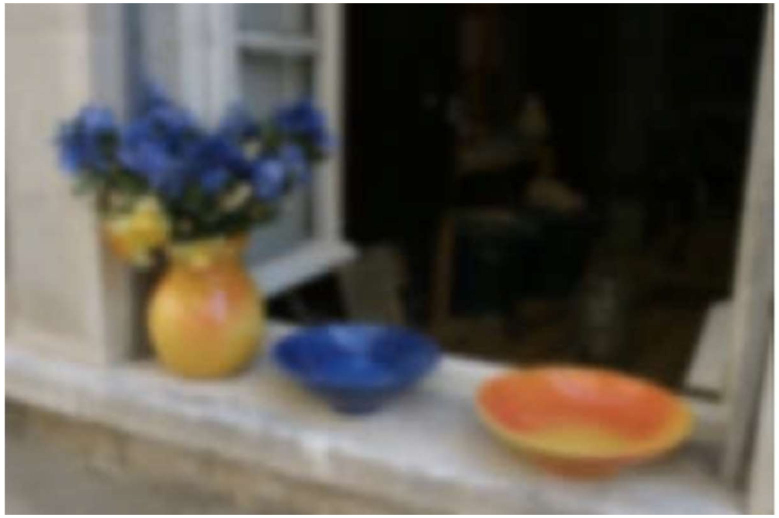
Part 2/2 – Point ops, filters and pyramids

Rafał Mantiuk

Computer Laboratory, University of Cambridge

Point operators and filters

Original



Blurred

Sharpenned

Edge-preserving filter

Point operators

- Modify each pixel independent from one another
- The simplest case: multiplication and addition

Pixel position **x**=(x,y)

Input pixel value

$$g(\boldsymbol{x}) = af(\boldsymbol{x}) + b$$

Resulting pixel value

Gain

Bias

Pixel precision for image processing

- Given an RGB image, 8-bit per color channel (uchar)
  - What happens if the value of 10 is subtracted from the pixel value of 5 ?
  - 250 + 10 = ?
  - How to multiply pixel values by 1.5 ?
    - a) Using floating point numbers
    - b) While avoiding floating point numbers

Image blending

- Cross-dissolve between two images

Pixel from image 1

Pixel from image 2

$$g(\boldsymbol{x}) = (1-\alpha) f_0(\boldsymbol{x}) + \alpha f_1(\boldsymbol{x})$$

Resulting pixel value

Blending parameter

- where α is between 0 and 1

Image matting and compositing

- Matting – the process of extracting an object from the original image
- Compositing – the process of inserting the object into a different image
- It is convenient to represent the extracted object as an RGBA image

Transparency, alpha channel

- RGBA – red, green, blue, alpha
  - alpha = 0 – transparent pixel
  - alpha = 1 – opaque pixel
- Compositing
  - Final pixel value:

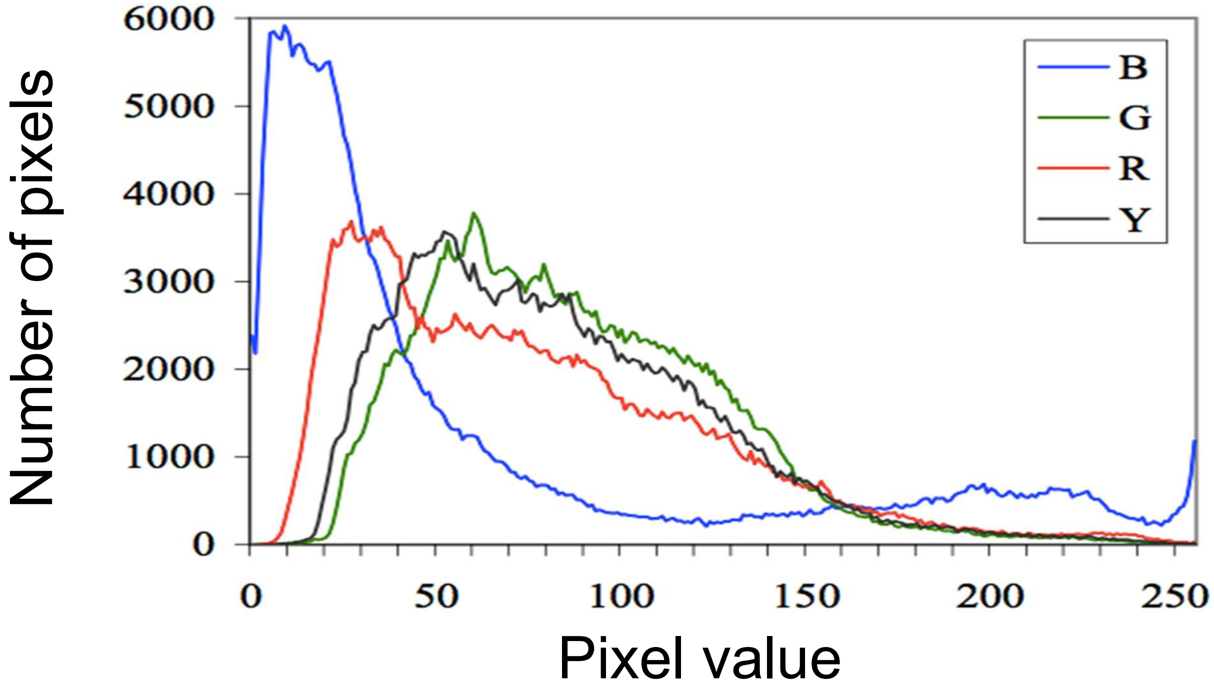
$$P = \alpha C_{pixel} + (1 - \alpha) C_{background}$$

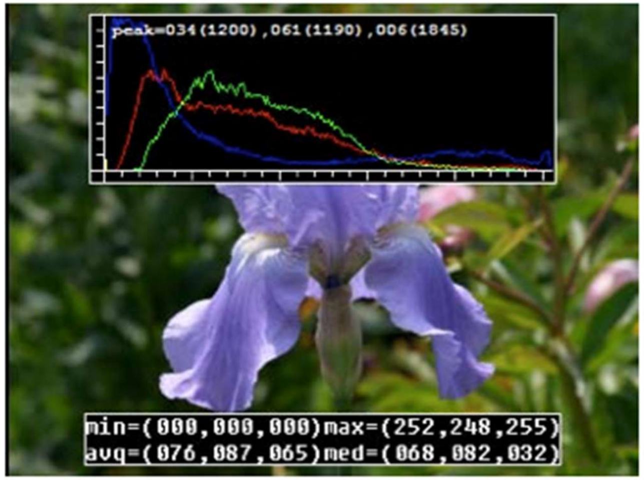
  - Multiple layers:

$$P_0 = C_{background}$$

$$P_i = \alpha_i C_i + (1 - \alpha_i) P_{i-1} \quad i = 1..N$$

Image histogram

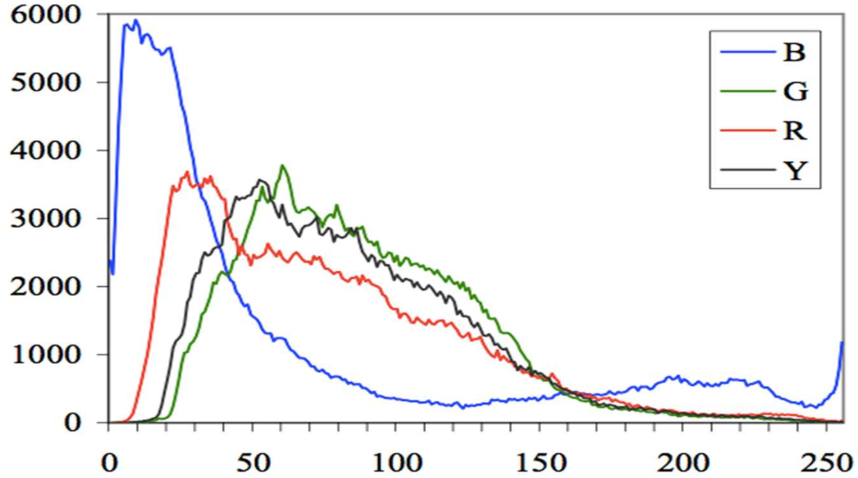




- histogram / total pixels = probability mass function
  - what probability does it represent?

Histogram equalization

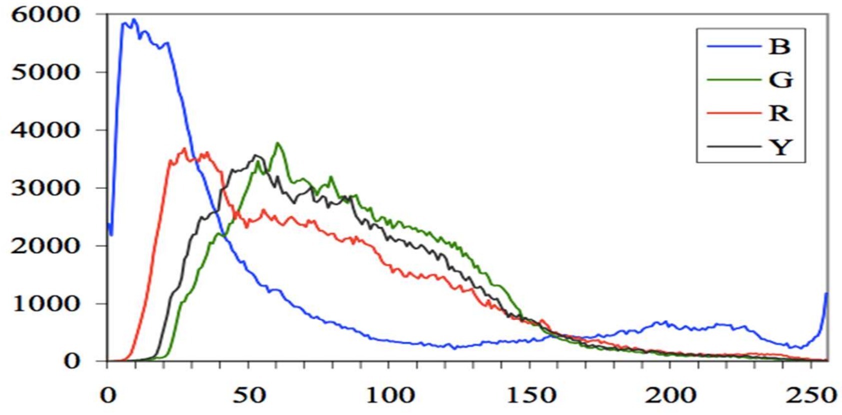
- Pixels are non-uniformly distributed across the range of values



- Would the image look better if we uniformly distribute pixel values (make the histogram more uniform)?
- How can this be done?

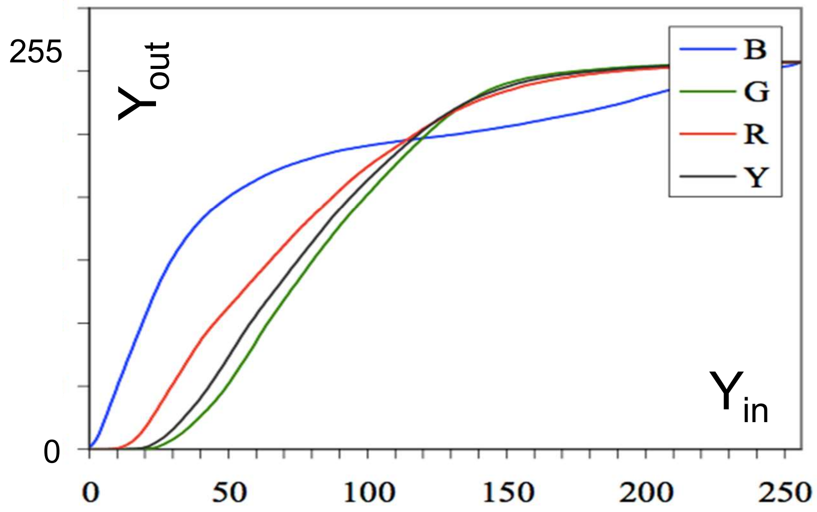
Histogram equalization

- Step 1: Compute image histogram



- Step 2: Compute a normalized cumulative histogram

$$c(I) = \frac{1}{N} \sum_{i=0}^{I} h(i)$$



- Step 3: Use the cumulative histogram to map pixels to the new values (as a look-up table)

$$Y_{out} = c(Y_{in})$$

Linear filtering

- Output pixel value is a weighted sum of neighboring pixels

Input pixel value

Kernel (filter)

$$g(i,j) = \sum_{k,l} f(i-k, j-l)h(k,l)$$

Resulting pixel value

Sum over neighboring pixels, e.g. k=-1,0,1, j=-1,0,1 for 3x3 neighborhood

compact notation $g = f * h$

Convolution operation

Linear filter: example

45	60	98	127	132	133	137	133
46	65	98	123	126	128	131	133
47	65	96	115	119	123	135	137
47	63	91	107	113	122	138	134
50	59	80	97	110	123	133	134
49	53	68	83	97	113	128	133
50	50	58	70	84	102	116	126
50	50	52	58	69	86	101	120

$f(x,y)$

$*$

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1

$h(x,y)$

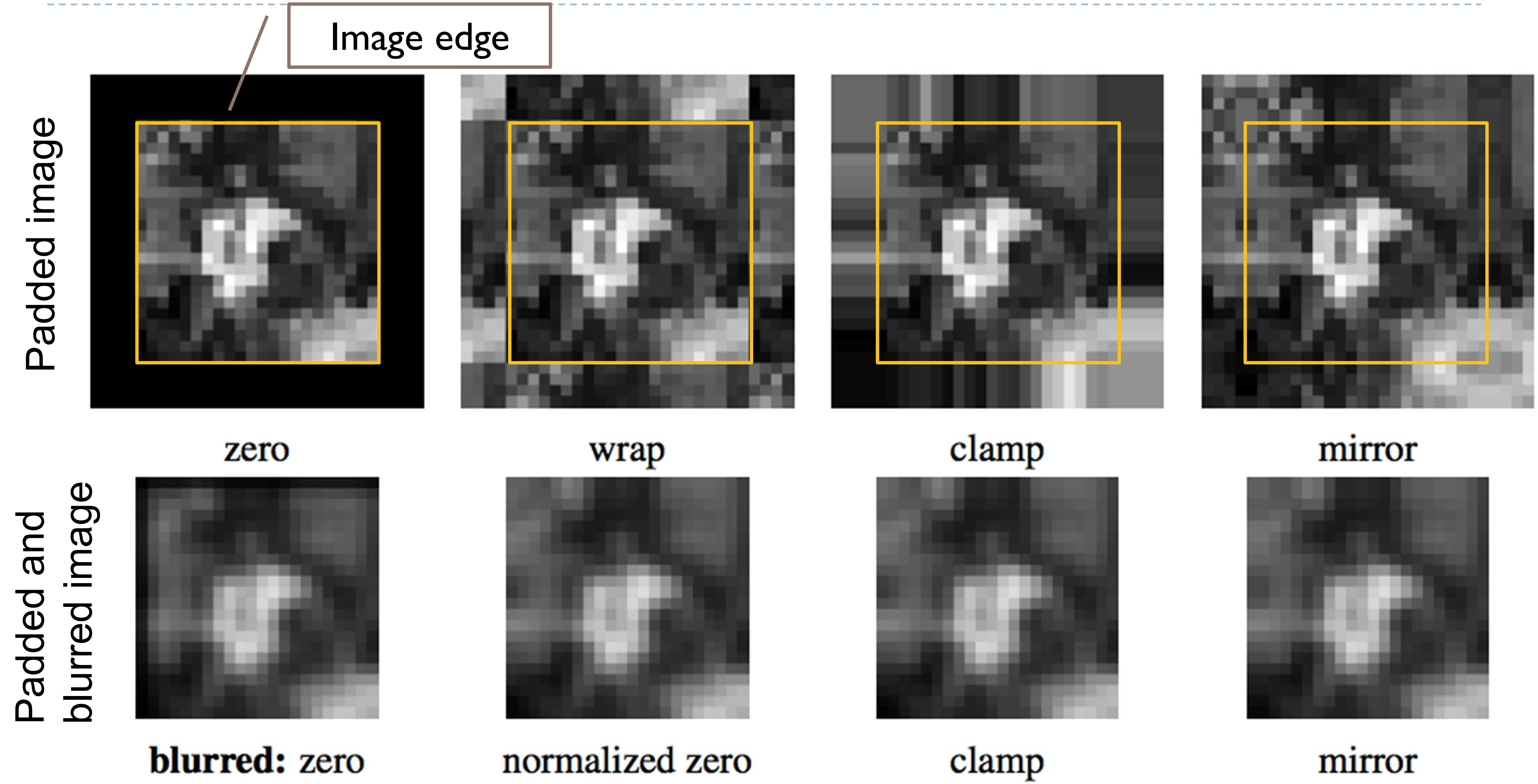
$=$

69	95	116	125	129	132
68	92	110	120	126	132
66	86	104	114	124	132
62	78	94	108	120	129
57	69	83	98	112	124
53	60	71	85	100	114

$g(x,y)$

Why is the matrix g smaller than f ?

Padding an image



What is the computational cost of the convolution?

$$g(i,j) = \sum_{k,l} f(i-k, j-l)h(k,l)$$

- How many multiplications do we need to do to convolve 100x100 image with 9x9 kernel ?
  - The image is padded, but we do not compute the values for the padded pixels

Separable kernels

- Convolution operation can be made much faster if split into two separate steps:
  - 1) convolve all rows in the image with a 1D filter
  - 2) convolve columns in the result of 1) with another 1D filter
- But to do this, the kernel must be separable

$$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \cdot \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix}$$

$$\vec{h} = \vec{u} \cdot \vec{v}$$

Examples of separable filters

- Box filter:

$$\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} = \begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}$$

- Gaussian filter:

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

  - What are the corresponding 1D components of this separable filter (u(x) and v(y))?

$$G(x,y) = u(x) \cdot v(y)$$

Unsharp masking

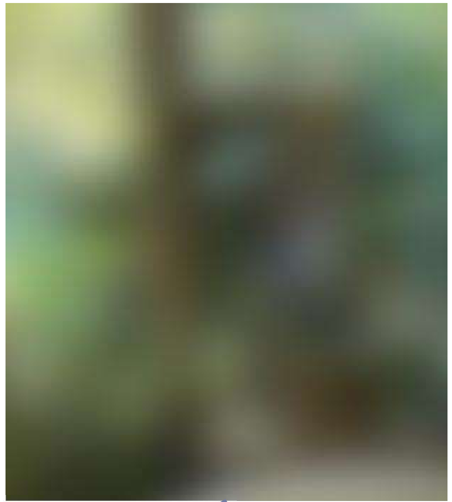
- How to use blurring to sharpen an image ?

results

original image

high-pass image

blurry image



$$g_{\text{sharp}} = f + \gamma(f - h_{\text{blur}} * f)$$

Why "linear" filters ?

- Linear functions have two properties:
  - Additivity: $f(x) + f(y) = f(x + y)$
  - Homogenity: $f(ax) = af(x)$ (where "f" is a linear function)
- Why is it important?
  - Linear operations can be performed in an arbitrary order

$$blur(aF + b) = a\, blur(F) + b$$

  - Linearity of the Gaussian filter could be used to improve the performance of your image processing operation
  - This is also how separable filters work:

Matrix multiplication

Convolution

$$(u \cdot v) * f = u * (v * f)$$

The components of a separable kernel

An image

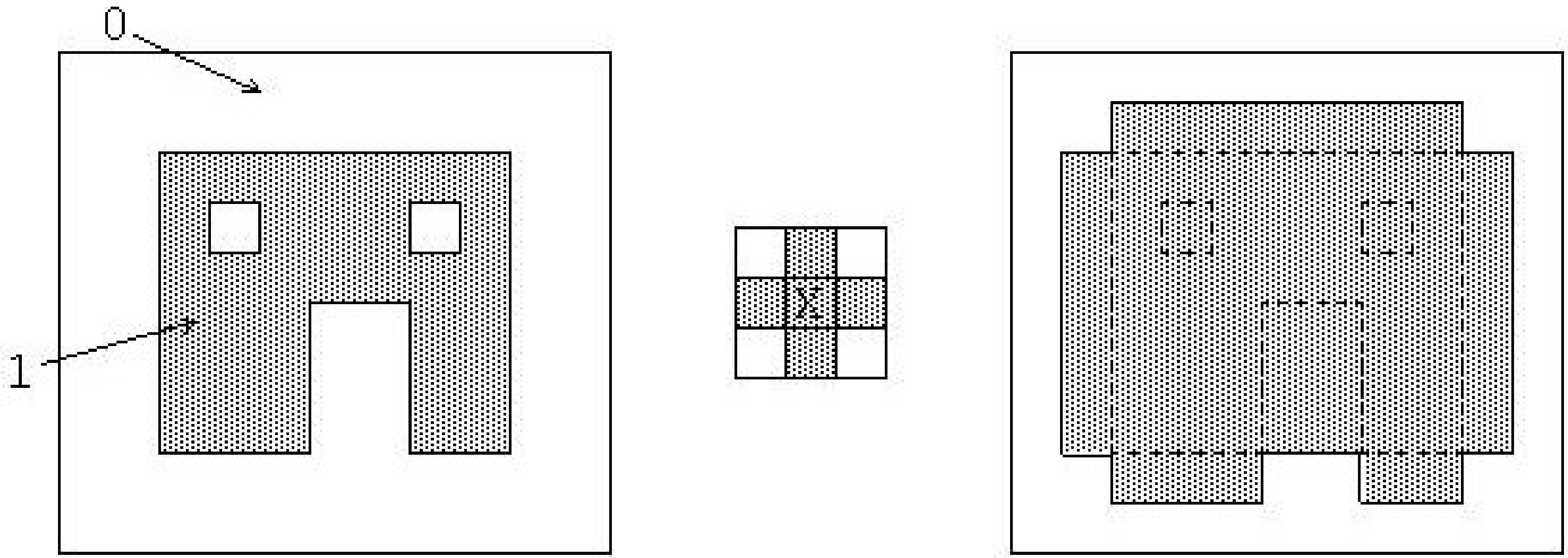
Operations on binary images

- Essential for many computer vision tasks

- Binary image can be constructed by thresholding a grayscale image

$$\theta(f,c) = \begin{cases} 1 & \text{if } f \geq c, \\ 0 & \text{else,} \end{cases}$$

Morphological filters: dilation



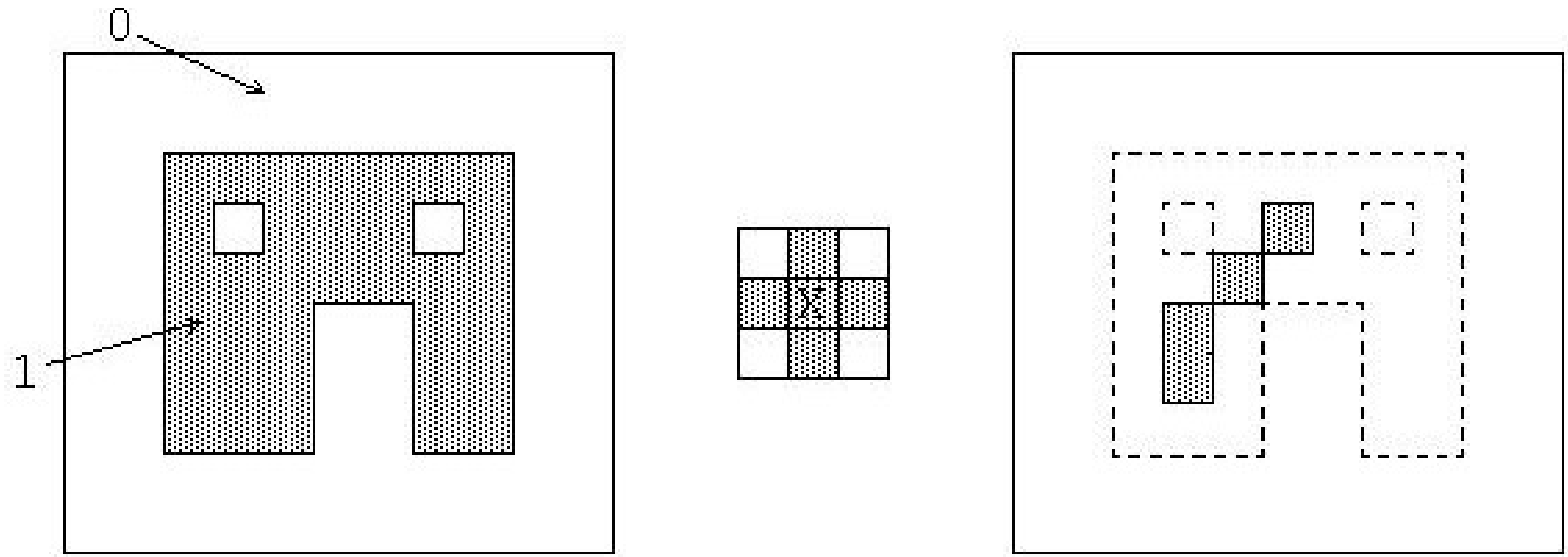
a) Original image

b) Structuring element; x = origin

c) Image after dilation; original in dashes

- Set the pixel to the maximum value of the neighboring pixels within the structuring element
- What could it be useful for ?

Morphological filters: erosion



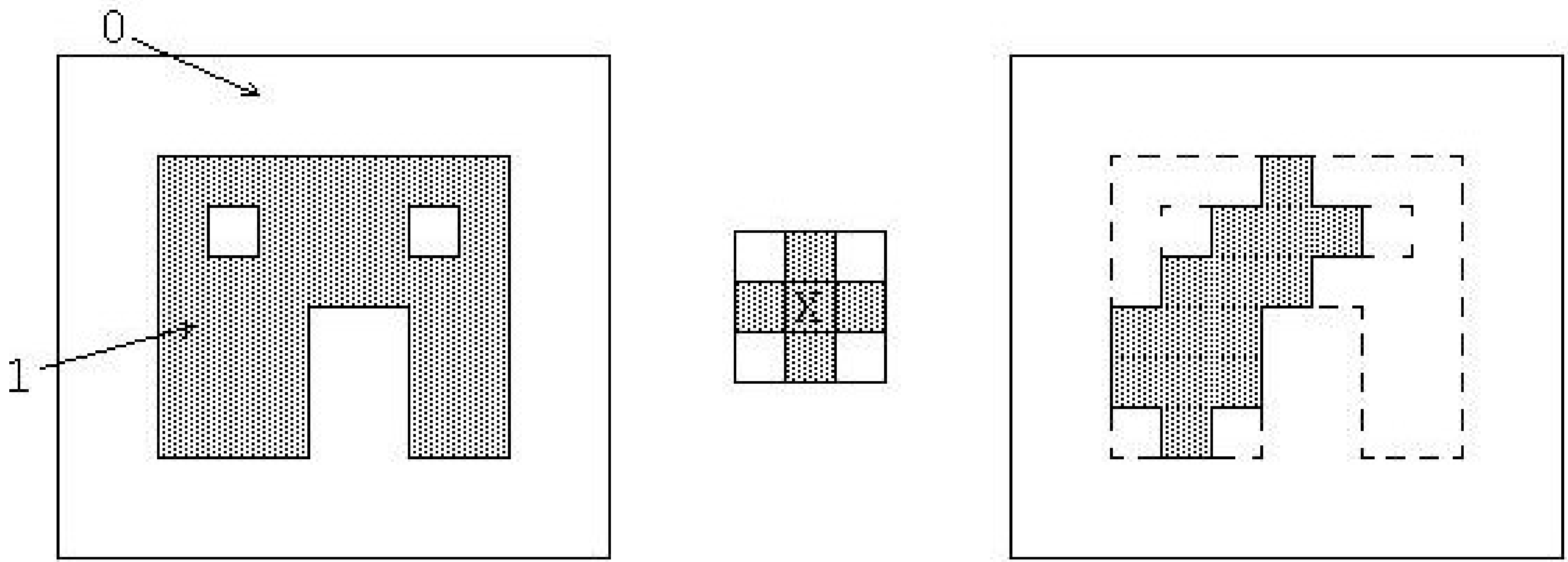
a) Original image

b) Structuring element; x = origin

c) Image after erosion; original in dashes

- Set the value to the minimum value of all the neighboring pixels within the structuring element
- What could it be useful for ?

Morphological filters: opening



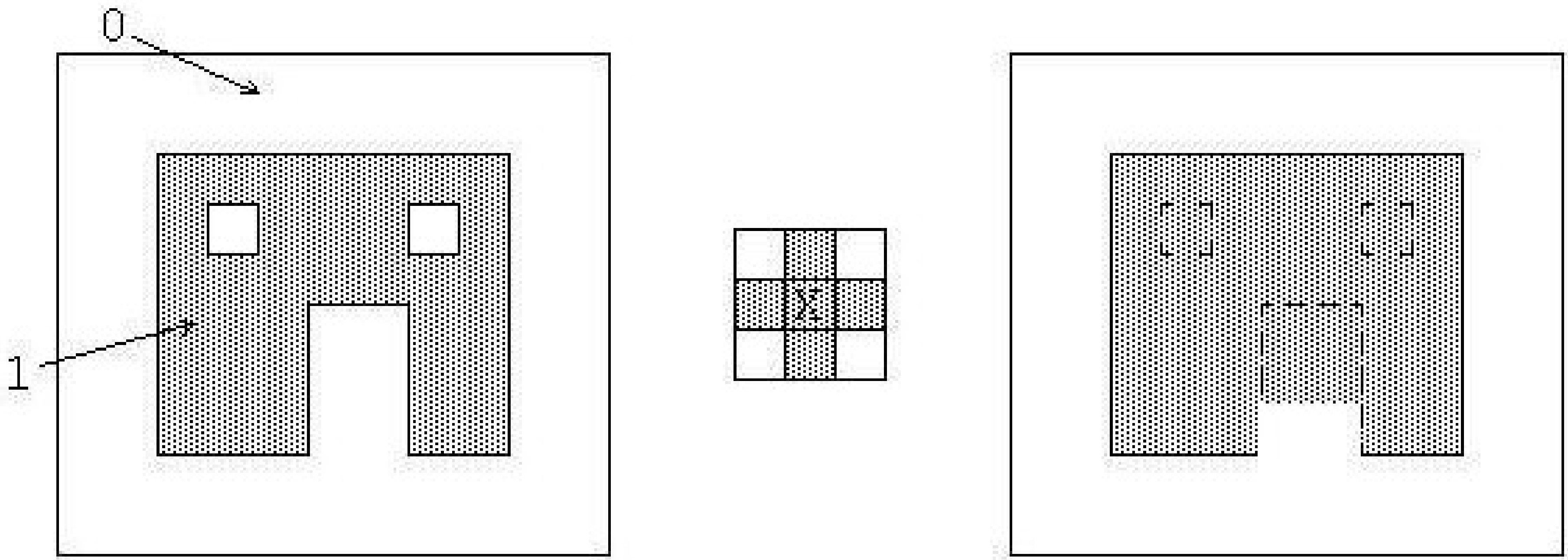
a) Original image

b) Structuring element; x = origin

c) Image after opening = erosion followed by dilation

- Erosion followed by dilation
- What could it be useful for?

Morphological filters: closing



a) Original image

b) Structuring element; x = origin

c) Image after closing = dilation followed by erosion; original in dashes.

- Dilation followed by erosion
- What could it be useful for ?

Binary morphological filters: formal definition

Number of 1s inside the region restricted by the structuring element

Binary image

Correlation (similar to convolution)

Structuring element

$$c = f \otimes s$$

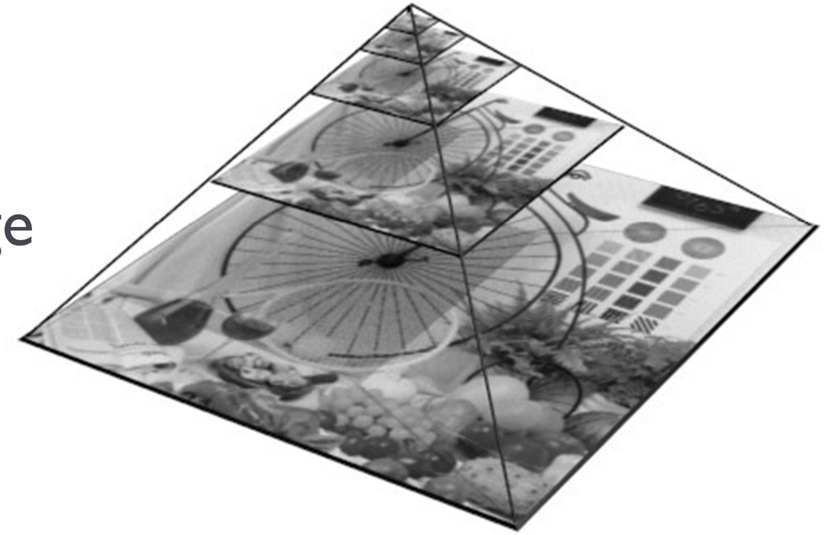
S – size of structuring element (number of 1s in the SI)

- **dilation**: $\text{dilate}(f, s) = \theta(c, 1)$;
- **erosion**: $\text{erode}(f, s) = \theta(c, S)$;
- **majority**: $\text{maj}(f, s) = \theta(c, S/2)$;
- **opening**: $\text{open}(f, s) = \text{dilate}(\text{erode}(f, s), s)$;
- **closing**: $\text{close}(f, s) = \text{erode}(\text{dilate}(f, s), s)$.

$$\theta(f, c) = \begin{cases} 1 & \text{if } f \geq c, \\ 0 & \text{else,} \end{cases}$$

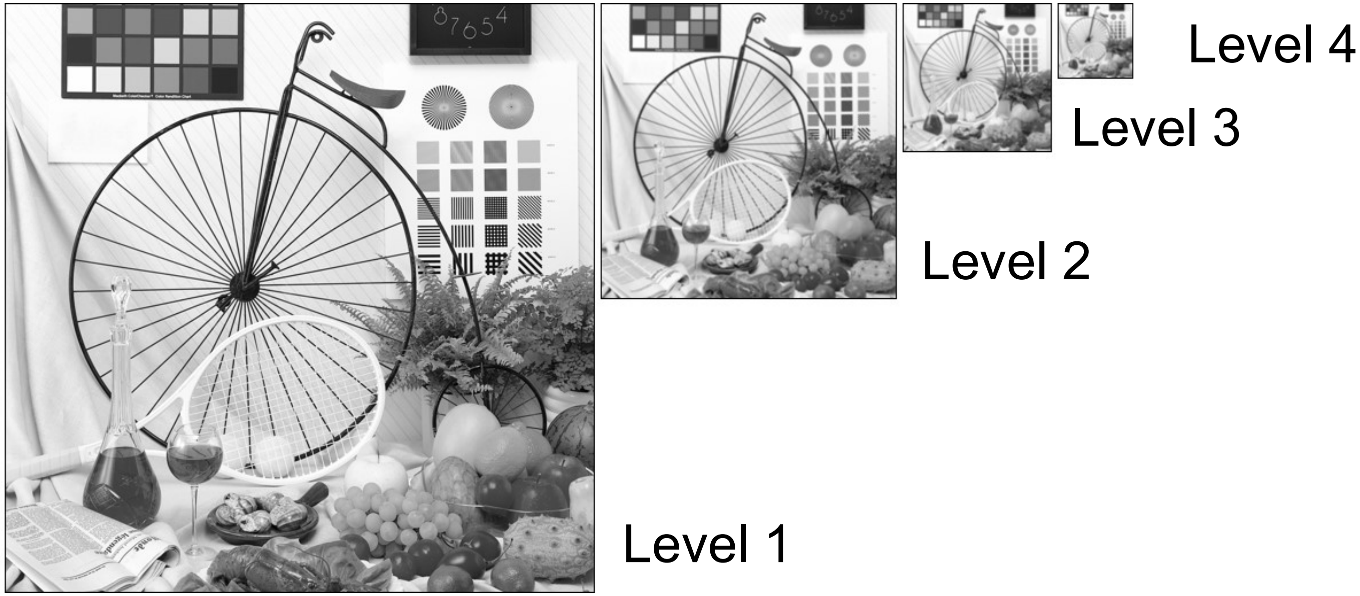
Multi-scale image processing (pyramids)

- Multi-scale processing operates on an image represented at several sizes (scales)
  - Fine level for operating on small details
  - Coarse level for operating on large features
- Example:
  - Motion estimation
    - Use fine scales for objects moving slowly
    - Use coarse scale for objects moving fast
  - Blending (to avoid sharp boundaries)



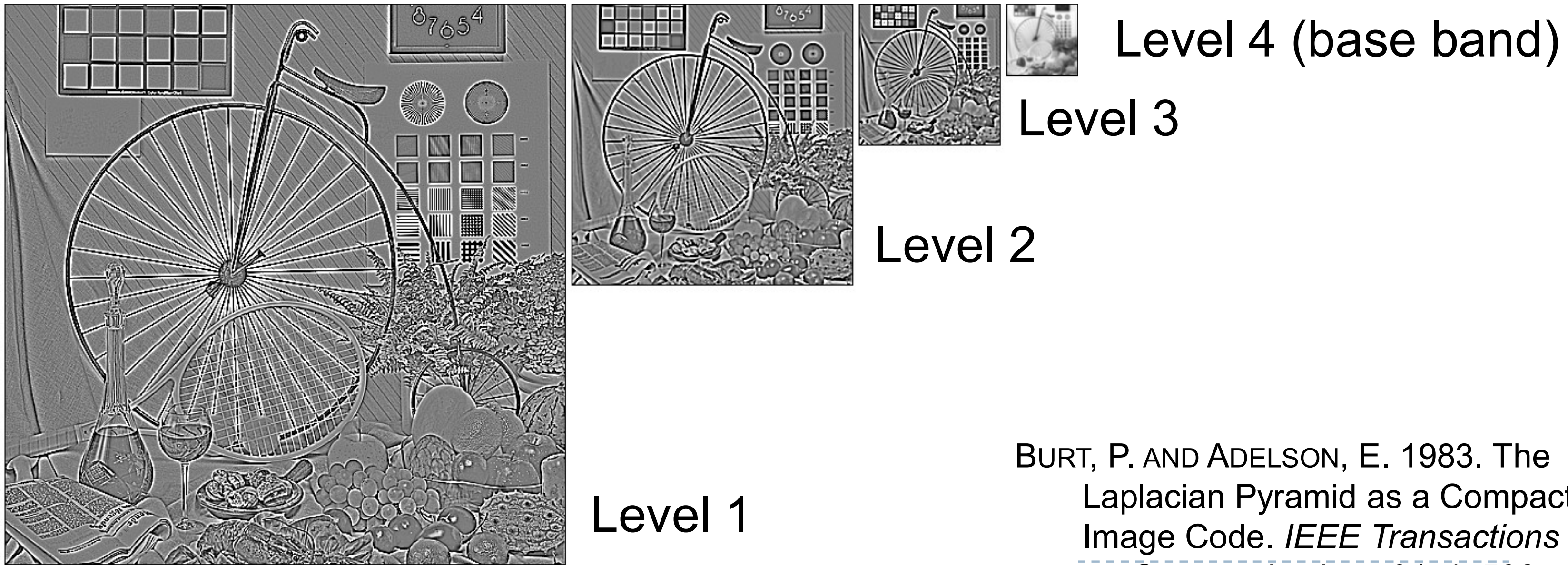
Two types of pyramids

Gaussian pyramid



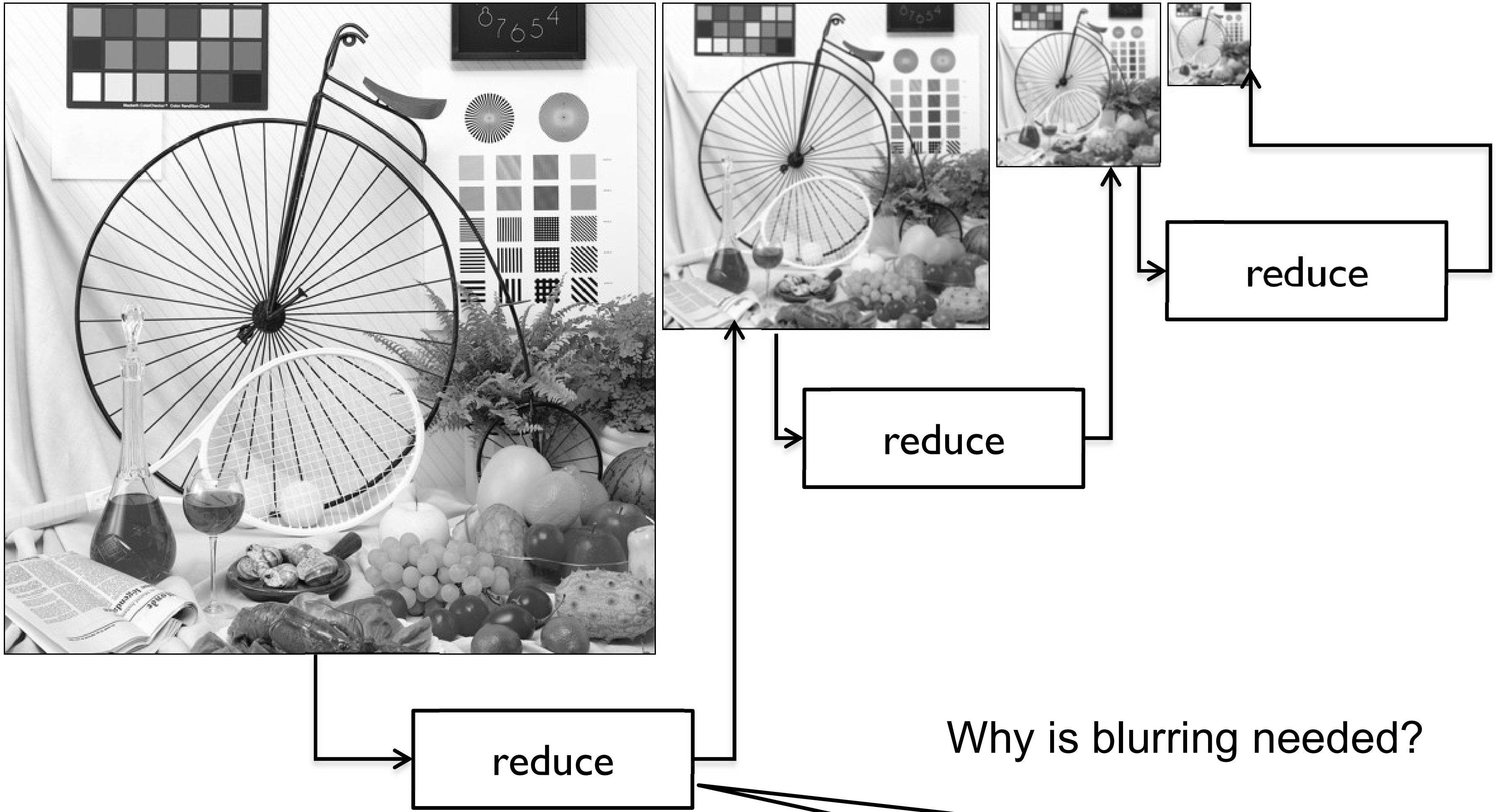
Laplacian pyramid

(a.k.a DoG Diffence of Gaussians)



BURT, P. AND ADELSON, E. 1983. The Laplacian Pyramid as a Compact Image Code. *IEEE Transactions on Communications 31*, 4, 532–540.

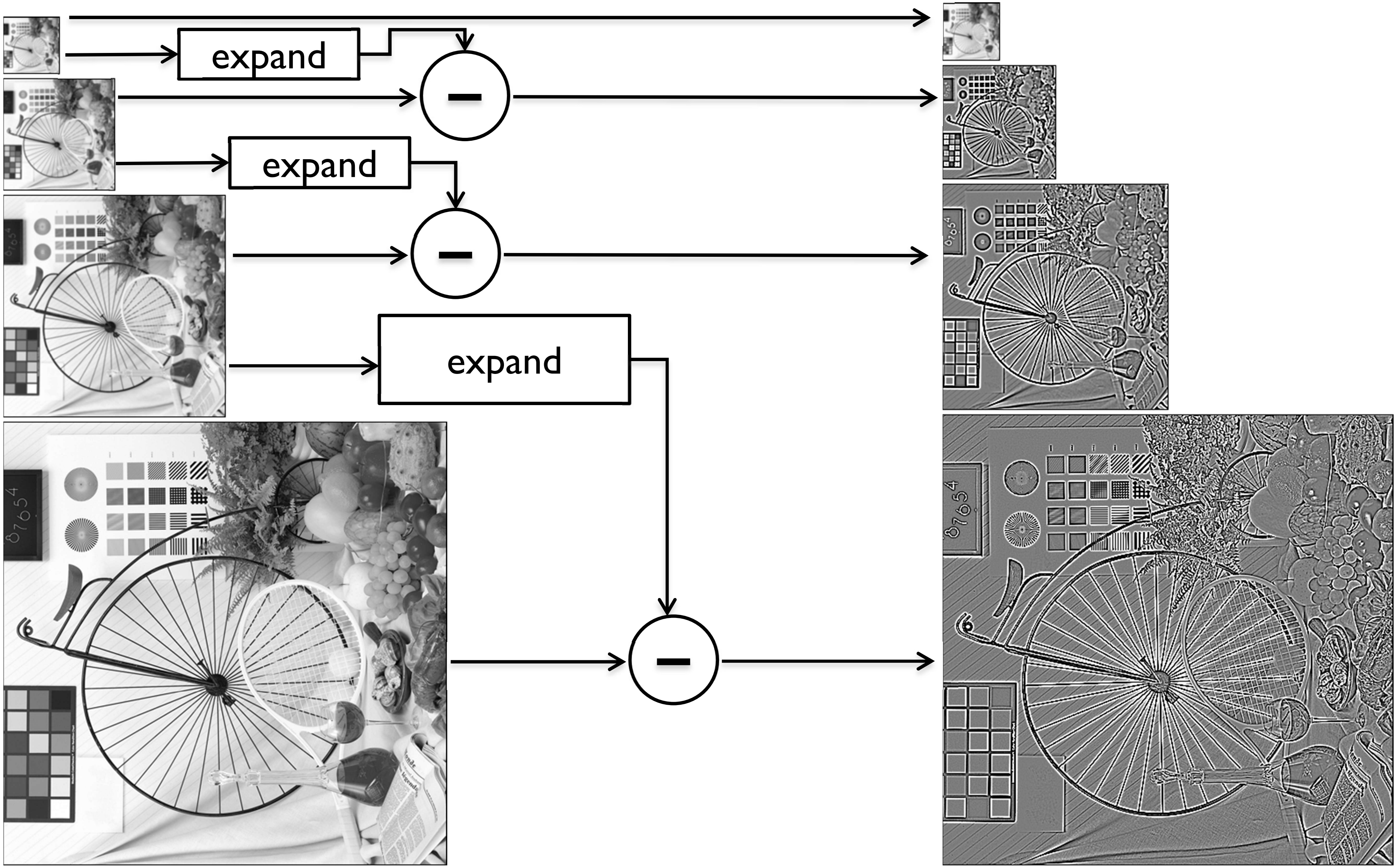
Gaussian Pyramid



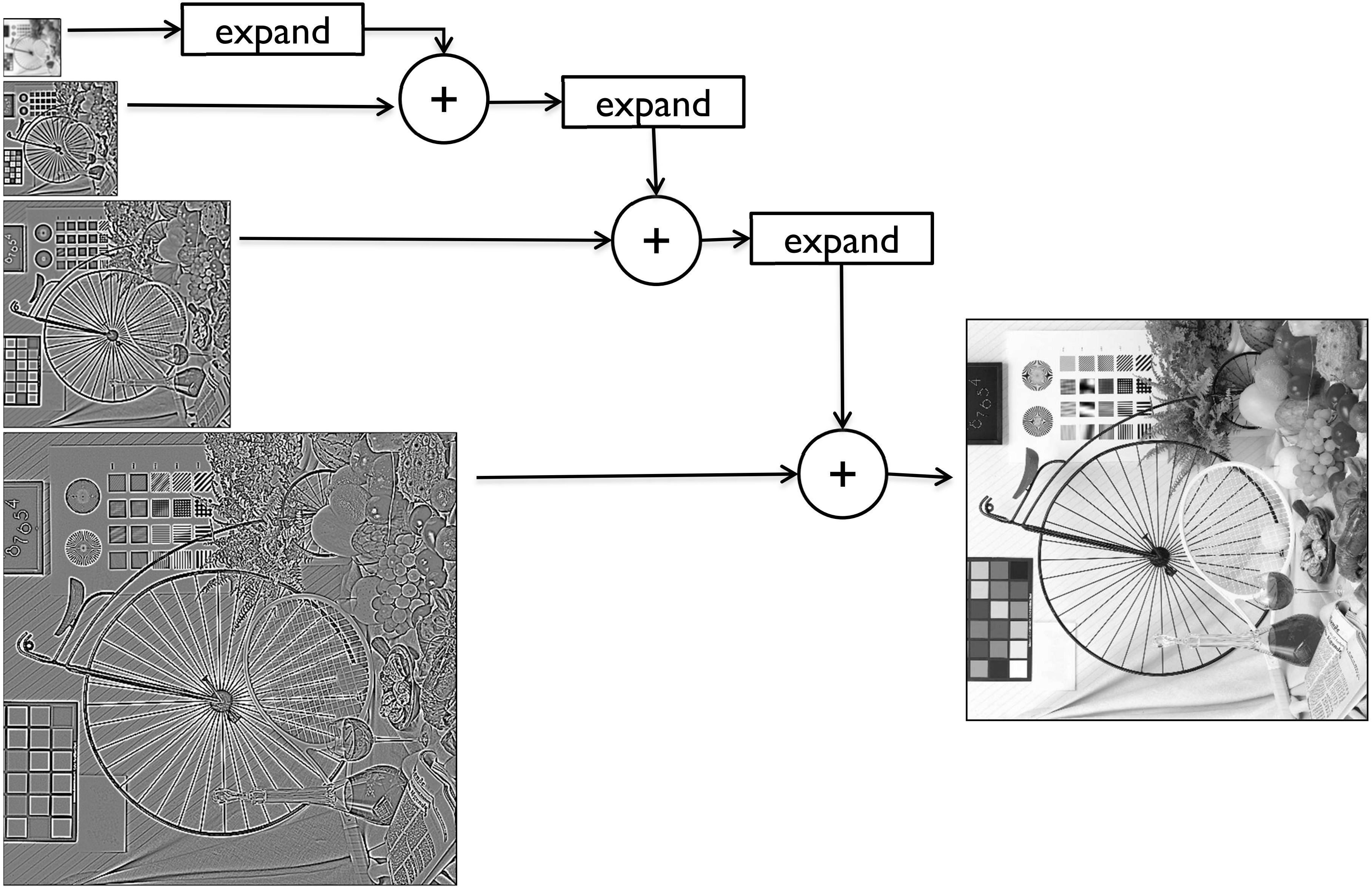
Why is blurring needed?

Blur the image and downsample
(take every 2nd pixel)

Laplacian Pyramid - decomposition

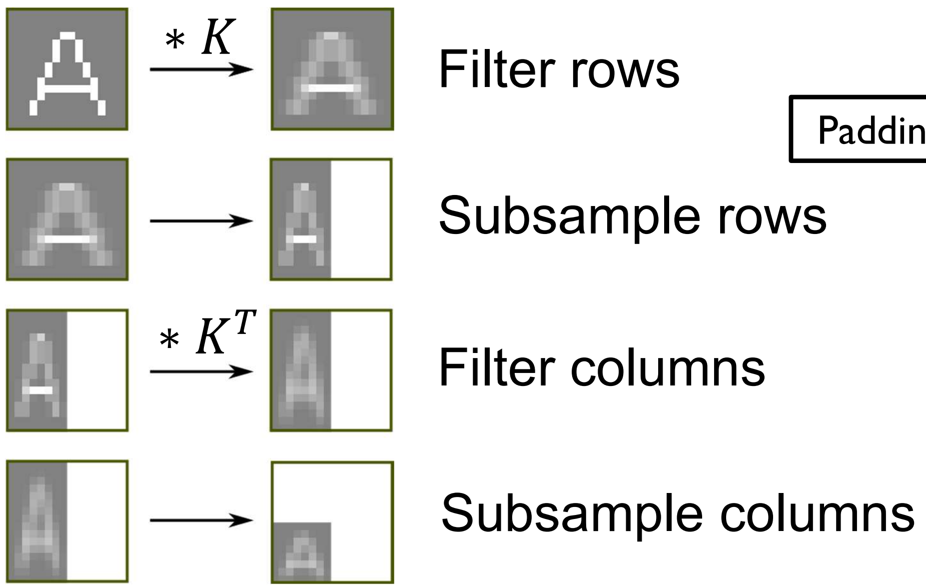


Laplacian Pyramid - synthesis

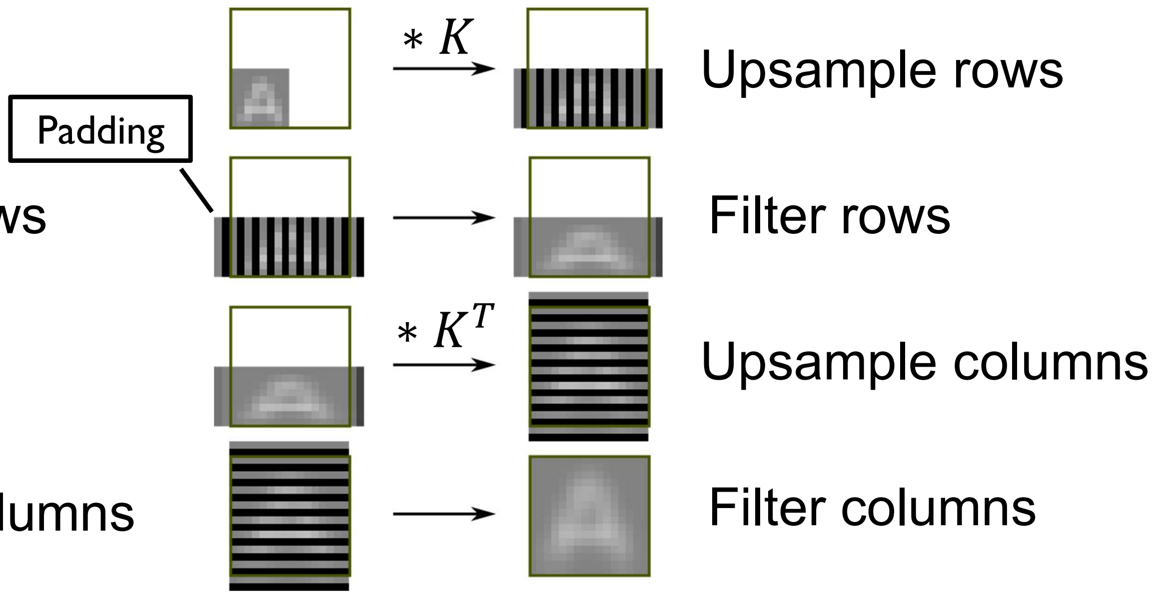


Reduce and expand

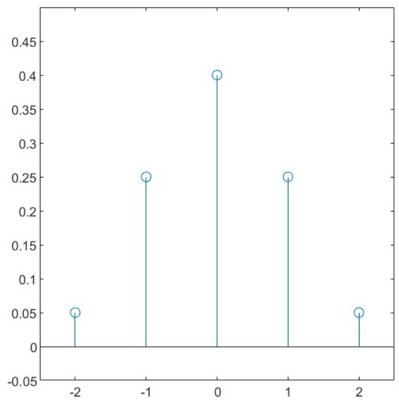
Reduce

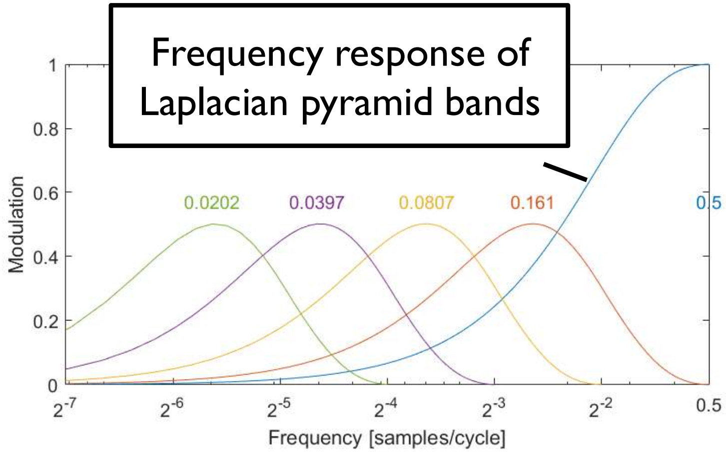


Expand



$K =$





Example: stitching and blending

Combine two images:

+

Image-space blending

Laplacian pyramid blending

References

- SZELISKI, R. 2010. *Computer Vision: Algorithms and Applications*. Springer-Verlag New York Inc.
  - Chapter 3
  - http://szeliski.org/Book