Image processing

Today

- Consequences of image formation

- Some basic primitives needed for computer vision problems
  - Convolution
  - Edge detection
  - Image resizing

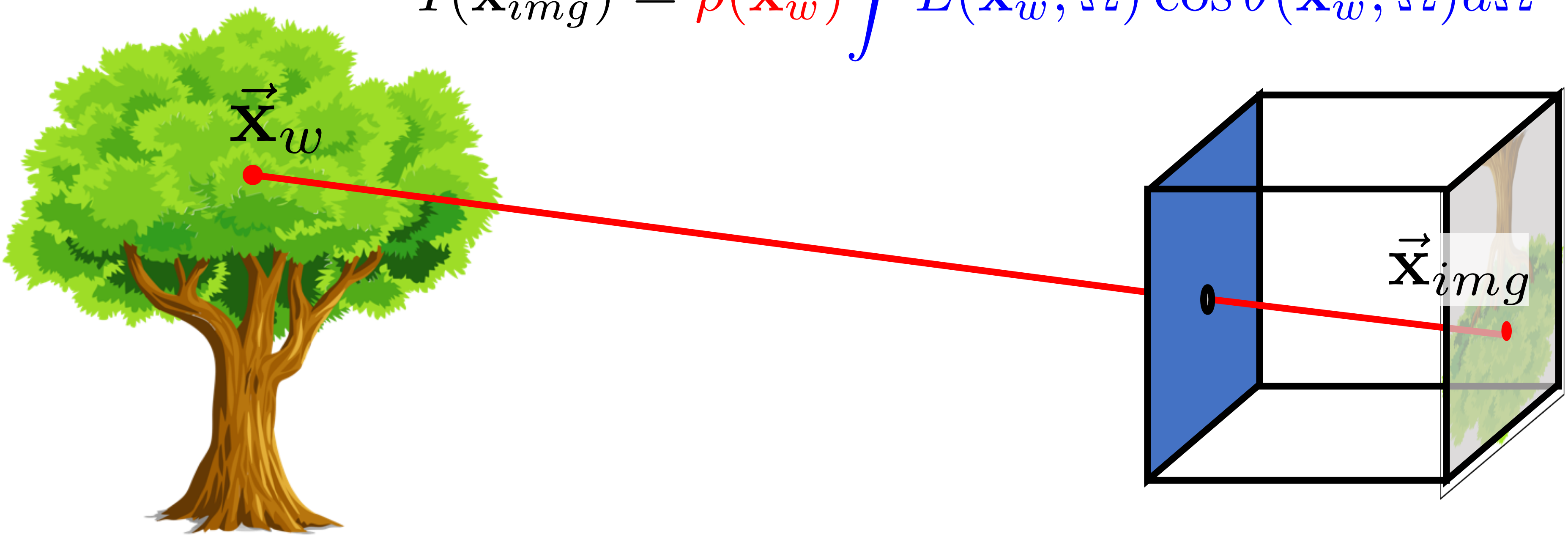
- Convolution as a basic operation
- Image pyramids as a basic structure

Recap

- Geometry: $\vec{\mathbf{x}}_{img} \equiv K \begin{bmatrix} R & \mathbf{t} \end{bmatrix} \vec{\mathbf{x}}_w$

- Color (Lambertian assumptipn):

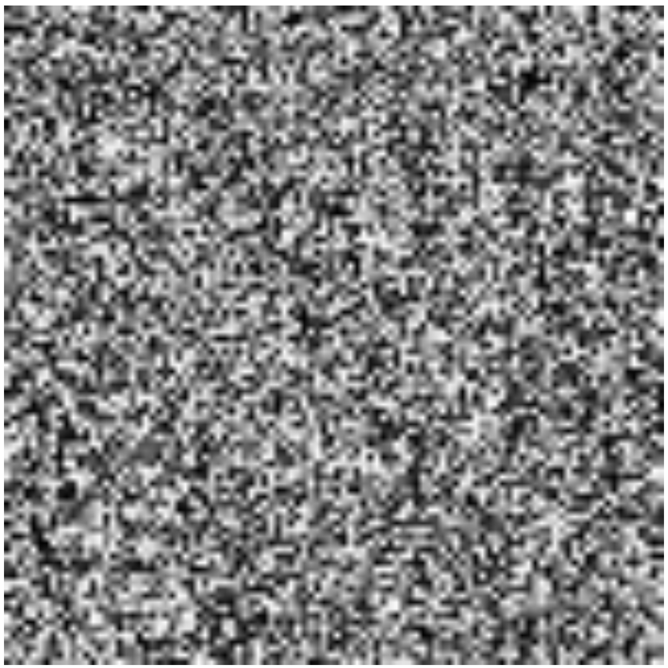
$$I(\vec{\mathbf{x}}_{img}) = \rho(\vec{\mathbf{x}}_w) \int L(\vec{\mathbf{x}}_w, \Omega) \cos\theta(\vec{\mathbf{x}}_w, \Omega) d\Omega$$

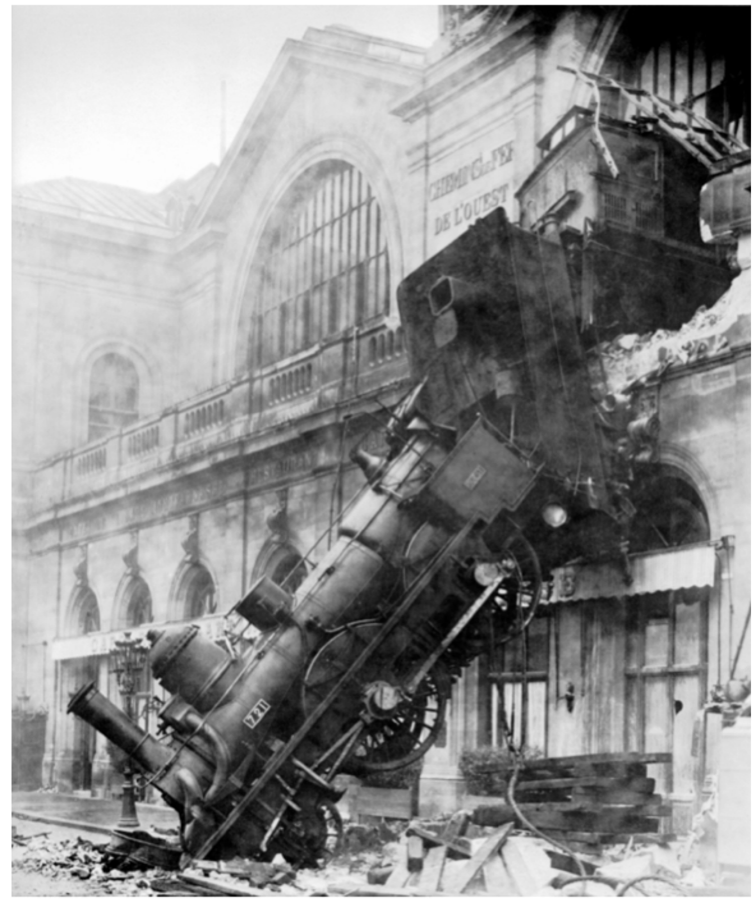


Consequences of image formation

- Nearby objects appear larger
- Parallel lines and planes converge
- Information lost: distance from camera
- Pixel color depends on light intensity, light direction and surface normal and paint on object
- So objects in images
  - can appear in many different sizes and many positions
  - can have very different color

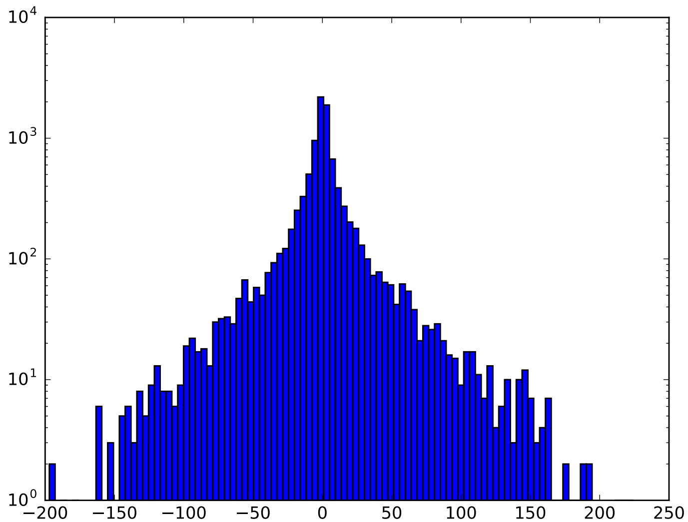
Consequence 1: nearby pixels are similar



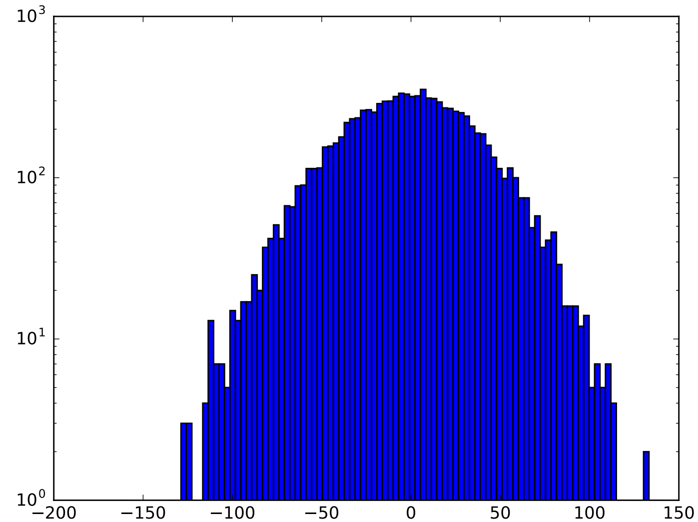


Consequence 1: nearby pixels are similar

Log histogram of differences between adjacent pixels



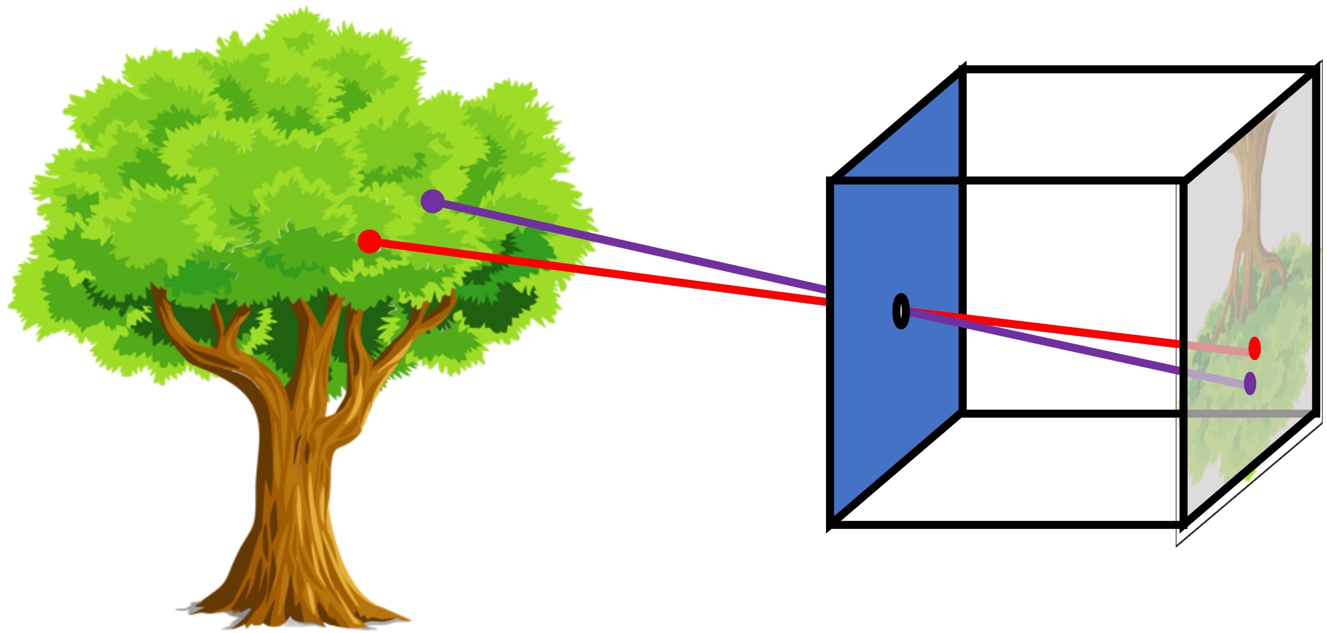
Natural images



Random arrays

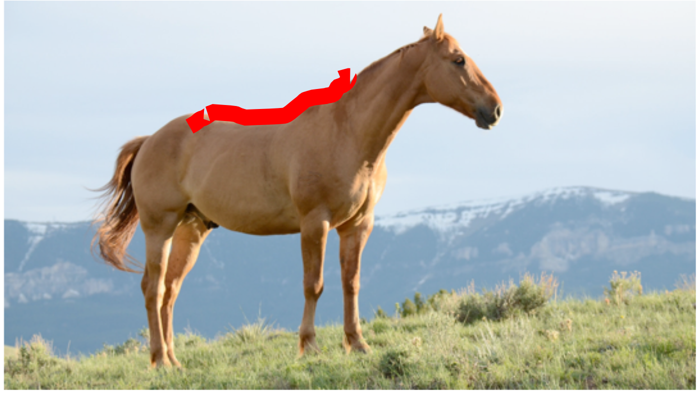
Consequence 1: nearby pixels are similar

- Why?
- Nearby pixels in pinhole camera lead to nearby rays
- Nearby rays *mostly* fall on the same object
- Objects have *mostly* smooth surfaces and *mostly* uniform color
- Lighting is *mostly* uniform



Consequence 1: nearby pixels are similar

- Nearby pixels that are *not* similar tend to have different depth, surface normal, paint or lighting
- Idea: *Abrupt changes in color can delineate objects, be a clue to shape, or be distinctive marks*



Depth discontinuities

Changes in albedo

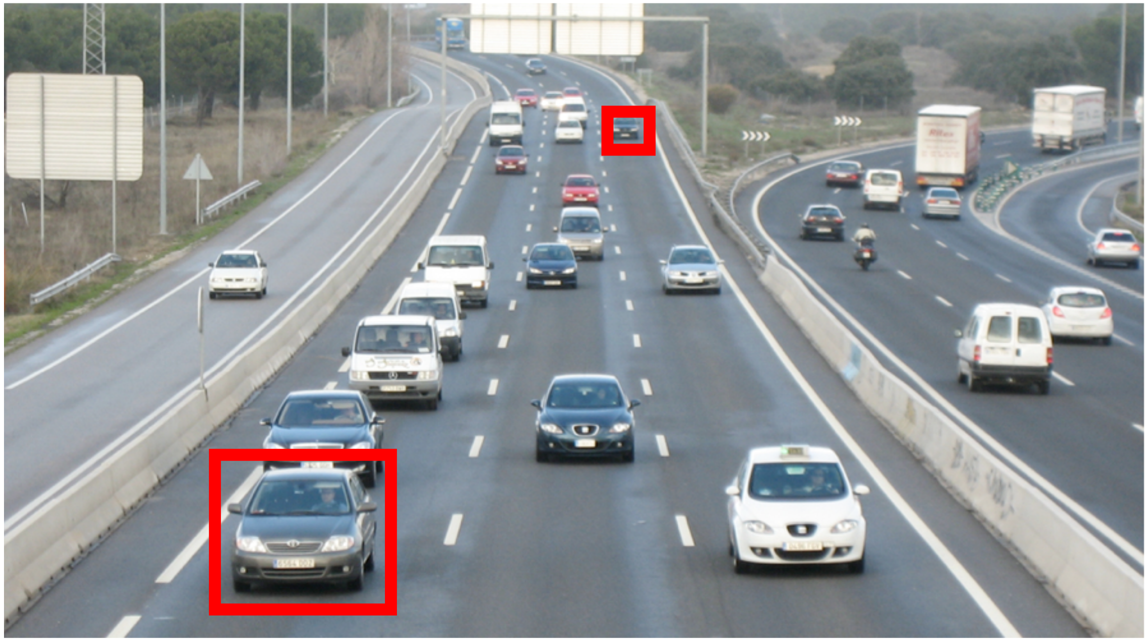
Normal discontinuities

Key primitive: edge detection

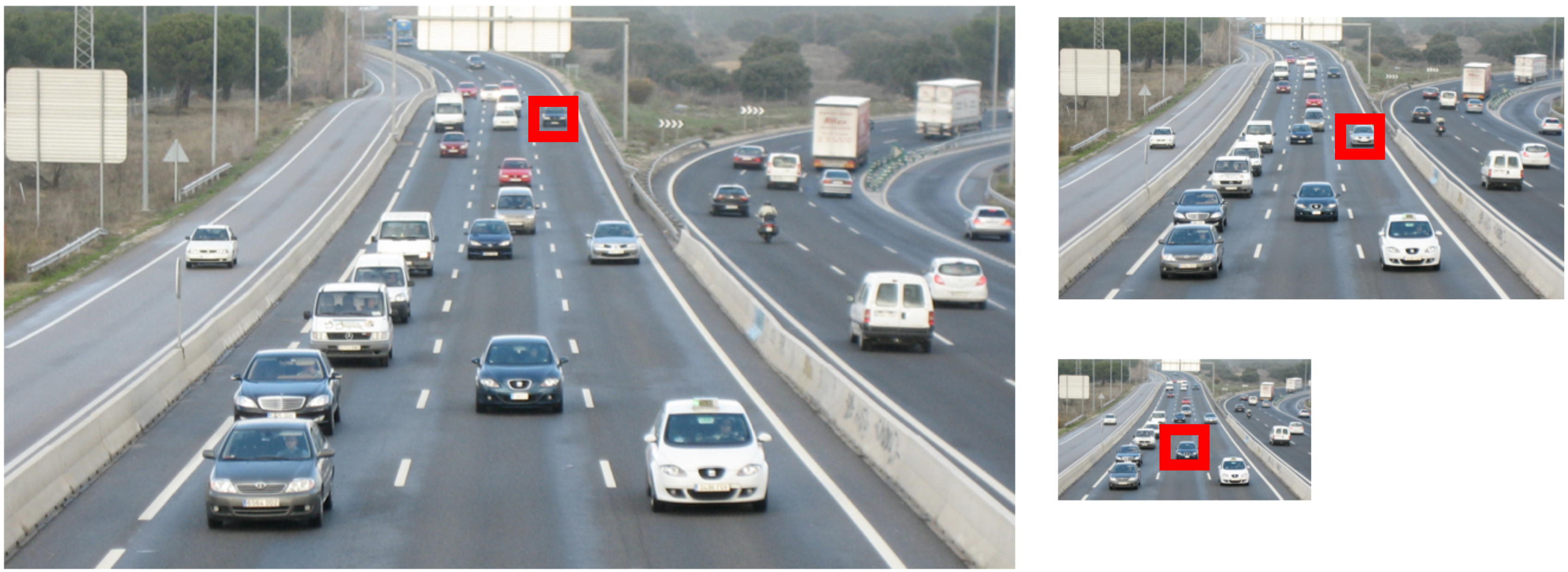
Consequence 2: Farther away objects appear smaller

Consequence 2: Farther away objects appear smaller

- Idea: search for objects over multiple scales



Key primitive: Image resizing



Some primitives

- Edge detection: identifying where pixels change color
  - Cue to object boundary
  - Cue to shape
  - More resilient to lighting than pixel color
- Image resizing: downsizing or upscaling images
  - Allows searching over scales
- Image processing: Operations that take images as input and produce images as output

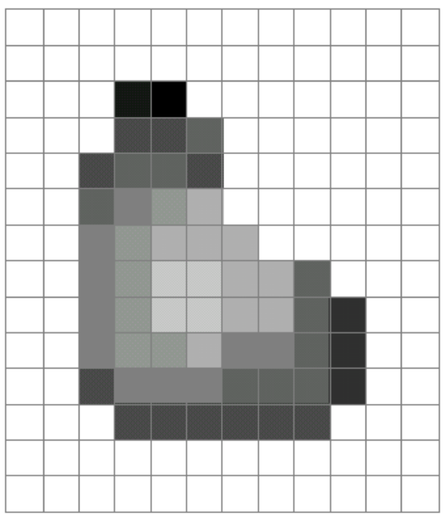
Prelude: Image denoising

Why would images have noise?

- Sensor noise
  - Sensors count photons: noise in count
- Dead pixels
- Old photographs
- …

What is an image?

- A grid (matrix) of intensity values: 1 color or 3 colors



=

255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	20	0	255	255	255	255	255	255	255
255	255	255	75	75	75	255	255	255	255	255	255
255	255	75	95	95	75	255	255	255	255	255	255
255	255	96	127	145	175	255	255	255	255	255	255
255	255	127	145	175	175	175	255	255	255	255	255
255	255	127	145	200	200	175	175	95	255	255	255
255	255	127	145	200	200	175	175	95	47	255	255
255	255	127	145	145	175	127	127	95	47	255	255
255	255	74	127	127	127	95	95	95	47	255	255
255	255	255	74	74	74	74	74	74	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255

(common to use one byte per value: 0 = black, 255 = white)

An assumption about noise

- Let us assume noise at a pixel is
  - independent of other pixels
  - distributed according to a Gaussian distribution
    - i.e., low noise values are more likely than high noise values
    - “grainy images”

Noise reduction

- Nearby pixels are likely to belong to same object
  - thus likely to have similar color
- Replace each pixel by *average of neighbors*

Mean filtering

0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	10	0	0	0	0
0	0	10	20	20	20	10	40	0	0
0	10	20	30	0	20	10	0	0	0
0	10	0	30	40	30	20	10	0	0
0	10	20	30	40	30	20	10	0	0
0	10	20	10	40	30	20	10	0	0
0	10	20	30	30	20	10	0	0	0
0	0	10	20	20	0	10	0	20	0
0	0	0	10	10	10	0	0	0	0

$$(0 + 0 + 0 + 10 + 40 + 0 + 10 + 0 + 0)/9 = 6.66$$

Mean filtering

0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	10	0	0	0	0
0	0	10	20	20	20	10	40	0	0
0	10	20	30	0	20	10	0	0	0
0	10	0	30	40	30	20	10	0	0
0	10	20	30	40	30	20	10	0	0
0	10	20	10	40	30	20	10	0	0
0	10	20	30	30	20	10	0	0	0
0	0	10	20	20	0	10	0	20	0
0	0	0	10	10	10	0	0	0	0

$$(0 + 0 + 0 + 0 + 0 + 10 + 0 + 0 + 0 + 0 + 20 + 10 + 40 + 0 + 0 + 20 + 10 + 0 + 0 + 0 + 30 + 20 + 10 + 0 + 0)/25 = 6.8$$

Mean filtering

0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	10	0	0	0	0
0	0	10	20	20	20	10	40	0	0
0	10	20	30	0	20	10	0	0	0
0	10	0	30	40	30	20	10	0	0
0	10	20	30	40	30	20	10	0	0
0	10	20	10	40	30	20	10	0	0
0	10	20	30	30	20	10	0	0	0
0	0	10	20	20	0	10	0	20	0
0	0	0	10	10	10	0	0	0	0

0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$(0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 10)/9 = 1.11$$

Mean filtering

0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	10	0	0	0	0
0	0	10	20	20	20	10	40	0	0
0	10	20	30	0	20	10	0	0	0
0	10	0	30	40	30	20	10	0	0
0	10	20	30	40	30	20	10	0	0
0	10	20	10	40	30	20	10	0	0
0	10	20	30	30	20	10	0	0	0
0	0	10	20	20	0	10	0	20	0
0	0	0	10	10	10	0	0	0	0

0	0	0	0	0	0	0	0	0	0
0	1	4	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$(0 + 0 + 0 + 0 + 0 + 10 + 0 + 10 + 20)/9 = 4.44$$

Mean filtering

0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	10	0	0	0	0
0	0	10	20	20	20	10	40	0	0
0	10	20	30	0	20	10	0	0	0
0	10	0	30	40	30	20	10	0	0
0	10	20	30	40	30	20	10	0	0
0	10	20	10	40	30	20	10	0	0
0	10	20	30	30	20	10	0	0	0
0	0	10	20	20	0	10	0	20	0
0	0	0	10	10	10	0	0	0	0

0	0	0	0	0	0	0	0	0	0
0	1	4	8	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

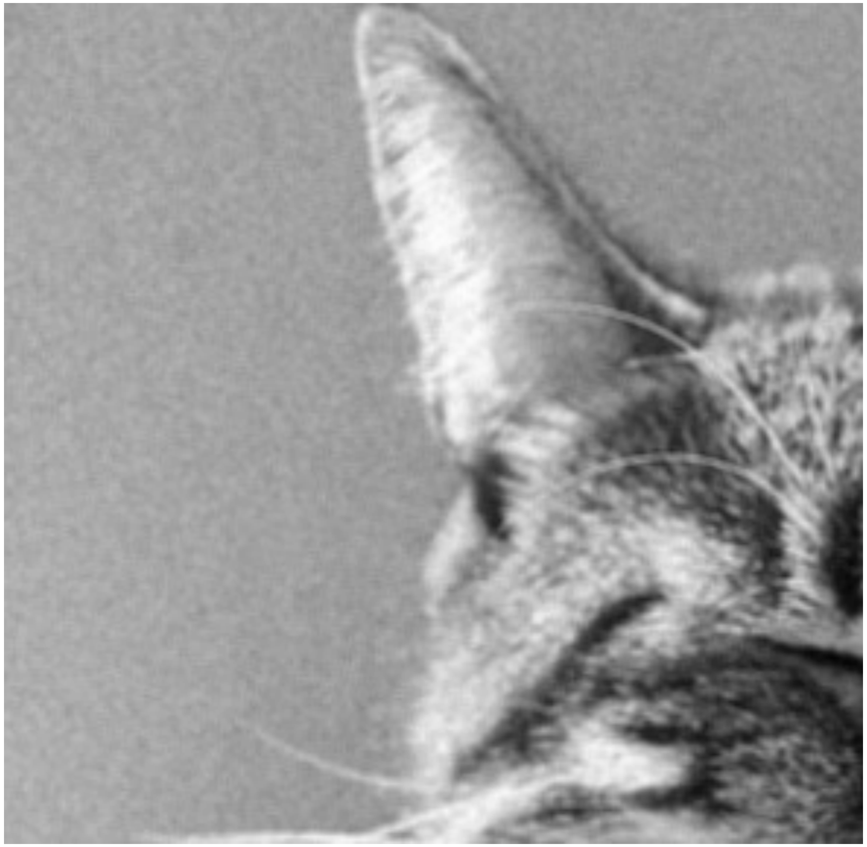
$$(0 + 0 + 0 + 0 + 10 + 10 + 10 + 20 + 20)/9 = 7.77$$

Mean filtering

0	0	0	0	0	0	0	0	0	0
0	0	0	10	10	10	0	0	0	0
0	0	10	20	20	20	10	40	0	0
0	10	20	30	0	20	10	0	0	0
0	10	0	30	40	30	20	10	0	0
0	10	20	30	40	30	20	10	0	0
0	10	20	10	40	30	20	10	0	0
0	10	20	30	30	20	10	0	0	0
0	0	10	20	20	0	10	0	20	0
0	0	0	10	10	10	0	0	0	0

0	0	0	0	0	0	0	0	0	0
0	1	4	8	10	8	9	6	4	0
0	4	11	13	16	11	12	7	4	0
0	6	14	19	23	19	18	10	6	0
0	8	18	23	28	23	17	8	2	0
0	8	16	26	31	30	20	10	3	0
0	10	18	27	29	27	17	8	2	0
0	8	14	22	22	20	11	8	3	0
0	4	11	17	17	12	6	4	2	0
0	0	0	0	0	0	0	0	0	0

Noise reduction using mean filtering



Mean filtering

- Replace pixel by mean of neighborhood

10	5	3
4	5	1
1	1	7

Local image data

f

	7	

Modified image data

$S[f]$

$$S[f](m,n) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} f(m+i, n+j)/9$$

A more general version

10	5	3
4	5	1
1	1	7

Local image data

	7	

Kernel / filter

$$S[f](m,n) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} w(i,j) f(m+i, n+j)$$

A more general version

0	10	5	7	0
5	11	6	8	3
9	22	4	5	1
2	9	14	6	7
3	10	15	12	9

Local image data

		7		

Kernel size = 2k+1

$$S[f](m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m+i, n+j)$$

A more general version

$$S[f](m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m+i, n+j)$$

- w(i,j) = $1/(2k+1)^2$ for mean filter
- If w(i,j) >=0 and sum to 1, *weighted mean*
- But w(i,j) can be *arbitrary real numbers!*

Convolution and cross-correlation

- Cross correlation

$$S[f] = w \otimes f$$

$$S[f](m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m+i, n+j)$$

- Convolution

$$S[f] = w * f$$

$$S[f](m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m-i, n-j)$$

Cross-correlation

1	2	3
4	5	6
7	8	9

w

1	2	3
4	5	6
7	8	9

f

1*1 + 2*2 + 3*3 + 4*4 + 5*5 + 6*6 + 7*7 + 8*8 + 9*9

Convolution

1	2	3
4	5	6
7	8	9

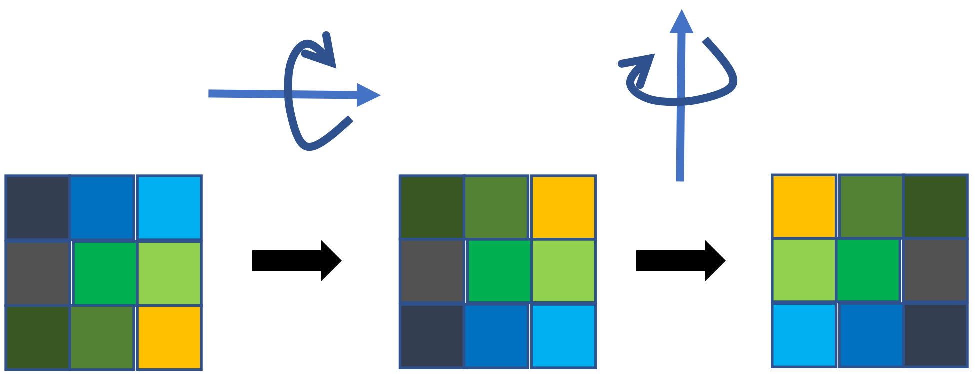
w

1	2	3
4	5	6
7	8	9

f

1*9 + 2*8 + 3*7 + 4*6 + 5*5 + 6*4 + 7*3 + 8*2 + 9*1

Convolution



F

Adapted from F. Durand

Properties: Linearity

$$(w \otimes f)(m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m+i, n+j)$$

$$f' = af + bg$$

$$w \otimes f' = a(w \otimes f) + b(w \otimes g)$$

Properties: Linearity

$$(w \otimes f)(m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m+i, n+j)$$

$$w' = aw + bv$$

$$w' \otimes f = a(w \otimes f) + b(v \otimes f)$$

Properties: Shift invariance

$$(w \otimes f)(m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m+i, n+j)$$

$$f'(m,n) = f(m - m_0, n - n_0)$$

f

f'

Shift invariance

$$(w \otimes f)(m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m+i, n+j)$$

$$f'(m,n) = f(m-m_0, n-n_0)$$

$$\begin{aligned}(w \otimes f')(m,n) &= \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f'(m+i, n+j) \\ &= \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m+i-m_0, n+j-n_0) \\ &= (w \otimes f)(m-m_0, n-n_0)\end{aligned}$$

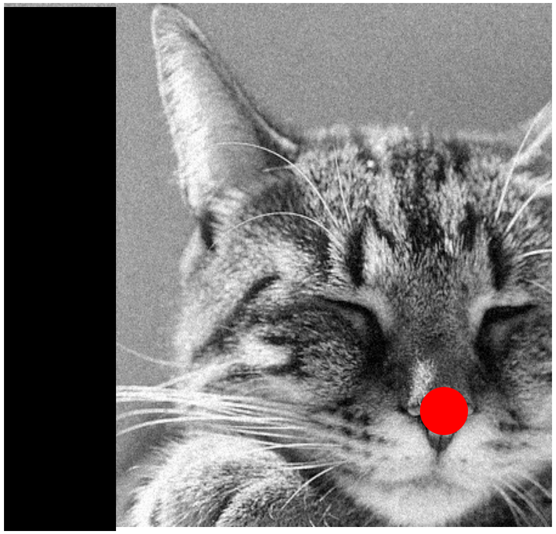
Shift invariance

$$f'(m,n) = f(m-m_0, n-n_0)$$

$$(w \otimes f')(m,n) = (w \otimes f)(m-m_0, n-n_0)$$

- Shift, then convolve = convolve, then shift
- Output of convolution does not depend on where the pixel is

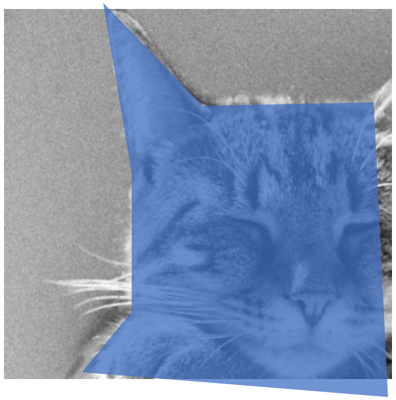
f



f'

Why is convolution important?

- Shift invariance is a crucial property

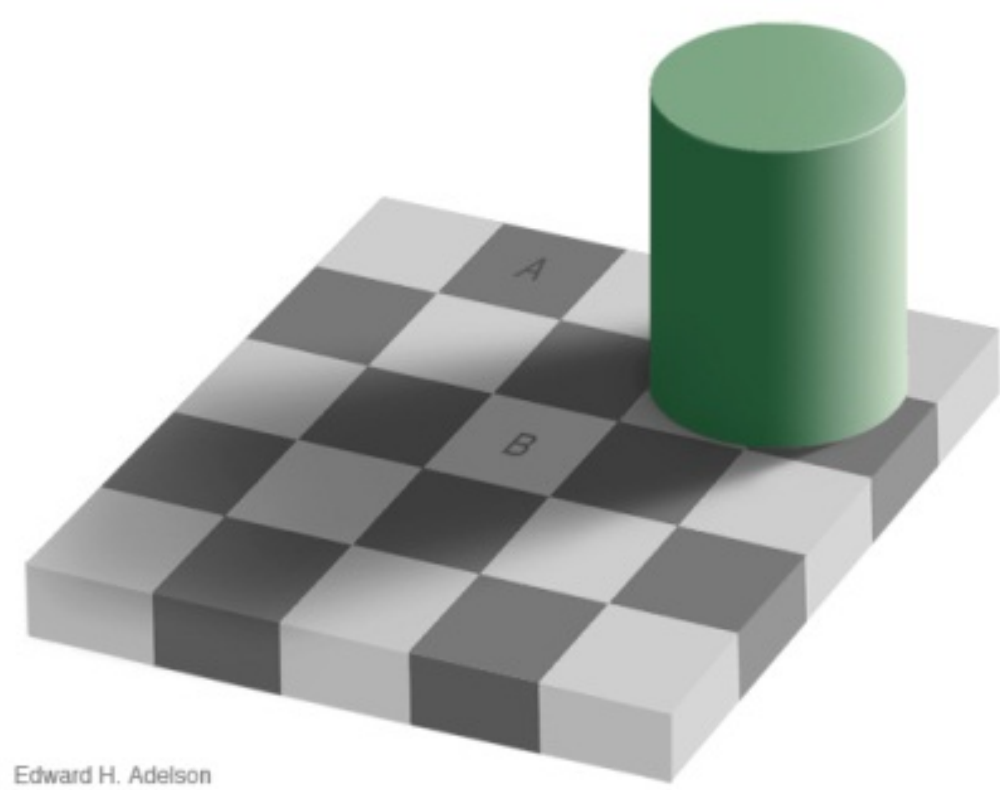


Why is convolution important?

- We *like* linearity
  - Linear functions behave predictably when input changes
  - Lots of theory just easier with linear functions
- *All linear shift-invariant systems can be expressed as a convolution*

Edges

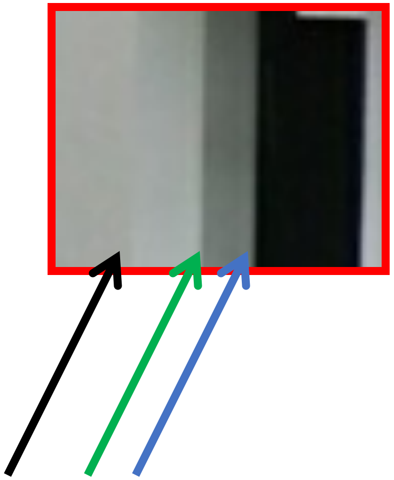
- Edges are curves in the image, across which the brightness changes “a lot”
- Corners/Junctions



Closeup of edges

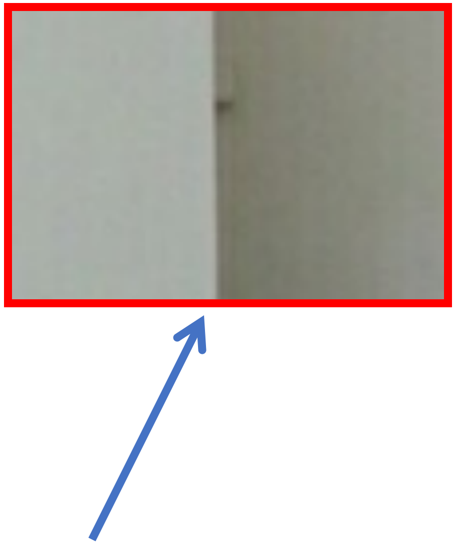
Source: D. Hoiem

Closeup of edges



Source: D. Hoiem

Closeup of edges



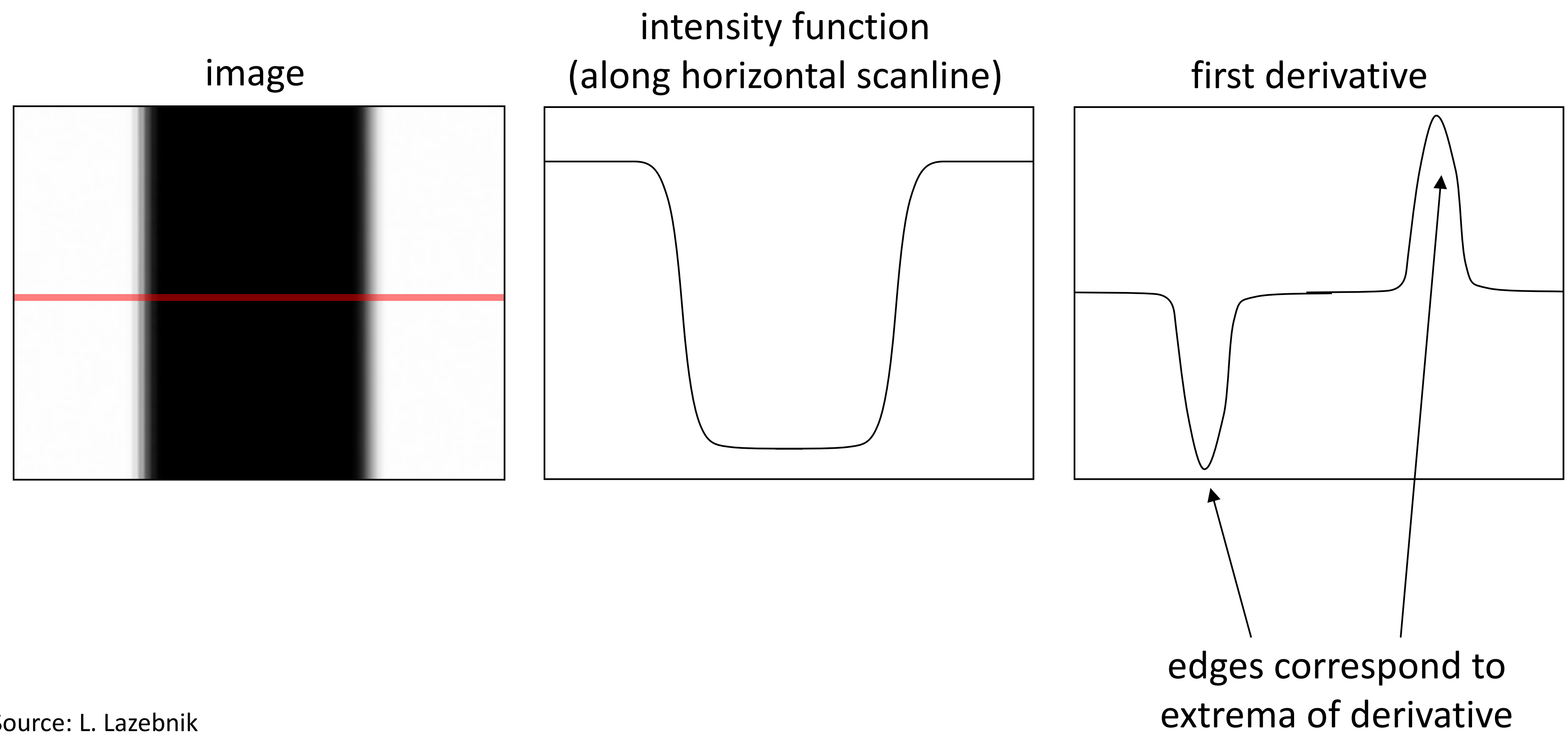
Source: D. Hoiem

Closeup of edges

Source: D. Hoiem

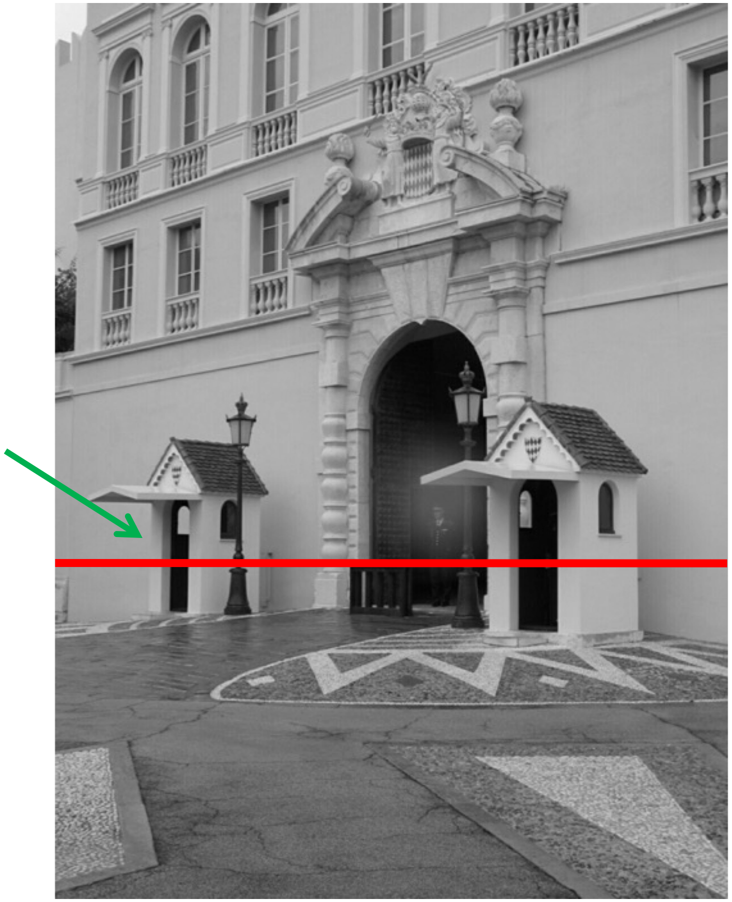
Characterizing edges

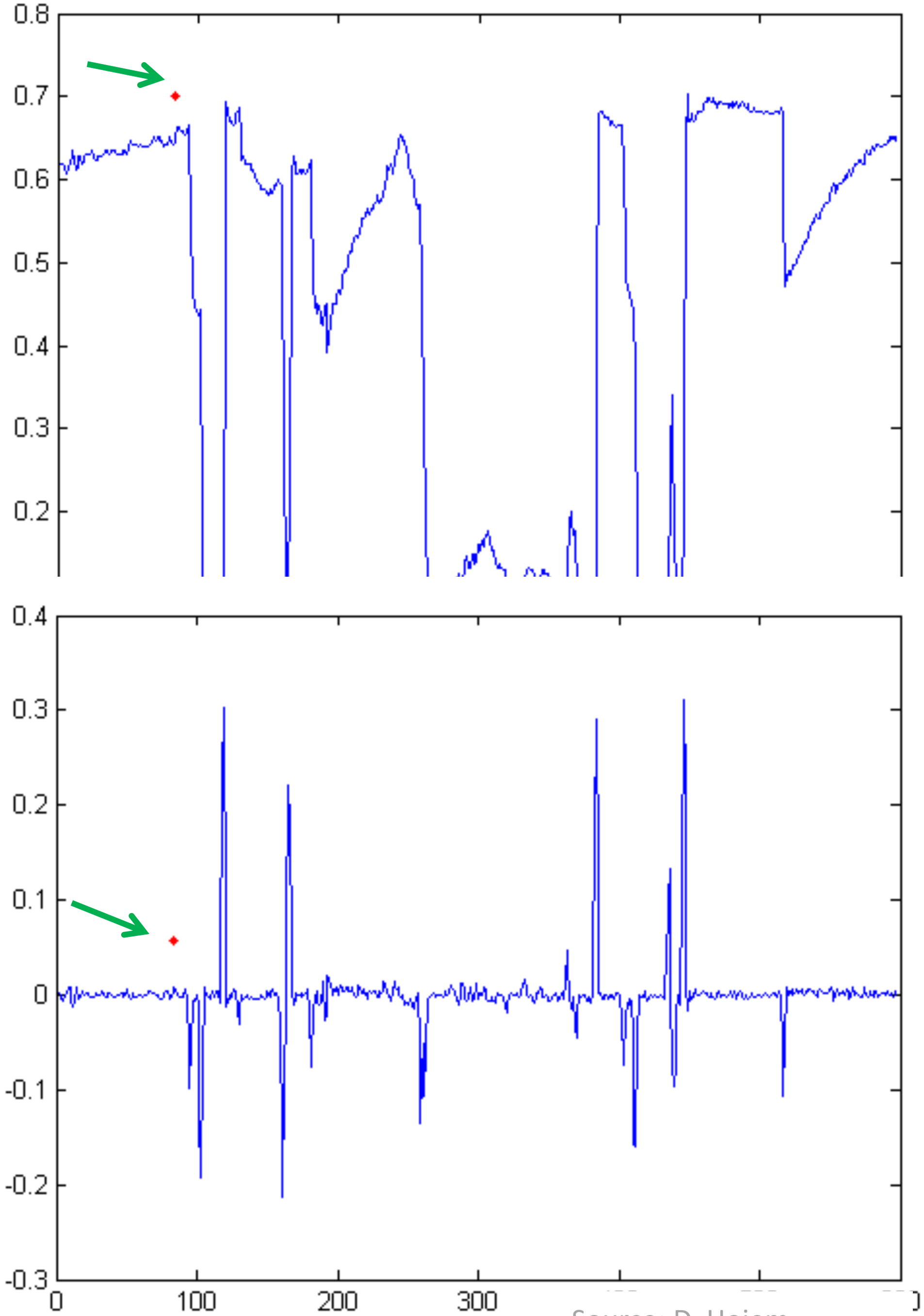
- An edge is a place of *rapid change* in the image intensity function



Source: L. Lazebnik

Intensity profile





Source: D. Hoiem

Derivatives and convolution

- Differentiation is *linear*

$$\frac{\partial(af(x) + bg(x))}{\partial x} = a\frac{\partial f(x)}{\partial x} + b\frac{\partial g(x)}{\partial x}$$

- Differentiation is *shift-invariant*
  - Derivative of shifted signal is shifted derivative

- Hence, differentiation can be represented as convolution!

Image derivatives

- How can we differentiate a *digital* image F[x,y]?
  – Option 1: reconstruct a continuous image, *f,* then compute the derivative
  – Option 2: take discrete derivative (finite difference)

$$\frac{\partial f}{\partial x}[x, y] \approx F[x+1, y] - F[x, y]$$

How would you implement this as a linear filter?

$\frac{\partial f}{\partial x}$:

H_x

$\frac{\partial f}{\partial y}$:

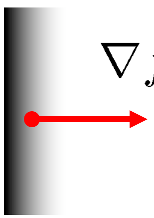
H_y

Source: S. Seitz

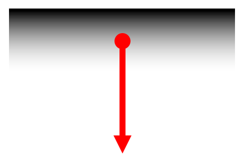
Image gradient

- The *gradient* of an image: $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

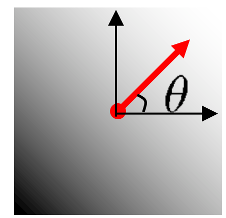
  The gradient points in the direction of most rapid increase in intensity



$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$



$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$



$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

The *edge strength* is given by the gradient magnitude:

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

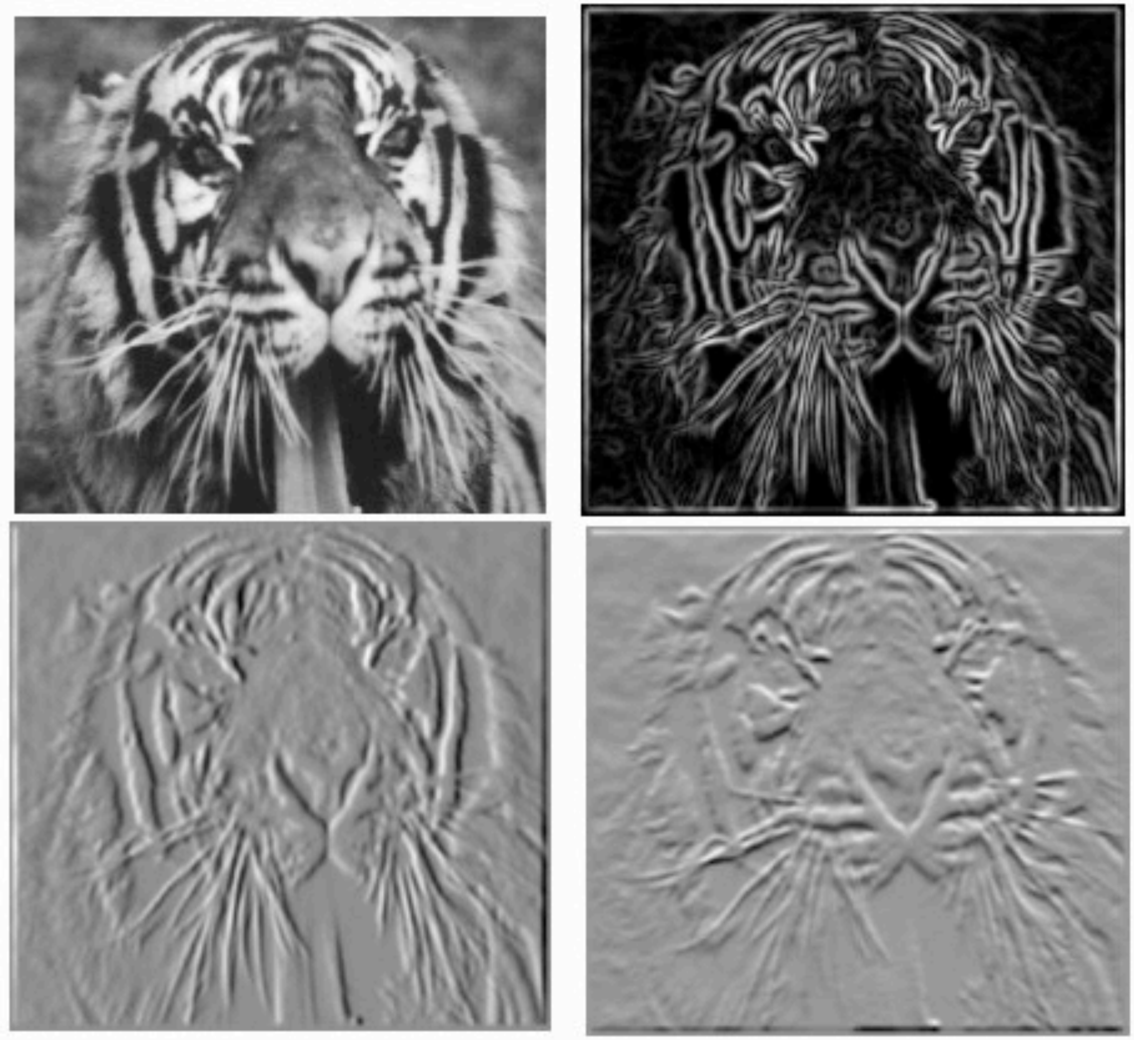
The gradient direction is given by:

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

- how does this relate to the direction of the edge?

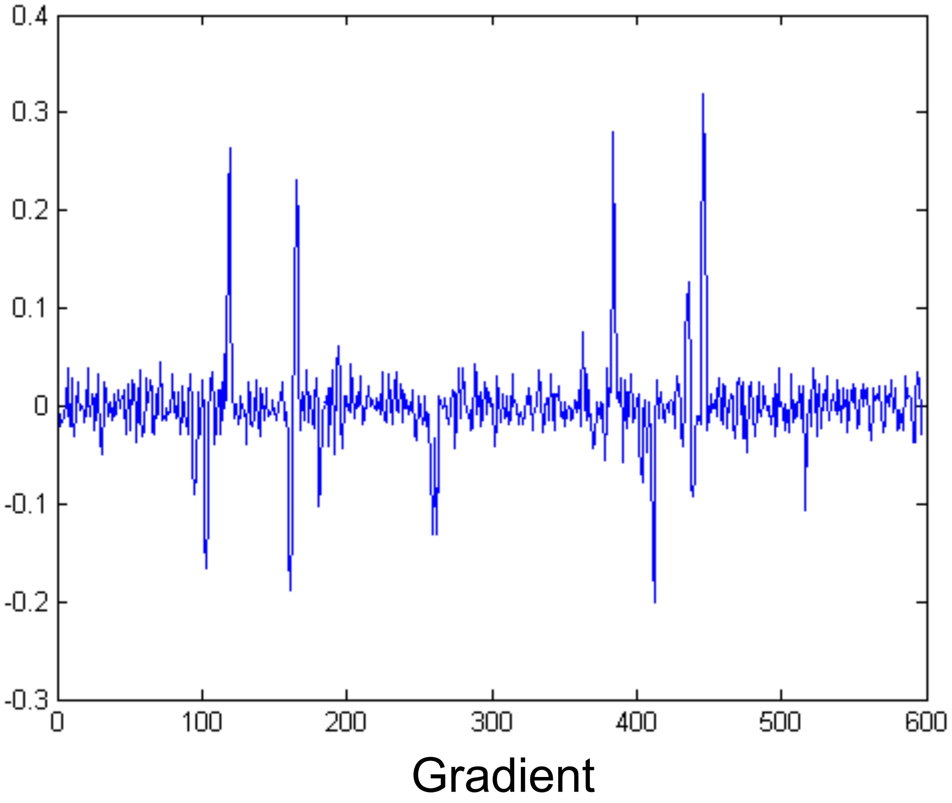
Source: Steve Seitz

Image gradient



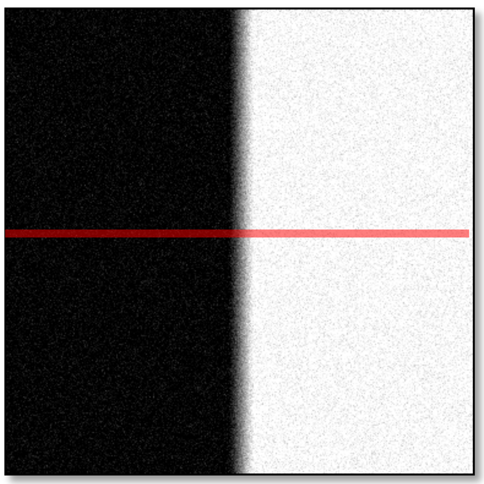
Source: L. Lazebnik

With a little Gaussian noise



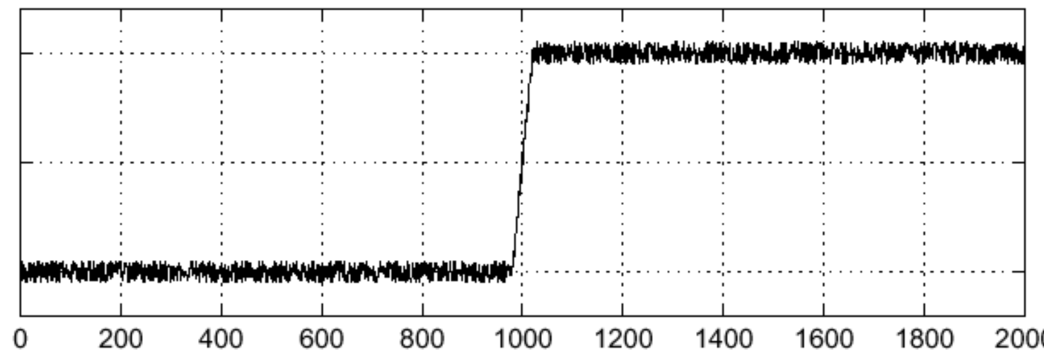
Source: D. Hoiem

Effects of noise

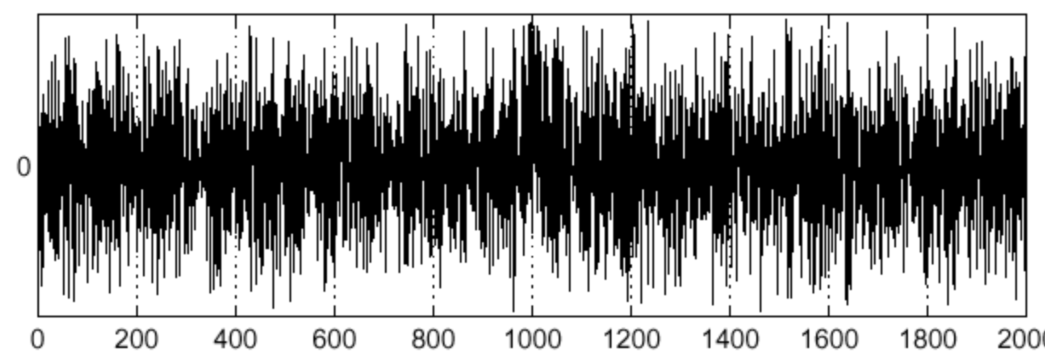


Noisy input image

$f(x)$



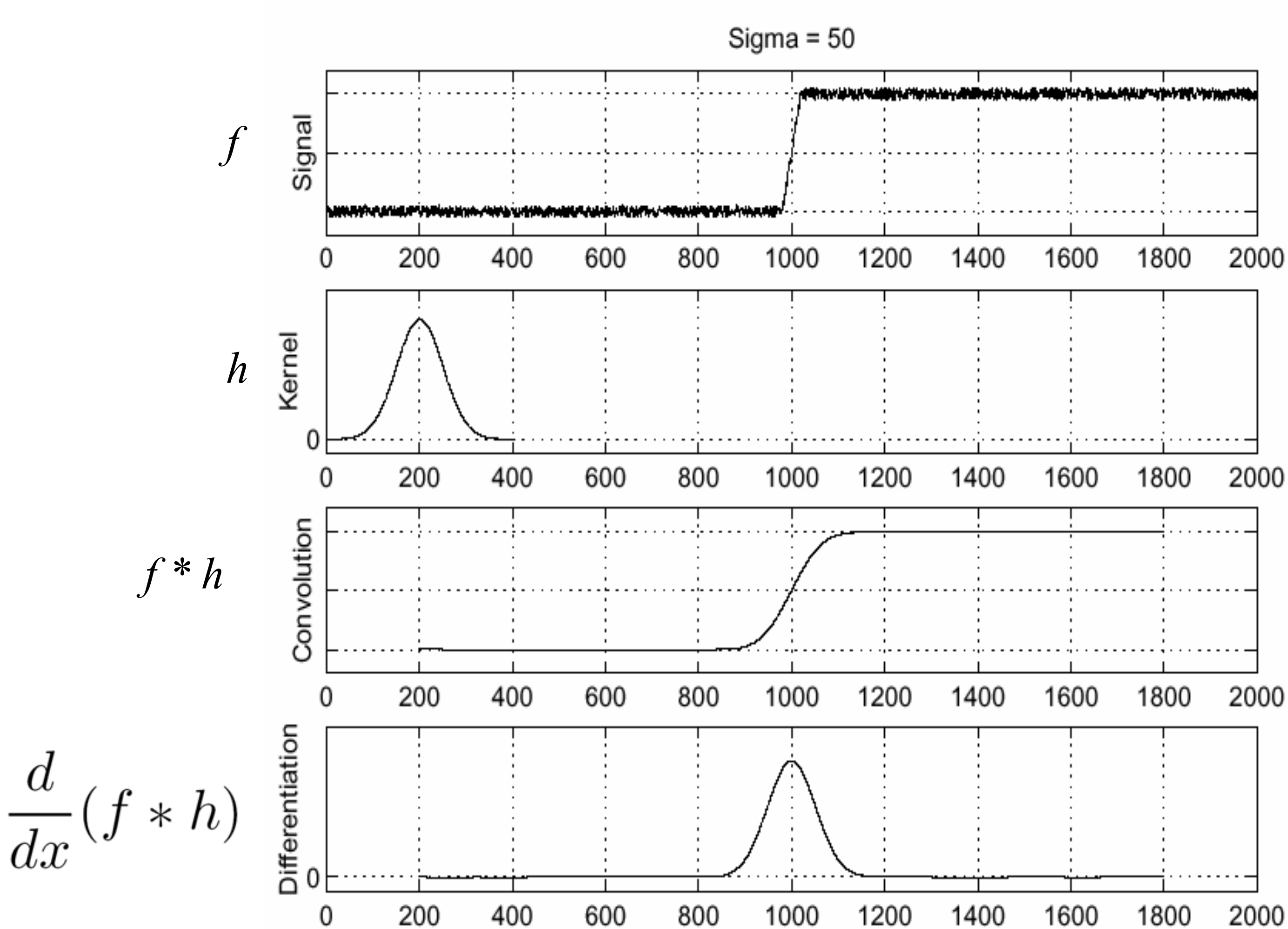
$\frac{d}{dx}f(x)$



Where is the edge?

Source: S. Seitz

Solution: smooth first



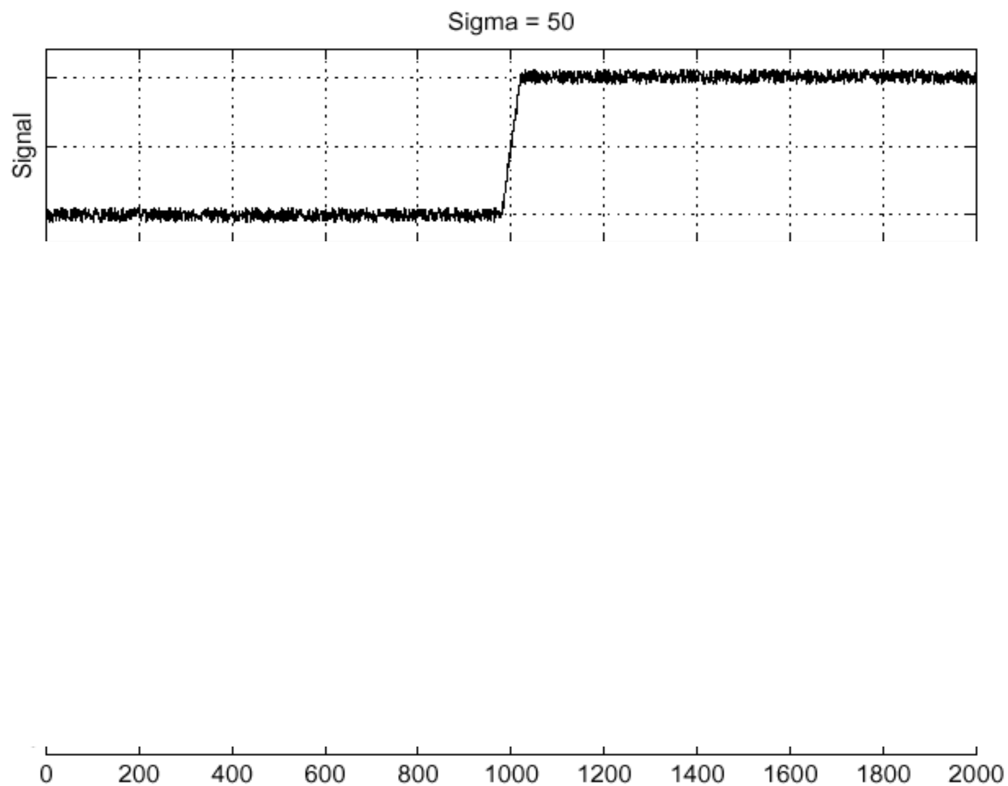
To find edges, look for peaks in $\frac{d}{dx}(f * h)$

Source: S. Seitz

Associative property of convolution

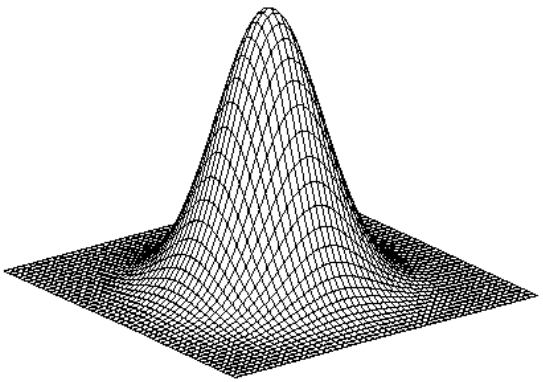
- Differentiation is a convolution
- Convolution is associative: $\frac{d}{dx}(f * h) = f * \frac{d}{dx}h$
- This saves us one operation:

f



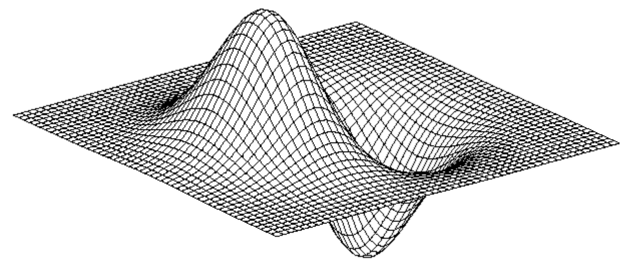
Source: S. Seitz

2D edge detection filters



Gaussian

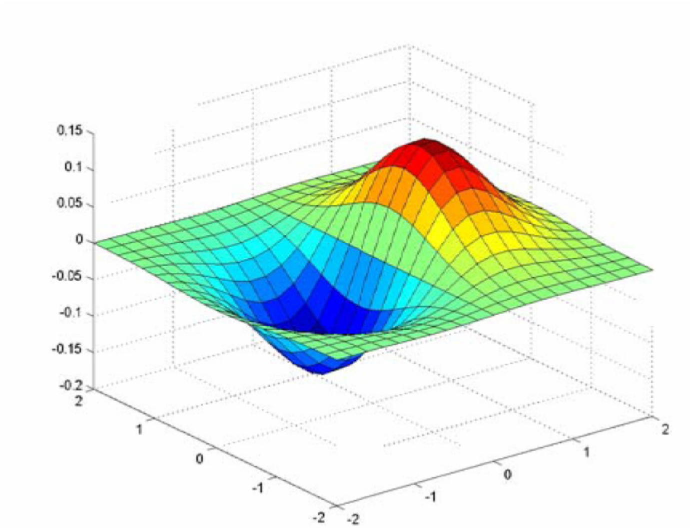
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



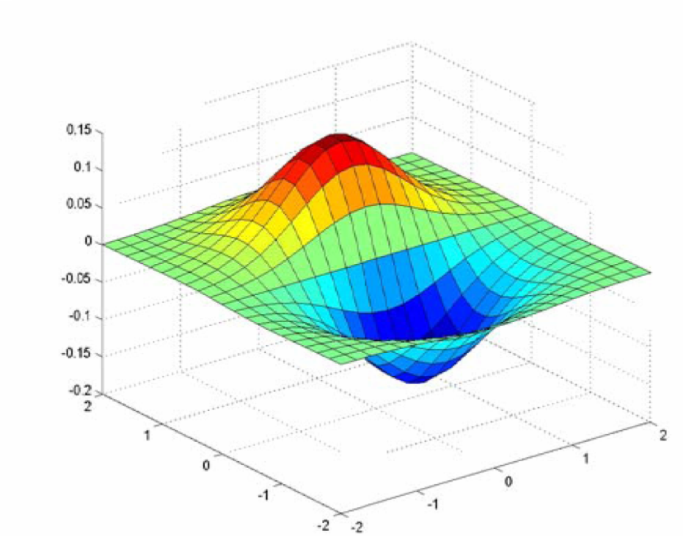
derivative of Gaussian (x)

$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

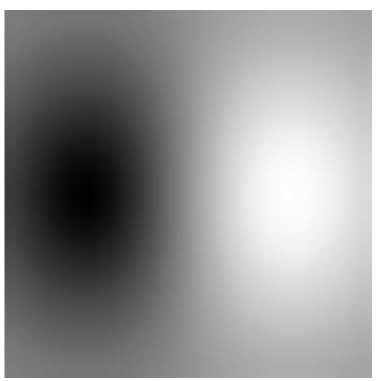
Derivative of Gaussian filter

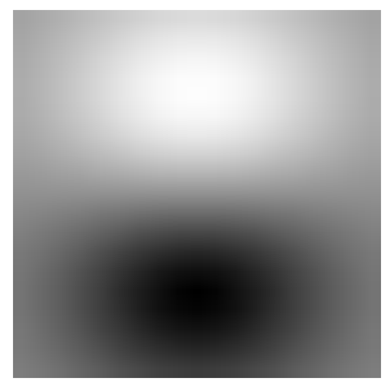


x-direction



y-direction





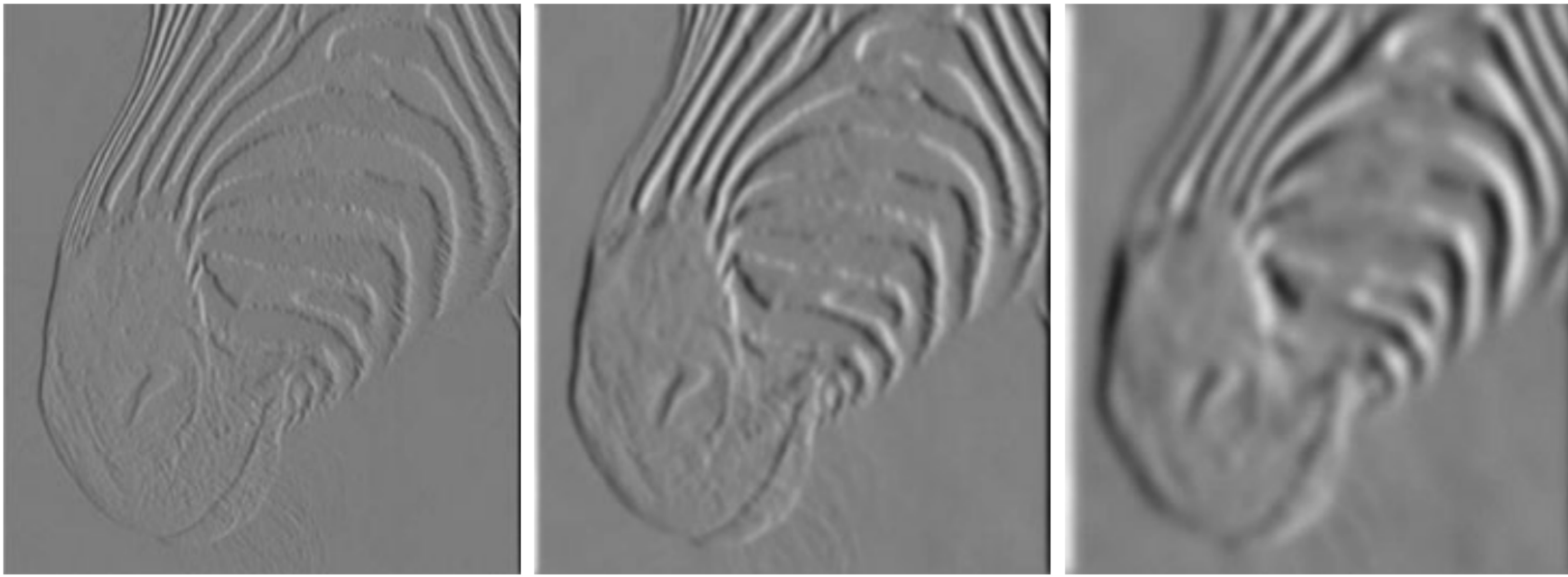


FIGURE 5.3: The scale (i.e., σ) of the Gaussian used in a derivative of Gaussian filter has significant effects on the results. The three images show estimates of the derivative in the x direction of an image of the head of a zebra obtained using a derivative of Gaussian filter with σ one pixel, three pixels, and seven pixels (**left** to **right**). Note how images at a finer scale show some hair, the animal's whiskers disappear at a medium scale, and the fine stripes at the top of the muzzle disappear at the coarser scale.

Two Dimensional Gaussian

Anisotropic: $G_{\sigma_x,\sigma_y}(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2}+\frac{y^2}{\sigma_y^2}\right)}$

Isotropic: $G_{\sigma}(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{r^2}{2\sigma^2}}$

Oriented Gaussian First and Second Derivatives

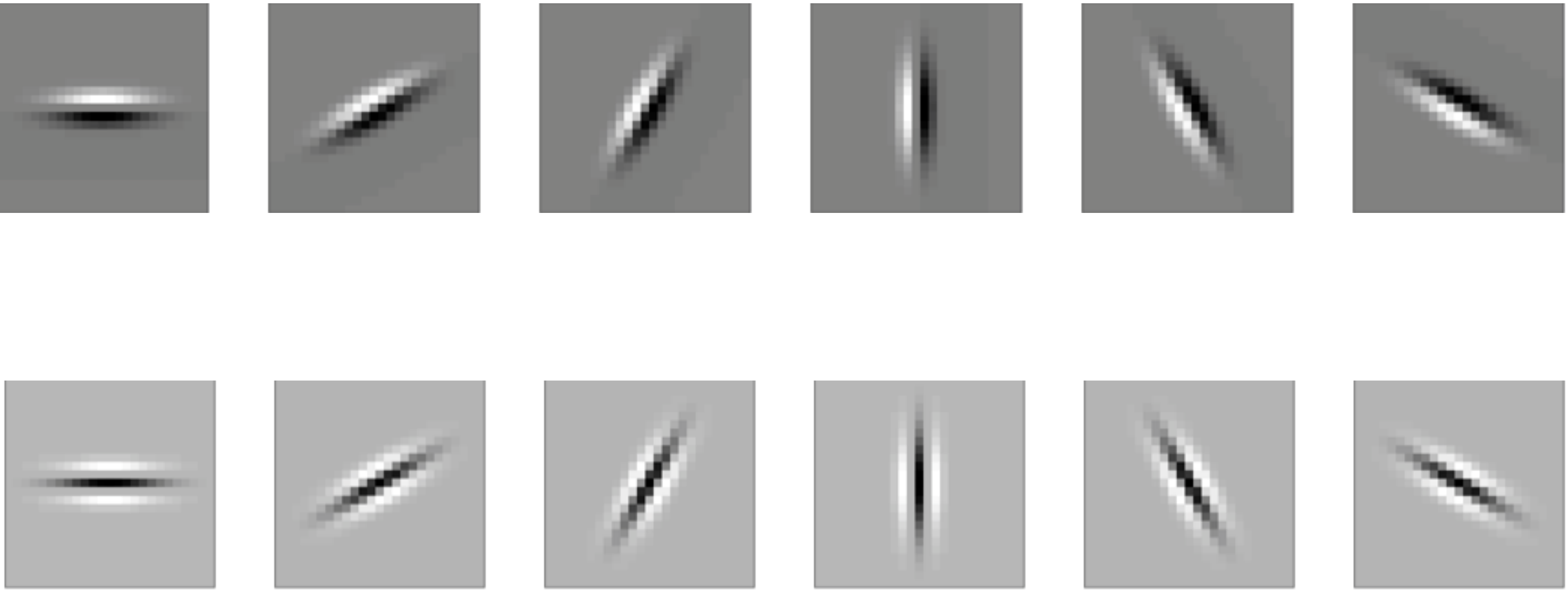
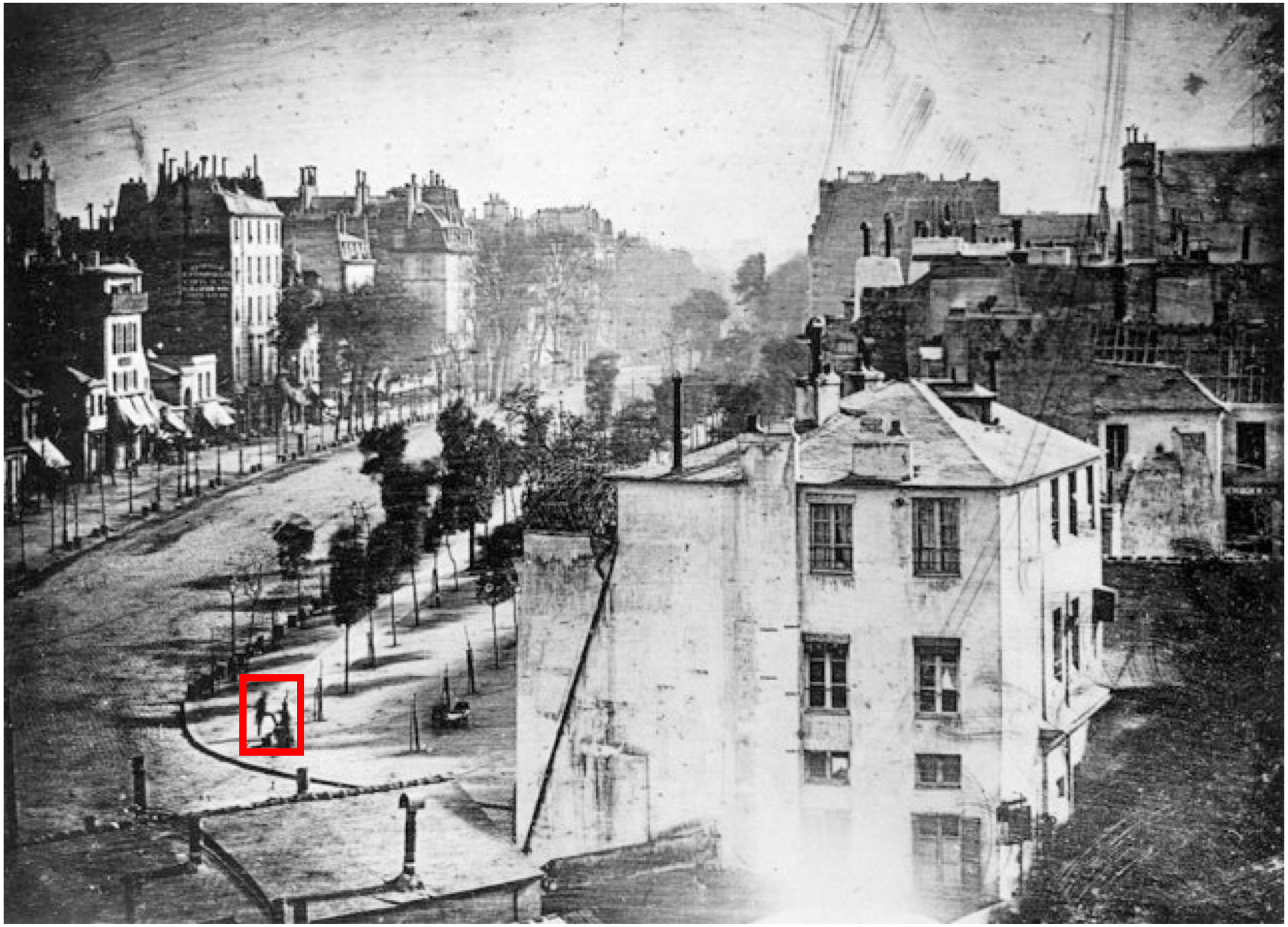


Image resizing

Image resizing

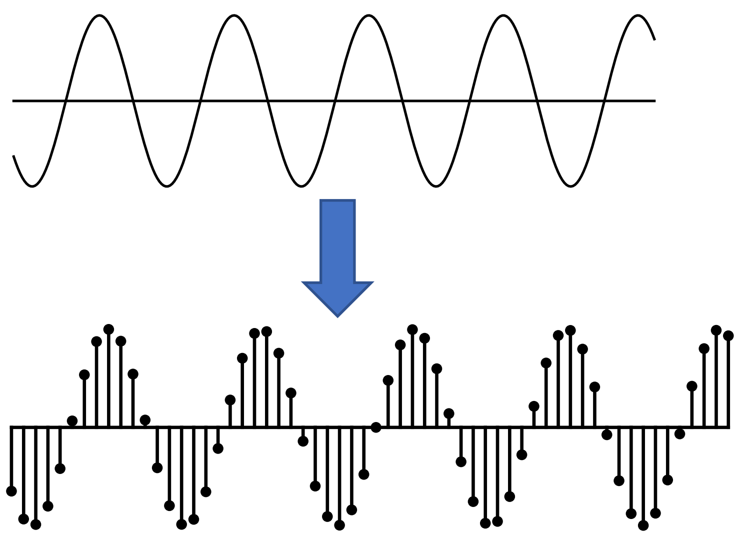
Let’s enhance!



Louis Daguerre, 1838

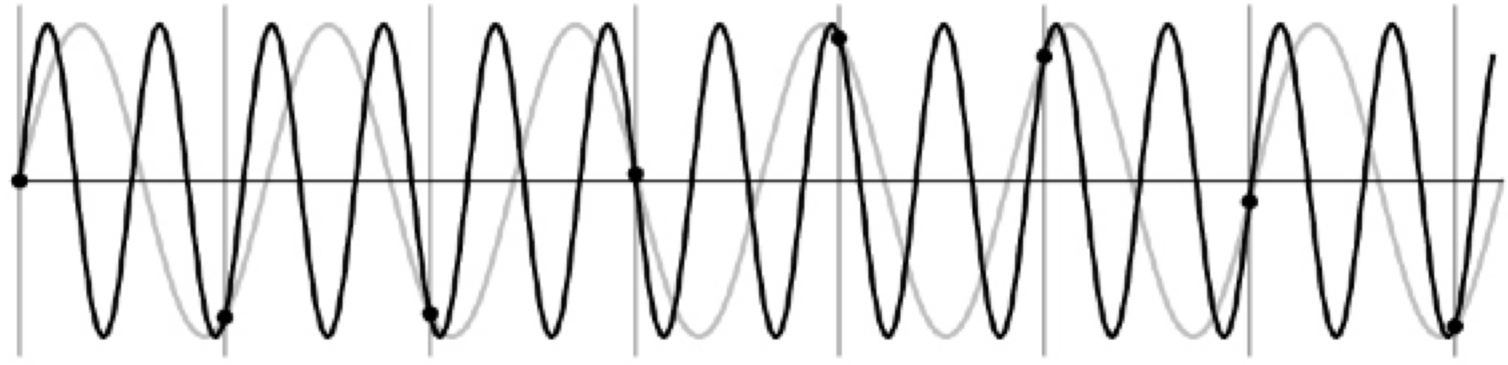
What is a (digital) image?

- True image is a function from R^2 to R
- Digital image is a sample from it
- 1D example:



- To enhance, we need to recover the original signal and sample again

Undersampling

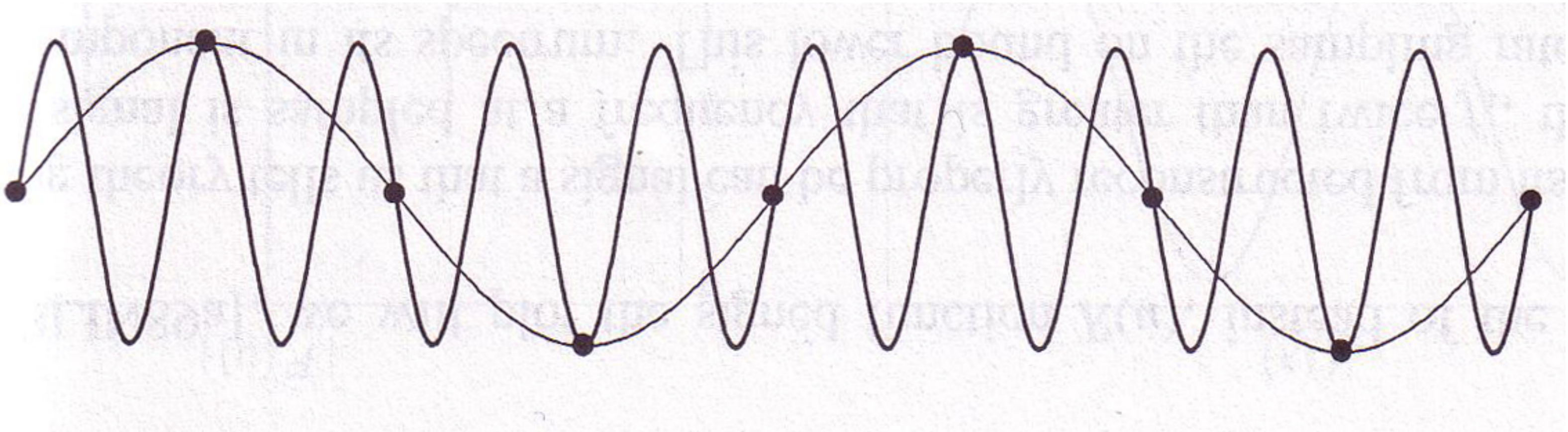


Undersampling

- What if we "missed" things between the samples?
- Simple example: undersampling a sine wave
  - unsurprising result: information is lost
  - surprising result: indistinguishable from lower frequency
  - also was always indistinguishable from higher frequencies
  - *aliasing*: signals "traveling in disguise" as other frequencies

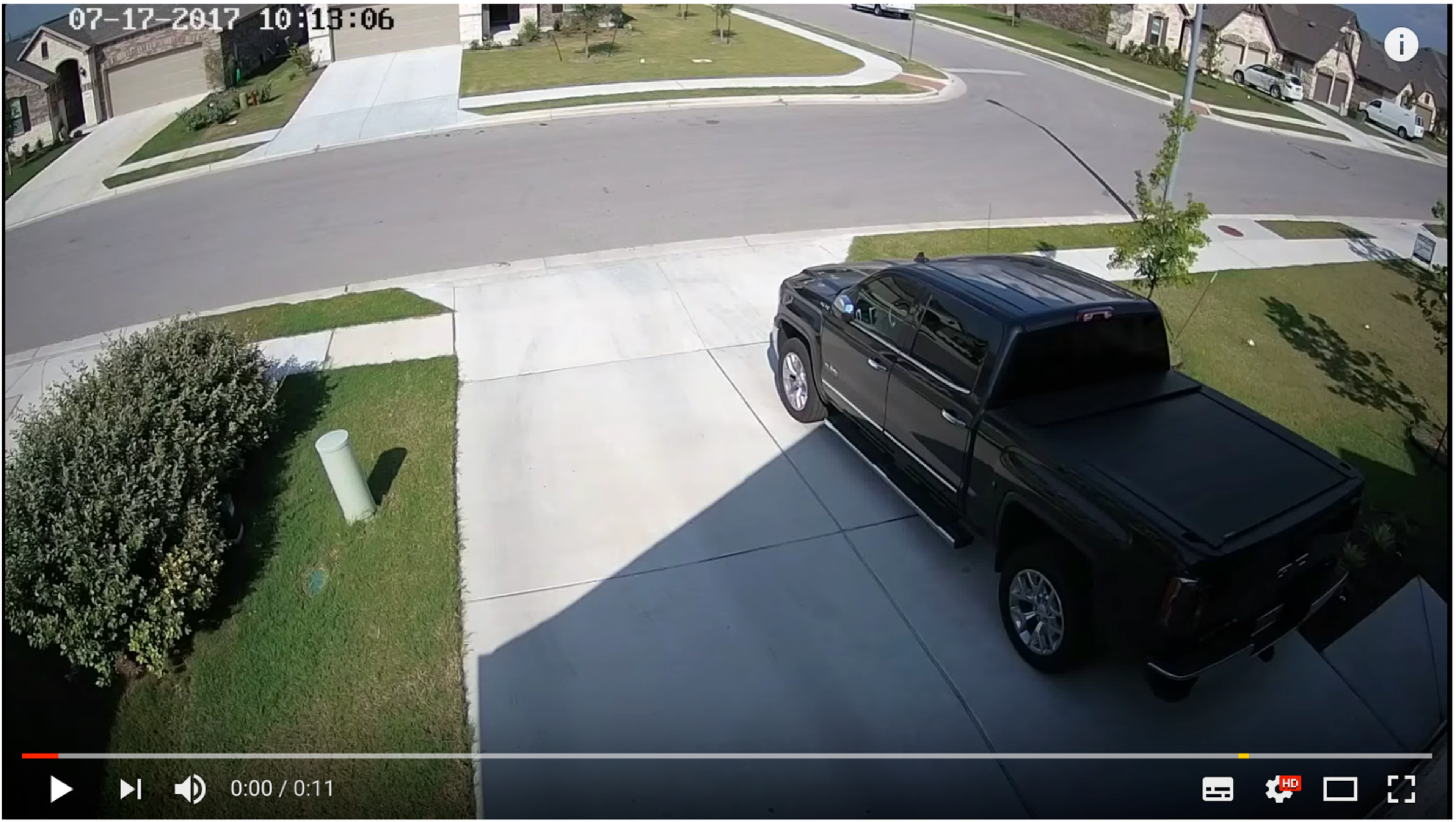
Aliasing

- When sampling is not adequate, impossible to distinguish between low and high frequency signal



Aliasing in time

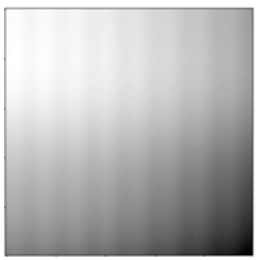
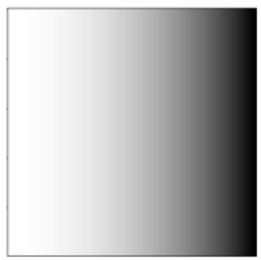
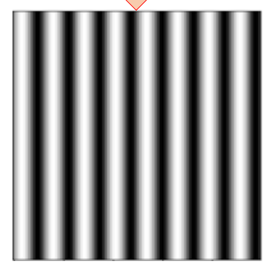
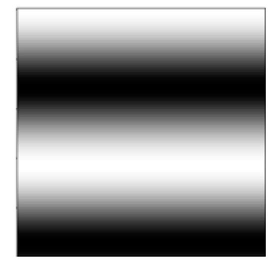
Aliasing in time

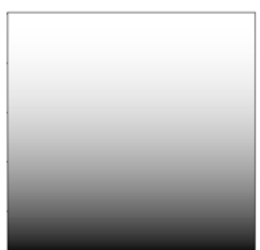
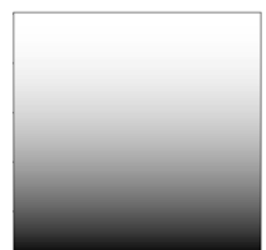


Beyond sines and cosines

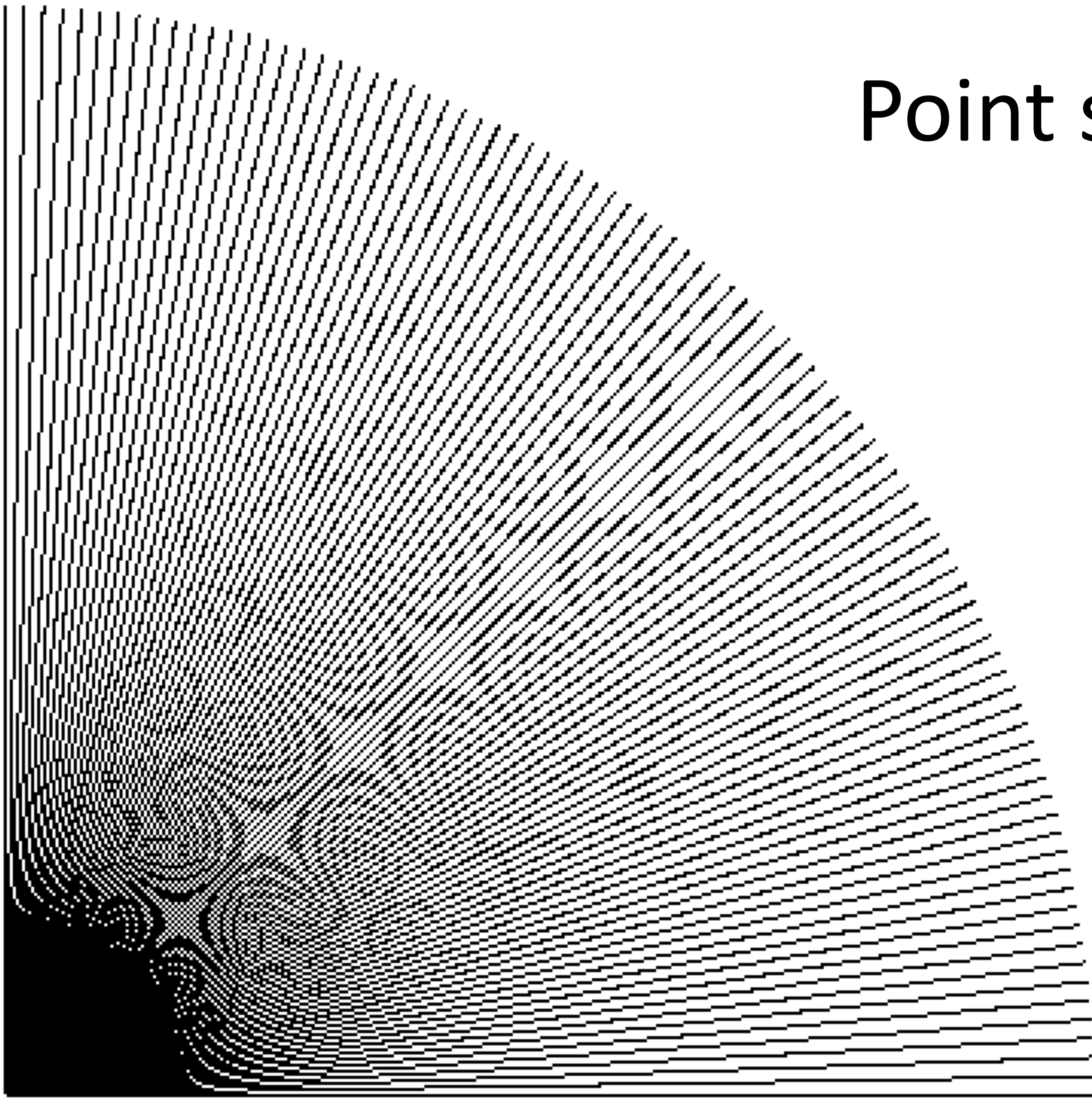
- Images are not sines and
- But they can be written a … es and cosines (**Fourier Transform**)

High frequency components

 = 30*  + 0.5*  - 

+ 100*  +10* 

Point sampling in action



Aliasing

- High frequency components when downsampled *masquerade* as low frequency components
- Key step: remove high frequency components. But how?

Smoothing and image frequencies

- Smoothing makes a pixel more like its neighbors
- Image intensities change more slowly with x and y in smoothed image
- ➔ Smoothing *removes high frequencies*

Subsampling images

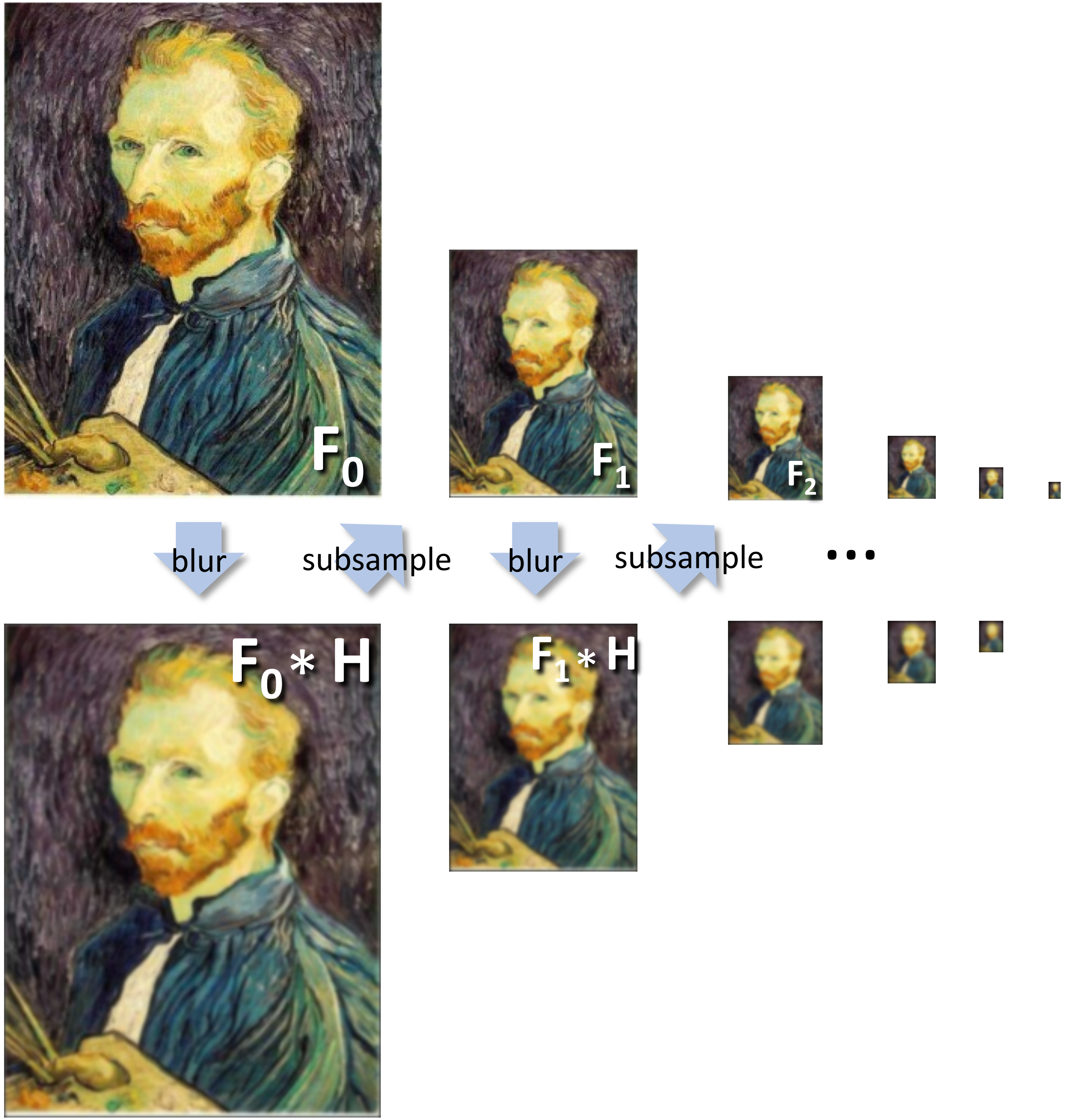
- Step 1: Convolve with Gaussian to eliminate high frequencies
- Step 2: Drop unneeded pixels

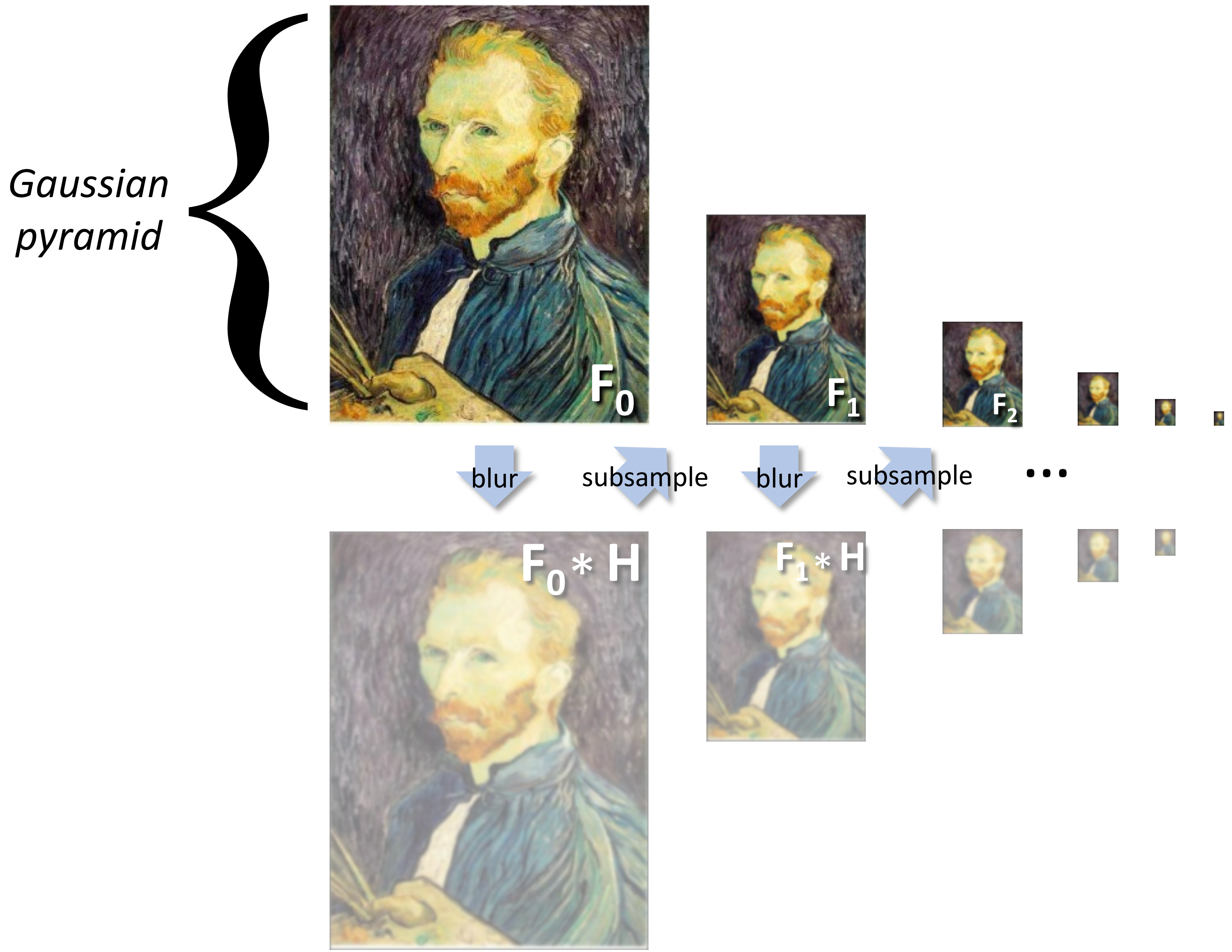
Subsampling without removing high frequencies

Subsampling after removing high frequencies

Gaussian pre-filtering

- Solution: filter the image, *then* subsample




Gaussian
pyramid
F0
F1
F2
blur
subsample
blur
subsample
...
F0 * H
F1 * H

Why Gaussian pyramids?

- The same neighborhood size captures differently-sized image patches
- Search for multiple sizes of an object using the same template!

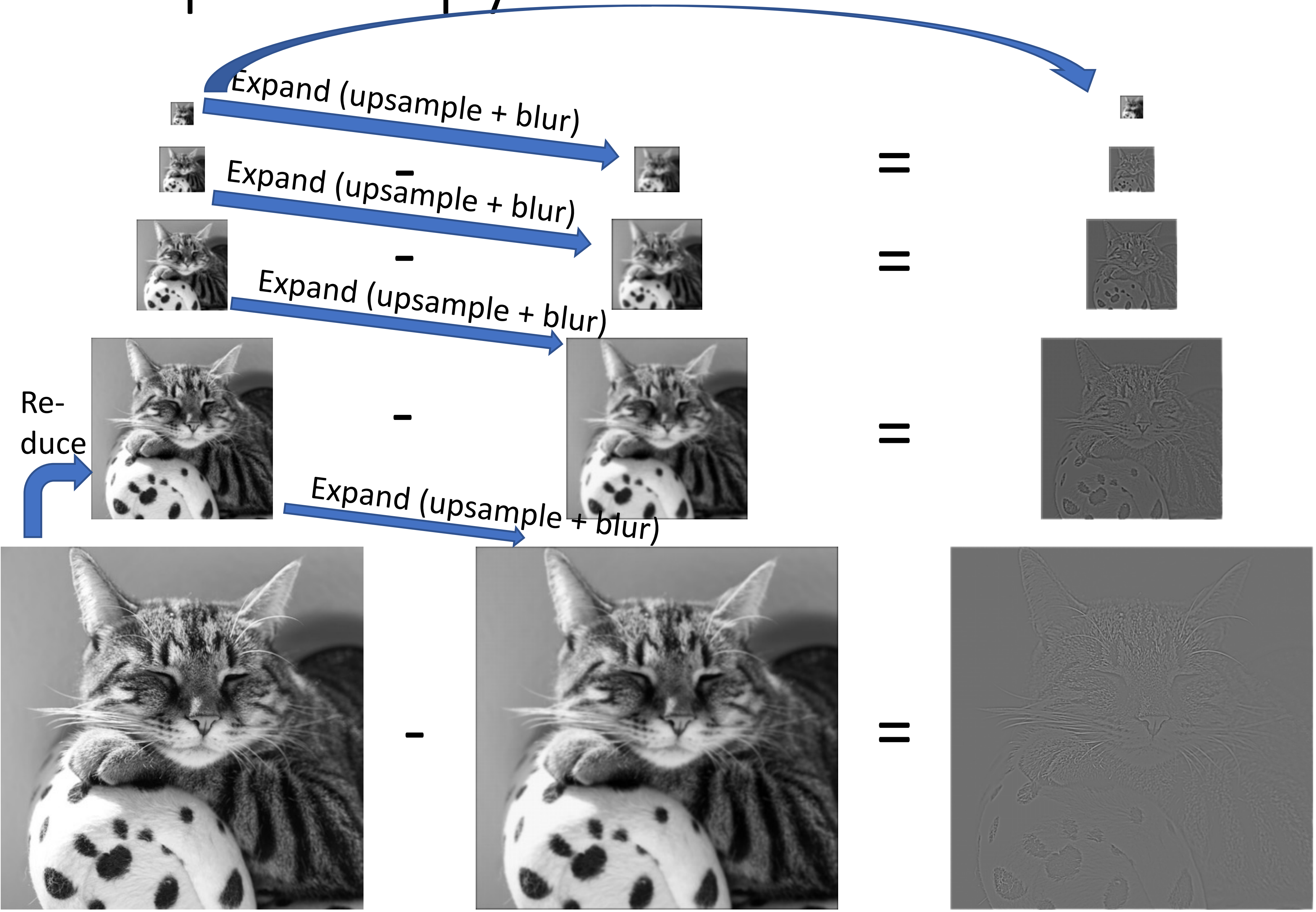
Upsampling images

Step 1: blow up to original size with 0's in between

Upsampling images

Step 2: Convolve with Gaussian to interpolate

Laplacian pyramid

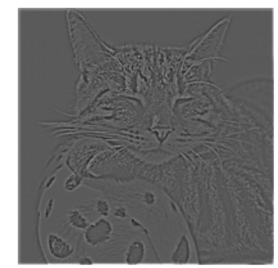


Laplacian pyramid

$L_4 = G_4 =$ 

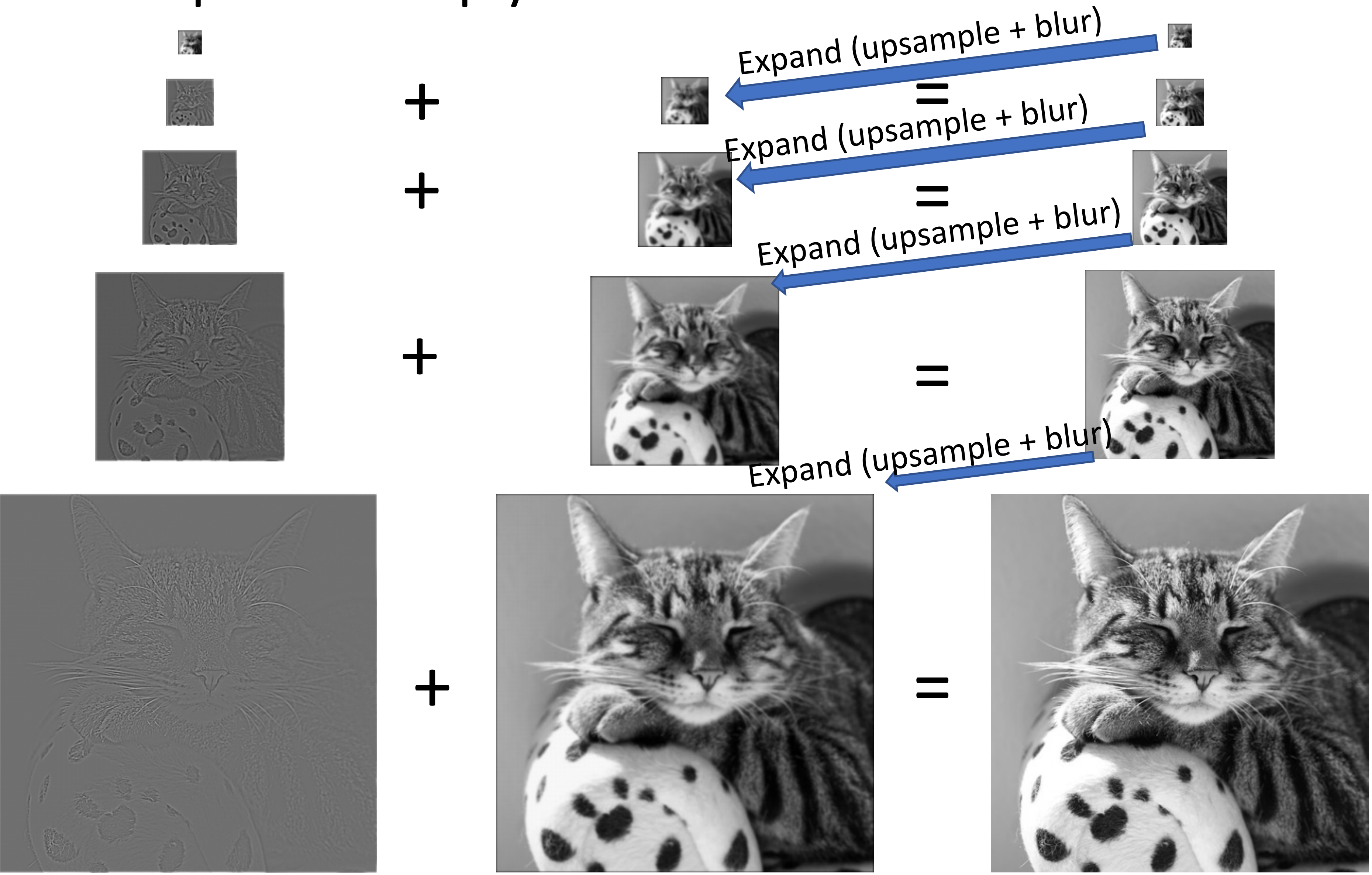
$L_3 = G_3 - \text{expand}(G_4) =$ 

$L_2 = G_2 - \text{expand}(G_3) =$ 

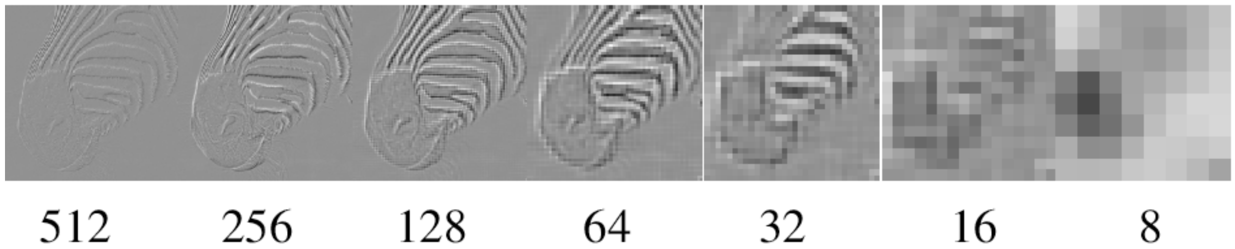
$L_1 = G_1 - \text{expand}(G_2) =$ 

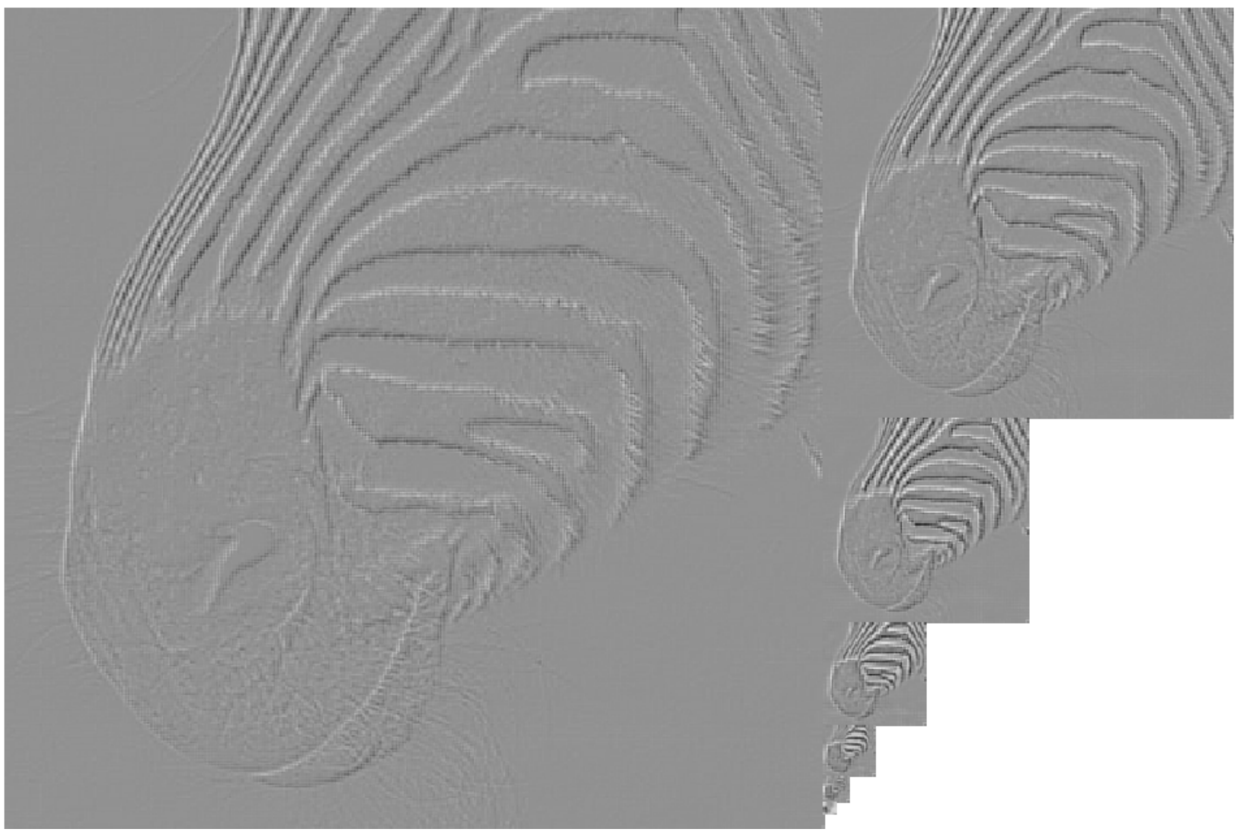
$L_0 = G_0 - \text{expand}(G_1) =$ 

Reconstructing the image from a Laplacian pyramid



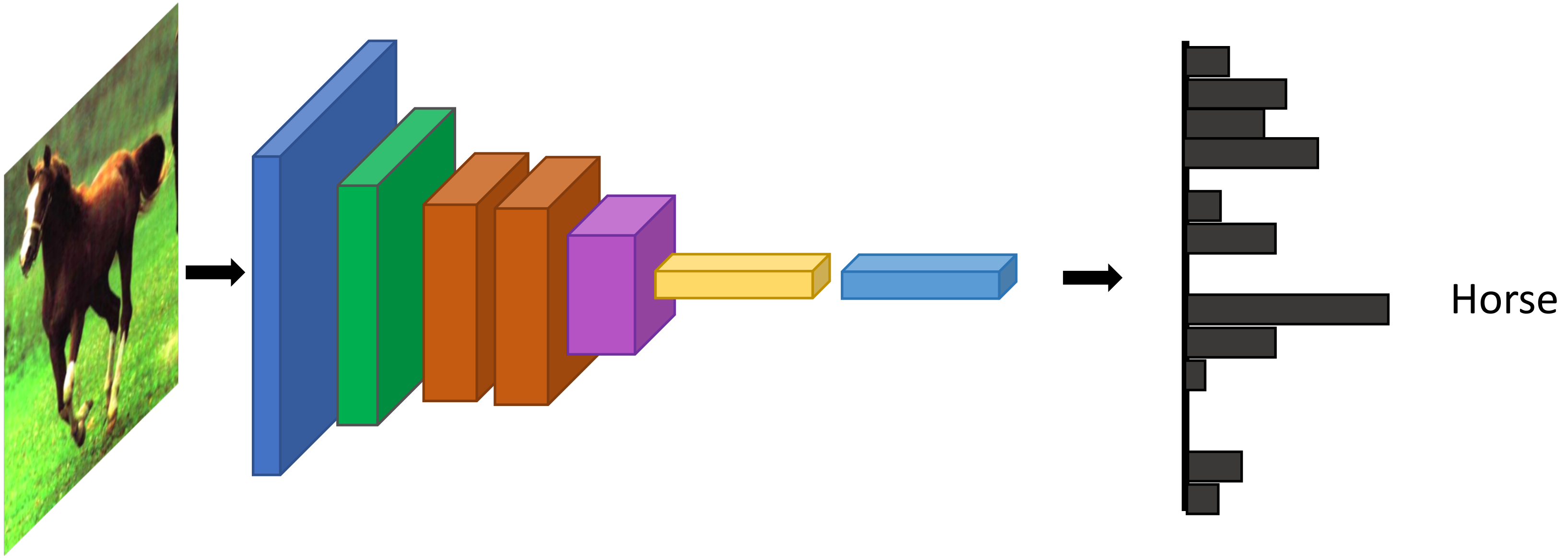
Laplacian pyramid





Source: Forsyth

Convolution and subsampling is familiar…



Key take-away

- A versatile tool : convolution
  - Any linear shift-invariant operation is a convolution
  - In particular edge detection, image smoothing
- A versatile structure : pyramids
  - Early layers capture low-level detail, higher layers capture global structure.