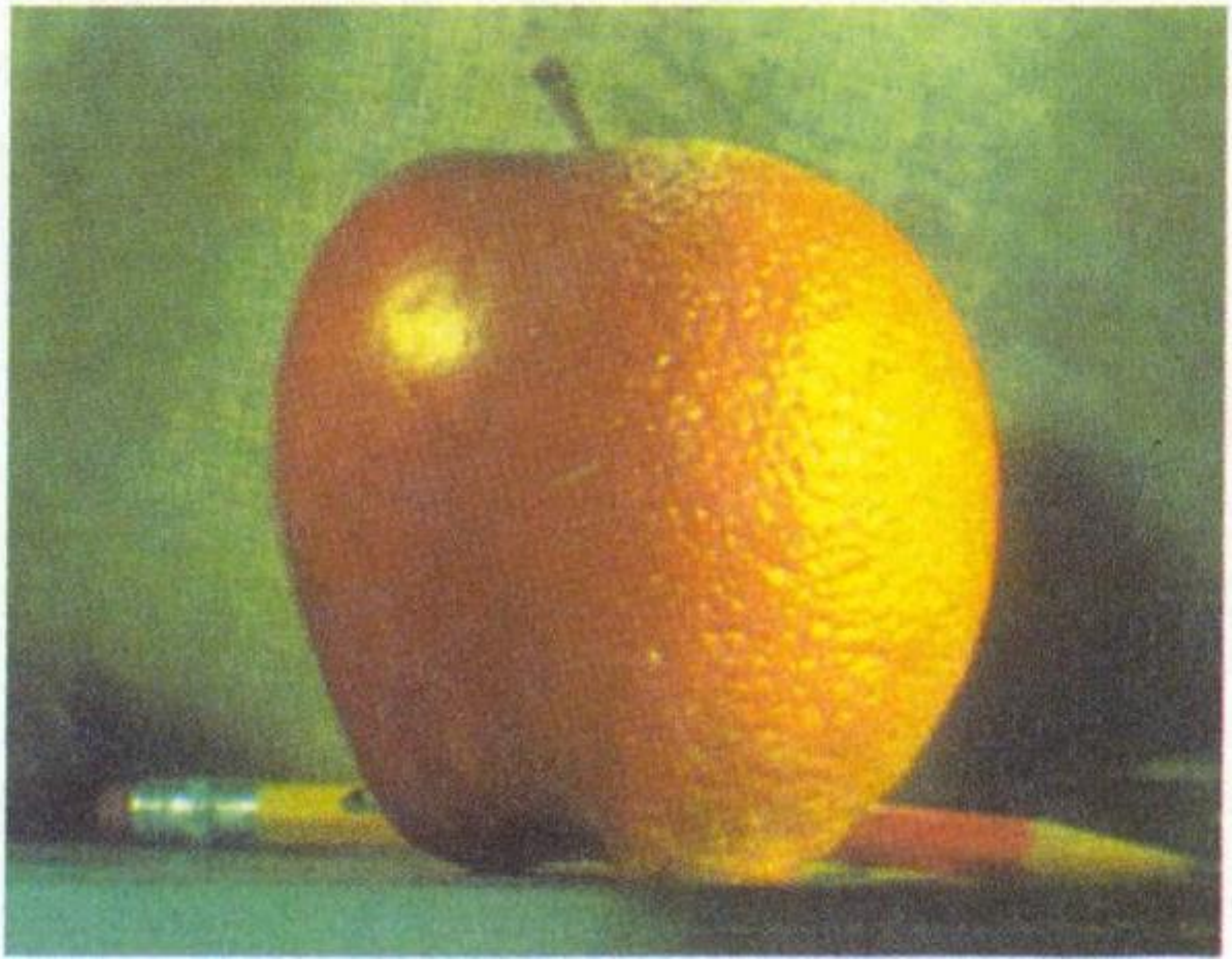
Pyramid Blending, Templates, NL Filters



CS180: Intro to Comp. Vision and Comp. Photo
Alexei Efros & Angjoo Kanazawa, UC Berkeley, Fall 2023

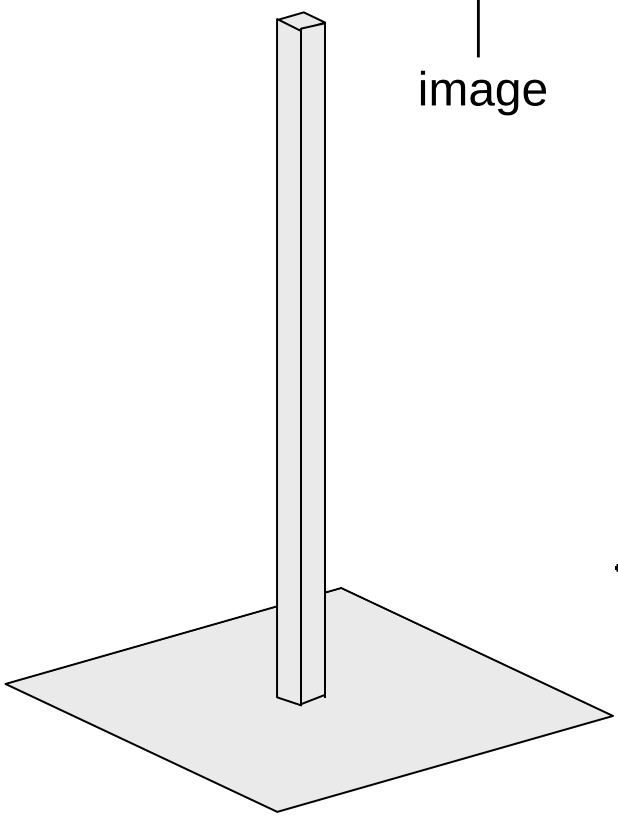
Unsharp mask filter

$$f + \alpha(f - f * g) = (1+\alpha)f - \alpha f * g = f * ((1+\alpha)e - \alpha g)$$

image

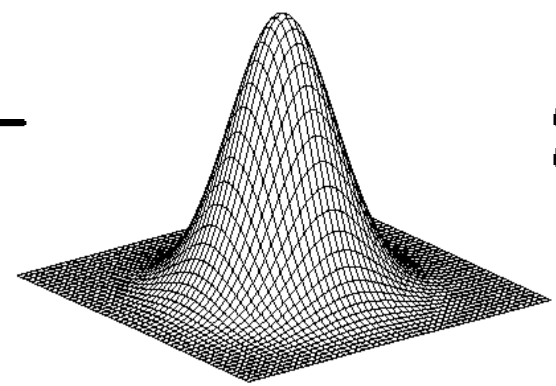
blurred image

unit impulse (identity)



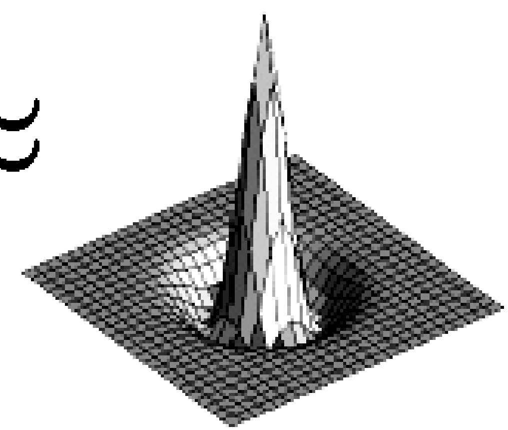
unit impulse

$-$



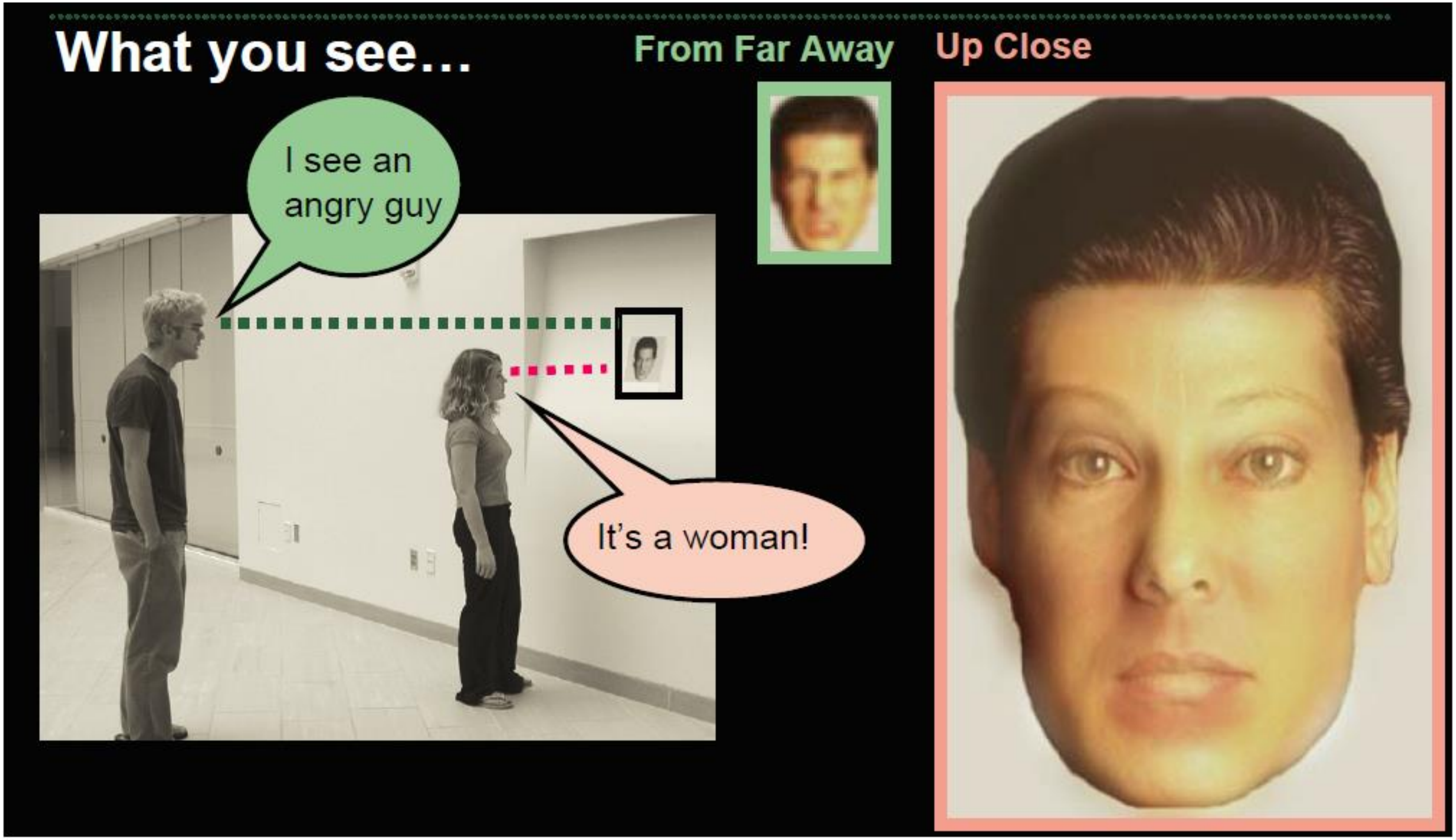
Gaussian

$\approx$



Laplacian of Gaussian

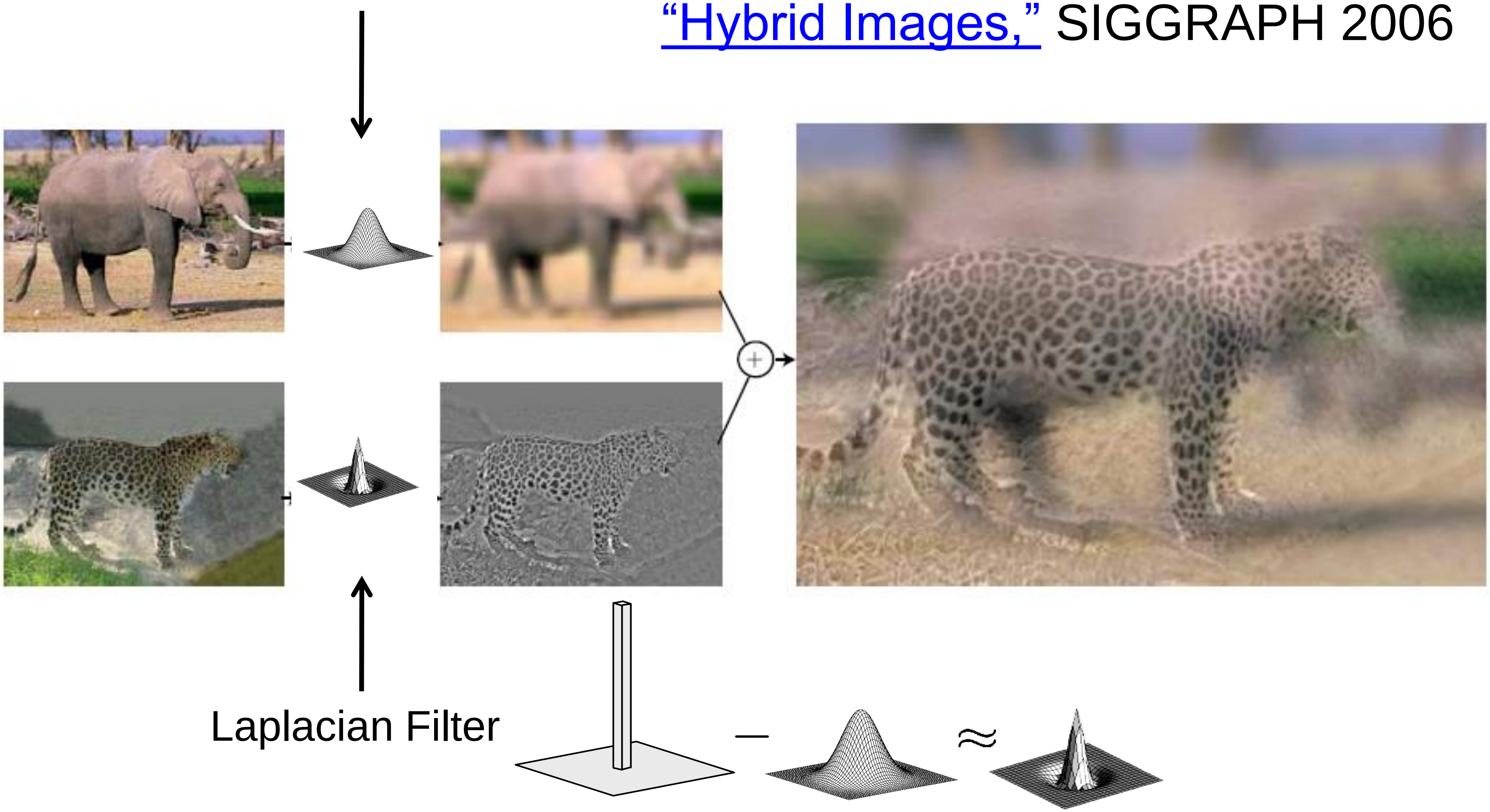
application: Hybrid Images



Aude Oliva & Antonio Torralba & Philippe G Schyns, SIGGRAPH 2006

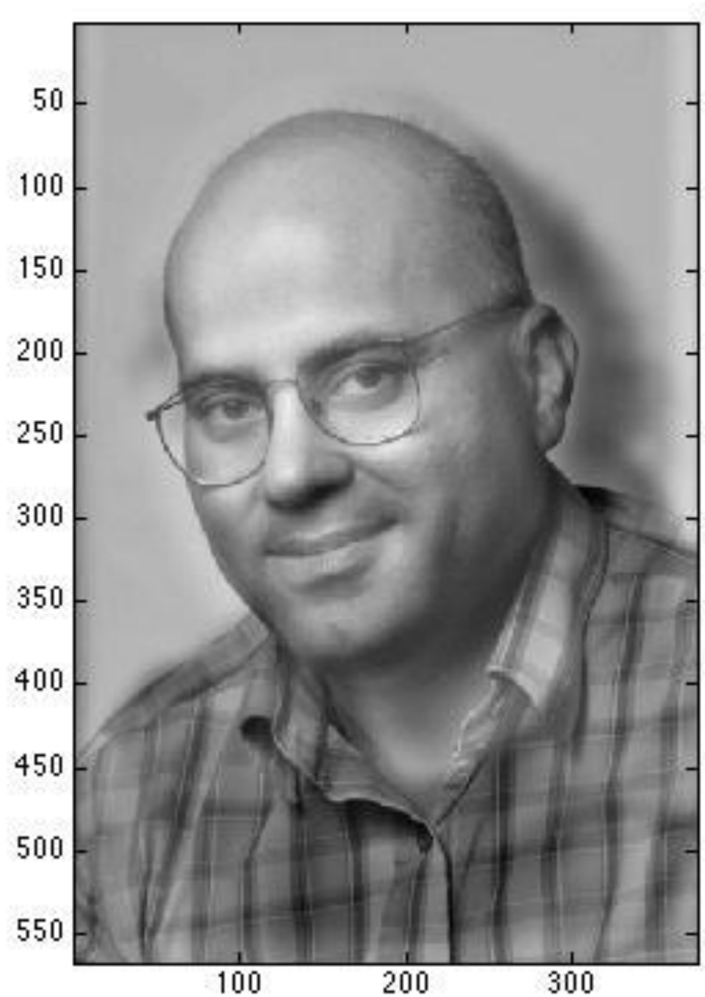
Application: Hybrid Images

A. Oliva, A. Torralba, P.G. Schyns, [“Hybrid Images,”] SIGGRAPH 2006



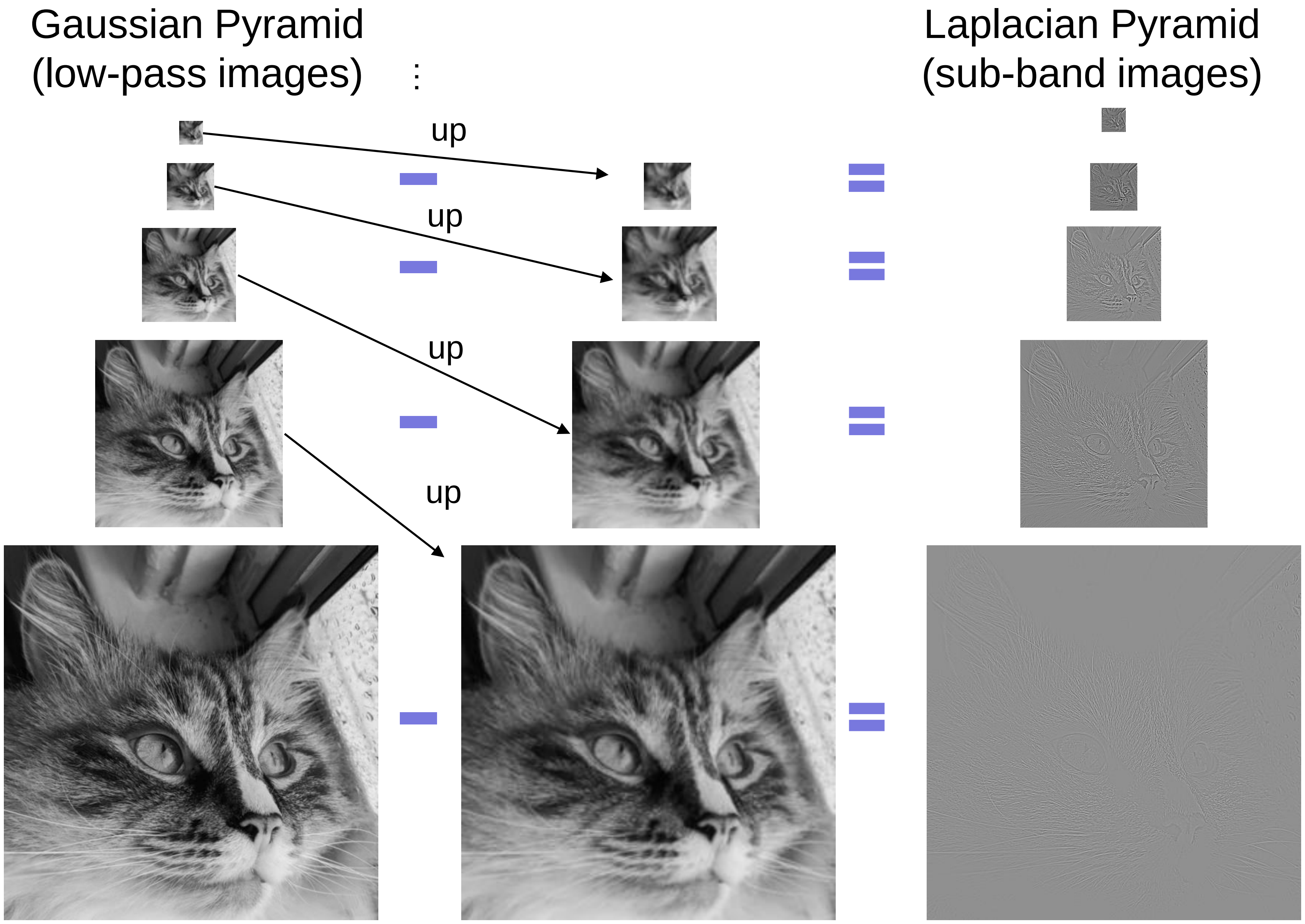
Yestaryear's homework

(CS194-26: Riyaz Faizullabhoy)



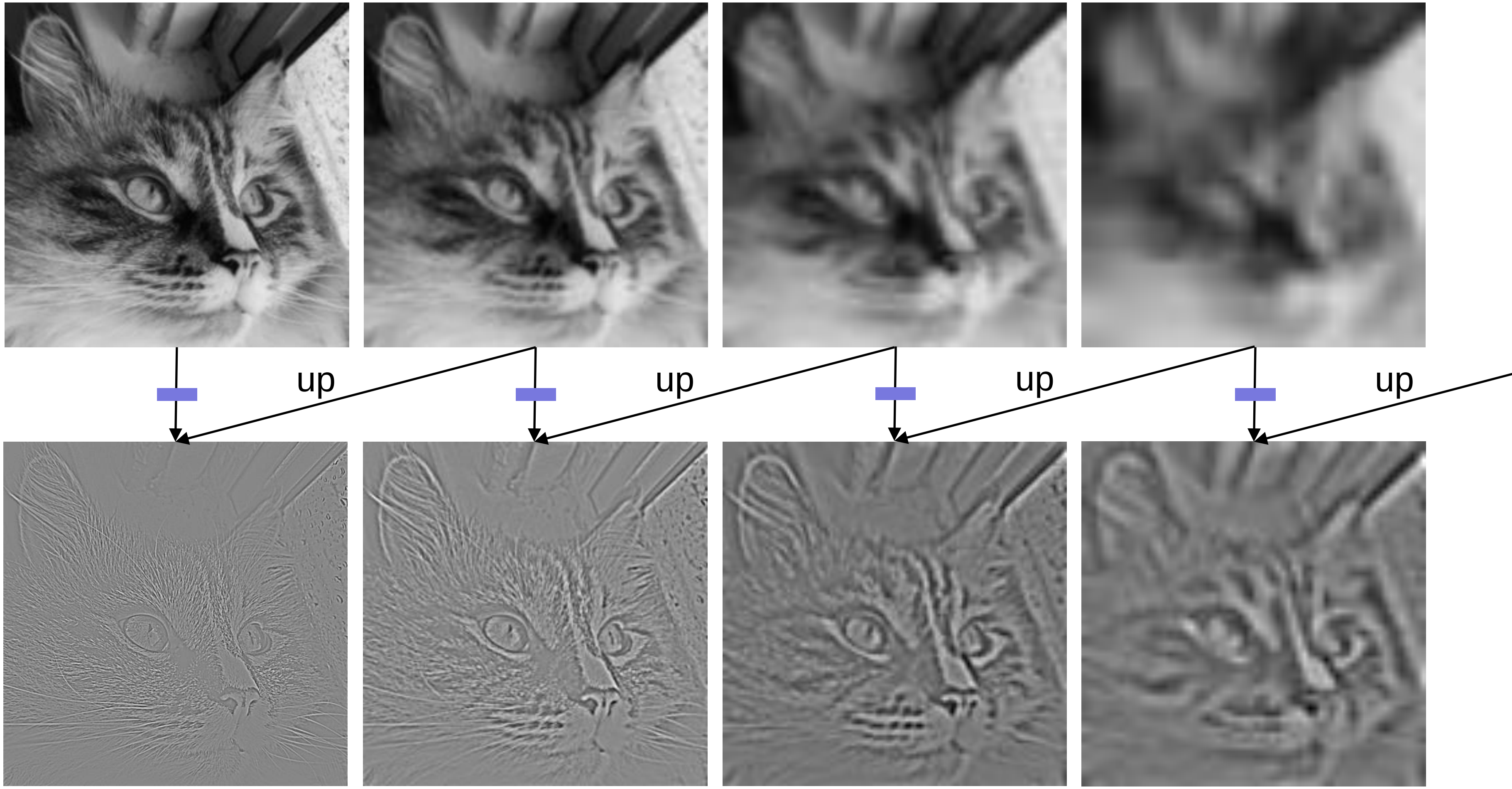
Prof. Jitendros Papadimalik

Band-pass filtering in spatial domain



As a stack

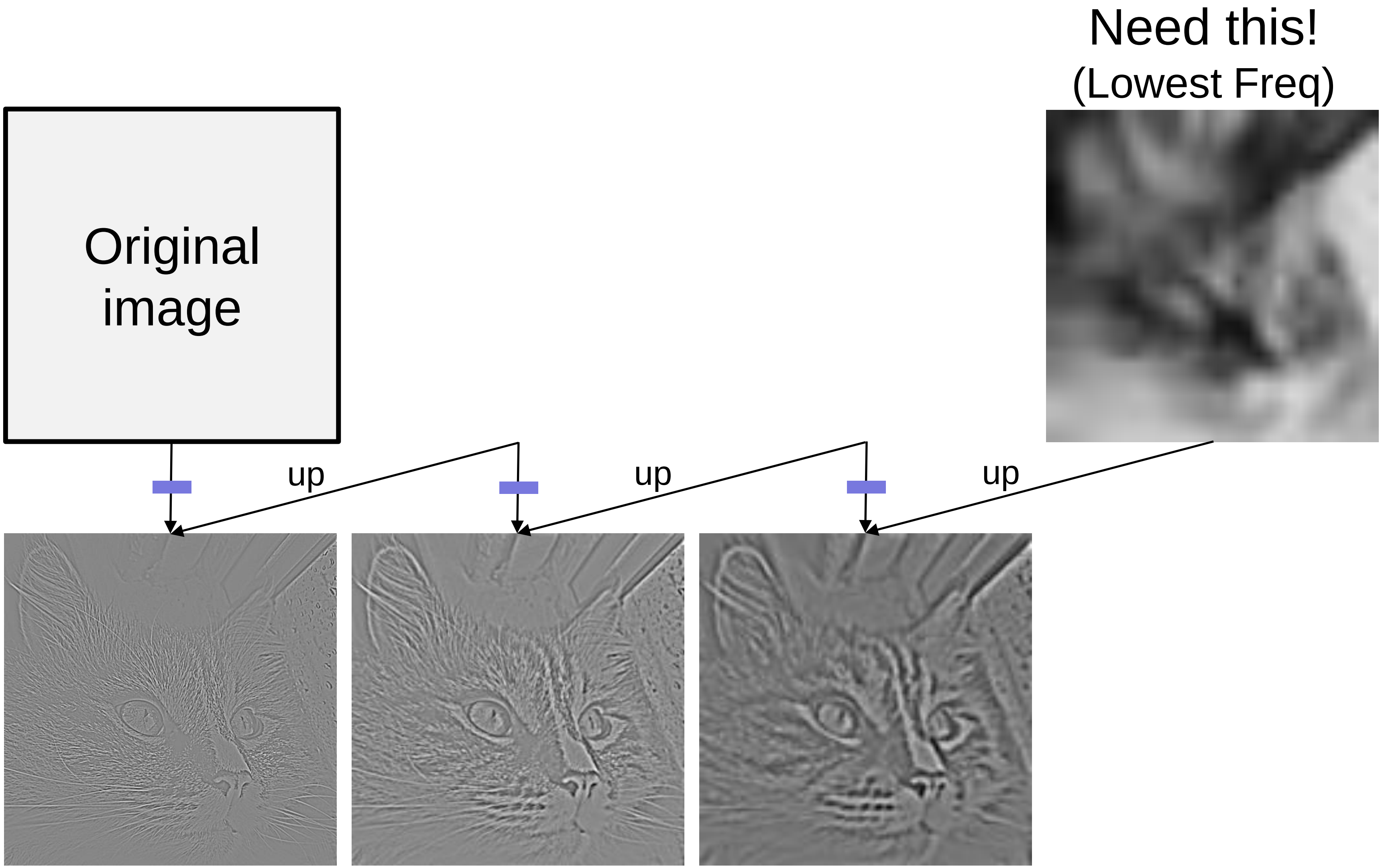
Gaussian Pyramid (low-pass images)



Laplacian Pyramid (sub-band images)

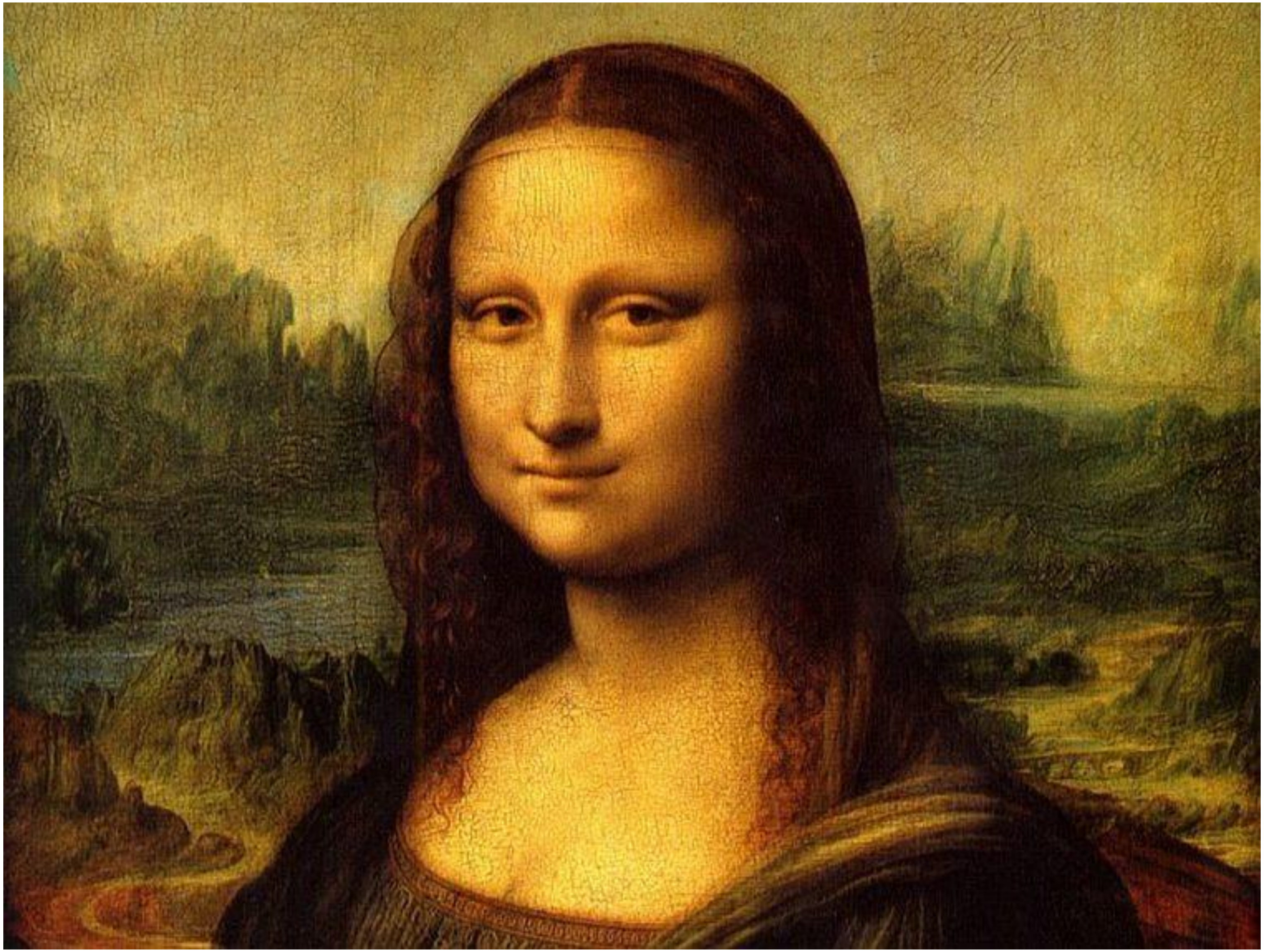
Created from Gaussian pyramid by subtraction

Laplacian Pyramid

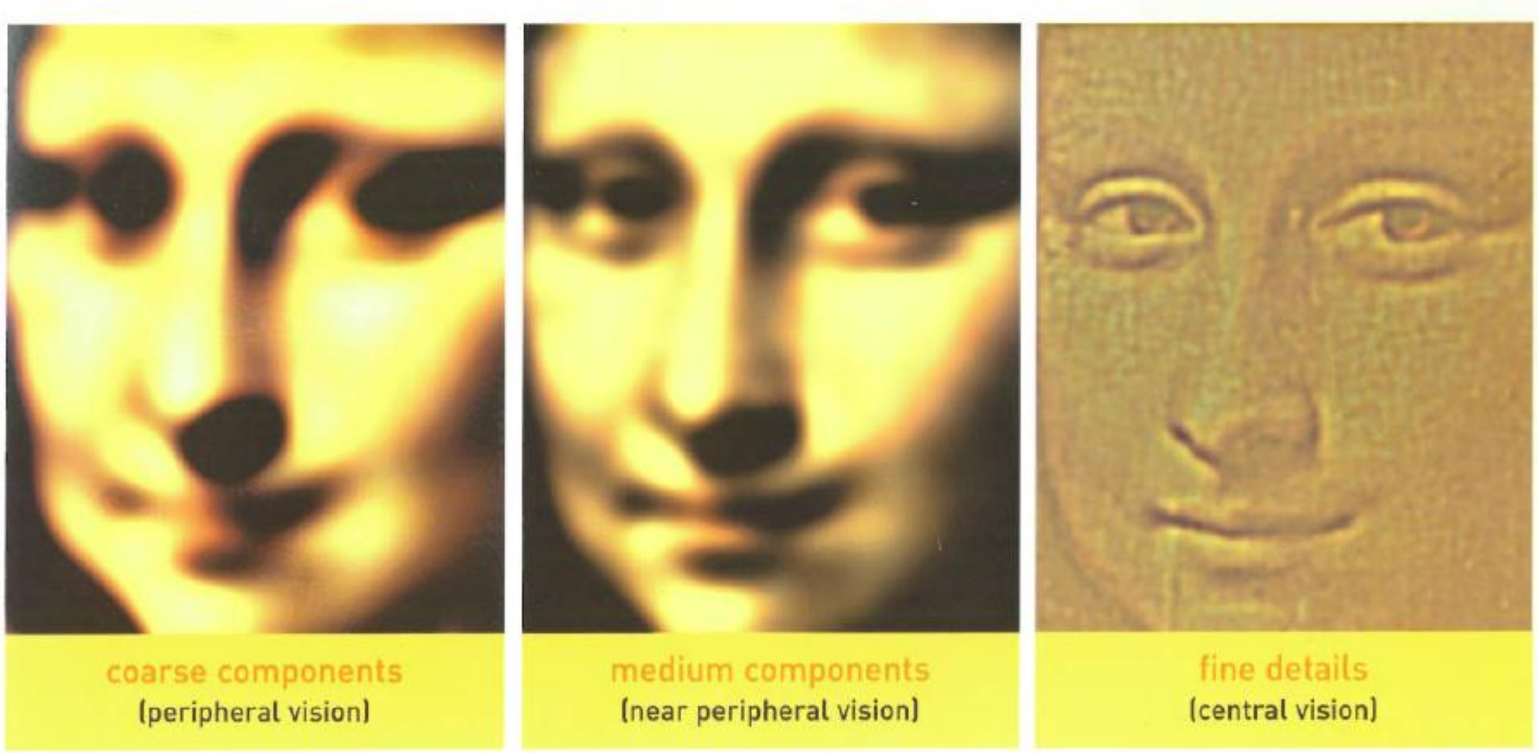


How can we reconstruct (collapse) this pyramid into the original image?

Da Vinci and The Laplacian Pyramid



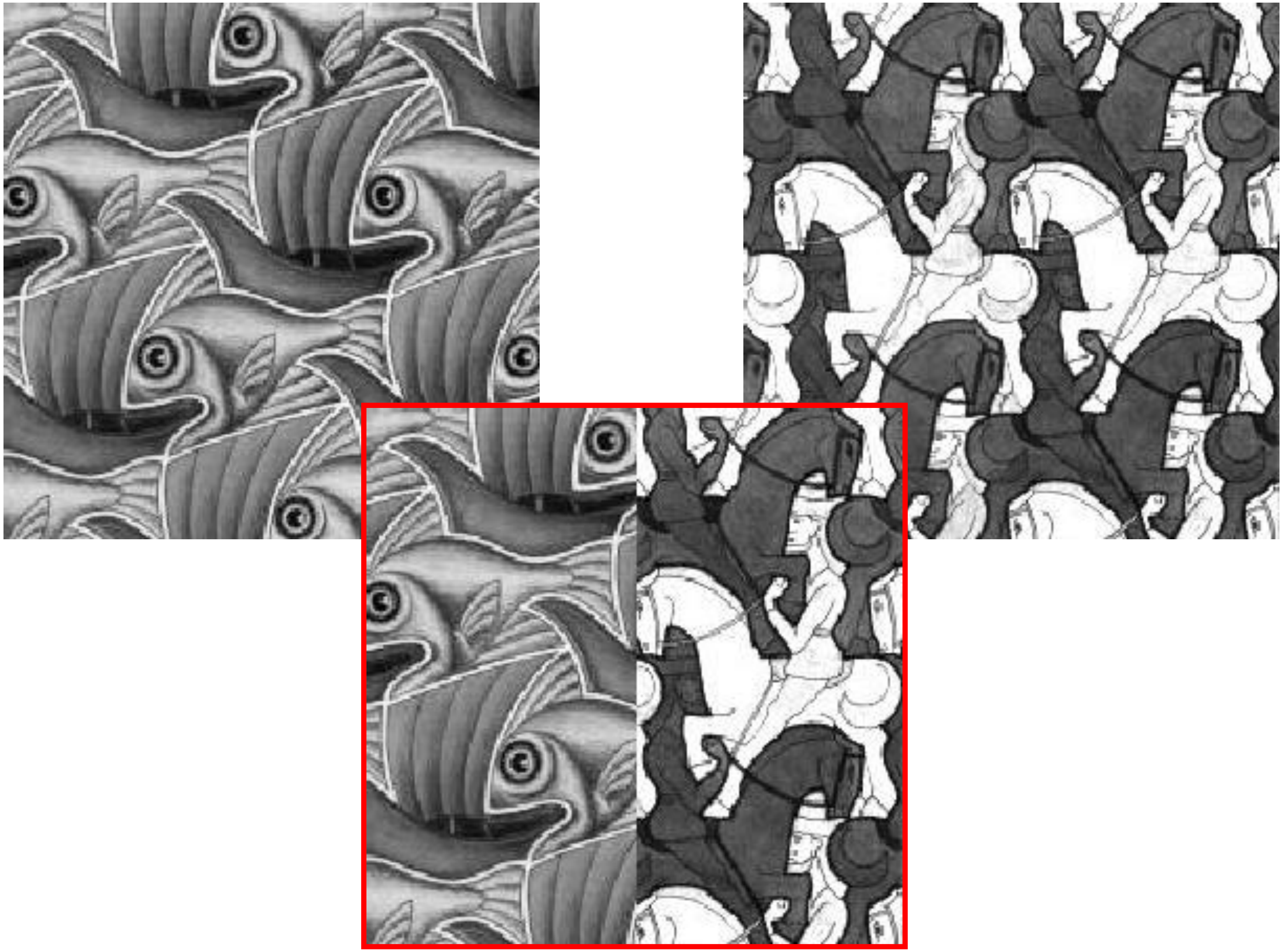
Da Vinci and The Laplacian Pyramid



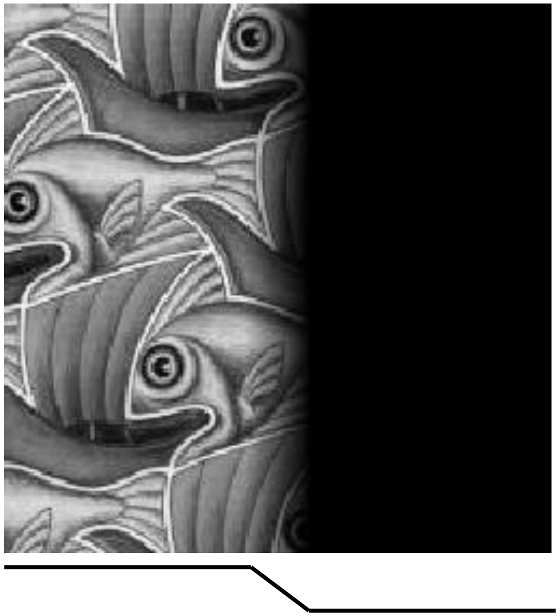
Leonardo playing with peripheral vision

[Livingstone, Vision and Art: The Biology of Seeing]

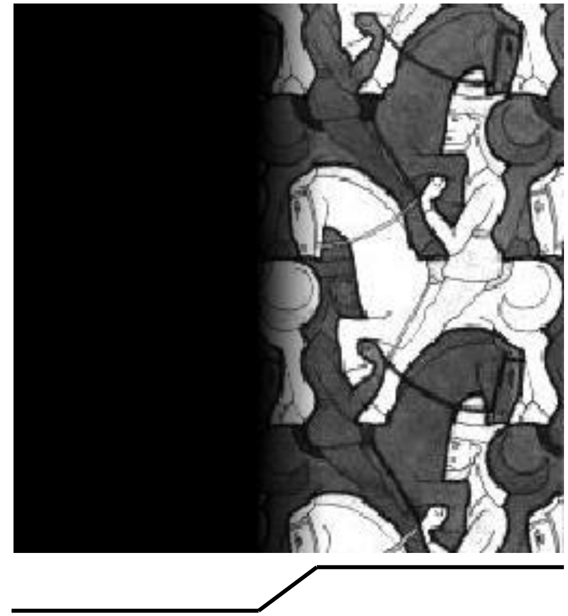
Blending



Alpha Blending / Feathering



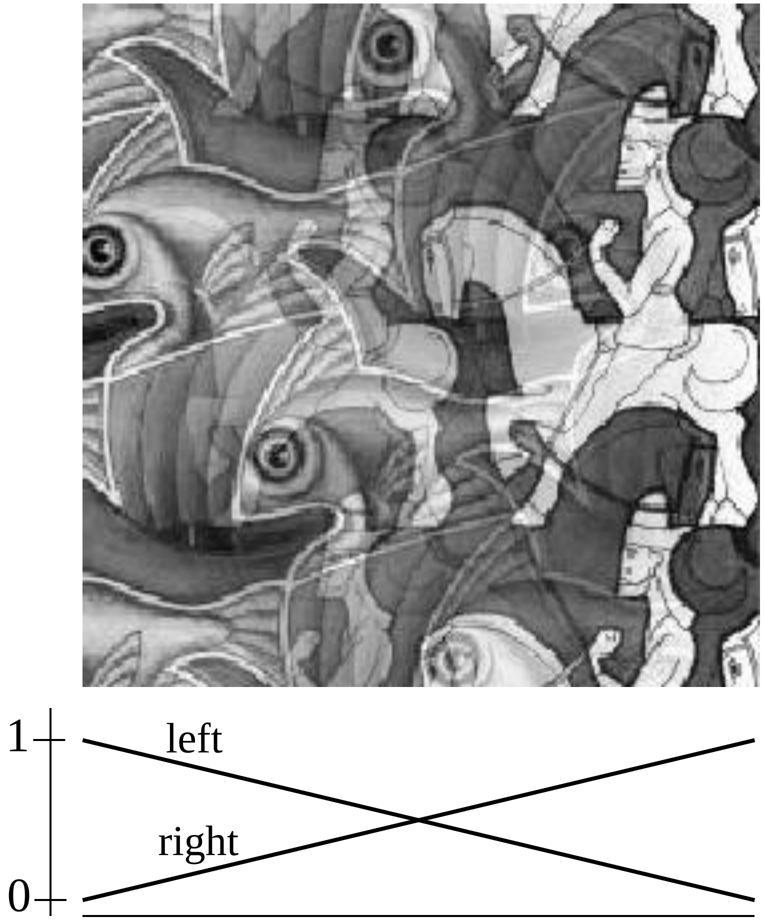
+

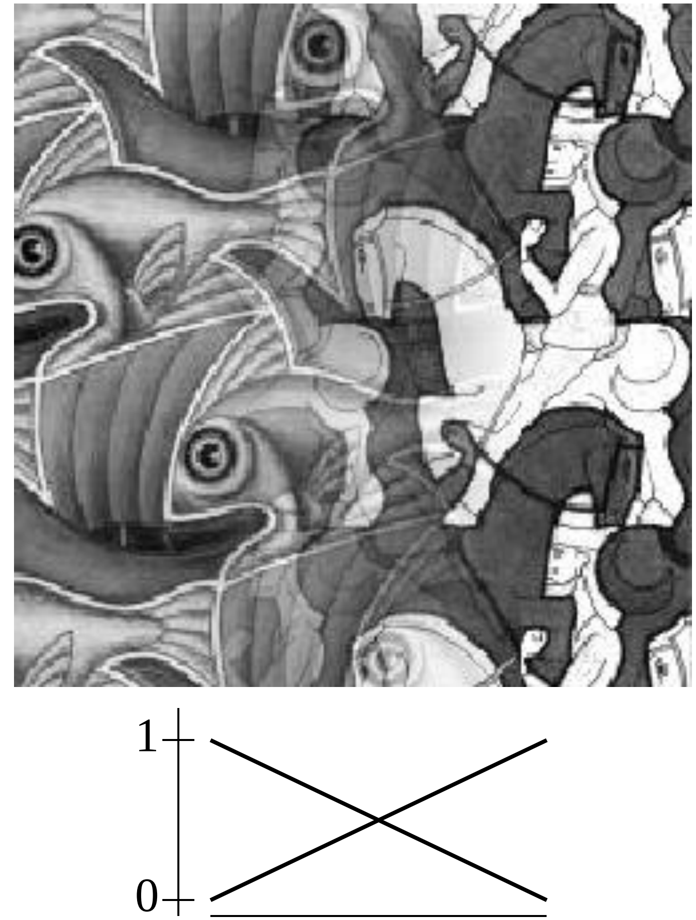


=

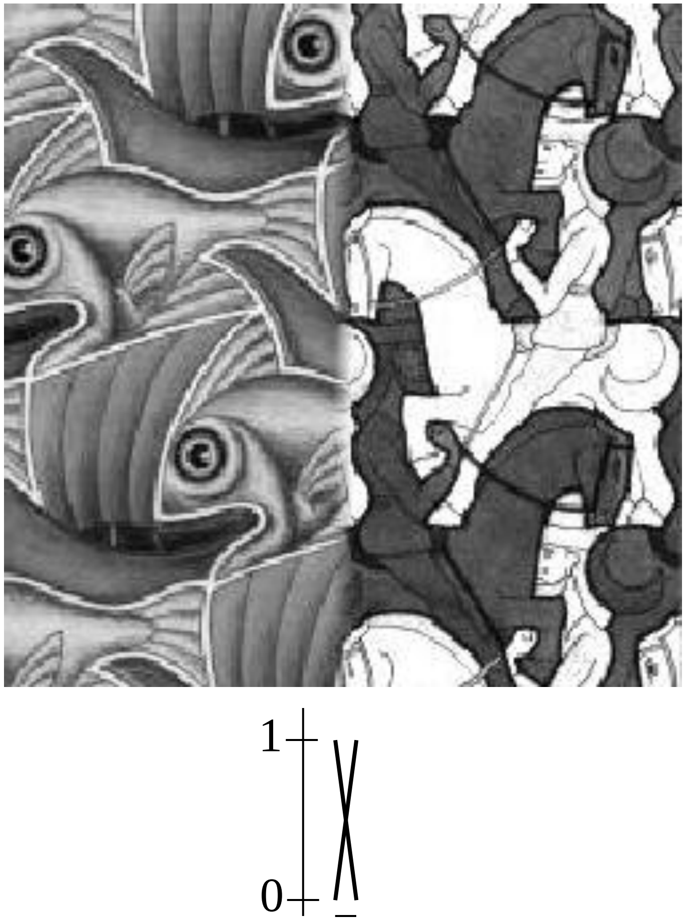
$$I_{blend} = \alpha I_{left} + (1-\alpha) I_{right}$$

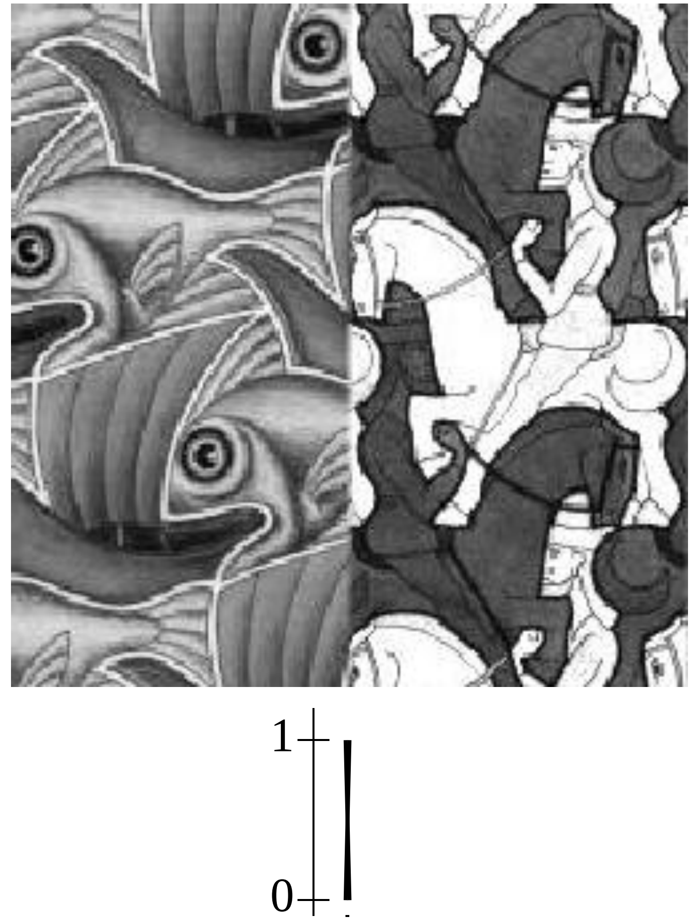
Affect of Window Size



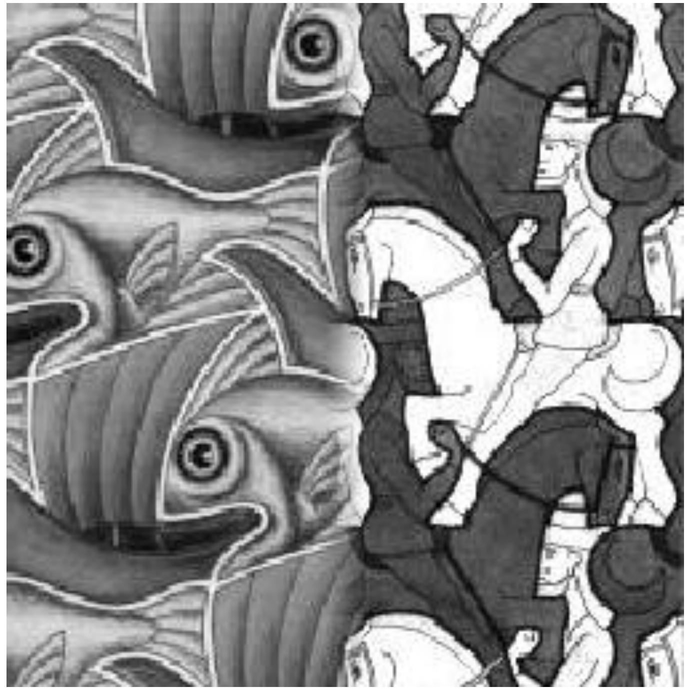


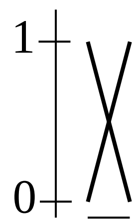
Affect of Window Size





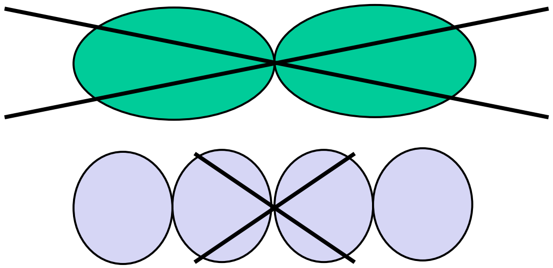
Good Window Size





“Optimal” Window: smooth but not ghosted

What is the Optimal Window?



To avoid seams

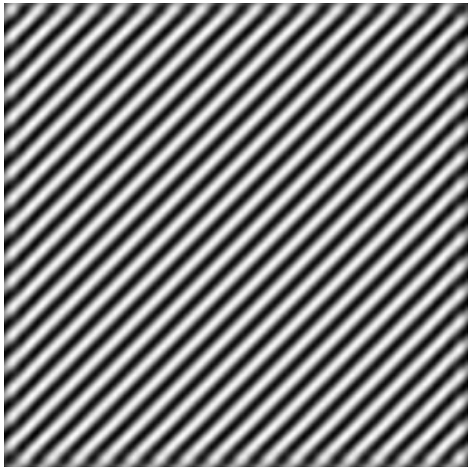
- window = size of largest prominent feature

To avoid ghosting

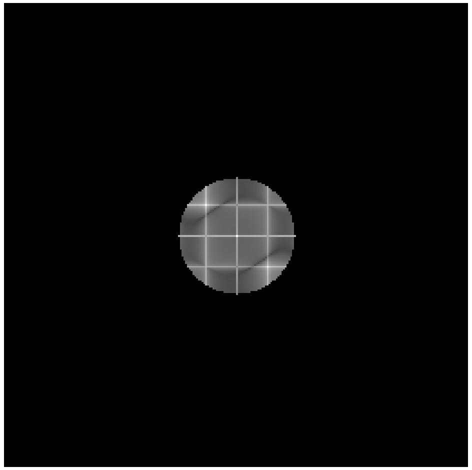
- window <= 2*size of smallest prominent feature

Natural to cast this in the *Fourier domain*

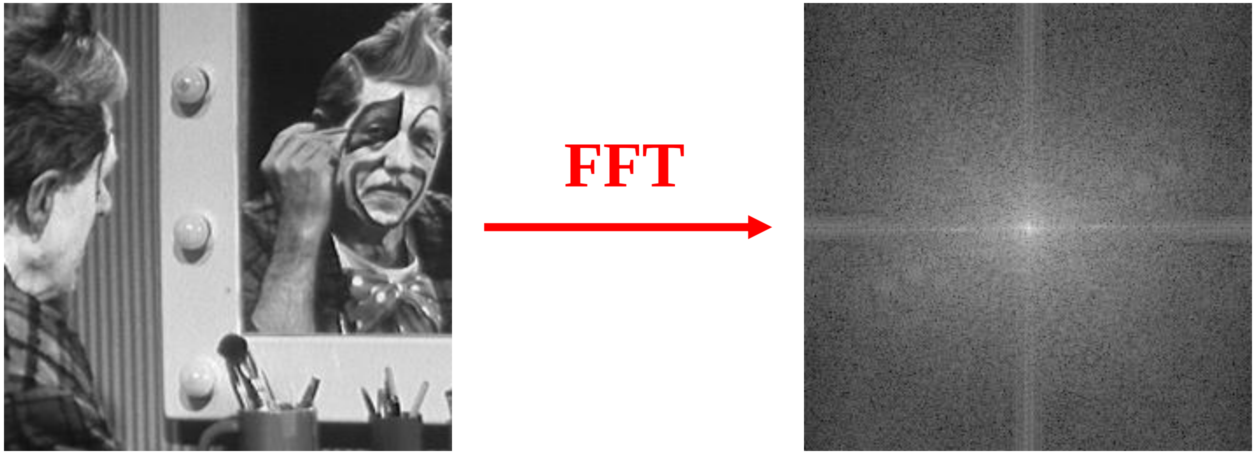
- largest frequency <= 2*size of smallest frequency
- image frequency content should occupy one “octave” (power of two)



FFT



What if the Frequency Spread is Wide

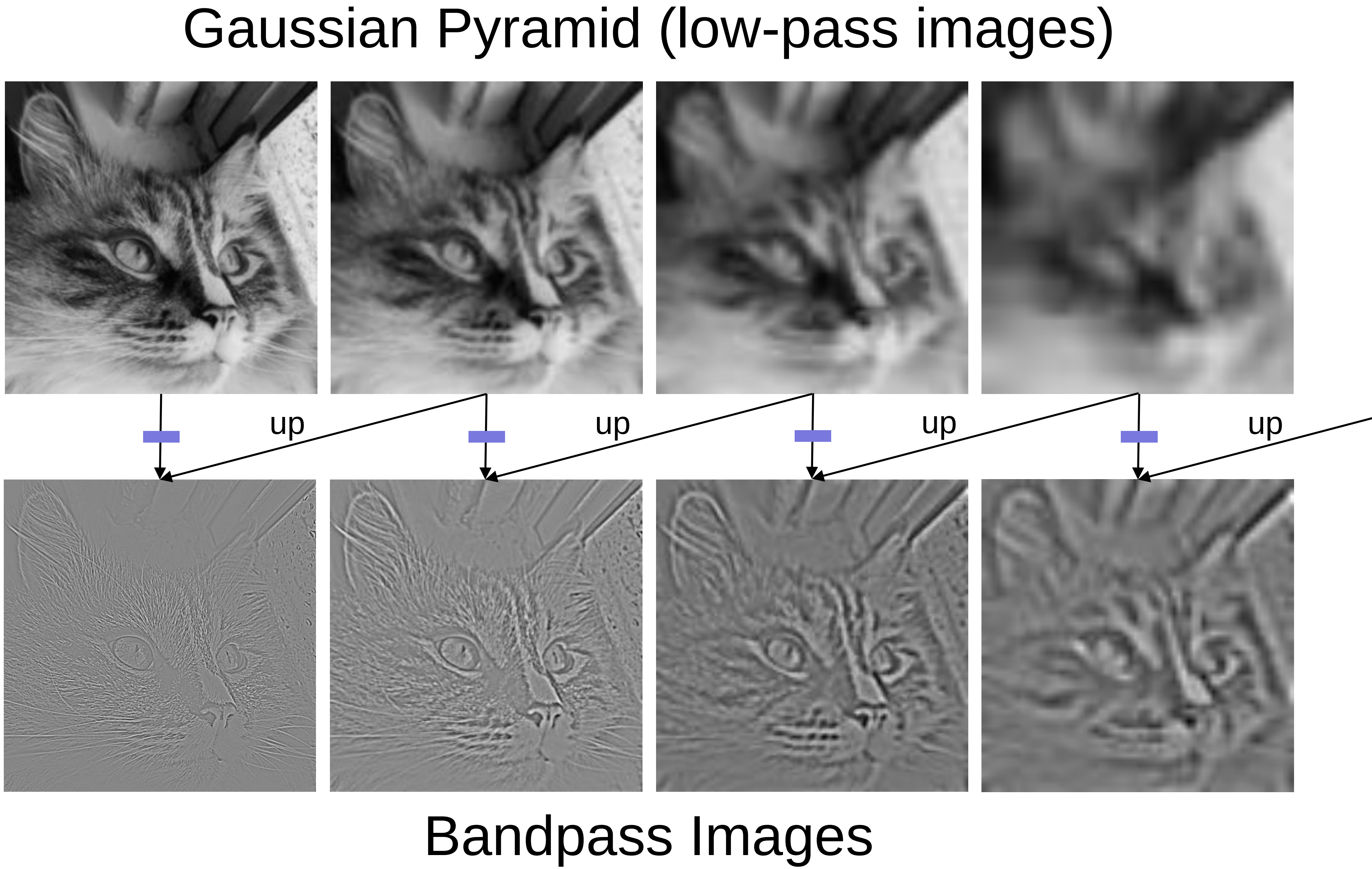


Idea (Burt and Adelson)

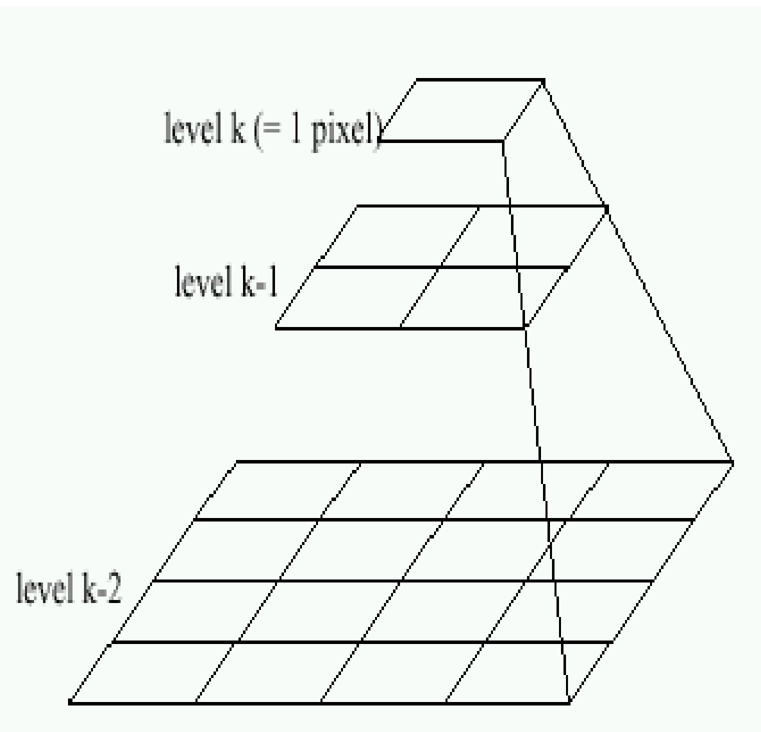
- Compute $F_{left} = FFT(I_{left})$, $F_{right} = FFT(I_{right})$
- Decompose Fourier image into octaves (bands)
  - $F_{left} = F_{left}^1 + F_{left}^2 + \ldots$
- Feather corresponding octaves F_{left}^i with F_{right}^i
  - Can compute inverse FFT and feather in spatial domain
- Sum feathered octave images in frequency domain

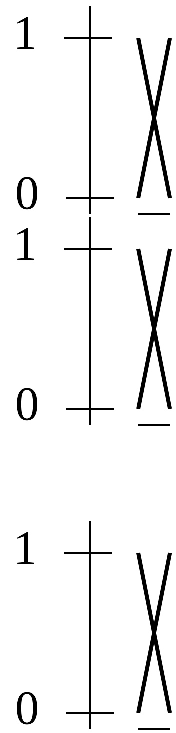
Better implemented in *spatial domain*

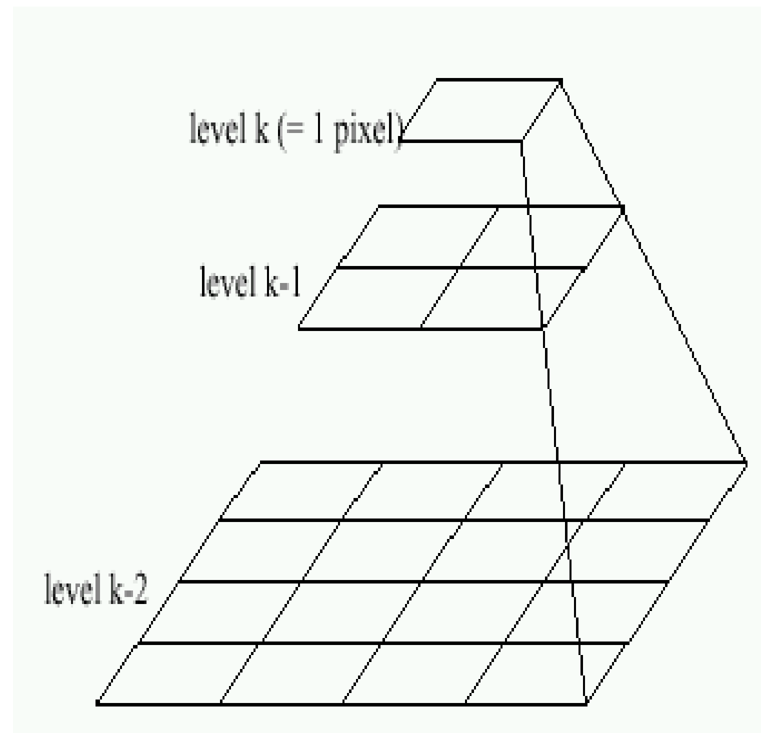
As a stack


Gaussian Pyramid (low-pass images)
up
up
up
up
Bandpass Images

Pyramid Blending





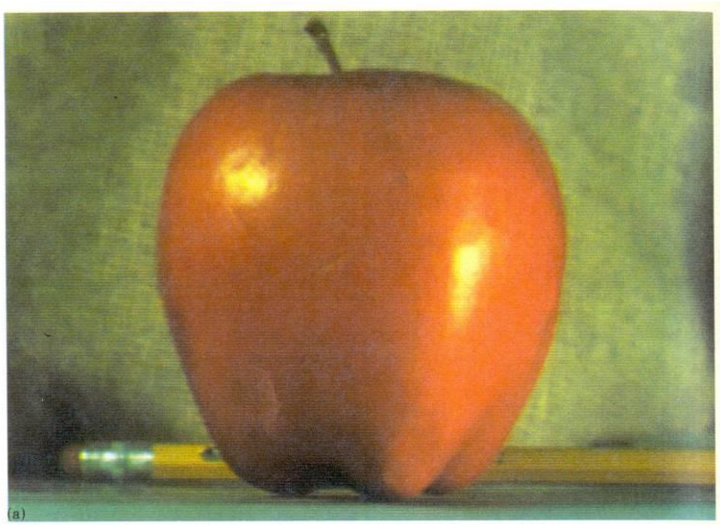


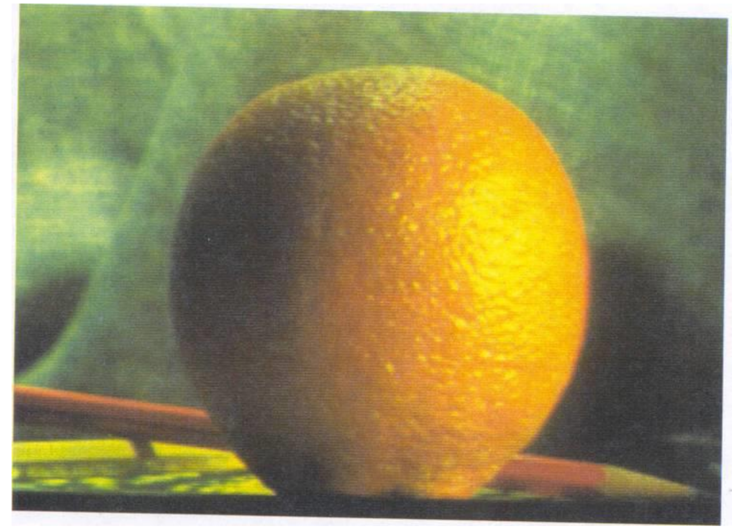
Left pyramid

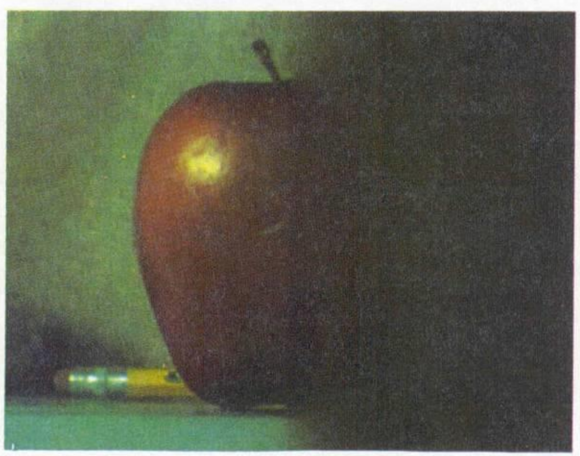
blend

Right pyramid

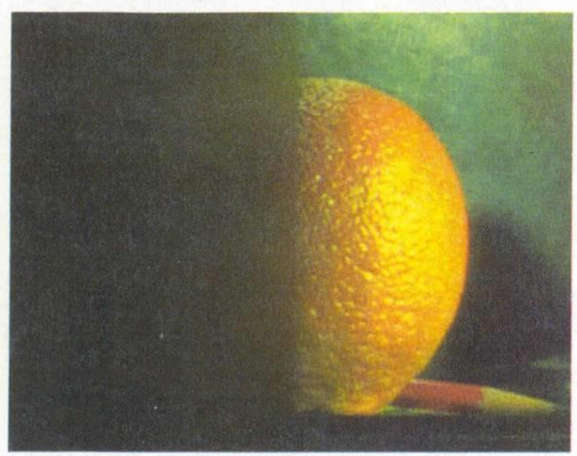
Pyramid Blending



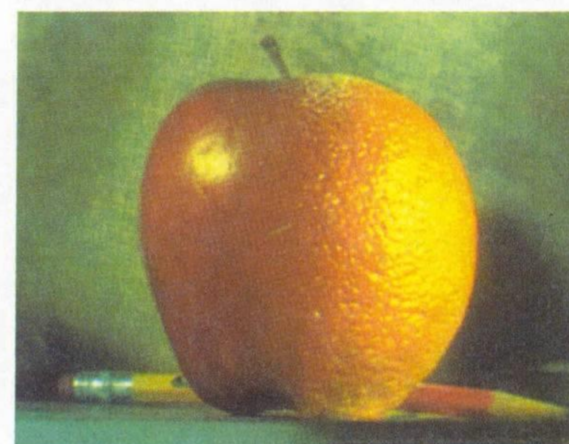




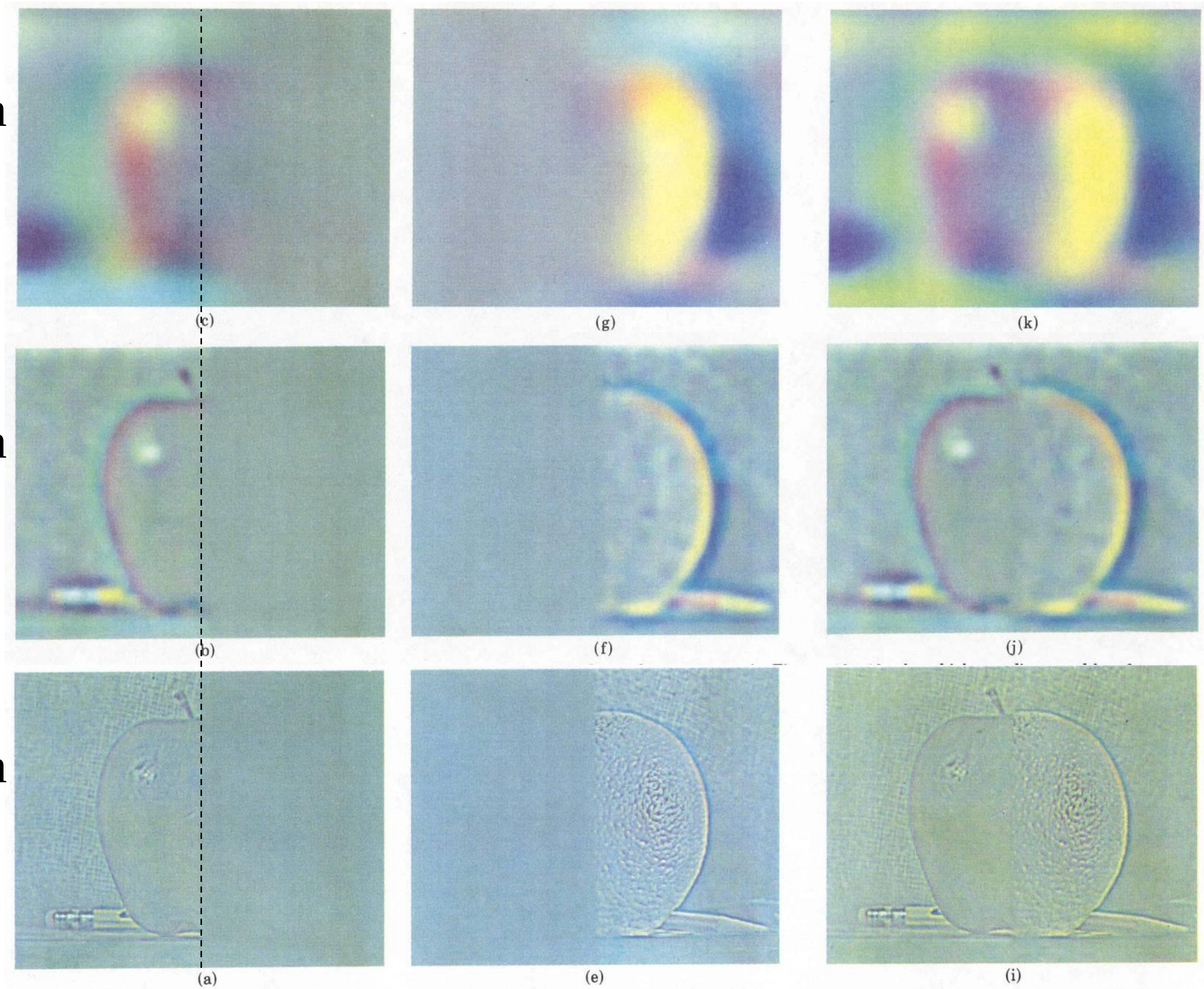
(d)



(h)



(l)



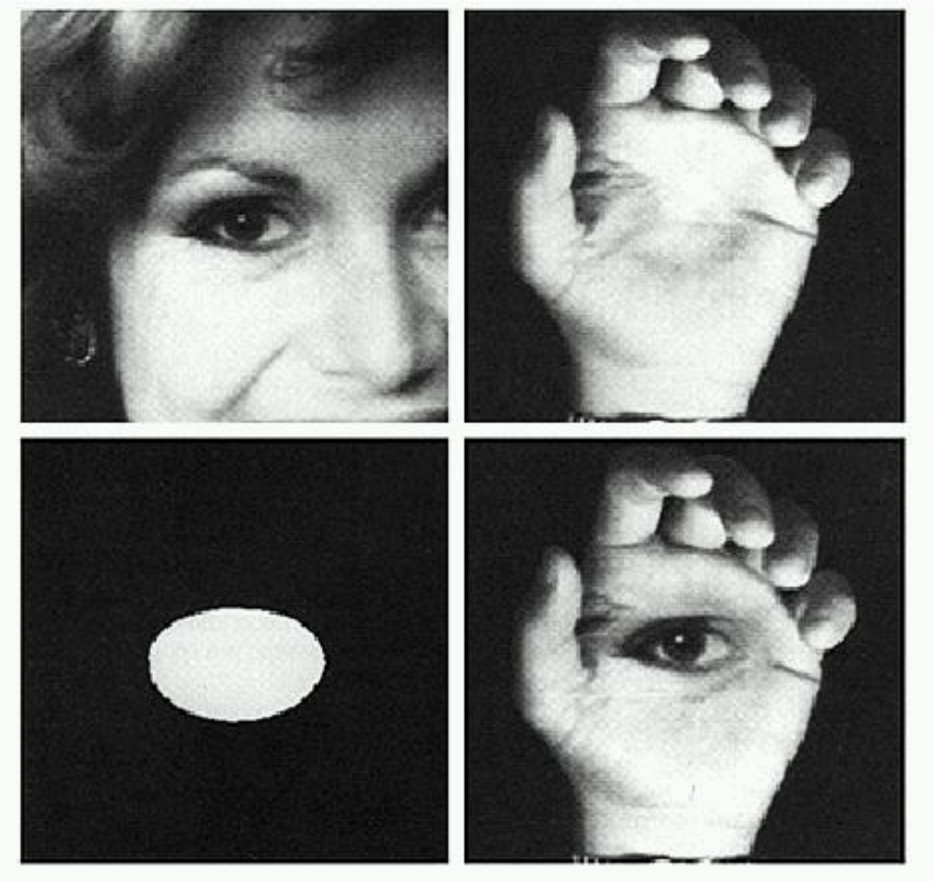
laplacian level 4

laplacian level 2

laplacian level 0

left pyramid | right pyramid | blended pyramid

Blending Regions

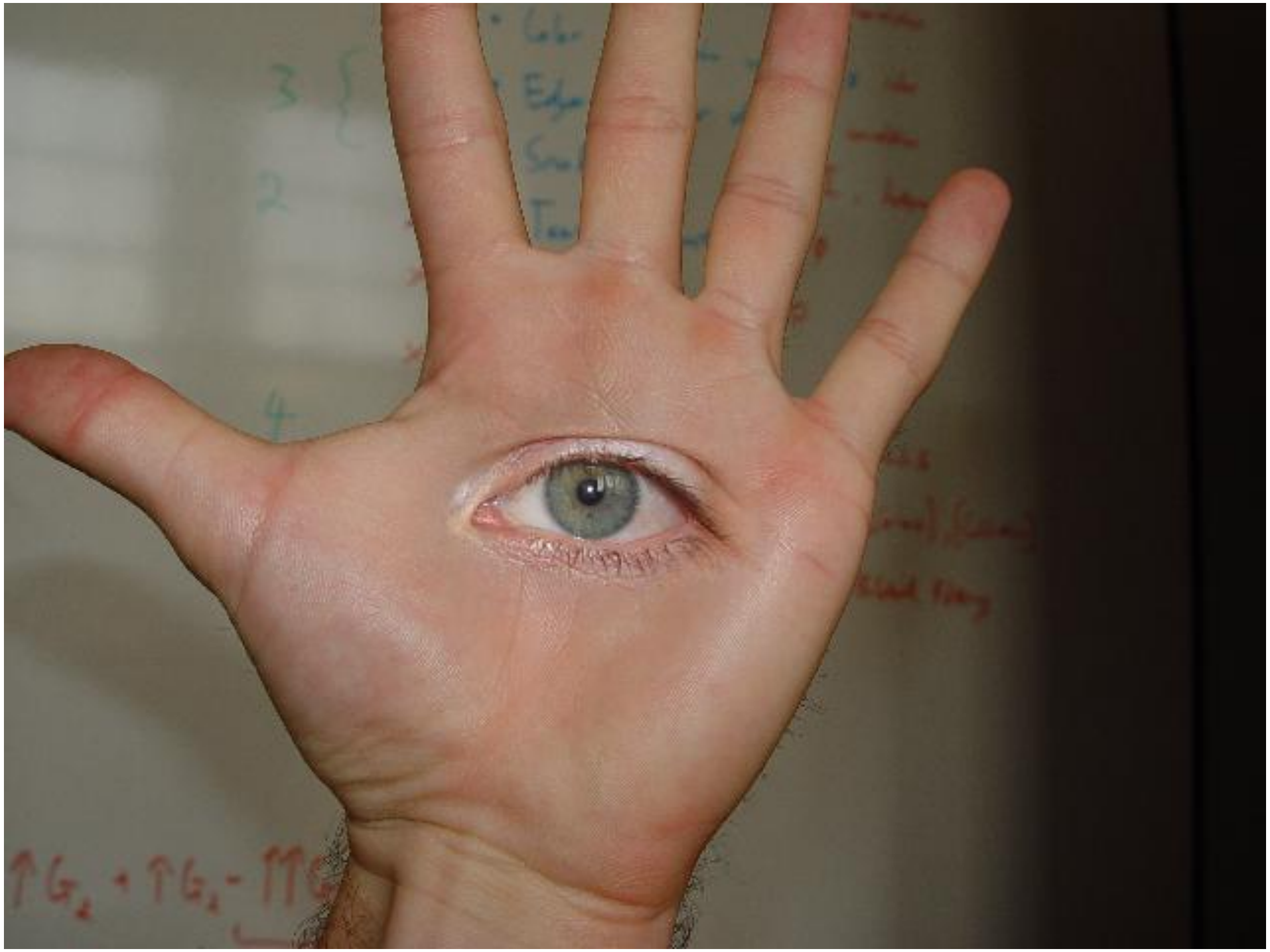


Laplacian Pyramid: Blending

General Approach:

1. Build Laplacian pyramids *LA* and *LB* from images *A* and *B*
2. Build a Gaussian pyramid *GR* from selected region *R*
3. Form a combined pyramid *LS* from *LA* and *LB* using nodes of *GR* as weights:
   - $LS(i,j) = GR(I,j,)*LA(I,j) + (1-GR(I,j))*LB(I,j)$
4. Collapse the *LS* pyramid to get the final blended image

Horror Photo



© david dmartin (Boston College)

Results from this class (fall 2005)

Simplification: Two-band Blending

Brown & Lowe, 2003

- Only use two bands -- high freq. and low freq. – without downsampling
- Blends low freq. smoothly
- Blend high freq. with no smoothing: use binary alpha

2-band "Laplacian Stack" Blending

Low frequency (λ > 2 pixels)

High frequency (λ < 2 pixels)

Linear Blending

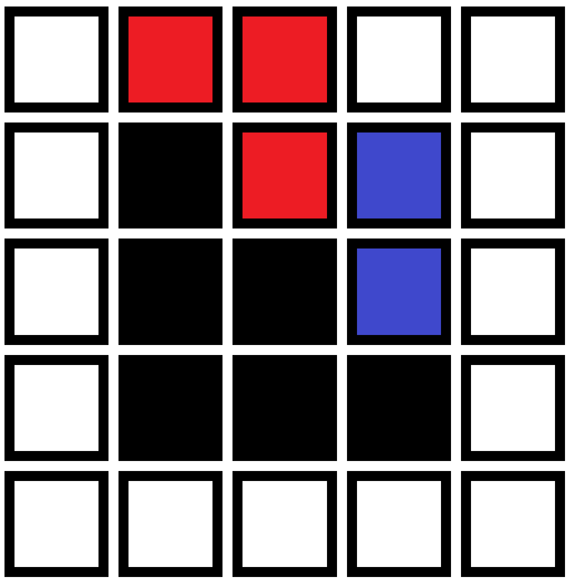
2-band Blending

Side note: Image Compression

89k

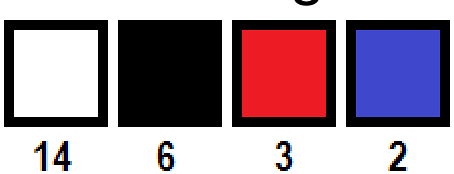
Lossless Compression (e.g. Huffman coding)

Input image:



Pixel code:

Pixel histogram:



Compressed image:

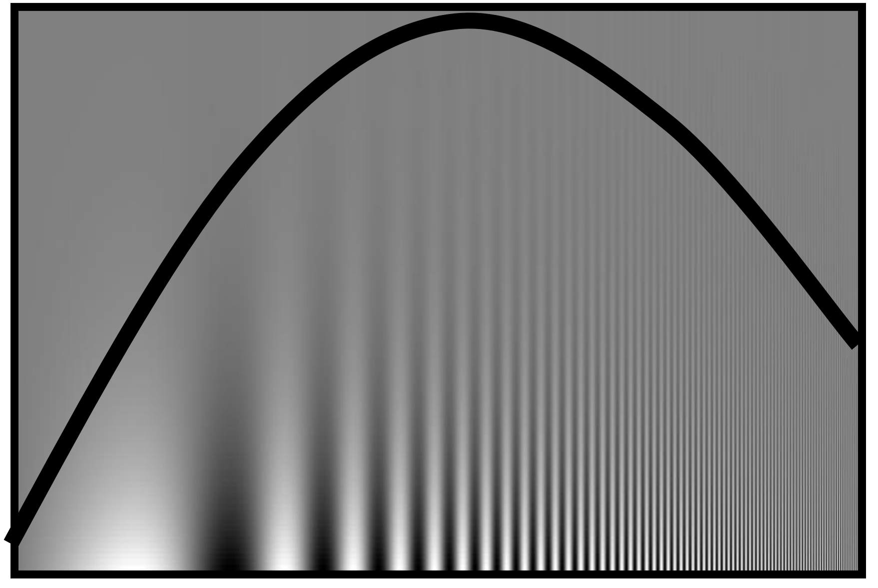
0 110 110 0 0

0 10 110 111 0

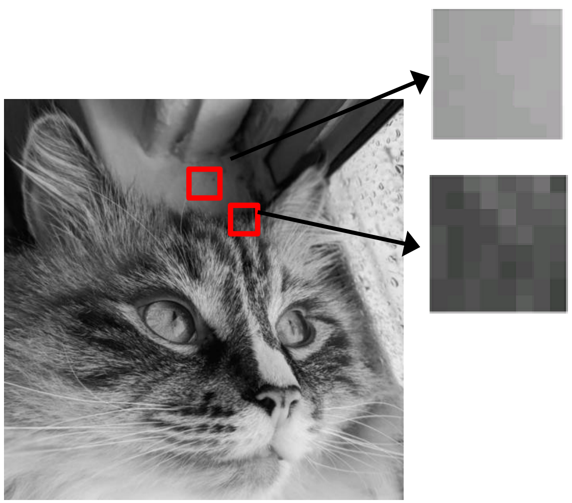
...

https://www.print-driver.com/stories/huffman-coding-jpeg

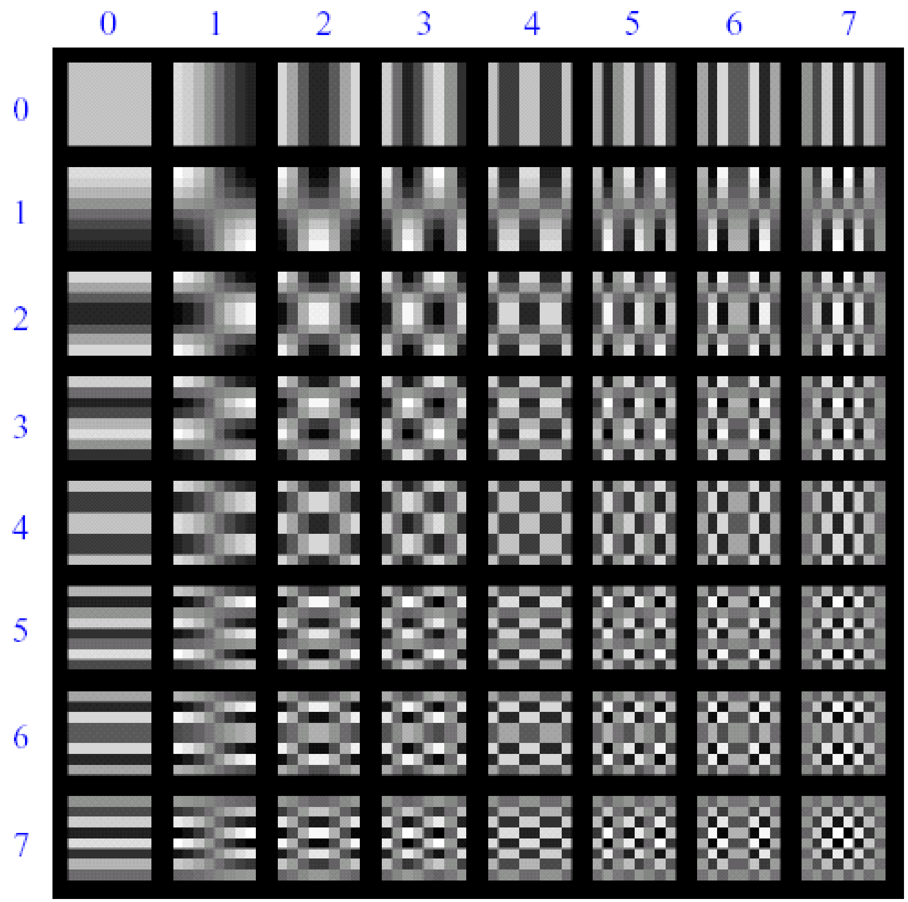
Lossless Compression not enough



Lossy Image Compression (JPEG)



cut up into 8x8 blocks

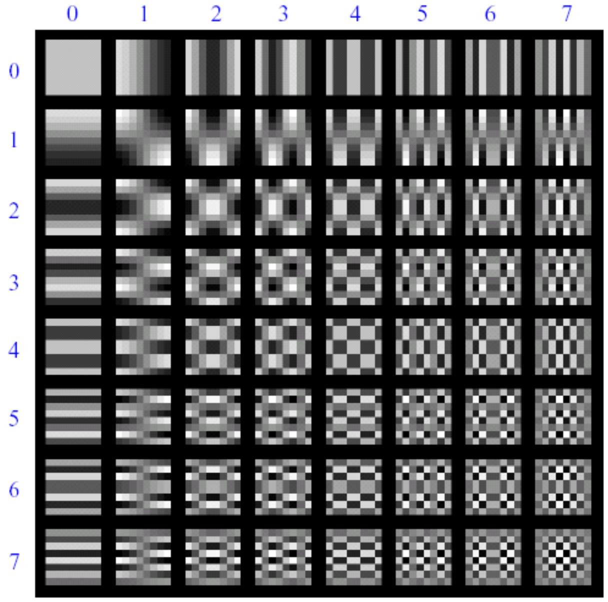


Block-based Discrete Cosine Transform (DCT)

Using DCT in JPEG

The first coefficient B(0,0) is the DC component, the average intensity

The top-left coeffs represent low frequencies, the bottom right – high frequencies



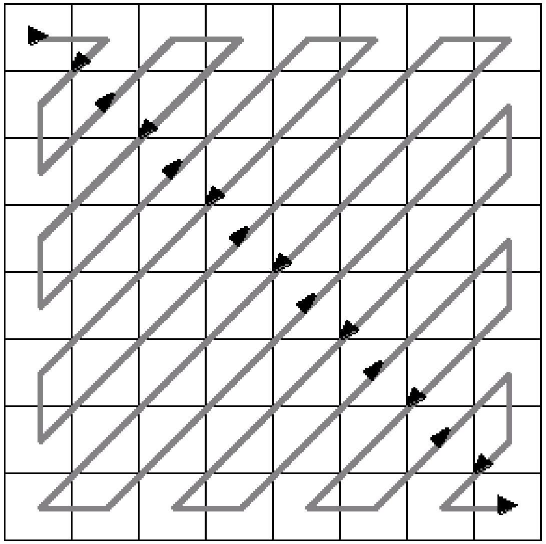


Image compression using DCT

Quantize

- More coarsely for high frequencies (tend to have smaller values anyway)
- Many quantized high frequency values will be zero

Encode

- Can decode with inverse dct

Filter responses

$$\xrightarrow{u}$$

$$G = \begin{bmatrix} -415.38 & -30.19 & -61.20 & 27.24 & 56.13 & -20.10 & -2.39 & 0.46 \\ 4.47 & -21.86 & -60.76 & 10.25 & 13.15 & -7.09 & -8.54 & 4.88 \\ -46.83 & 7.37 & 77.13 & -24.56 & -28.91 & 9.93 & 5.42 & -5.65 \\ -48.53 & 12.07 & 34.10 & -14.76 & -10.24 & 6.30 & 1.83 & 1.95 \\ 12.12 & -6.55 & -13.20 & -3.95 & -1.88 & 1.75 & -2.79 & 3.14 \\ -7.73 & 2.91 & 2.38 & -5.94 & -2.38 & 0.94 & 4.30 & 1.85 \\ -1.03 & 0.18 & 0.42 & -2.42 & -0.88 & -3.02 & 4.12 & -0.66 \\ -0.17 & 0.14 & -1.07 & -4.19 & -1.17 & -0.10 & 0.50 & 1.68 \end{bmatrix} \Big\downarrow v$$

Quantized values

$$B = \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Quantization table

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

JPEG Compression Summary

Subsample color by factor of 2

- People have bad resolution for color

Split into blocks (8x8, typically), subtract 128

For each block

a. Compute DCT coefficients
b. Coarsely quantize
   – Many high frequency components will become zero
c. Encode (e.g., with Huffman coding)

Spatial dimension of color channels are reduced by 2 (lecture 2)!

http://en.wikipedia.org/wiki/YCbCr
http://en.wikipedia.org/wiki/JPEG

Block size in JPEG

Block size

- small block
  - faster
  - correlation exists between neighboring pixels
- large block
  - better compression in smooth regions
- It's 8x8 in standard JPEG

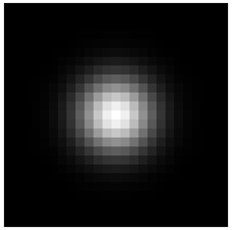
JPEG compression comparison

89k

12k

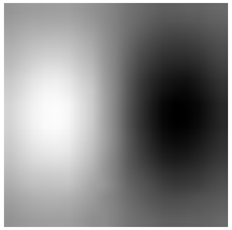
Review: Smoothing vs. derivative filters

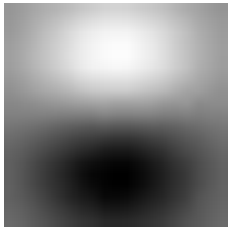
Smoothing filters



- Gaussian: remove “high-frequency” components; “low-pass” filter
- Can the values of a smoothing filter be negative?
- What should the values sum to?
  - **One:** constant regions are not affected by the filter

Derivative filters

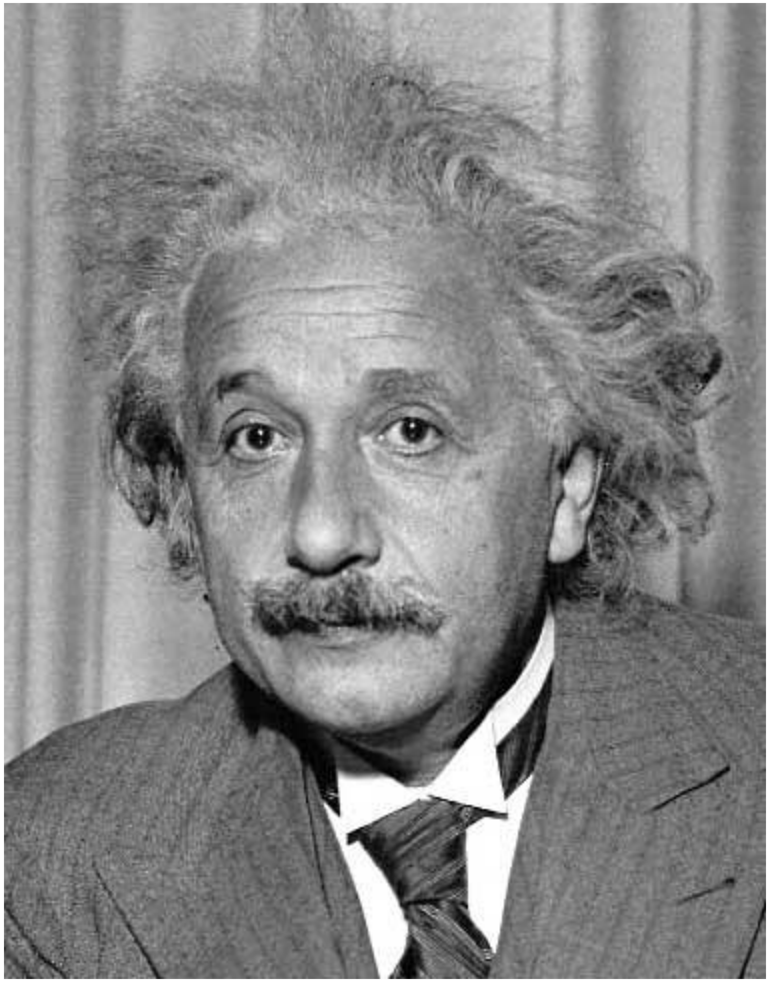




- Derivatives of Gaussian
- Can the values of a derivative filter be negative?
- What should the values sum to?
  - **Zero:** no response in constant regions
- High absolute value at points of high contrast

Template matching

Goal: find  in image



Main challenge: What is a good similarity or distance measure between two patches?

- Correlation
- Zero-mean correlation
- Sum Square Difference
- Normalized Cross Correlation

Side by Derek Hoiem

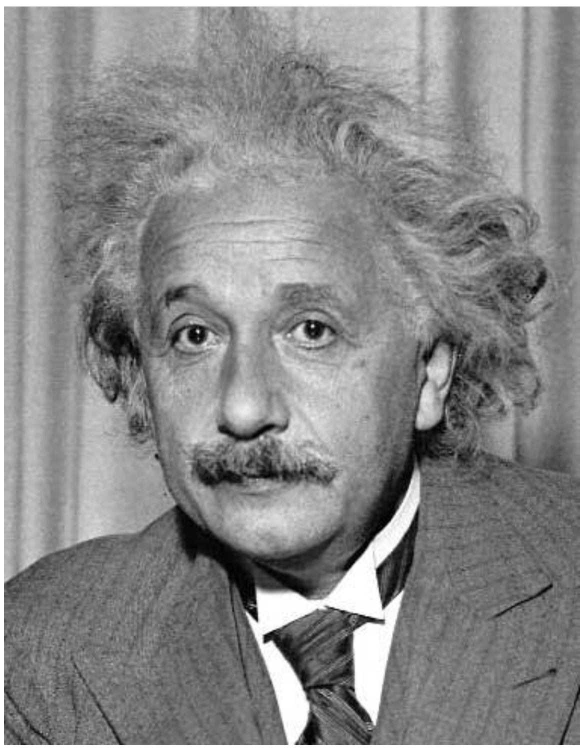
Matching with filters

Goal: find  in image

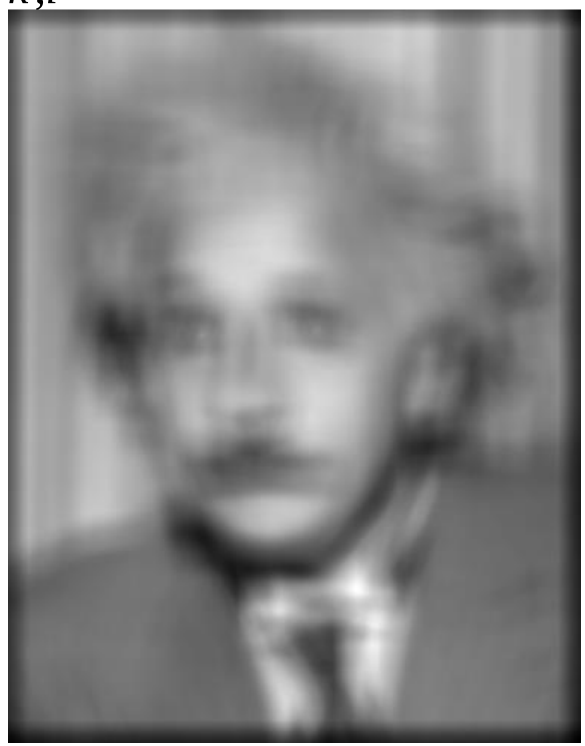
Method 0: filter the image with eye patch

$$h[m,n]=\sum_{k,l} g[k,l]\ f[m+k,n+l]$$

f = image
g = filter



Input



Filtered Image

What went wrong?

Side by Derek Hoiem

Matching with filters

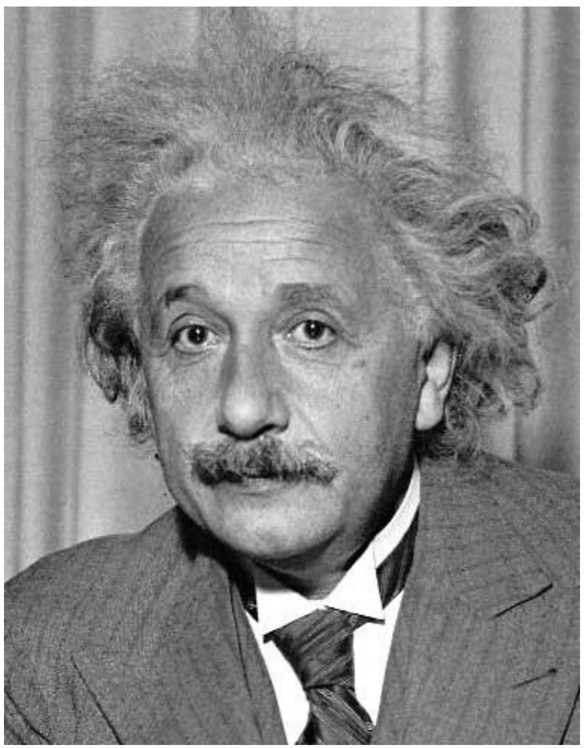
Goal: find  in image

f = image
g = filter

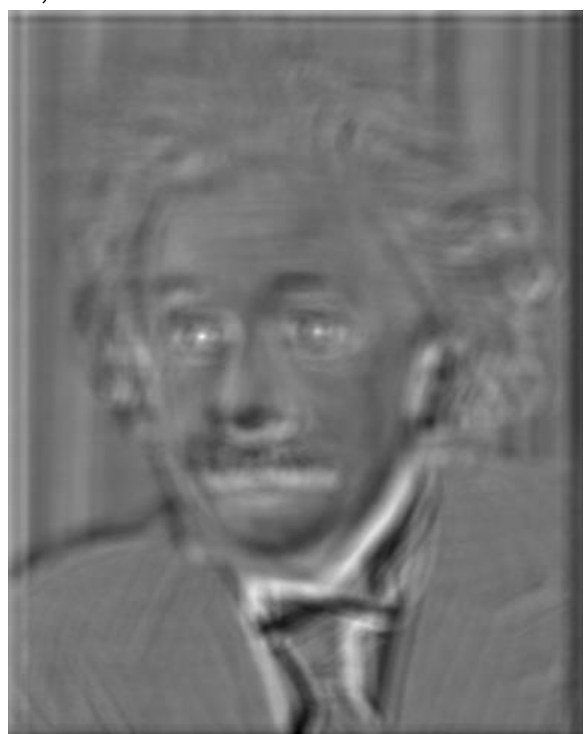
Method 1: filter the image with zero-mean eye

$$h[m,n] = \sum_{k,l} (g[k,l] - \bar{g})(f[m+k, n+l])$$

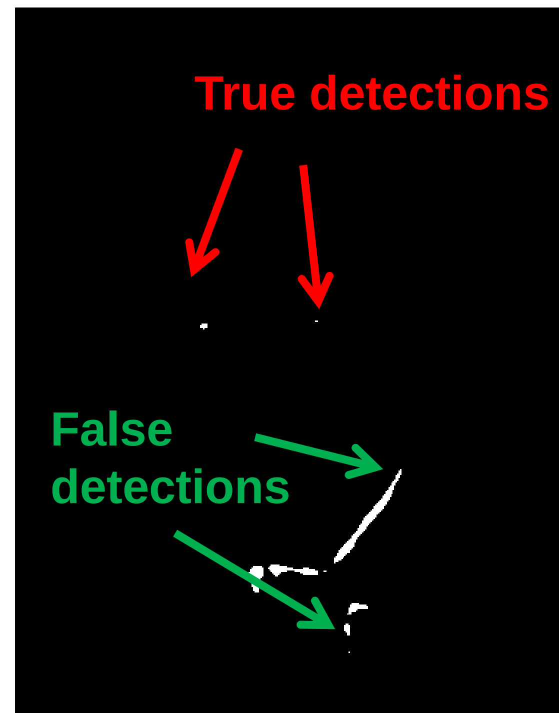
mean of g



Input



Filtered Image (scaled)



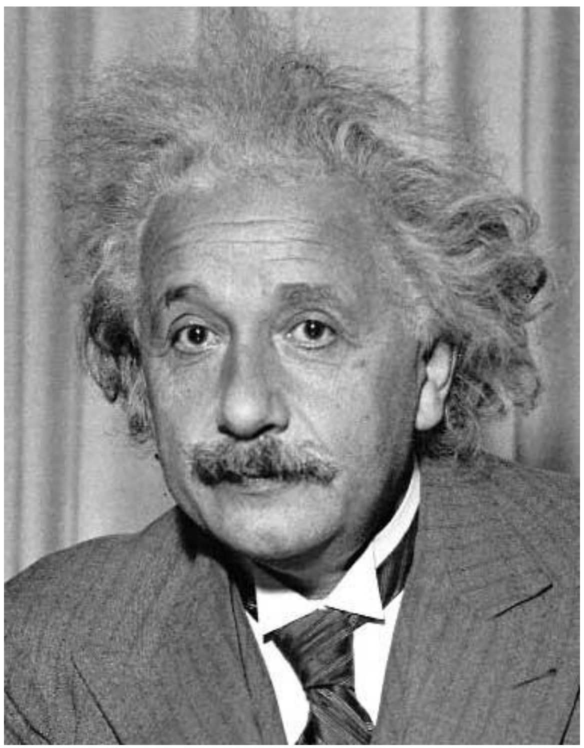
Thresholded Image

Matching with filters

Goal: find  in image

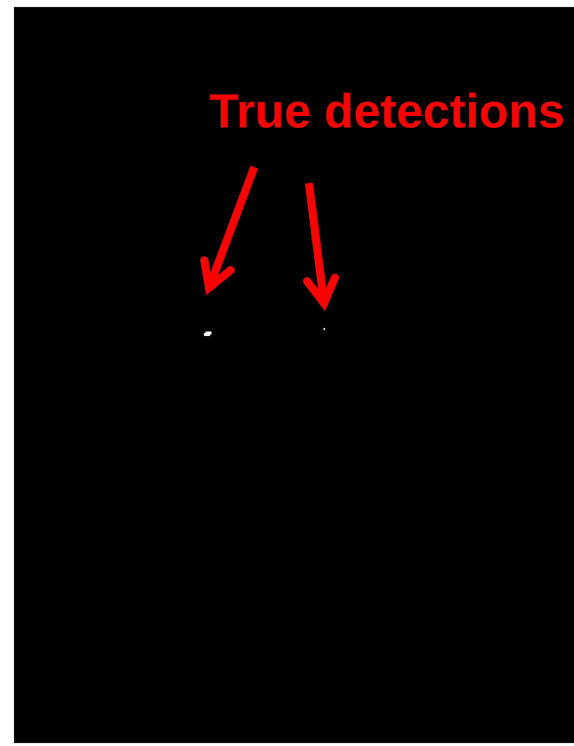
Method 2: SSD (L2)

$$h[m,n]=\sum_{k,l}(g[k,l]-f[m+k,n+l])^2$$



Input

1- sqrt(SSD)



Thresholded Image

Matching with filters

Can SSD be implemented with linear filters?

$$h[m,n] = \sum_{k,l} (g[k,l] - f[m+k, n+l])^2$$

Side by Derek Hoiem

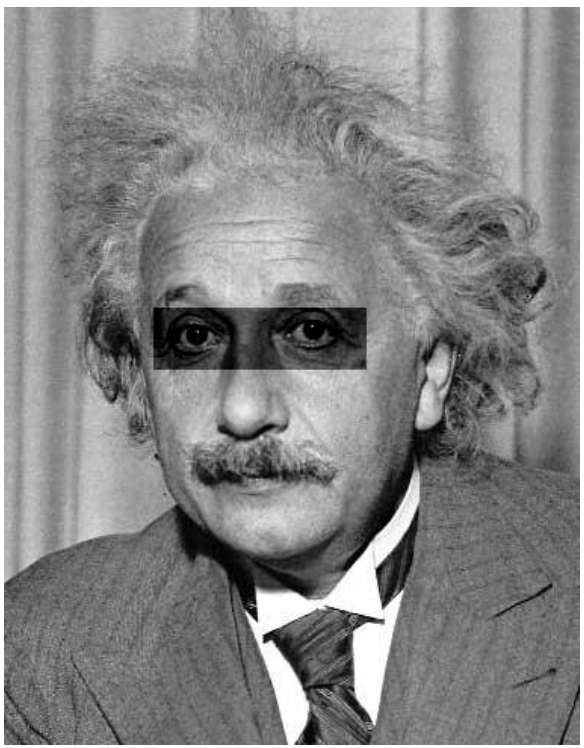
Matching with filters

Goal: find  in image

What's the potential downside of SSD?

Method 2: SSD

$$h[m,n]=\sum_{k,l}(g[k,l]-f[m+k,n+l])^2$$



Input

1- sqrt(SSD)

Side by Derek Hoiem

Matching with filters

Goal: find  in image

Method 3: Normalized cross-correlation

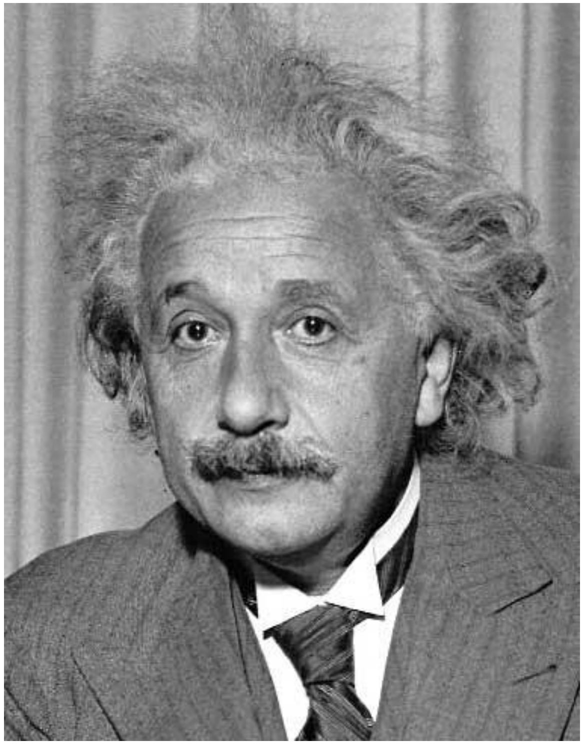
$$h[m,n] = \frac{\sum_{k,l} (g[k,l] - \overset{\text{mean template}}{\bar{g}})(f[m+k,n+l] - \overset{\text{mean image patch}}{\bar{f}_{m,n}})}{\left(\sum_{k,l} (g[k,l] - \bar{g})^2 \sum_{k,l} (f[m+k,n+l] - \bar{f}_{m,n})^2 \right)^{0.5}}$$

Side by Derek Hoiem

Matching with filters

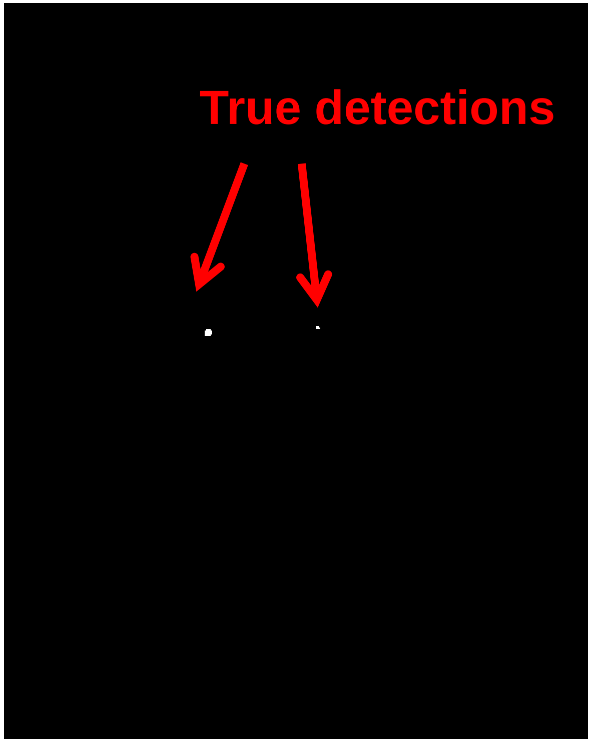
Goal: find  in image

Method 3: Normalized cross-correlation



Input

Normalized X-Correlation

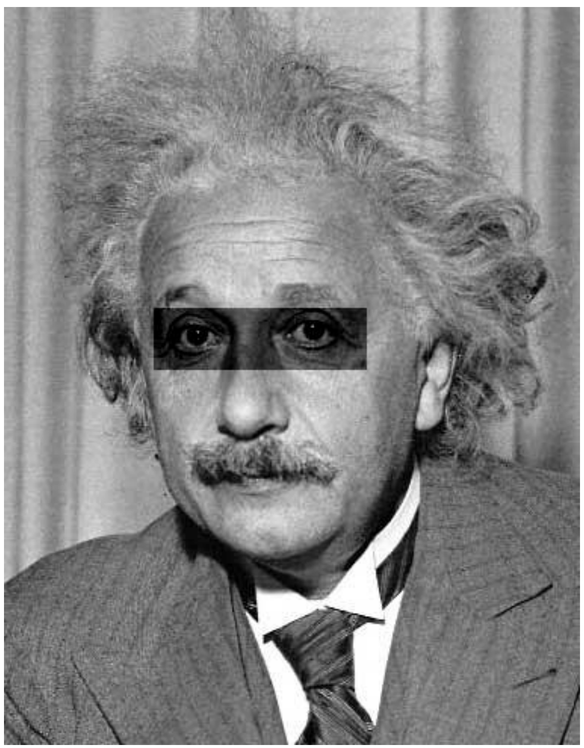


Thresholded Image

Matching with filters

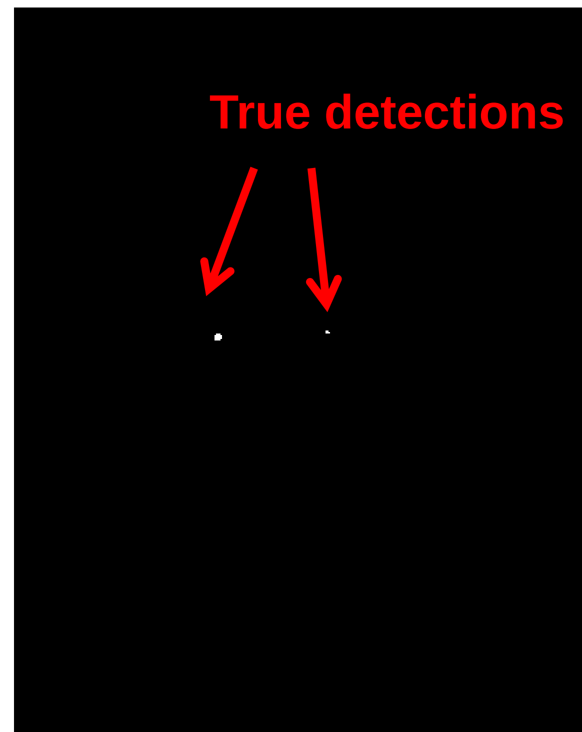
Goal: find  in image

Method 3: Normalized cross-correlation



Input

Normalized X-Correlation



Thresholded Image

Q: What is the best method to use?

A: Depends

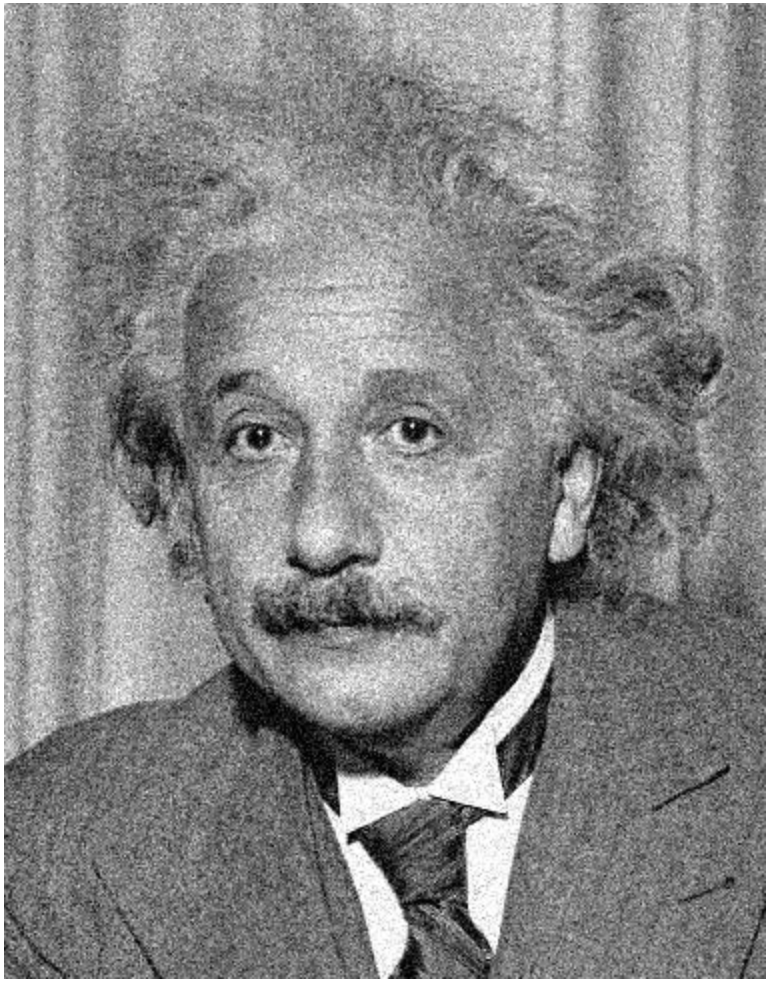
Zero-mean filter: fastest but not a great matcher

SSD: next fastest, sensitive to overall intensity

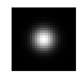
Normalized cross-correlation: slowest, invariant to local average intensity and contrast

Side by Derek Hoiem

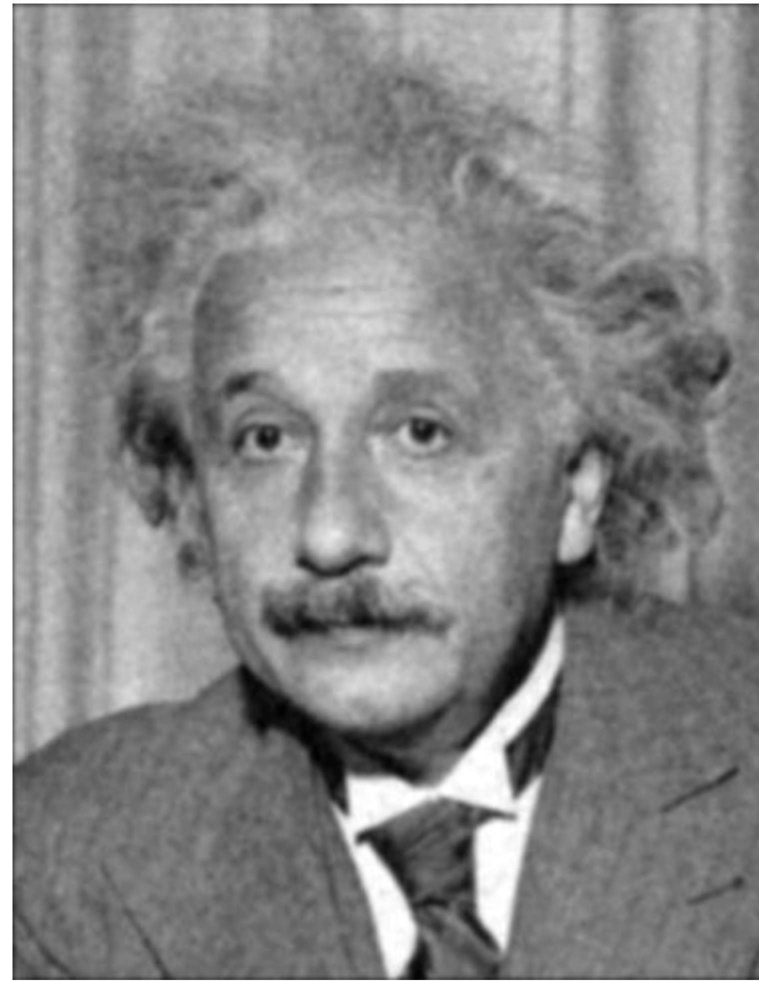
Denoising



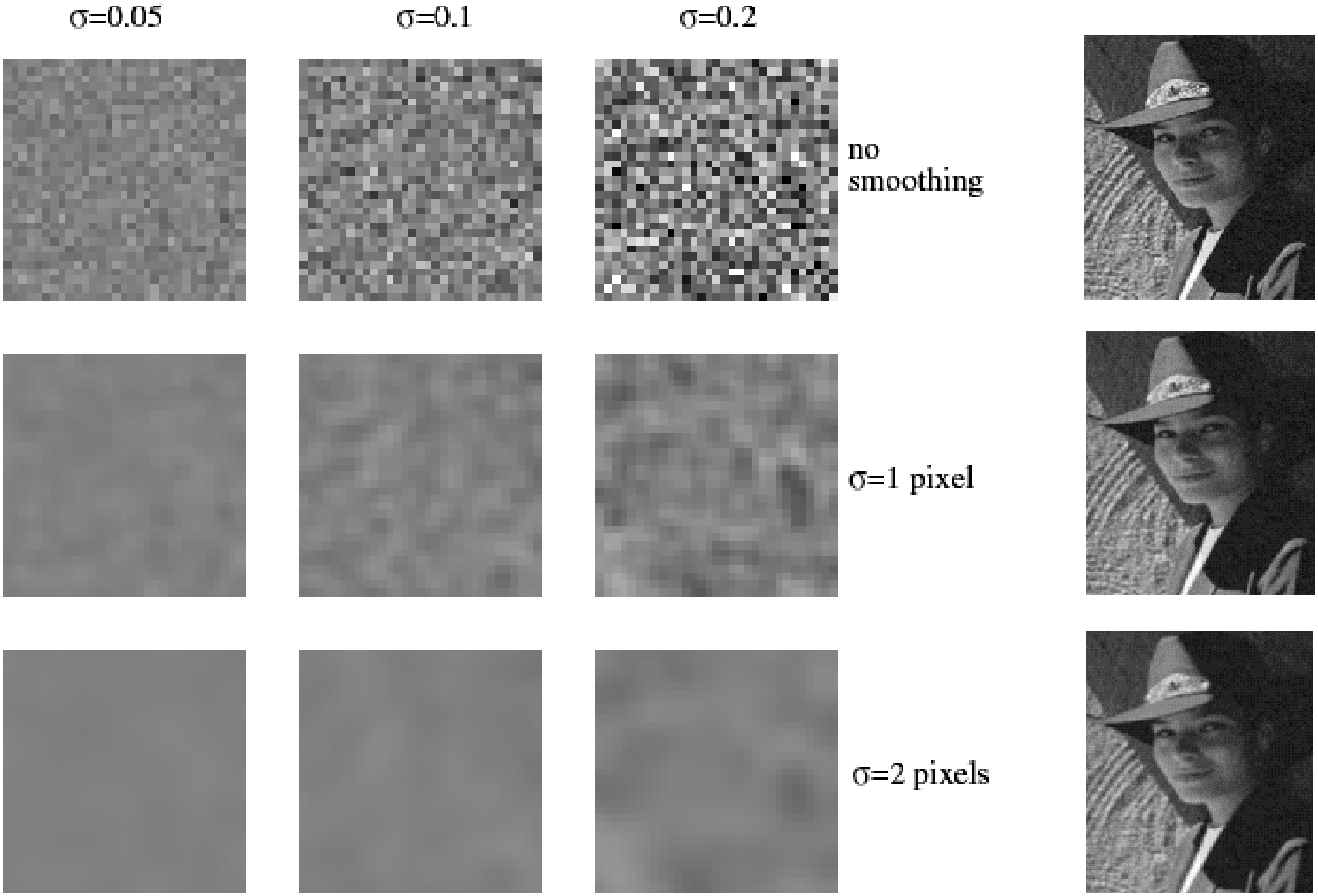
Additive Gaussian Noise



Gaussian
Filter



Reducing Gaussian noise



Smoothing with larger standard deviations suppresses noise, but also blurs the image

Source: S. Lazebnik

Reducing salt-and-pepper noise by Gaussian smoothing

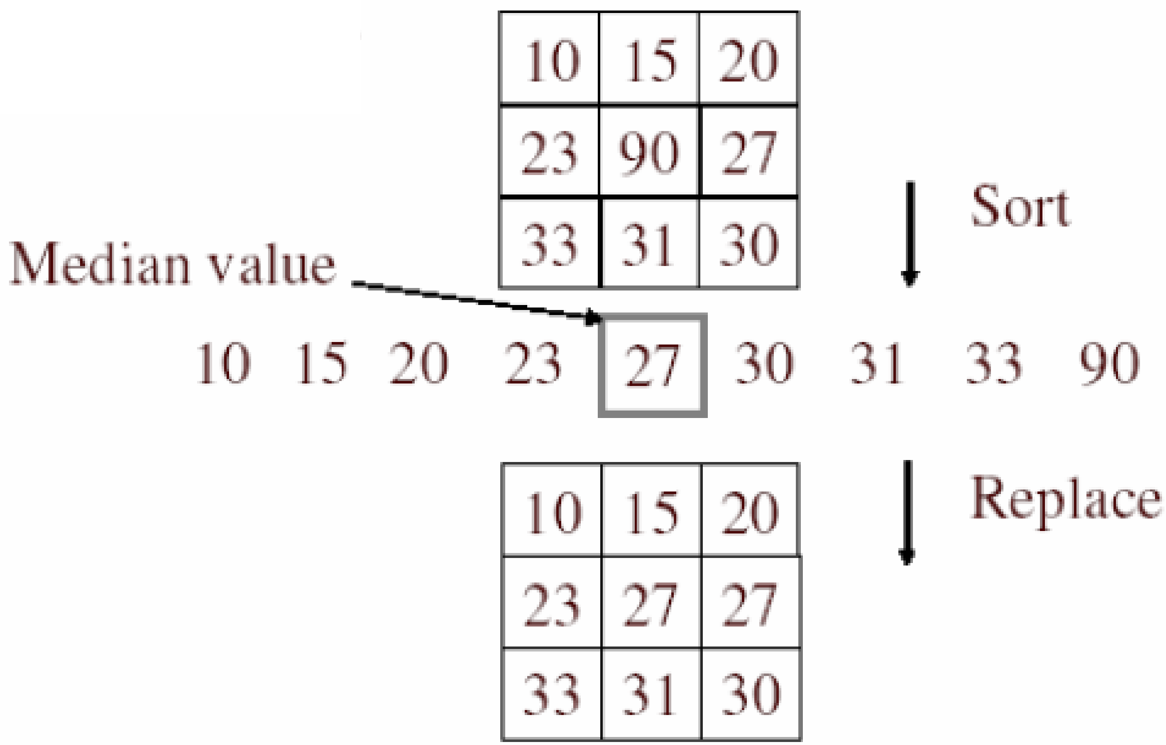
3x3

5x5

7x7

Alternative idea: Median filtering

A **median filter** operates over a window by selecting the median intensity in the window



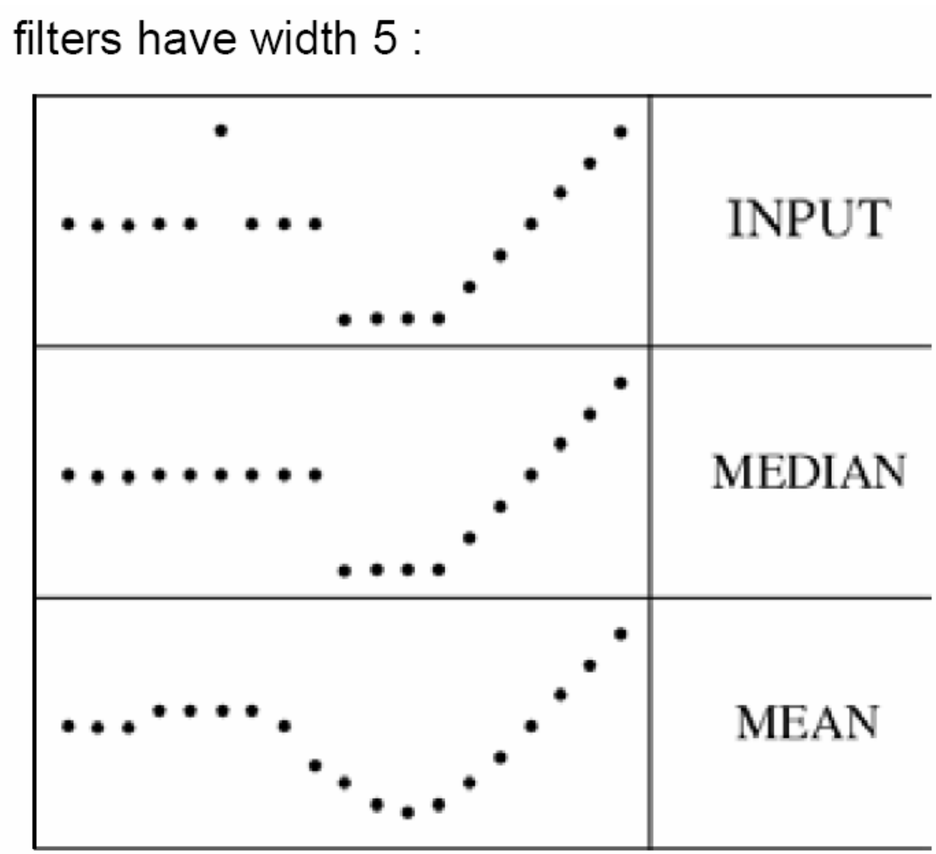
- Is median filtering linear?

Source: K. Grauman

Median filter

What advantage does median filtering have over Gaussian filtering?

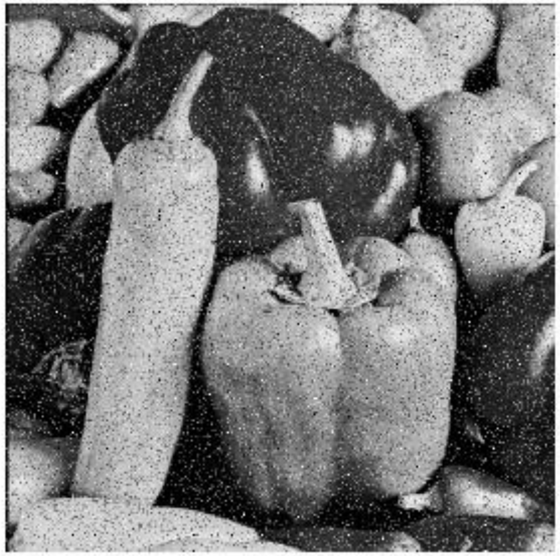
- Robustness to outliers

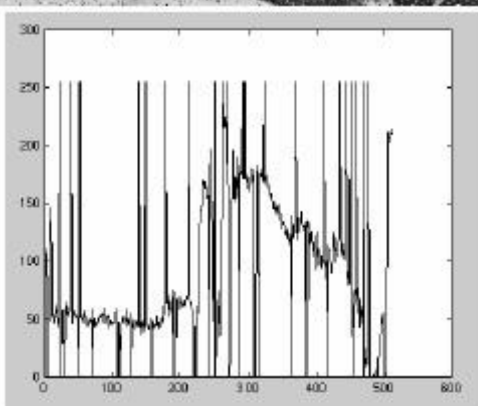


Source: K. Grauman

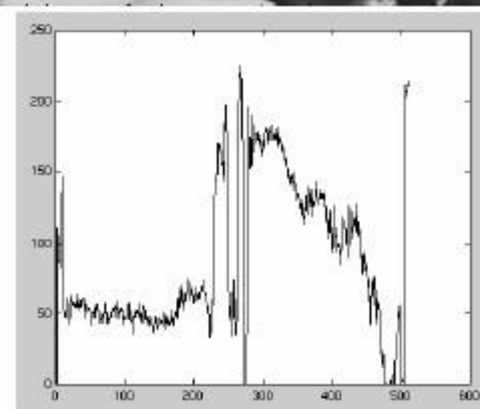
Median filter

Salt-and-pepper noise





Median filtered



MATLAB: medfilt2(image, [h w])

Source: M. Hebert

Median vs. Gaussian filtering

3x3 5x5 7x7

Gaussian

Median