# **Bilateral** Filter

image $I(X,Y)$

| | | | | | |
|---|---|---|---|---|---|
| 25 | 0 | 25 | 255 | 255 | 255 |
| 0 | 0 | 0 | 230 | 255 | 255 |
| 0 | 25 | 25 | 255 | 230 | 255 |
| 0 | 0 | 25 | 255 | 255 | 255 |

$\rightarrow$

Normalised

image $I(X,Y)$

| | | | | | |
|---|---|---|---|---|---|
| 0.1 | 0 | 0.1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0.9 | 1 | 1 |
| 0 | 0.1 | 0.1 | 1 | 0.9 | 1 |
| 0 | 0 | 0.1 | 1 | 1 | 1 |

**Domain** Kernel

$\sigma_d = 1$

| | | |
|---|---|---|
| 0.08 | 0.12 | 0.08 |
| 0.12 | 0.20 | 0.12 |
| 0.08 | 0.12 | 0.08 |

| **domain** kernel | $\exp^{-\frac{x^2+y^2}{2\sigma_d^2}}$ |
|---|---|

# **Bilateral** Filter

image $I(X,Y)$

| | | | | | |
|---|---|---|---|---|---|
| 25 | 0 | 25 | 255 | 255 | 255 |
| 0 | 0 | 0 | 230 | 255 | 255 |
| 0 | 25 | 25 | 255 | 230 | 255 |
| 0 | 0 | 25 | 255 | 255 | 255 |

$\rightarrow$

Normalised

image $I(X,Y)$

| | | | | | |
|---|---|---|---|---|---|
| 0.1 | 0 | 0.1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0.9 | 1 | 1 |
| 0 | 0.1 | 0.1 | 1 | 0.9 | 1 |
| 0 | 0 | 0.1 | 1 | 1 | 1 |

**Domain** Kernel

$\sigma_d = 1$

| | | |
|---|---|---|
| 0.08 | 0.12 | 0.08 |
| 0.12 | 0.20 | 0.12 |
| 0.08 | 0.12 | 0.08 |

**Range** Kernel

$\sigma_r = 0.45$

| | | |
|---|---|---|
| 0.98 | 0.98 | 0.2 |
| 1 | 1 | 0.1 |
| 0.98 | 1 | 0.1 |

(differences based on **centre pixel**)

$$\exp^{-\frac{(I(X+x,Y+y)-I(X,Y))^2}{2\sigma_r^2}}$$

**range** kernel

# **Bilateral** Filter

image $I(X,Y)$

| | | | | | |
|---|---|---|---|---|---|
| 25 | 0 | 25 | 255 | 255 | 255 |
| 0 | 0 | 0 | 230 | 255 | 255 |
| 0 | 25 | 25 | 255 | 230 | 255 |
| 0 | 0 | 25 | 255 | 255 | 255 |

Normalised

image $I(X,Y)$

| | | | | | |
|---|---|---|---|---|---|
| 0.1 | 0 | 0.1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0.9 | 1 | 1 |
| 0 | 0.1 | 0.1 | 1 | 0.9 | 1 |
| 0 | 0 | 0.1 | 1 | 1 | 1 |

**Domain** Kernel

$\sigma_d = 1$

| | | |
|---|---|---|
| 0.08 | 0.12 | 0.08 |
| 0.12 | 0.20 | 0.12 |
| 0.08 | 0.12 | 0.08 |

**Range** Kernel

$\sigma_r = 0.45$

| | | |
|---|---|---|
| 0.98 | 0.98 | 0.2 |
| 1 | 1 | 0.1 |
| 0.98 | 1 | 0.1 |

(differences based on **centre pixel**)

multiply

**Range** * **Domain** Kernel

| | | |
|---|---|---|
| 0.08 | 0.12 | 0.02 |
| 0.12 | 0.20 | 0.01 |
| 0.08 | 0.12 | 0.01 |

# **Bilateral** Filter

image $I(X,Y)$

| | | | | | |
|---|---|---|---|---|---|
| 25 | 0 | 25 | 255 | 255 | 255 |
| 0 | 0 | 0 | 230 | 255 | 255 |
| 0 | 25 | 25 | 255 | 230 | 255 |
| 0 | 0 | 25 | 255 | 255 | 255 |

Normalised

image $I(X,Y)$

| | | | | | |
|---|---|---|---|---|---|
| 0.1 | 0 | 0.1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0.9 | 1 | 1 |
| 0 | 0.1 | 0.1 | 1 | 0.9 | 1 |
| 0 | 0 | 0.1 | 1 | 1 | 1 |

**Domain** Kernel

$\sigma_d = 1$

| | | |
|---|---|---|
| 0.08 | 0.12 | 0.08 |
| 0.12 | 0.20 | 0.12 |
| 0.08 | 0.12 | 0.08 |

**Range** Kernel

$\sigma_r = 0.45$

| | | |
|---|---|---|
| 0.98 | 0.98 | 0.2 |
| 1 | 1 | 0.1 |
| 0.98 | 1 | 0.1 |

(differences based on **centre pixel**)

multiply

**Range** * **Domain** Kernel

| | | |
|---|---|---|
| 0.08 | 0.12 | 0.02 |
| 0.12 | 0.20 | 0.01 |
| 0.08 | 0.12 | 0.01 |

sum to 1

| | | |
|---|---|---|
| 0.11 | 0.16 | 0.03 |
| 0.16 | 0.26 | 0.01 |
| 0.11 | 0.16 | 0.01 |

# **Bilateral** Filter

image $I(X,Y)$

| | | | | | |
|---|---|---|---|---|---|
| 25 | 0 | 25 | 255 | 255 | 255 |
| 0 | 0 | 0 | 230 | 255 | 255 |
| 0 | 25 | 25 | 255 | 230 | 255 |
| 0 | 0 | 25 | 255 | 255 | 255 |

Normalised image $I(X,Y)$

| | | | | | |
|---|---|---|---|---|---|
| 0.1 | 0 | 0.1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0.9 | 1 | 1 |
| 0 | 0.1 | 0.1 | 1 | 0.9 | 1 |
| 0 | 0 | 0.1 | 1 | 1 | 1 |

**Domain** Kernel

$\sigma_d = 1$

| | | |
|---|---|---|
| 0.08 | 0.12 | 0.08 |
| 0.12 | 0.20 | 0.12 |
| 0.08 | 0.12 | 0.08 |

**Range** Kernel

$\sigma_r = 0.45$

| | | |
|---|---|---|
| 0.98 | 0.98 | 0.2 |
| 1 | 1 | 0.1 |
| 0.98 | 1 | 0.1 |

(differences based on **centre pixel**)

multiply

**Range** * **Domain** Kernel

| | | |
|---|---|---|
| 0.08 | 0.12 | 0.02 |
| 0.12 | 0.20 | 0.01 |
| 0.08 | 0.12 | 0.01 |

$$\sum \begin{bmatrix} 0.11 & 0.16 & 0.03 \\ 0.16 & 0.26 & 0.01 \\ 0.11 & 0.16 & 0.01 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 0.9 \\ 0.1 & 0.1 & 1 \\ 0 & 0.1 & 1 \end{bmatrix} = 0.1$$

**Bilateral** Filter

# **Bilateral** Filter

image $I(X,Y)$

| | | | | | |
|---|---|---|---|---|---|
| 25 | 0 | 25 | 255 | 255 | 255 |
| 0 | 0 | 0 | 230 | 255 | 255 |
| 0 | 25 | 25 | 255 | 230 | 255 |
| 0 | 0 | 25 | 255 | 255 | 255 |

→

Normalised

image $I(X,Y)$

| | | | | | |
|---|---|---|---|---|---|
| 0.1 | 0 | 0.1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0.9 | 1 | 1 |
| 0 | 0.1 | 0.1 | 1 | 0.9 | 1 |
| 0 | 0 | 0.1 | 1 | 1 | 1 |

**Domain** Kernel

$\sigma_d = 1$

$$\sum \begin{bmatrix} 0.08 & 0.12 & 0.08 \\ 0.12 & 0.20 & 0.12 \\ 0.08 & 0.12 & 0.08 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 0.9 \\ 0.1 & 0.1 & 1 \\ 0 & 0.1 & 1 \end{bmatrix} = 0.3$$

**Gaussian** Filter (only)

**Range** Kernel

$\sigma_r = 0.45$

| | | |
|---|---|---|
| 0.98 | 0.98 | 0.2 |
| 1 | 1 | 0.1 |
| 0.98 | 1 | 0.1 |

(differences based on **centre pixel**)

multiply →

**Range** * **Domain** Kernel

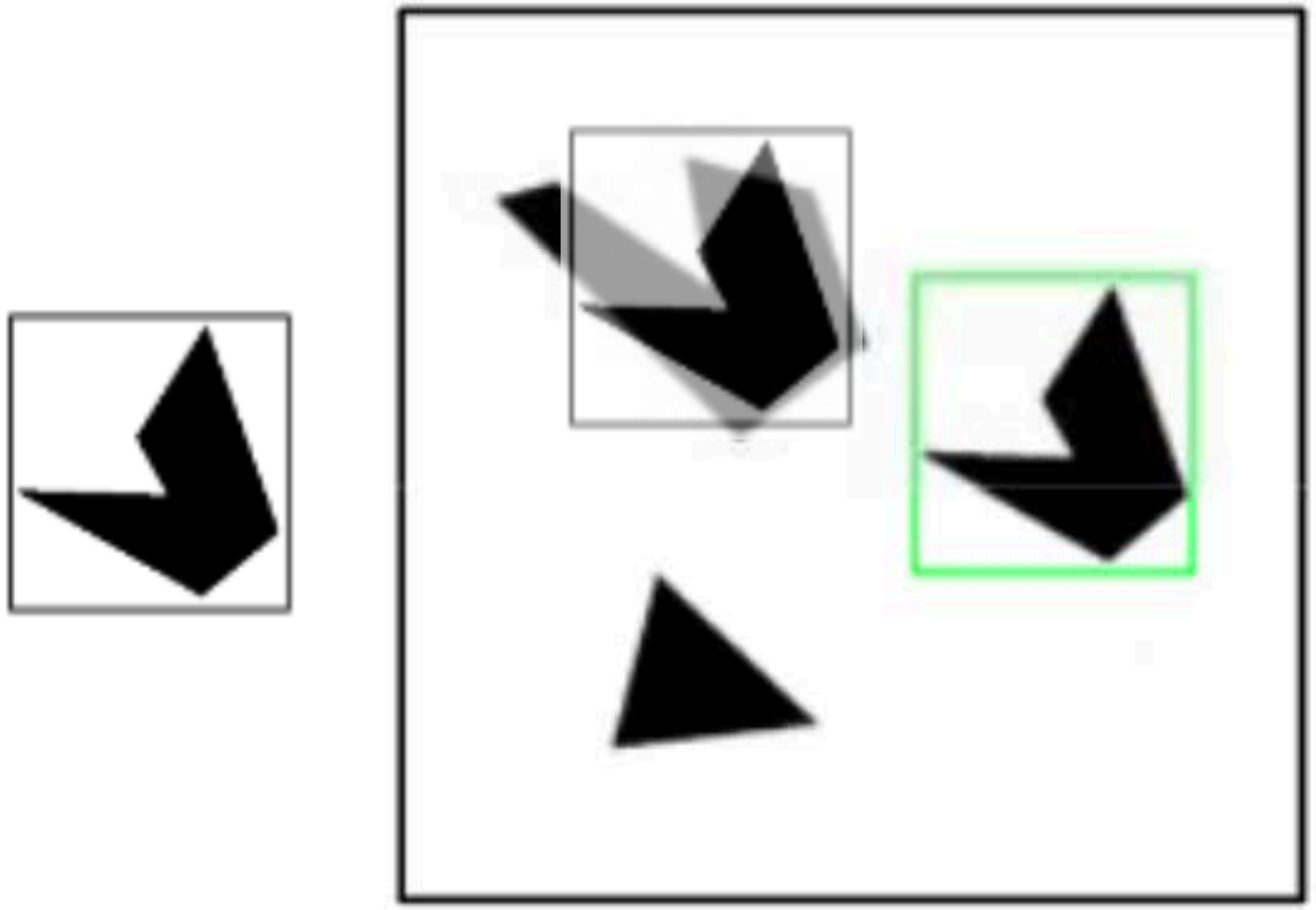
| | | |
|---|---|---|
| 0.08 | 0.12 | 0.02 |
| 0.12 | 0.20 | 0.01 |
| 0.08 | 0.12 | 0.01 |

$$\sum \begin{bmatrix} 0.11 & 0.16 & 0.03 \\ 0.16 & 0.26 & 0.01 \\ 0.11 & 0.16 & 0.01 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 0.9 \\ 0.1 & 0.1 & 1 \\ 0 & 0.1 & 1 \end{bmatrix} = 0.1$$

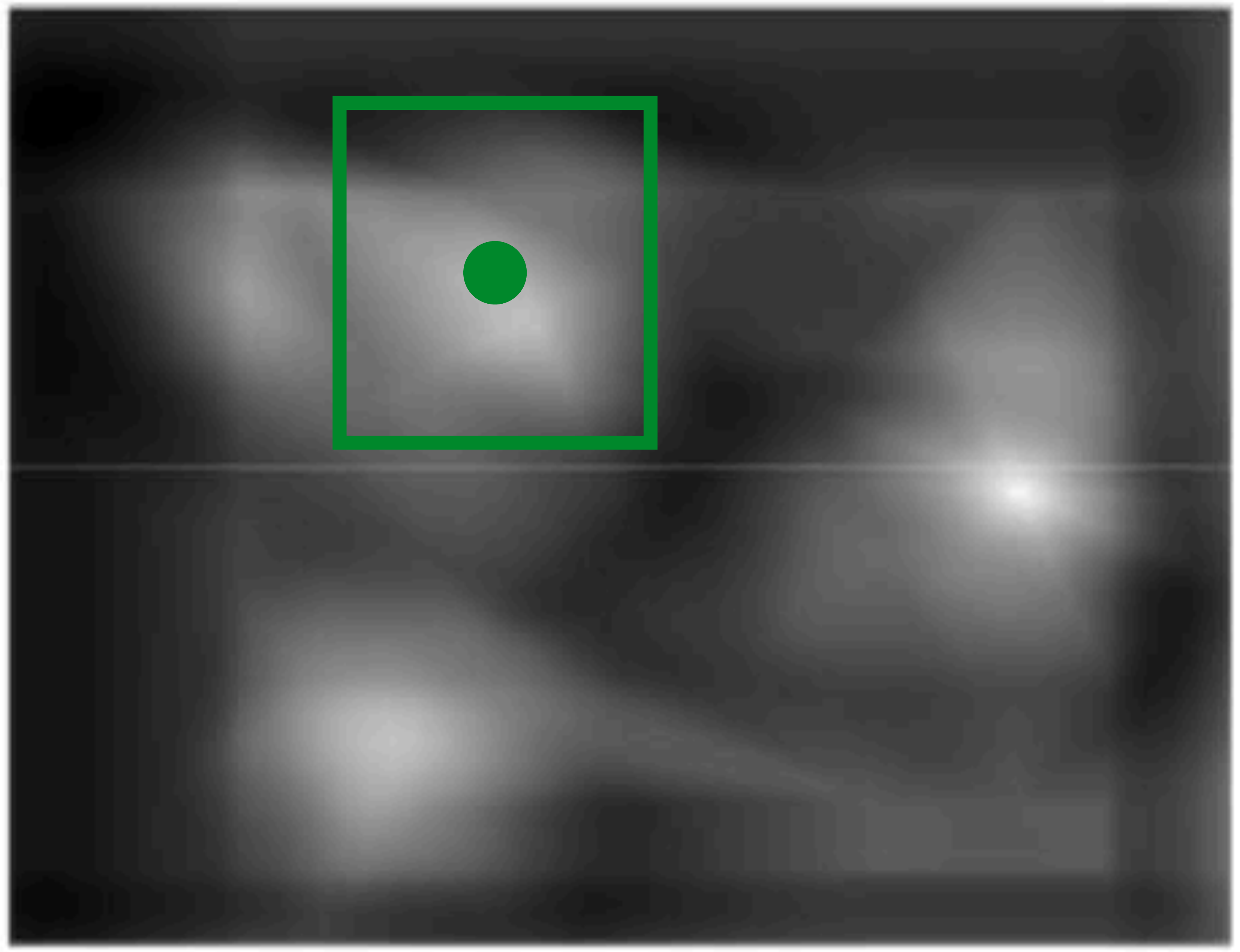
**Bilateral** Filter

# **Template** Matching

Assuming template is all positive, what does this tell us about correlation map?



Detected template



Correlation map

$$\frac{a}{|a|}\frac{b}{|b|} = ?$$

**Slide Credit**: Kristen Grauman

# **Template** Matching

We can think of convolution/**correlation** as comparing a template (the filter) with each local image patch.

— Consider the filter and image patch as vectors.

— Applying a filter at an image location can be interpreted as computing the dot product between the filter and the local image patch.

Image Patch 1 $\begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 1 & 1 \\ \hline\end{array}$

Template $\begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 1 & 1 \\ \hline\end{array}$ $\xrightarrow{\text{element multiply}}$ $\begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 1 & 1 \\ \hline\end{array} = 3$

Image Patch 2 $\begin{array}{|c|c|c|}\hline 1 & 0 & 1 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline\end{array} \times 255$

$\begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline\end{array} = 1 \times 255$

# **Template** Matching

Let $a$ and $b$ be vectors. Let $\theta$ be the angle between them. We know

$$\cos\theta = \frac{a \cdot b}{|a||b|} = \frac{a \cdot b}{\sqrt{(a \cdot a)(b \cdot b)}} = \frac{a}{|a|}\frac{b}{|b|}$$

where · is dot product and | | is vector magnitude

1. Normalize the template / filter ($b$) in the beginning
2. Compute norm of $|a|$ by convolving squared image with a filter of all 1's of equal size to the the template and square-rooting the response
3. We can compute the dot product by correlation of image ($a$) with normalized filter ($b$)
4. We can finally compute the normalized correlation by dividing element-wise result in Step 3 by result in Step 2

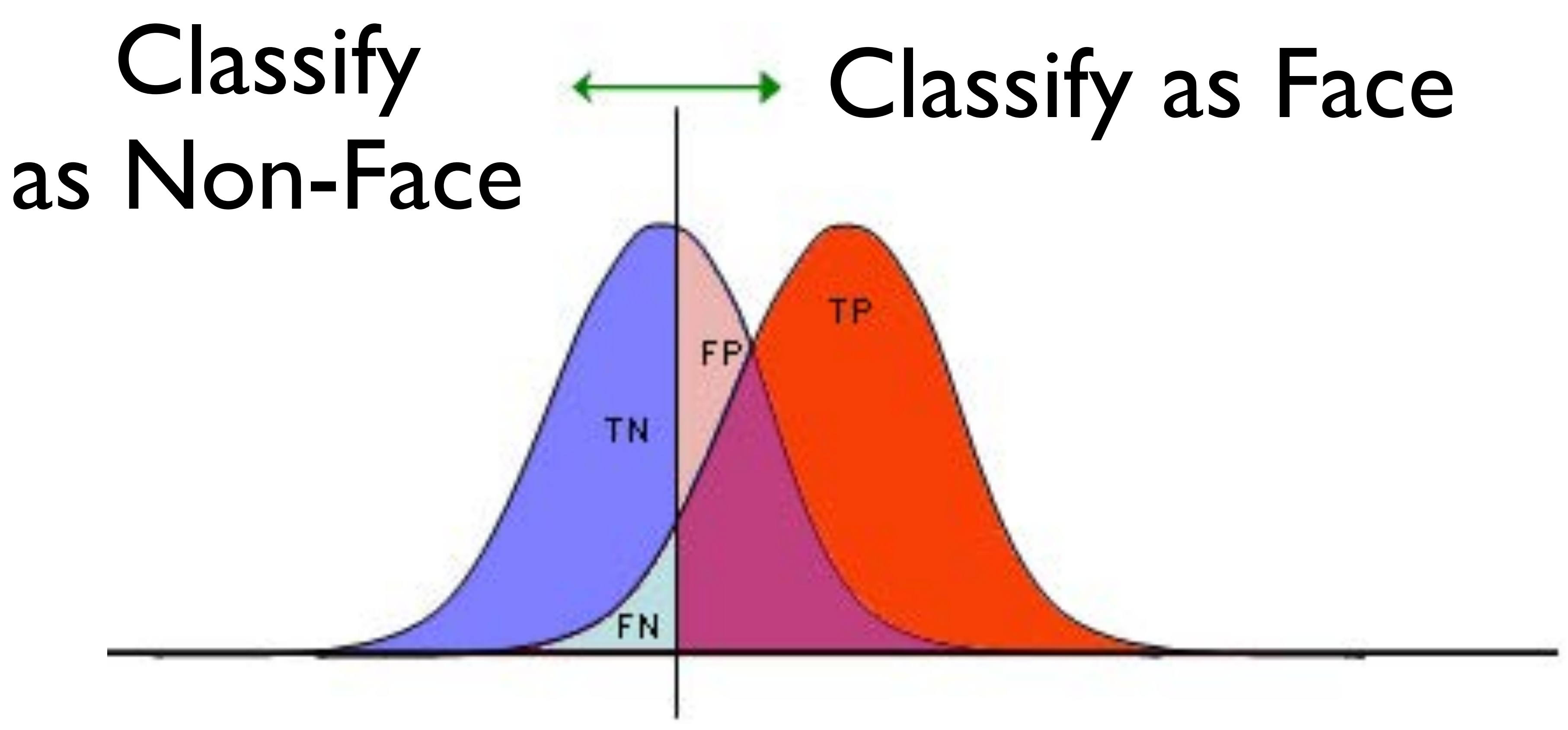
# ROC Curves

Note that we can easily get 100% true positives (if we are prepared to get 100% false positives as well!)

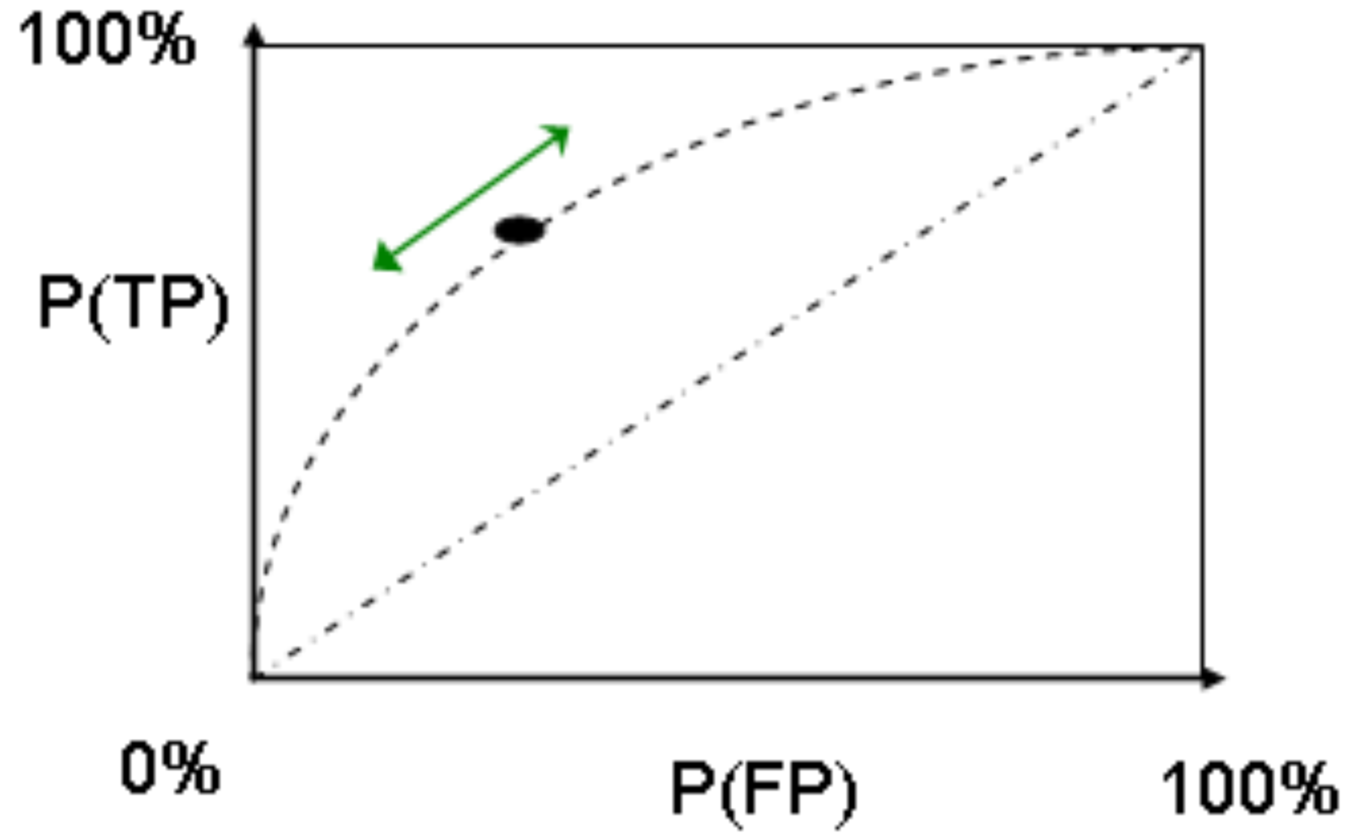
It is a tradeoff between **true positive rate (TP)** and **false positive rate (FP)**

We can plot a curve of all TP rates vs FP rates by varying the classifier threshold

This is a **Receiver Operating Characteristic (ROC)** curve




red = actual faces, blue = actual non-faces



# CPSC 425: Computer Vision

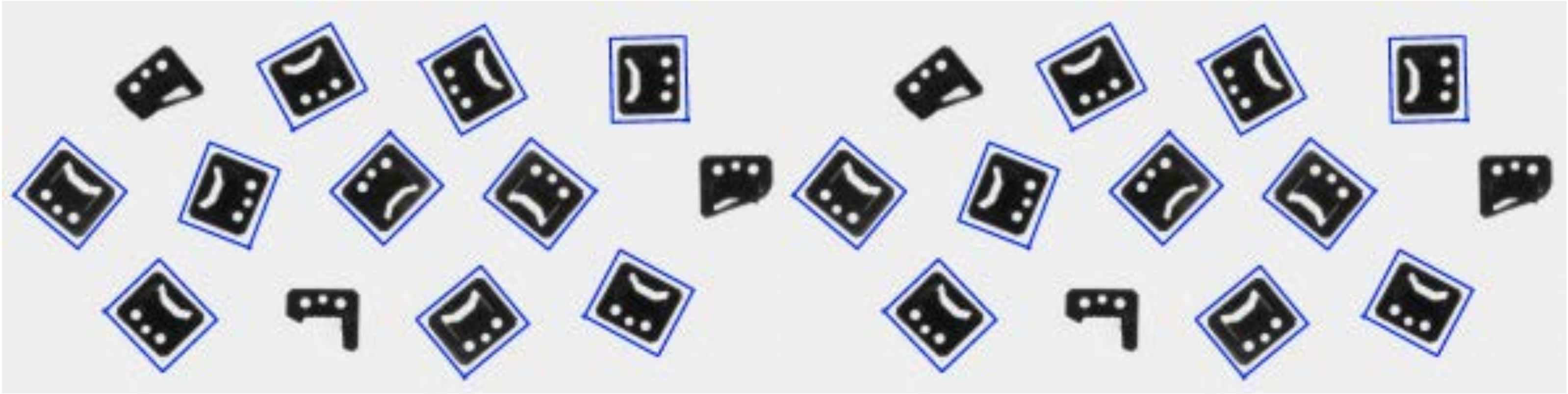


**Image Credit**: https://docs.adaptive-vision.com/4.7/studio/machine_vision_guide/TemplateMatching.html

## **Lecture 8:** Scaled Representations

( unless otherwise stated slides are taken or adopted from **Bob Woodham, Jim Little** and **Fred Tung** )

# **Menu** for Today

## Topics:

— **Scaled** Representations

— Image **Pyramid**

— Multi-scale Template **Matching**

## Readings:

— **Today's** Lecture: Szeliski 2.3, 3.5, Forsyth & Ponce (2nd ed.) 4.5 - 4.7

## Reminders:

— **Quiz 2** is up. (Open until tomorrow midnight)

— **Assignment 2**: Scaled Representations, Face Detection and Image Blending available now

# Goal

1. Understand the idea behind image pyramids

2. Understand laplacian pyramids

# **Multi-Scale** Template Matching

**Problem**: Make template matching robust to changes in 2D (spatial) scale.

**Key Idea**(s): Build a scaled representation: the Gaussian image pyramid

**Alternatives**:

— use multiple sizes for each given template

— ignore the issue of 2D (spatial) scale

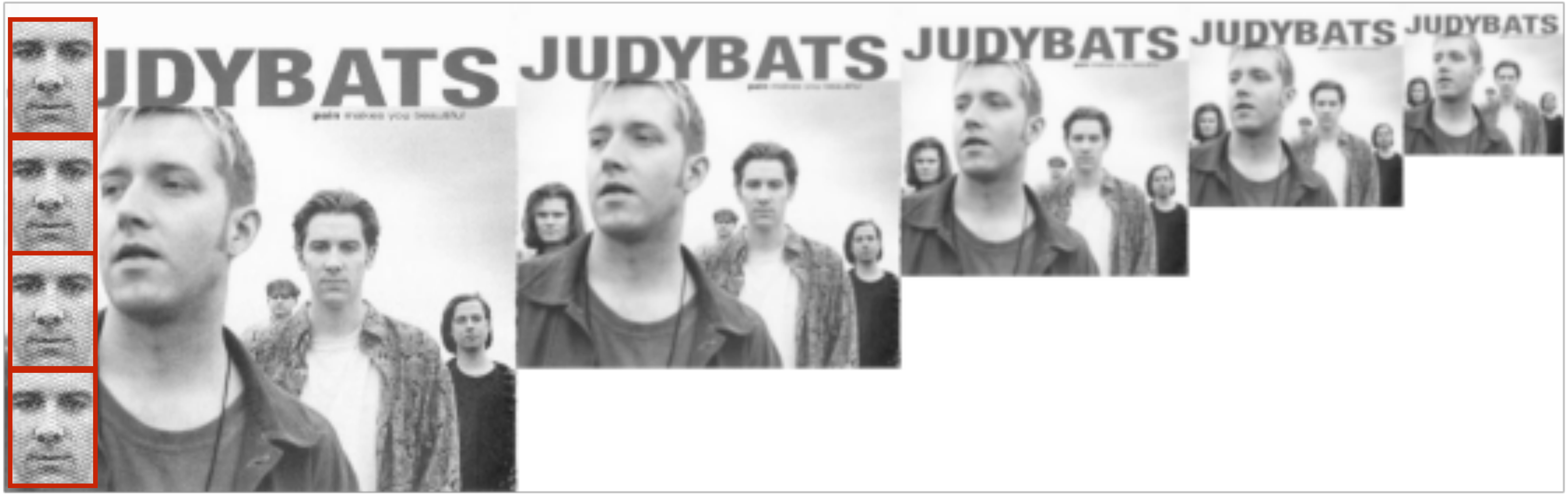
**Theory**: Sampling theory allows us to build image pyramids in a principled way

“**Gotchas**:”

— template matching remains sensitive to 2D orientation, 3D pose and illumination

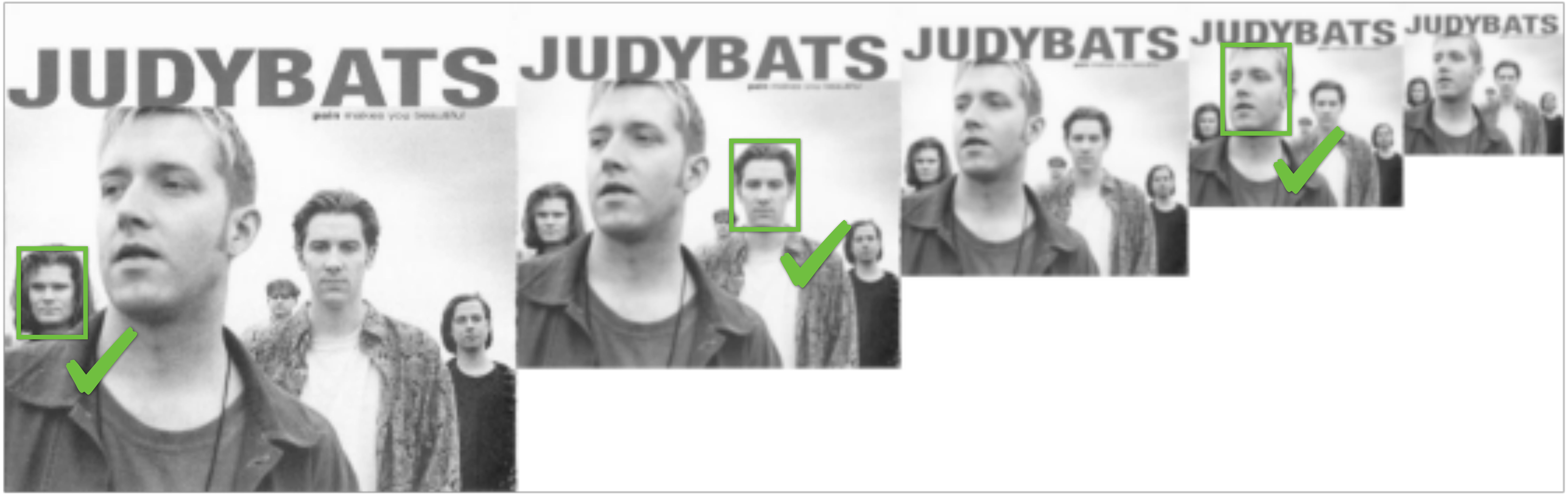
# **Multi-Scale** Template Matching

**Correlation** with a **fixed-sized template** only detects faces at **specific scales**



# **Multi-Scale** Template Matching

**Solution:** form a Gaussian Pyramid and convolve with the template at each scale

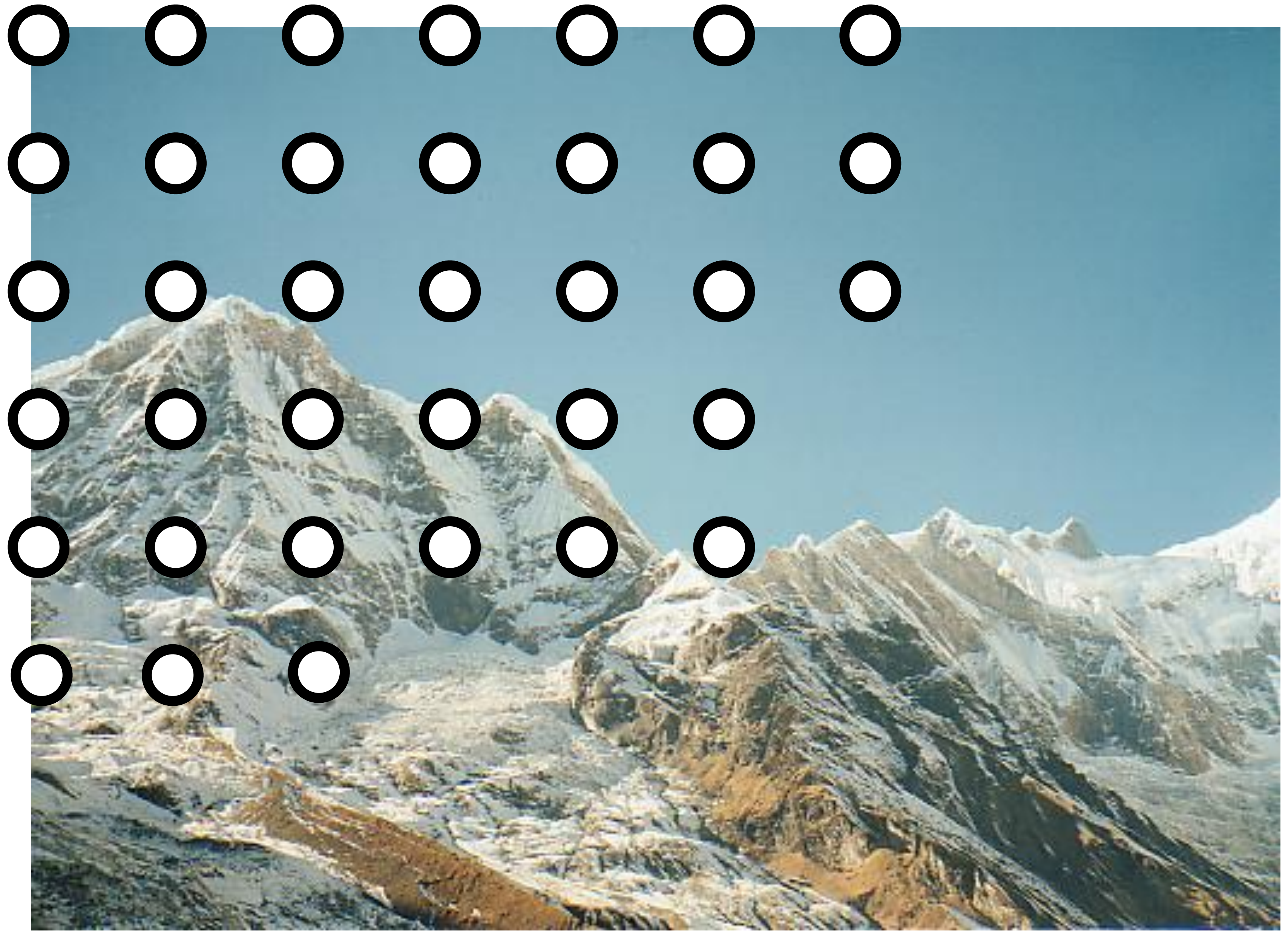


# **Shrinking** the Image

We can't shrink an image simply by taking every second pixel

 Why?

# Aliasing Example

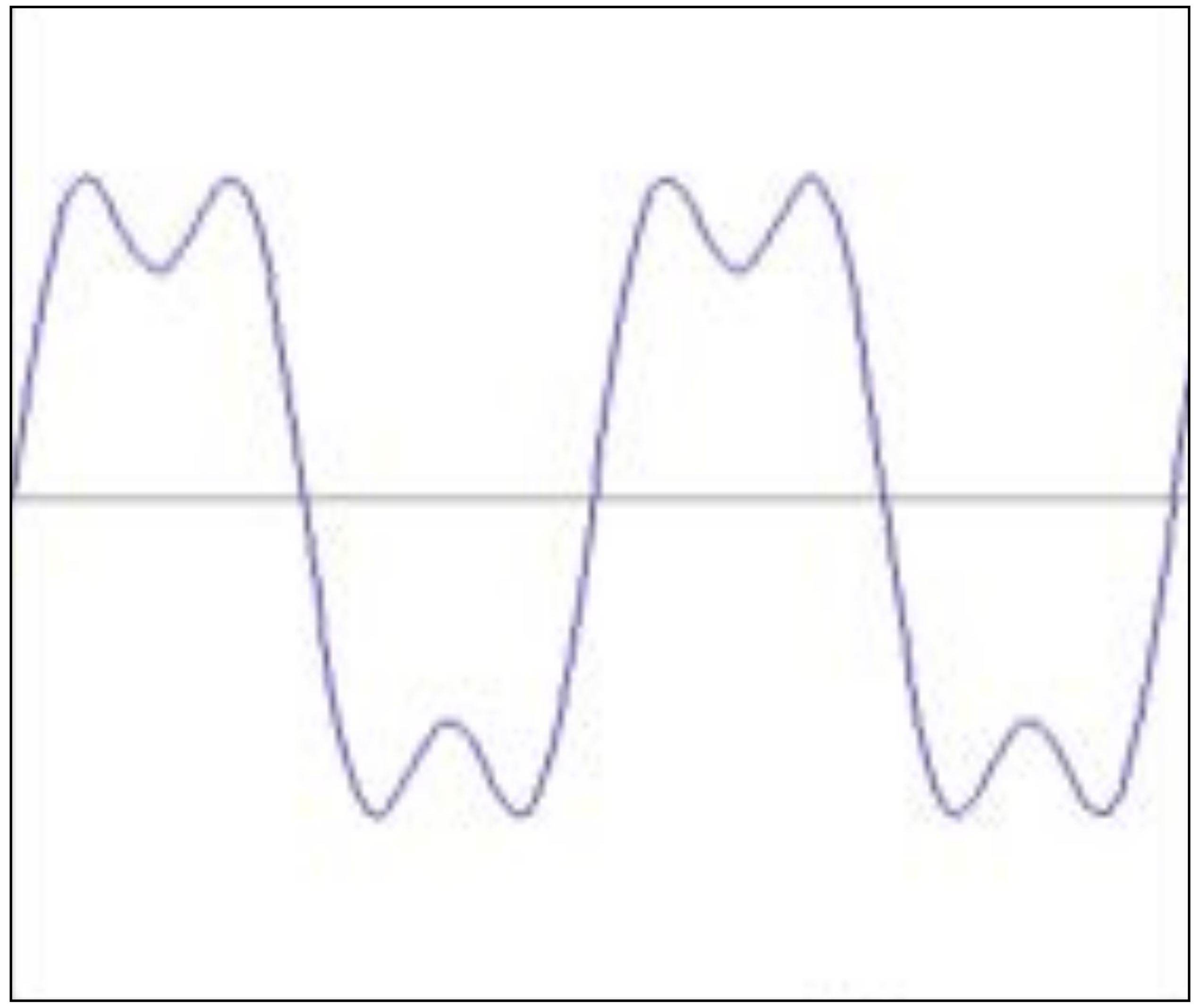


No filtering

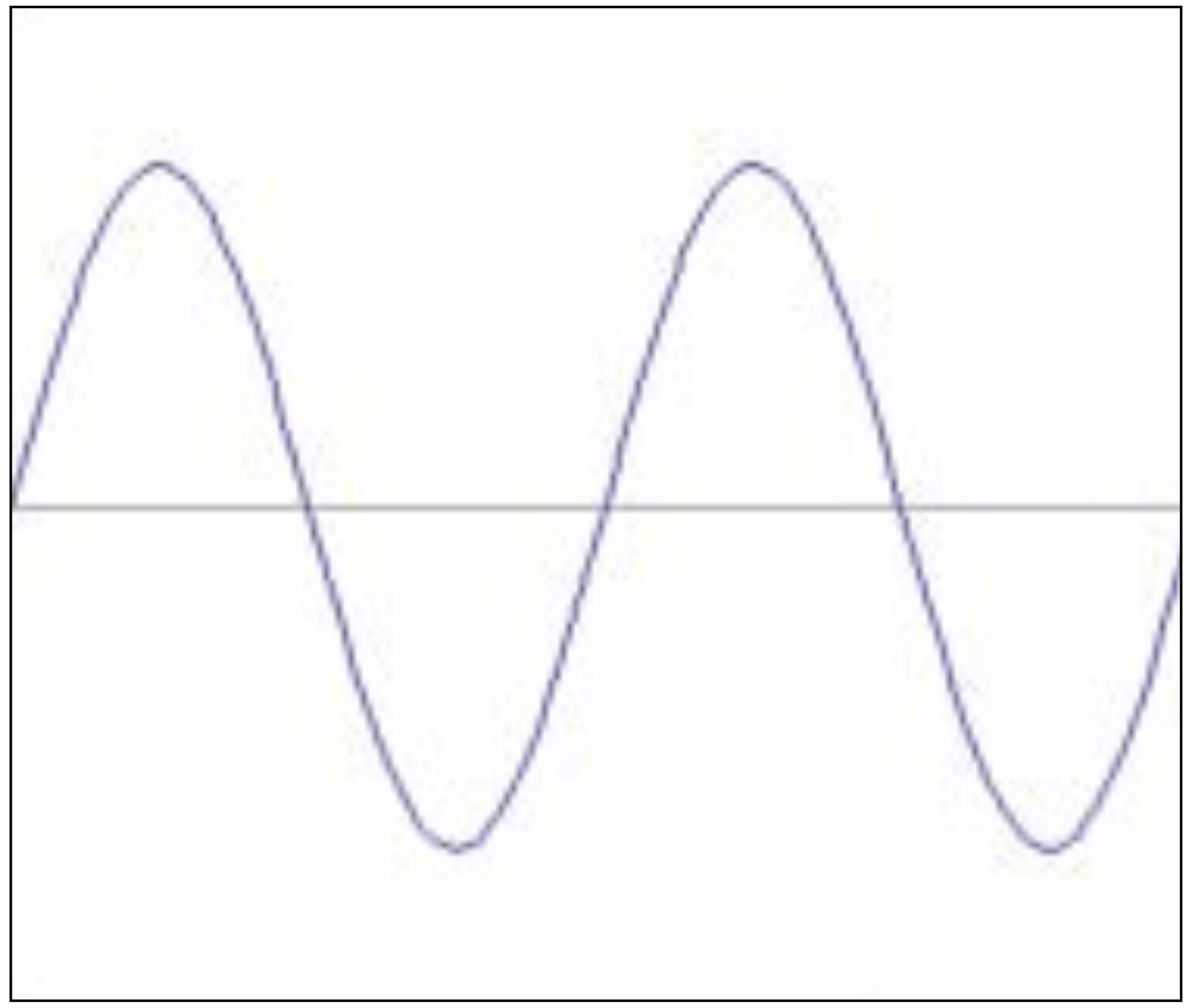
Gaussian Blur $\sigma = 3.0$

# Recall: **Fourier** Representation

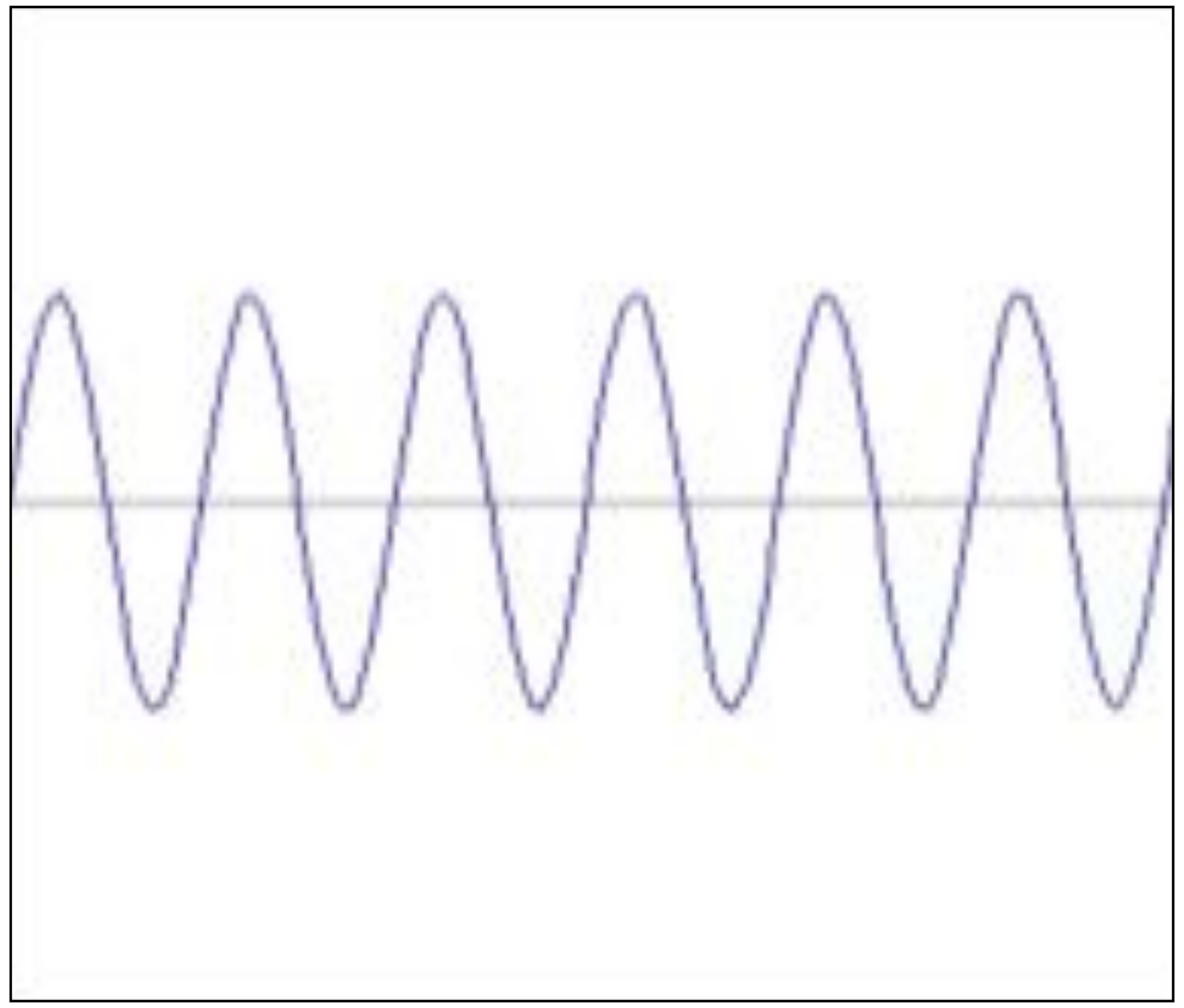
Any signal can be written as a sum of sinusoidal functions



=



+



$$f(x) = \sin(2\pi x) + \frac{1}{3}\sin(2\pi 3x)$$

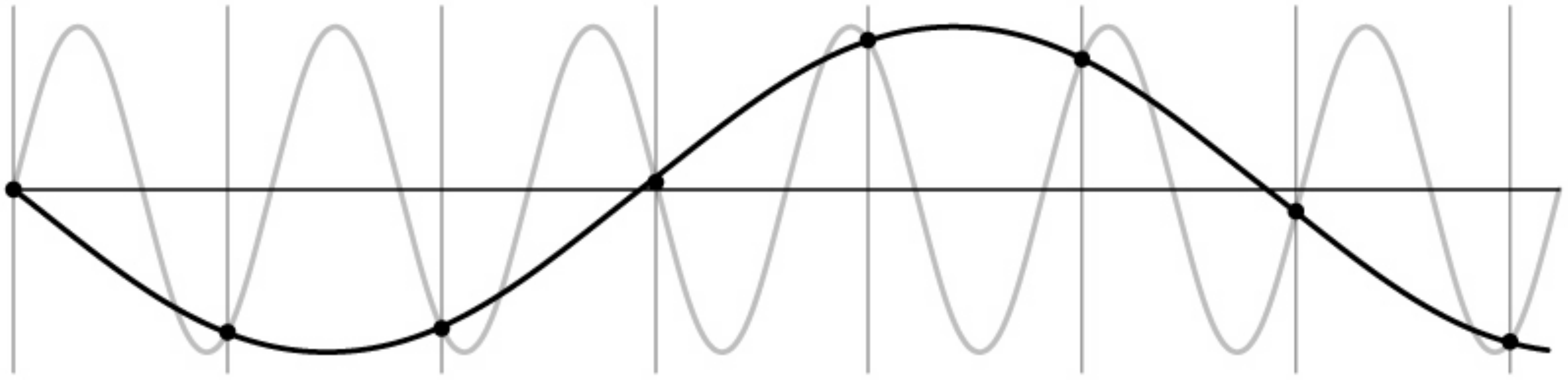
$$\sin(2\pi x)$$

$$\frac{1}{3}\sin(2\pi 3x)$$

**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# **Recall**: Aliasing

Signal has been sampled too infrequently — result = **Aliasing**



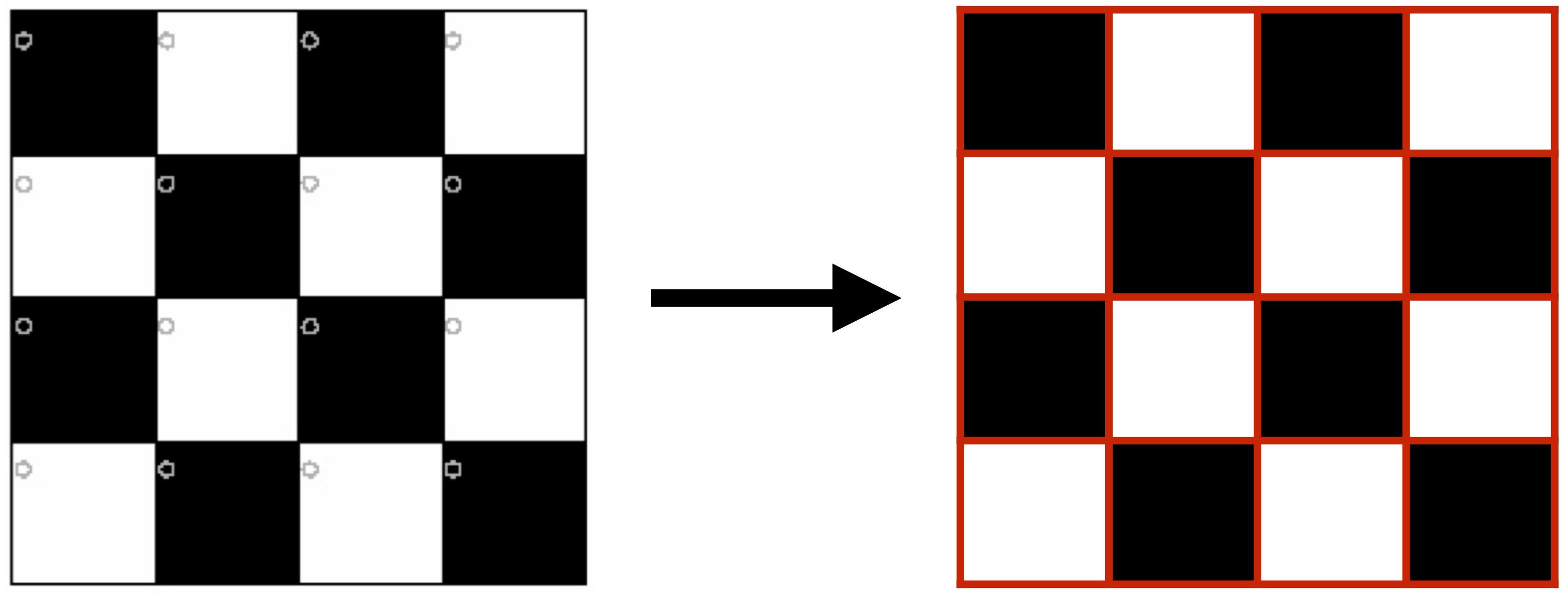
**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

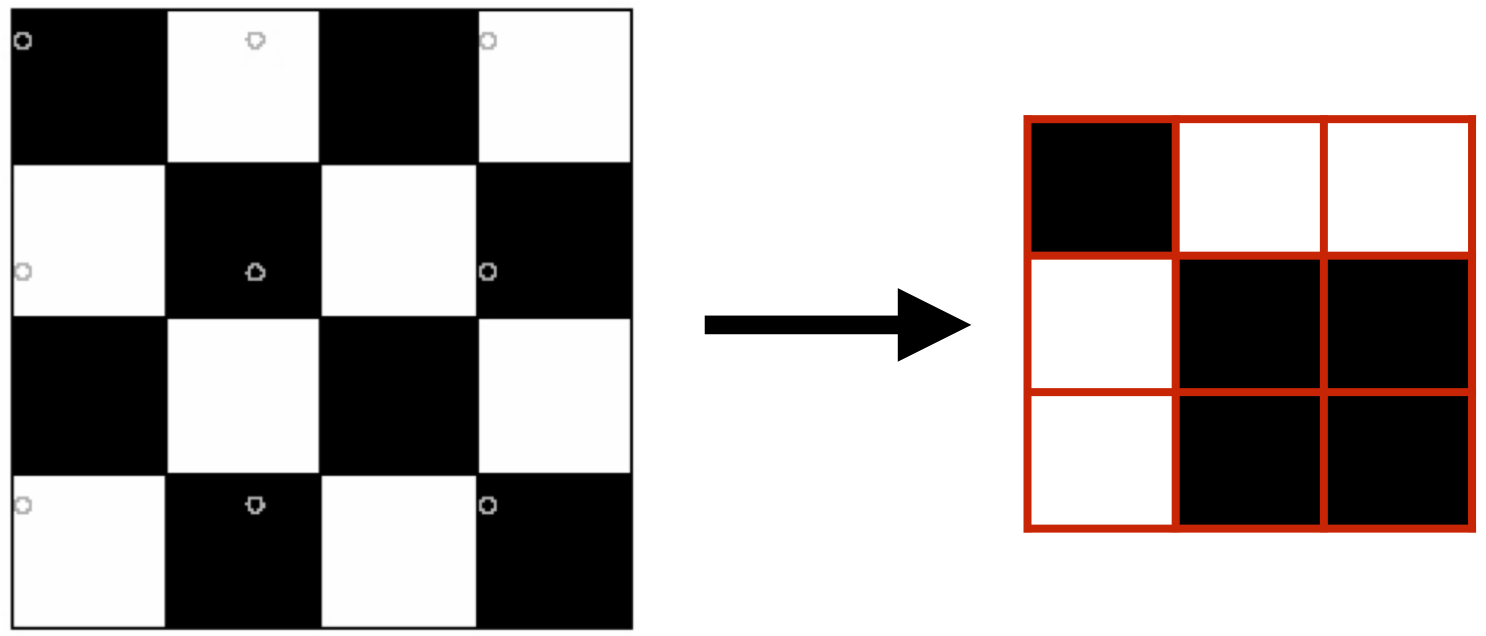
# **Nyquist** Sampling

To avoid aliasing a signal must be sampled at twice the maximum frequency:

$$f_s > 2 \times f_{max}$$

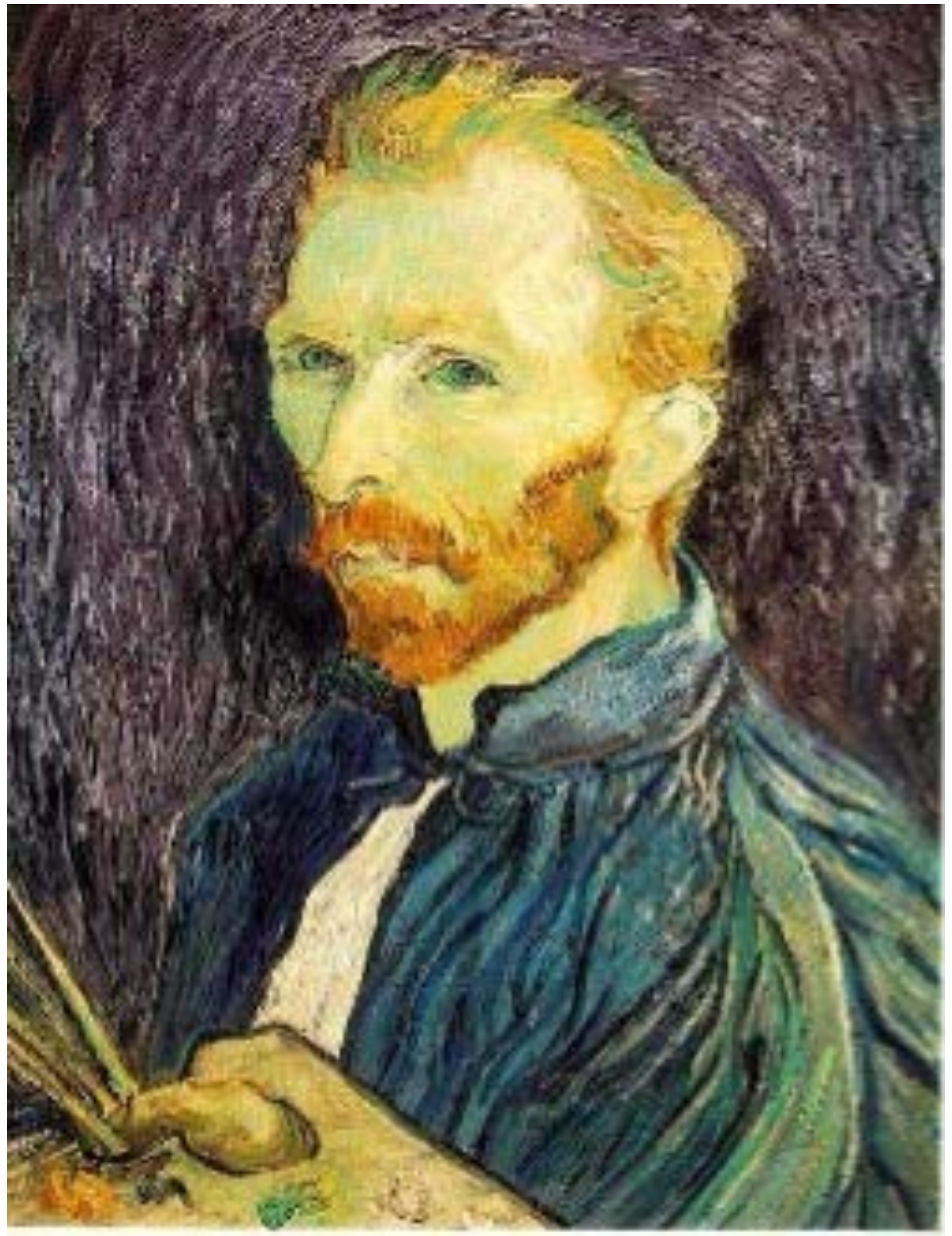
For Images: We need to sample the underlying continuous signal **at least once per pixel** to avoid aliasing (assuming a correctly sampled image)





undersampling = aliasing

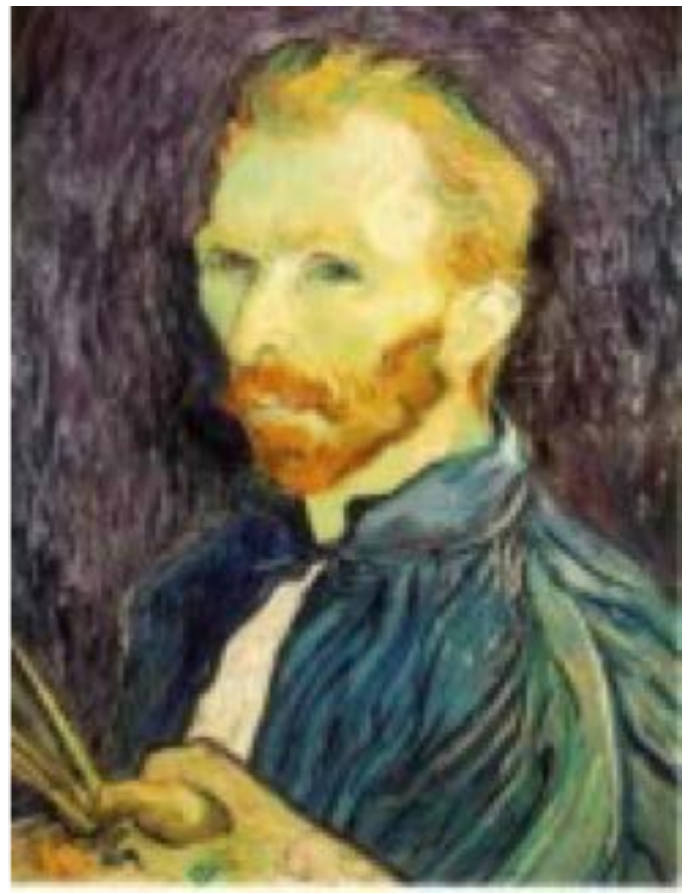
# **Template** Matching: Sub-sample with Gaussian Pre-filtering



1/2

Apply a smoothing filter first, then throw away half the rows and columns

Gaussian filter
delete even rows
delete even columns



1/4

Gaussian filter
delete even rows
delete even columns

1/8

**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

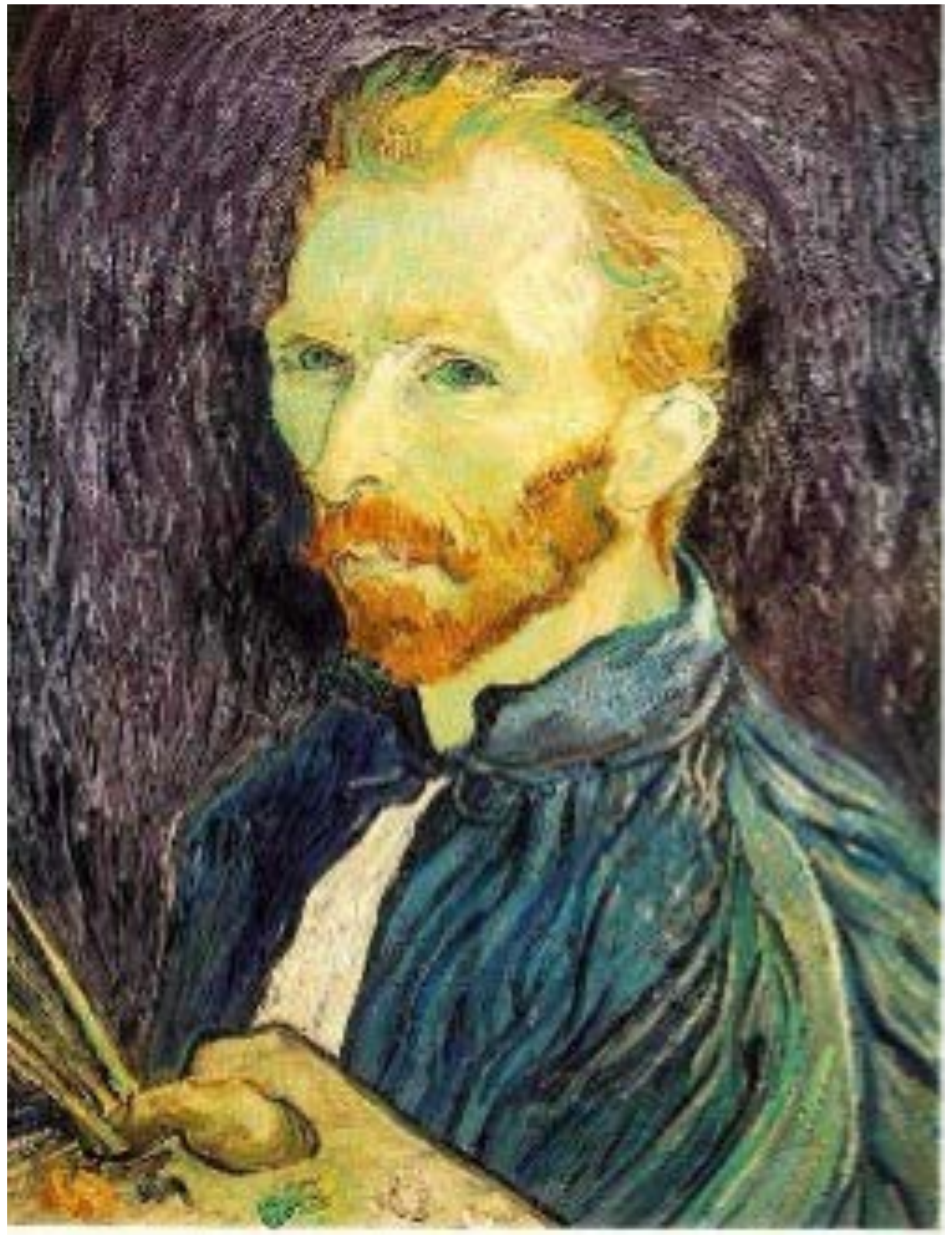
# **Gaussian** Pre-filtering

**Question**: How much smoothing is needed to avoid aliasing?

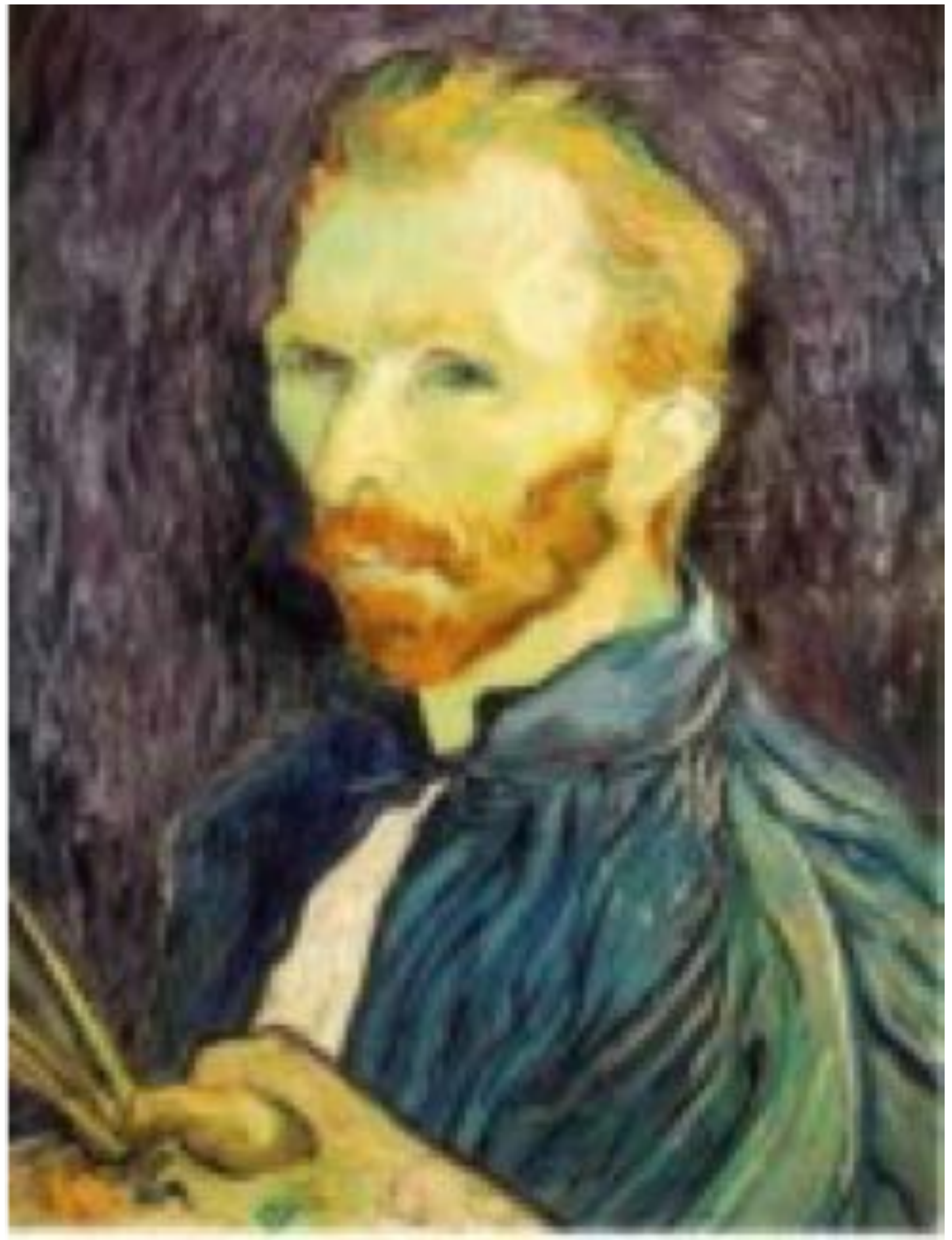
**Answer:** Smoothing should be sufficient to ensure that the resulting image is band limited “enough” to ensure we can sample every other pixel.

**Practically:** For every image reduction of 0.5, smooth by $\sigma = 1$

# **Template** Matching: Sub-sample with Gaussian Pre-filtering



1/2

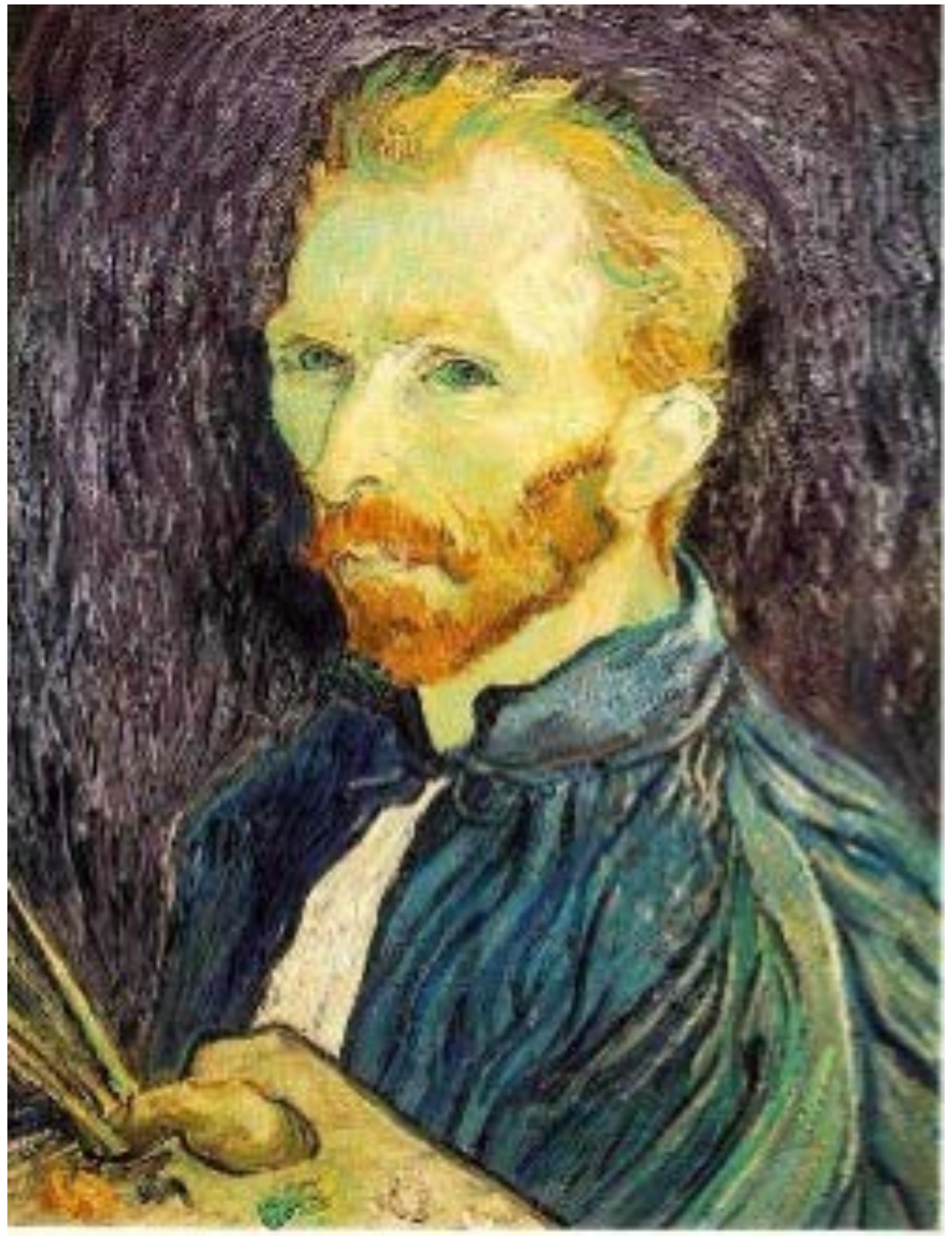


1/4 (2x zoom)

1/8 (4x zoom)

**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# **Template** Matching: Sub-sample with NO Pre-filtering



1/2

1/4 (2x zoom)

1/8 (4x zoom)

**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# Image **Pyramid**

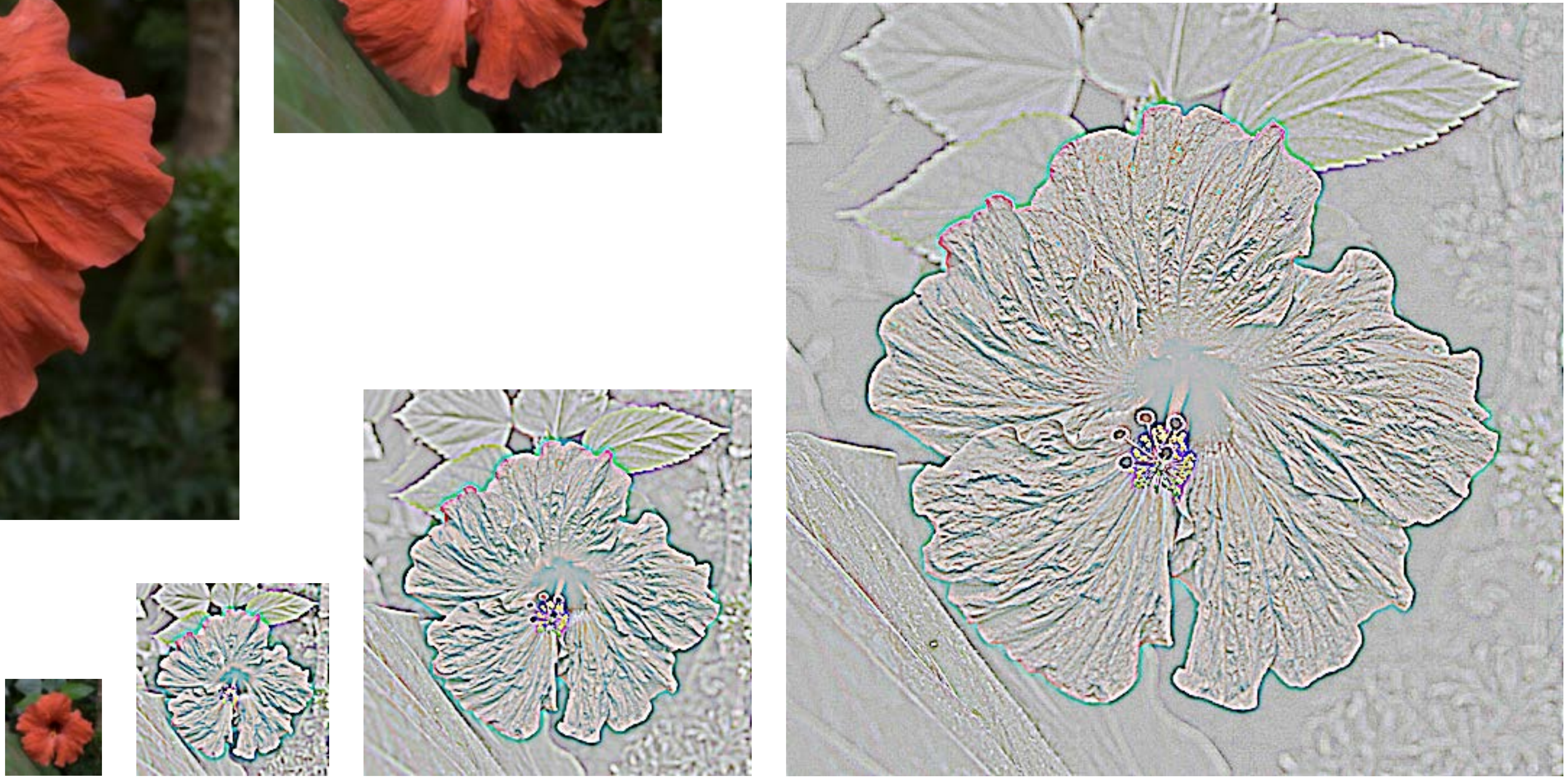
An **image pyramid** is an efficient way to represent an image at multiple scales

In a **Gaussian pyramid**, each layer is smoothed by a Gaussian filter and resampled to get the next layer, taking advantage of the fact that

$$G_{\sigma_1}(x) \otimes G_{\sigma_2}(x) = G_{\sqrt{\sigma_1^2+\sigma_2^2}}(x)$$

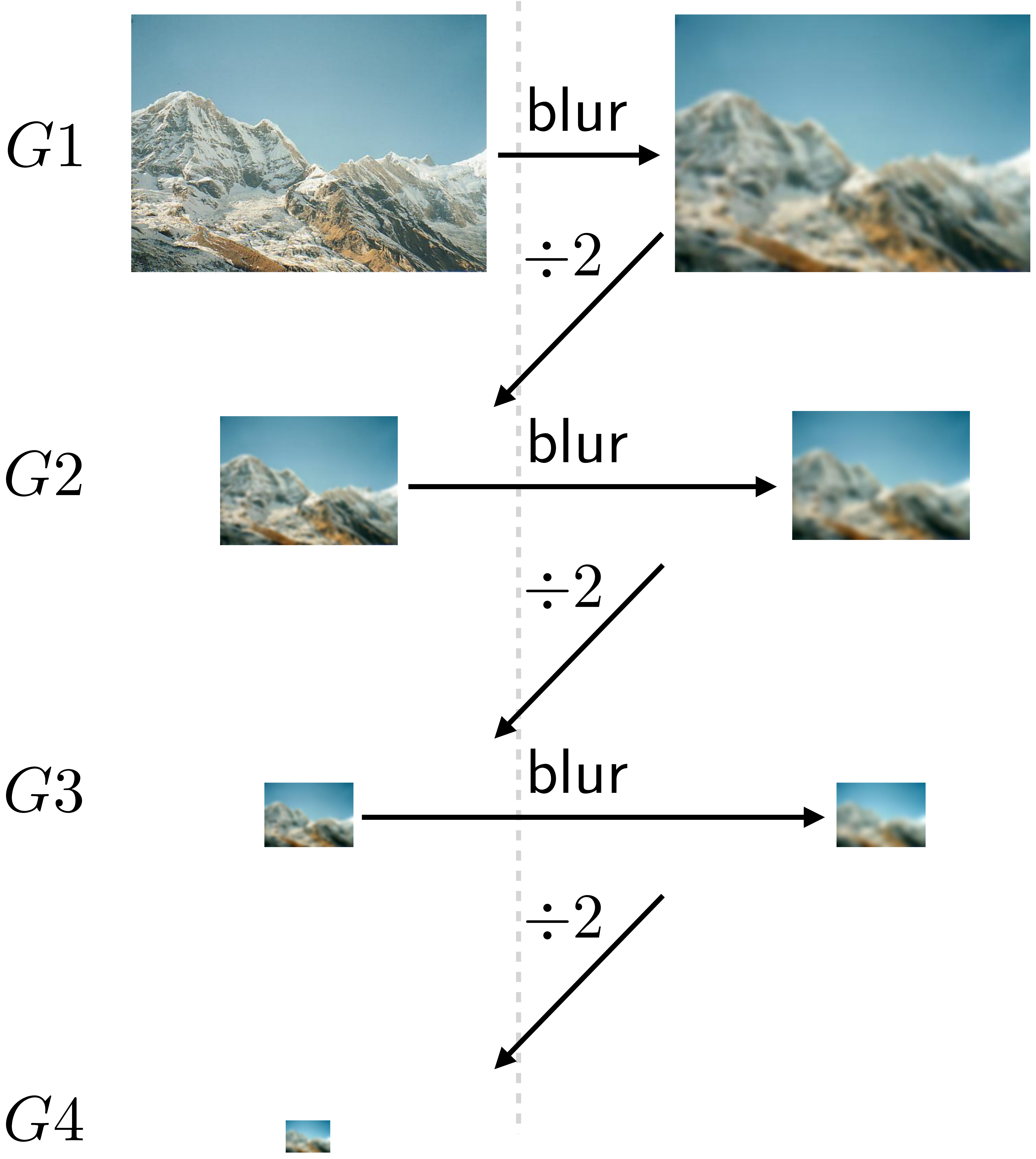
# **Gaussian** vs **Laplacian** Pyramid

Shown in opposite order for space



Which one takes more space to store?

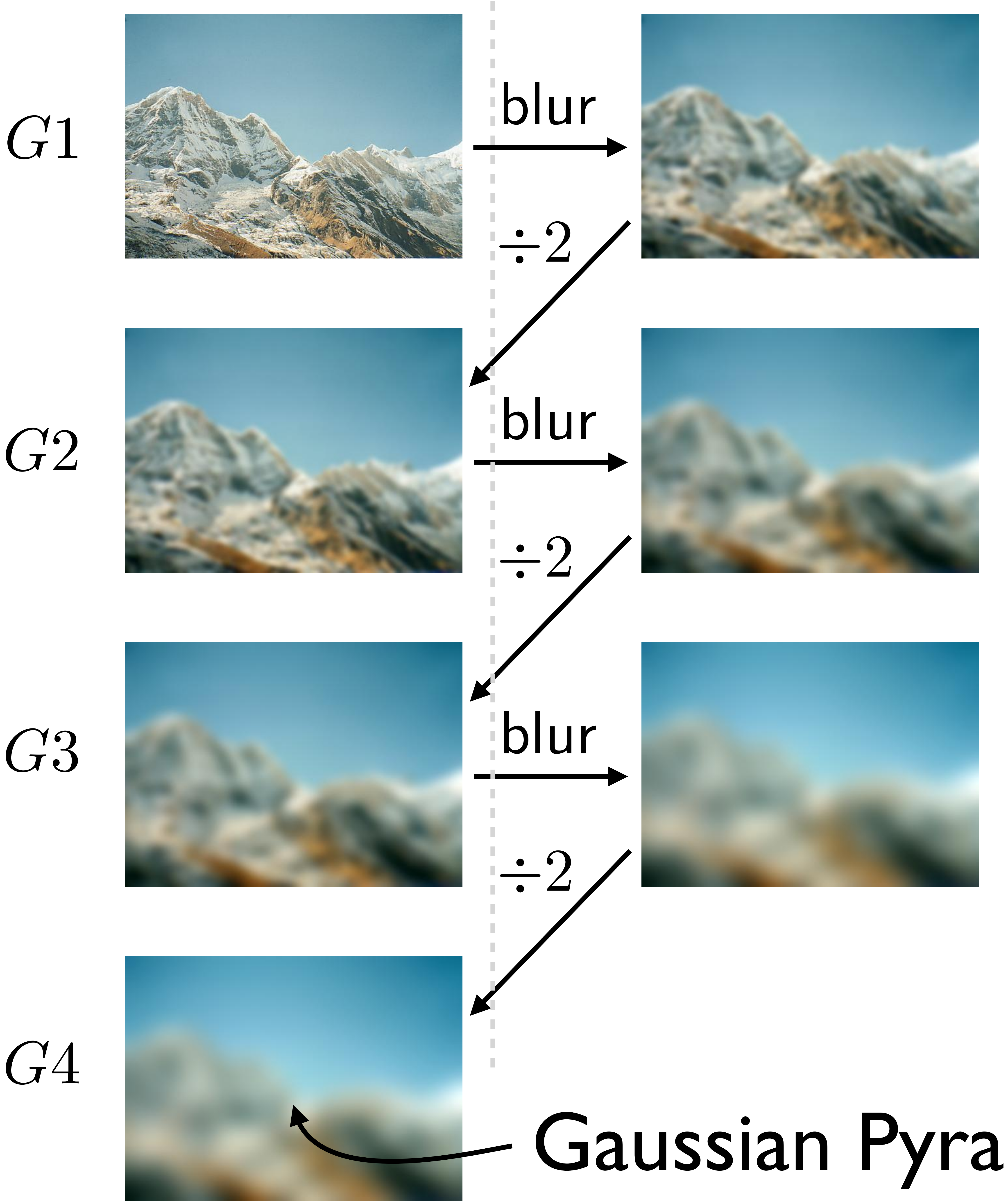
**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)




Blur with a Gaussian kernel, then select every 2nd pixel

$$I_s(x, y) = I(x, y) * g_\sigma(x, y)$$

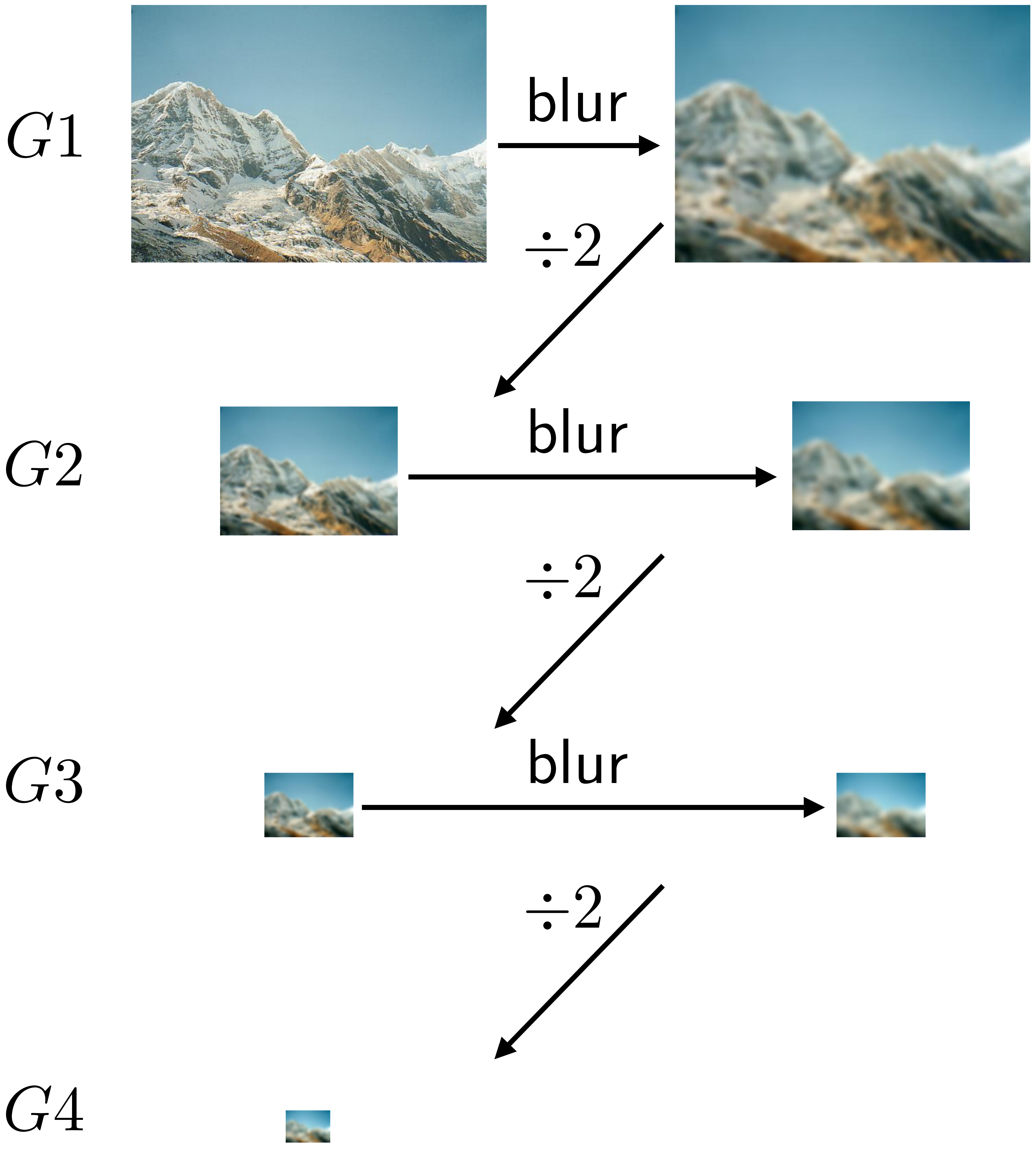
Gaussian Pyramid



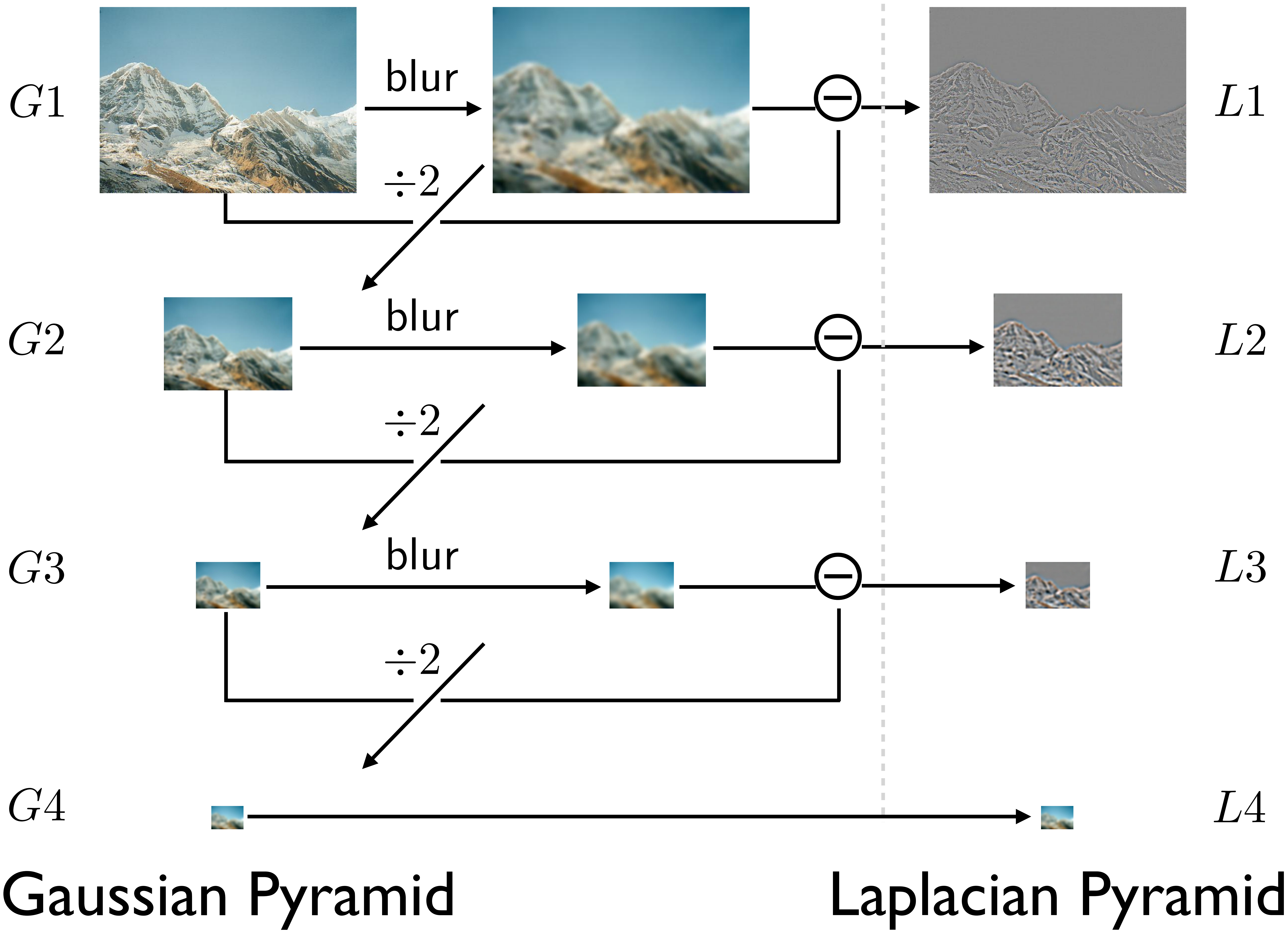

Blur with a Gaussian kernel, then select every 2nd pixel

$$I_s(x,y) = I(x,y) * g_\sigma(x,y)$$

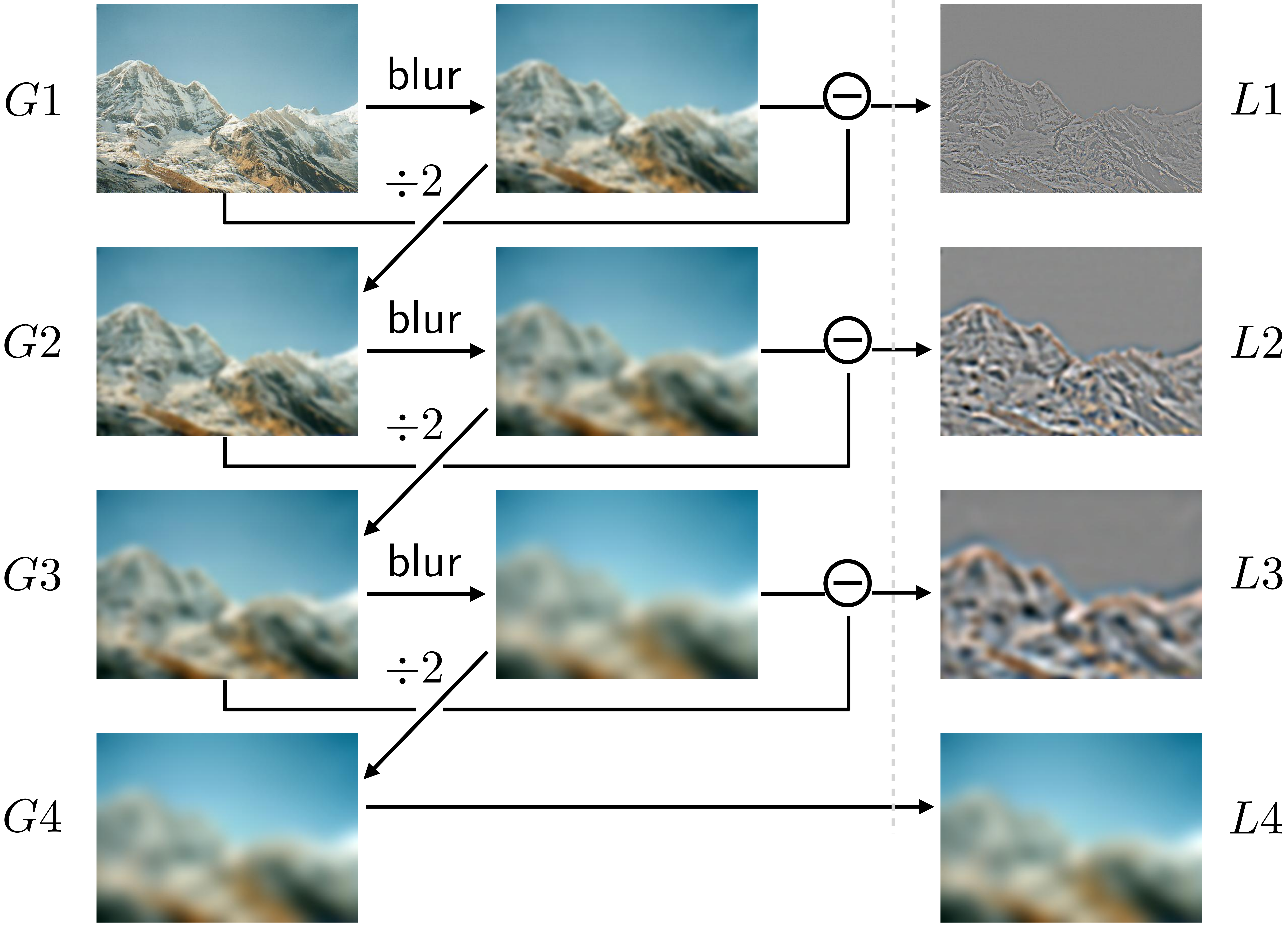
Gaussian Pyramid




# Gaussian Pyramid

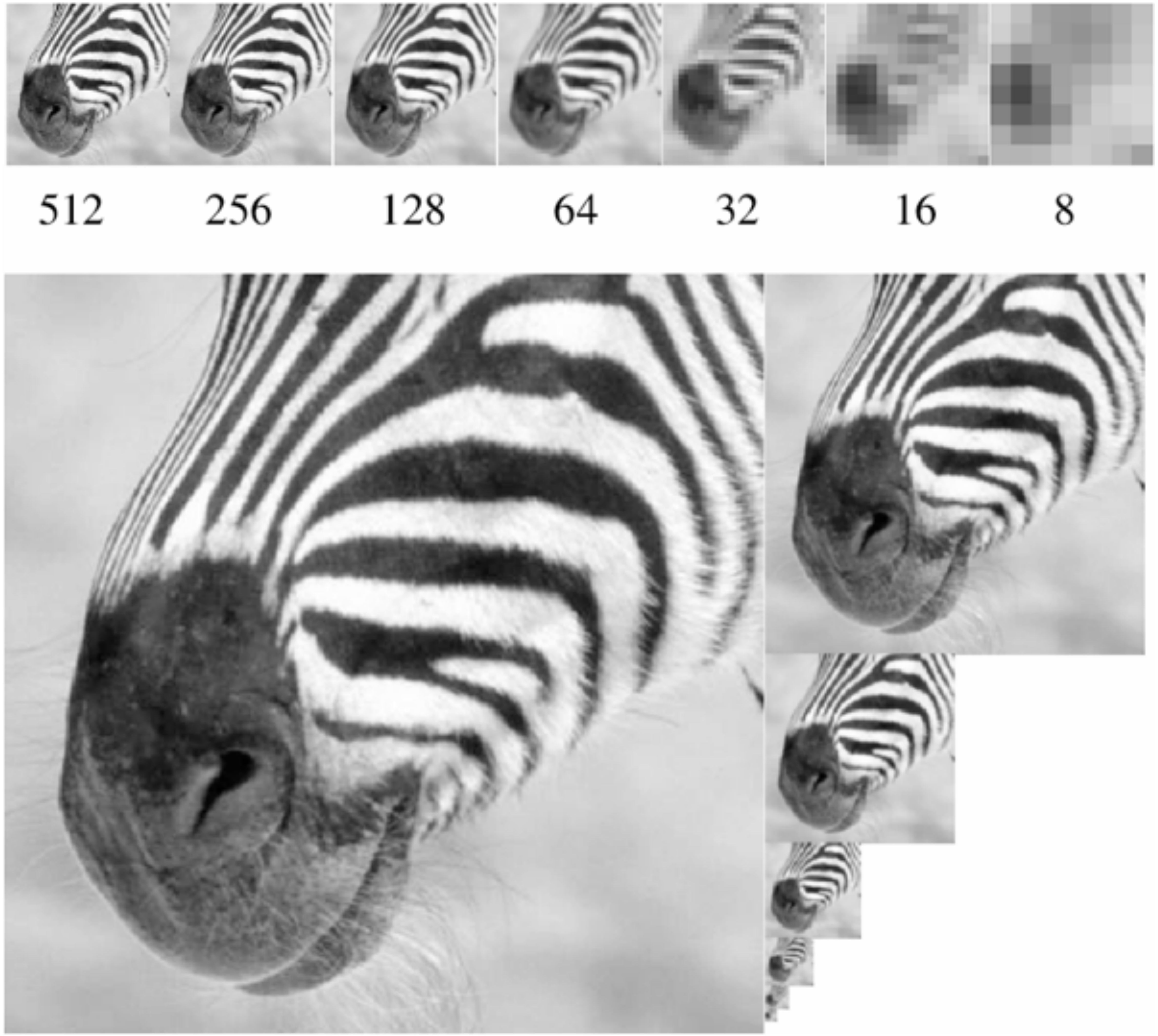


$G1$
blur
$\div 2$
$L1$
$G2$
blur
$\div 2$
$L2$
$G3$
blur
$\div 2$
$L3$
$G4$
$L4$
Gaussian Pyramid
Laplacian Pyramid



$G1$
blur
$\div 2$
$L1$
$G2$
blur
$\div 2$
$L2$
$G3$
blur
$\div 2$
$L3$
$G4$
$L4$

# **Gaussian** Pyramid




Forsyth & Ponce (2nd ed.) Figure 4.17

What happens to the details?

— They get smoothed out as we move to higher levels

What is preserved at the higher levels?

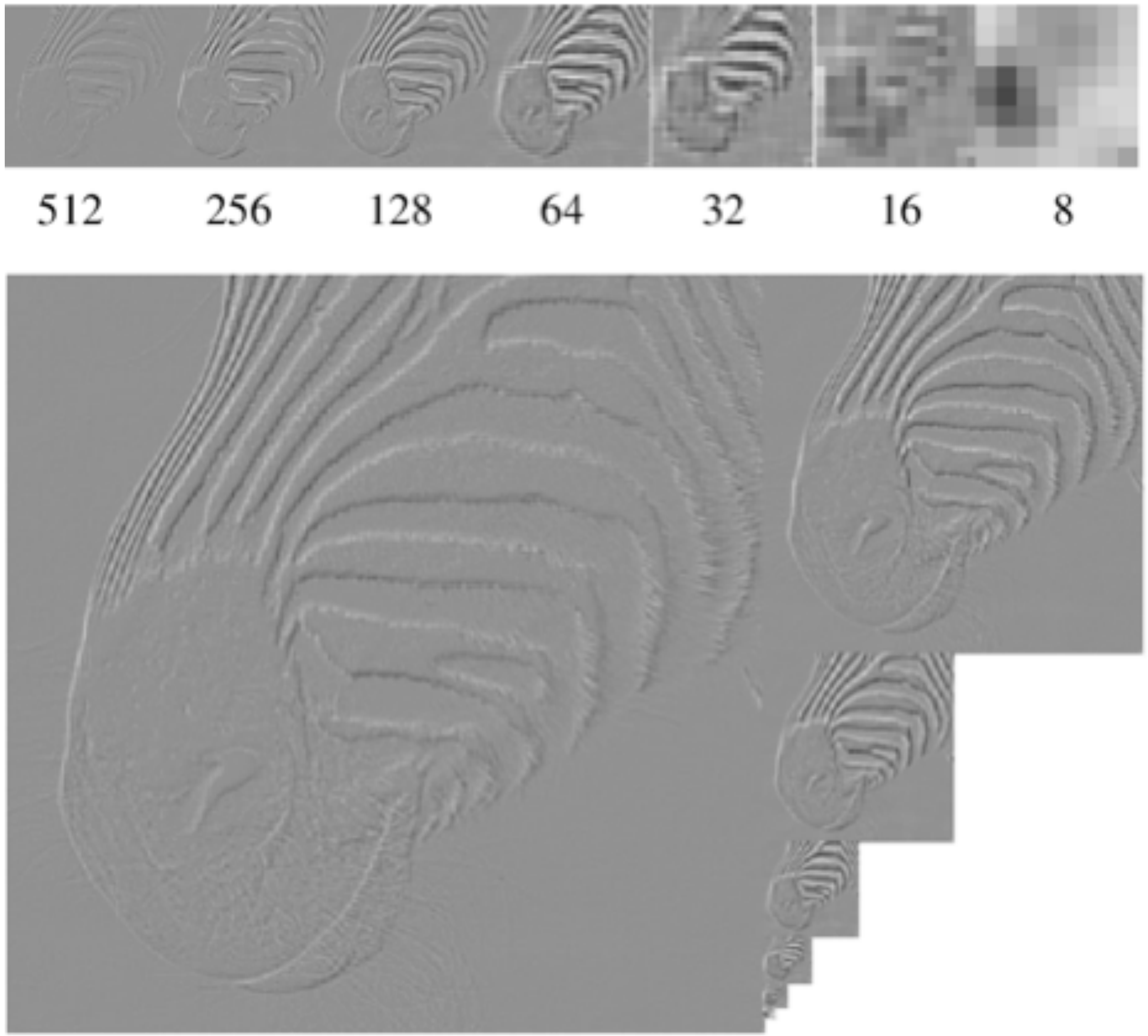
— Mostly large uniform regions in the original image

How would you reconstruct the original image from the image at the upper level?

— That's not possible

**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# **Laplacian** Pyramid




At each level, retain the residuals instead of the blurred images themselves.

Why is it called Laplacian Pyramid?

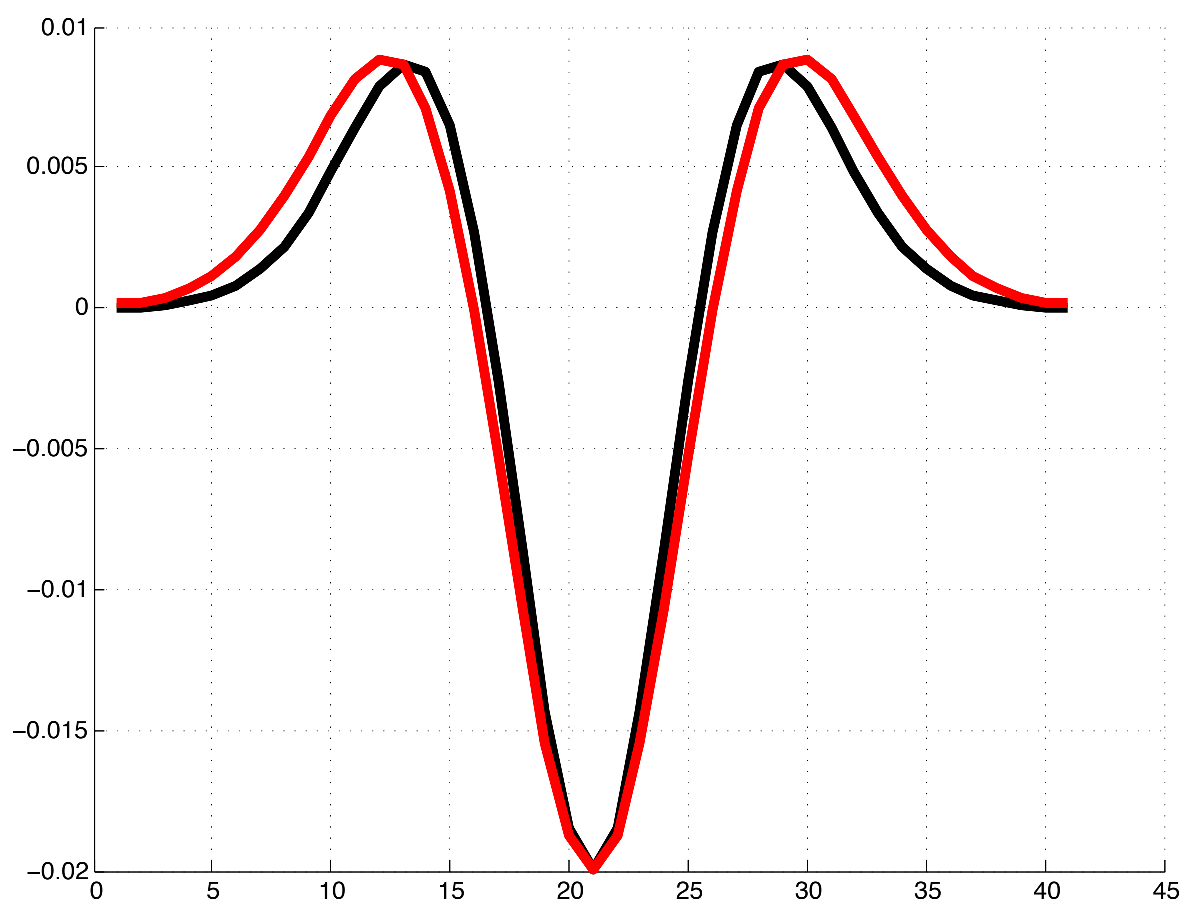
Can we reconstruct the original image using the pyramid?
— Yes we can!

What do we need to store to be able to reconstruct the original image?

**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

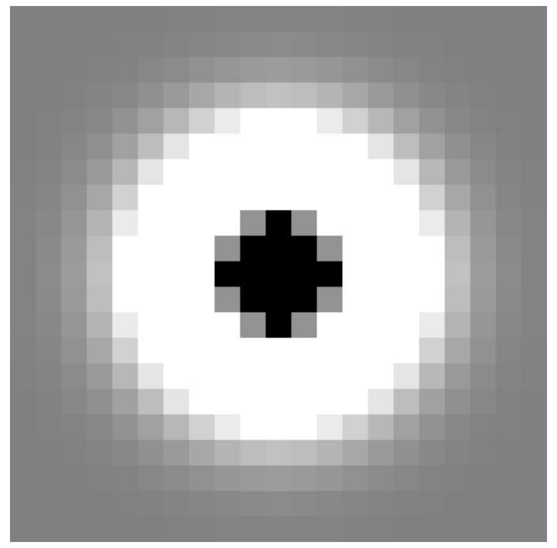
# Why **Laplacian** Pyramid?

$$\text{red} = [1\ \ -2\ 1] * g(x;\ 5.0)$$
$$\text{black} = g(x;\ 5.0) - g(x;\ 4.0)$$




- Laplacian/DoG = centre-surround filter

$*$



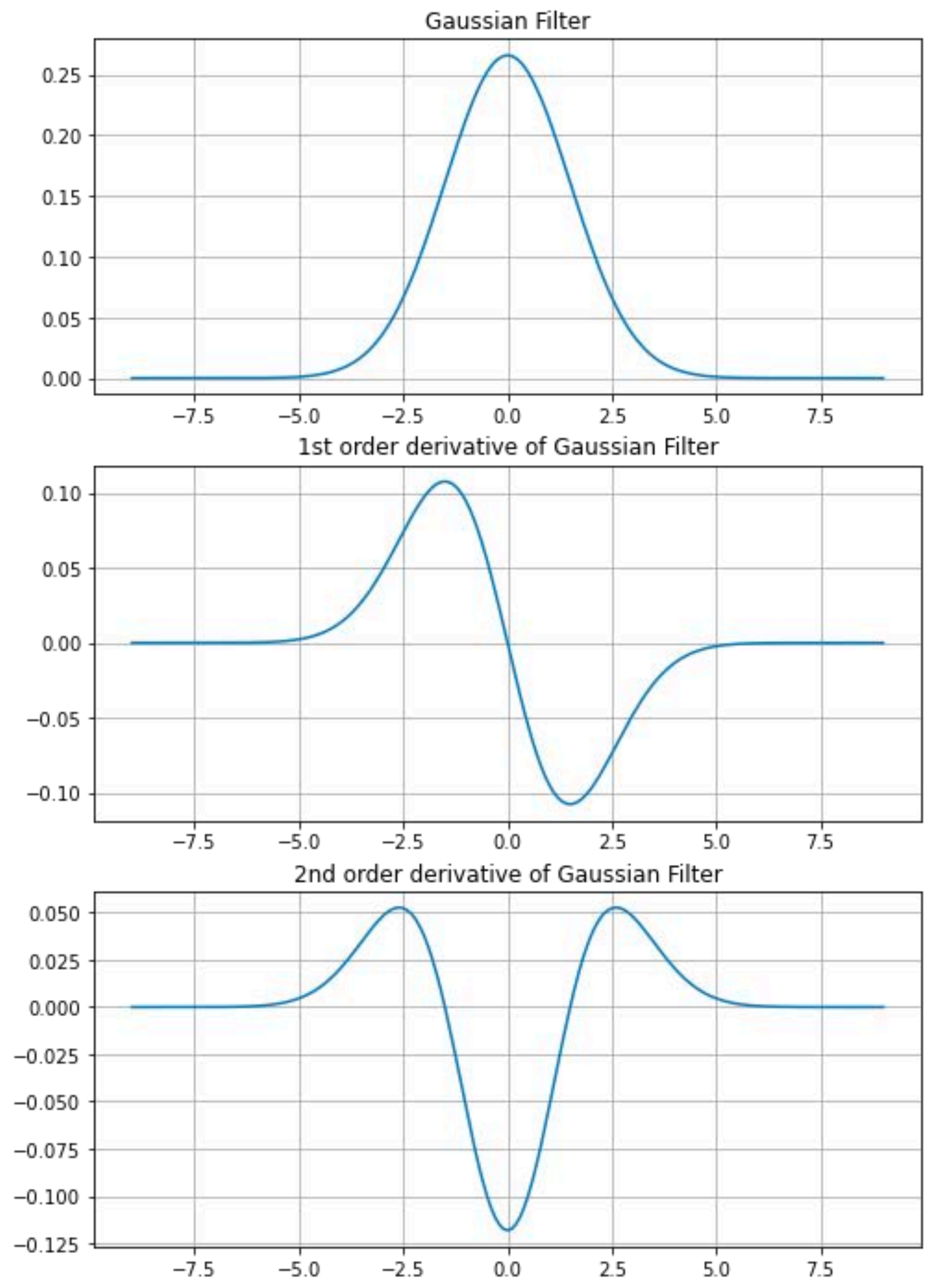
$=$

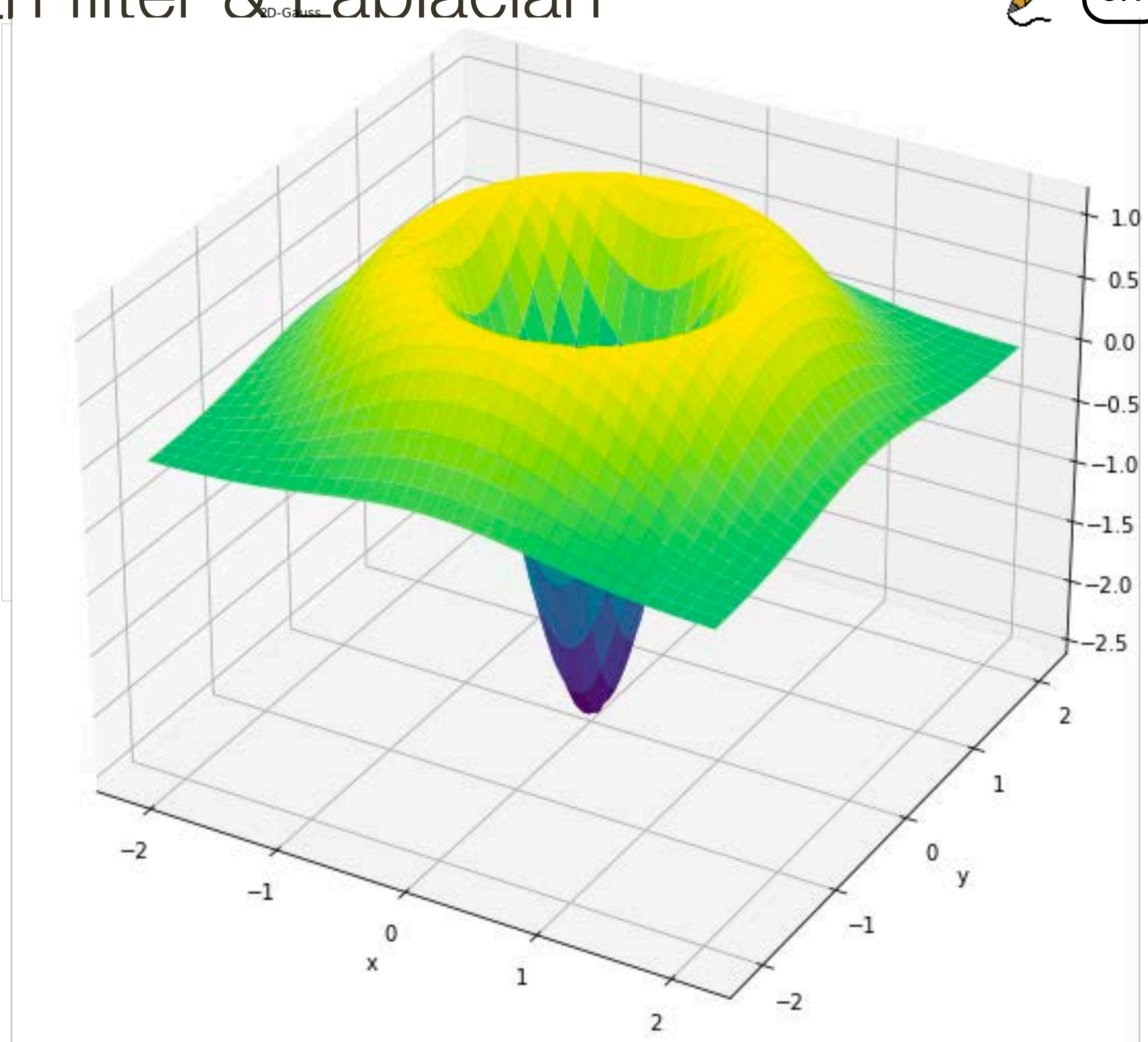
# Why **Laplacian** Pyramid?

8.1

# Derivatives of a Gaussian filter & Laplacian



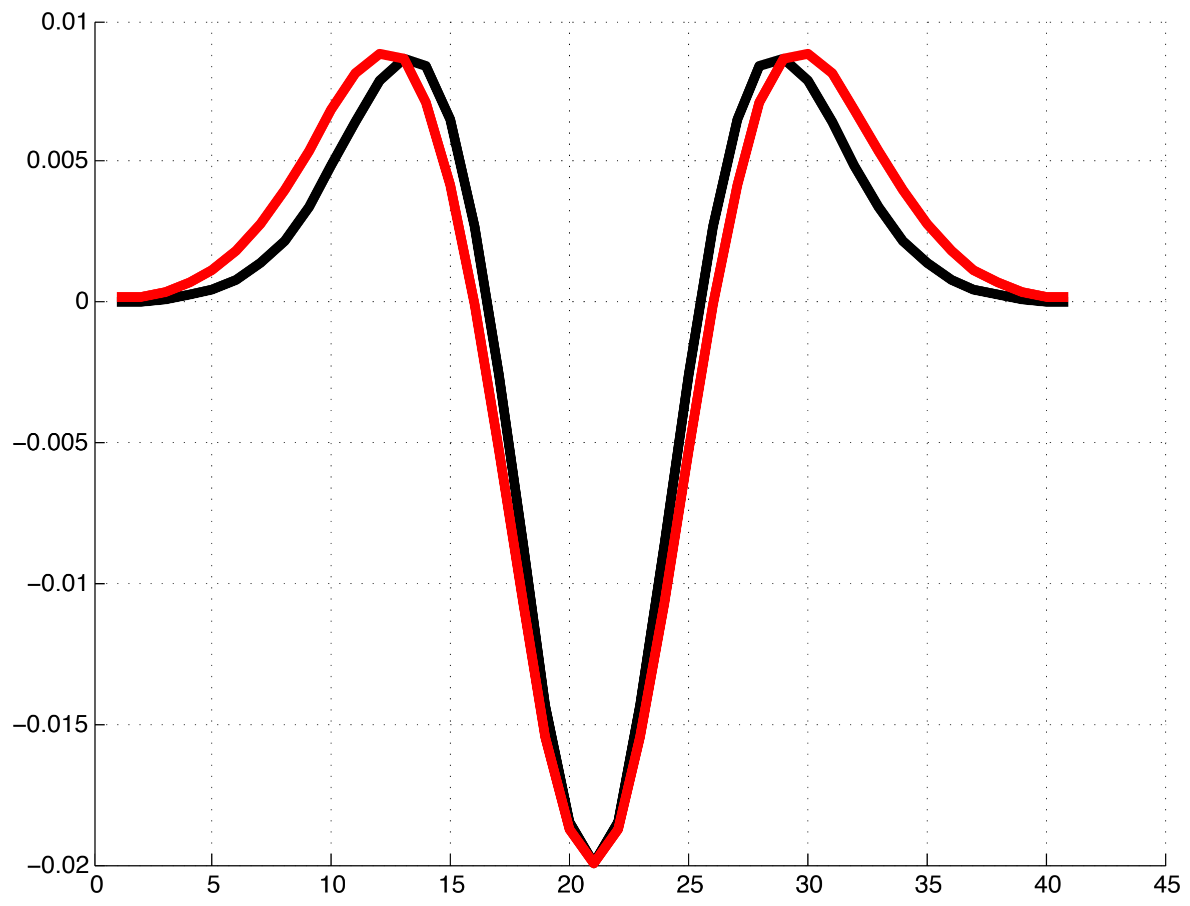


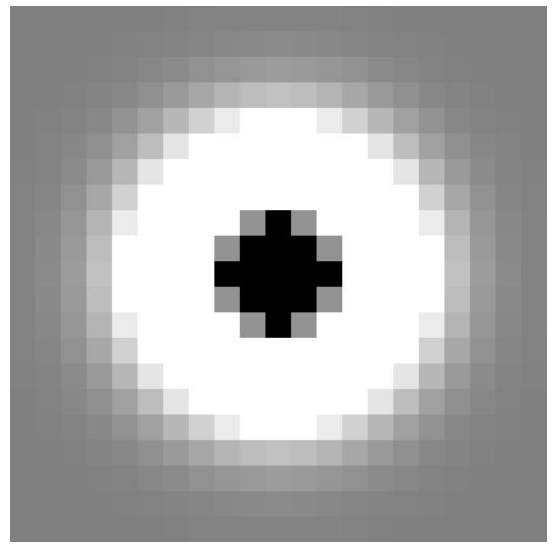

Images from https://hannibunny.github.io/orbook/preprocessing/04gaussianDerivatives.html

# Why **Laplacian** Pyramid?

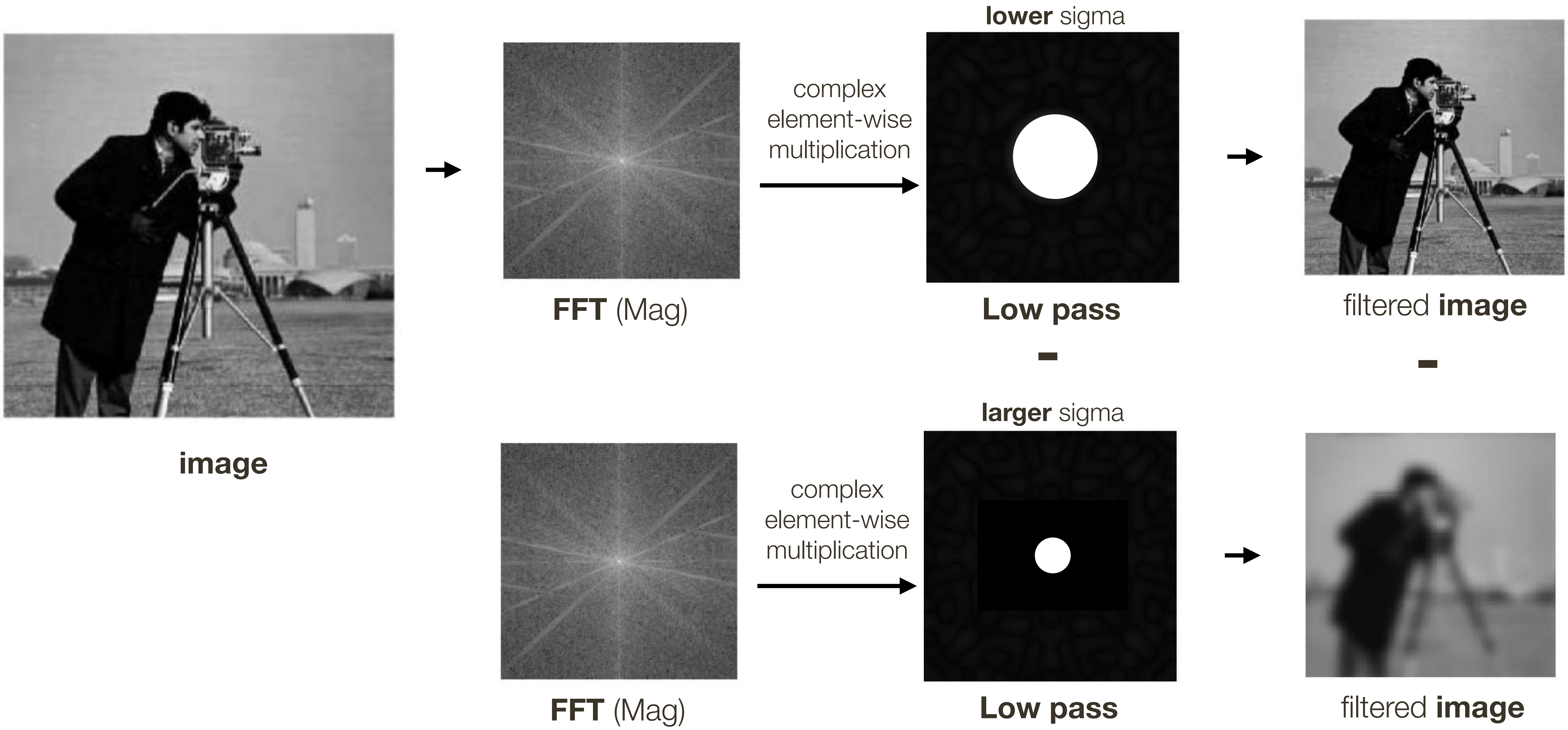
$$\text{red} = [1 \ \ -2 \ 1] * g(x;\ 5.0)$$
$$\text{black} = g(x;\ 5.0) - g(x;\ 4.0)$$




- Laplacian/DoG = centre-surround filter

 $*$  $=$ 

# **Laplacian** is a Bandpass Filter



# **Laplacian** is a Bandpass Filter

# **Laplacian** Pyramid

Building a **Laplacian** pyramid:

— Create a Gaussian pyramid

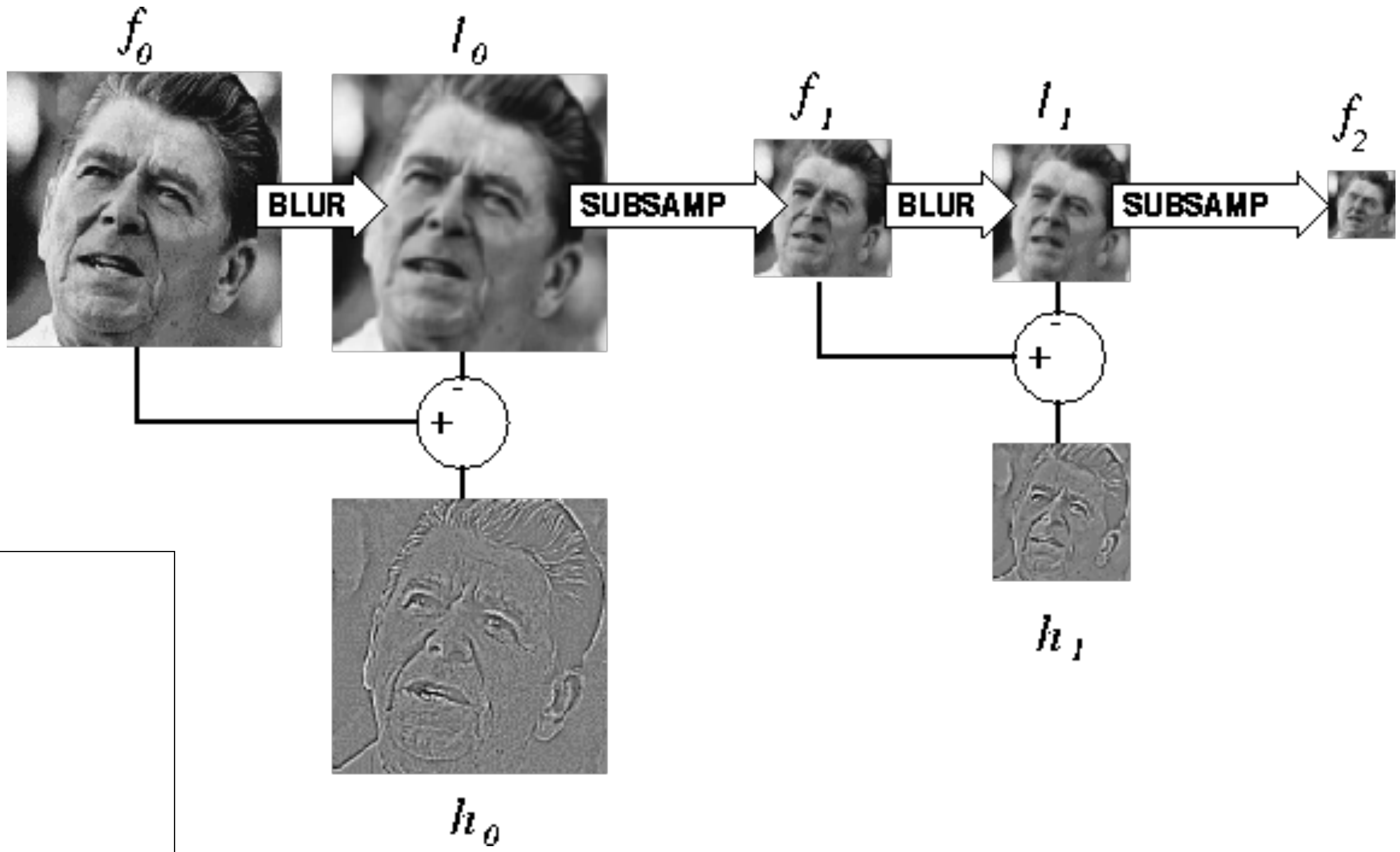
— Take the difference between one Gaussian pyramid level and the next

**Properties**

— Computes a Laplacian / Difference-of-Gaussian (DoG) function of the image at multiple scales

— It is a band pass filter – each level represents a different band of spatial frequencies

# Constructing a **Laplacian** Pyramid

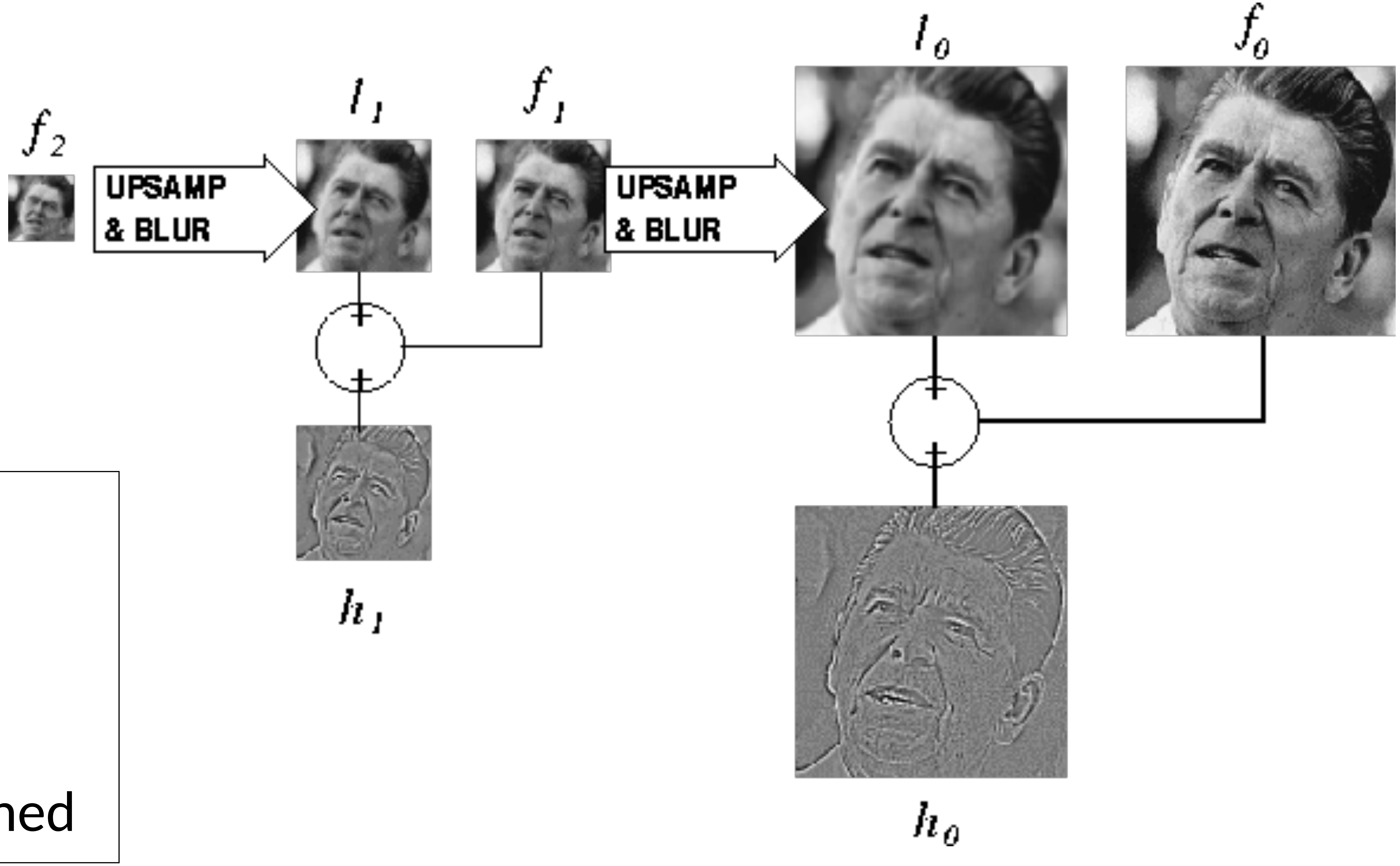



**Algorithm**

repeat:
filter
compute residual
subsample
until min resolution reached

**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

# **Reconstructing** the Original Image




**Algorithm**

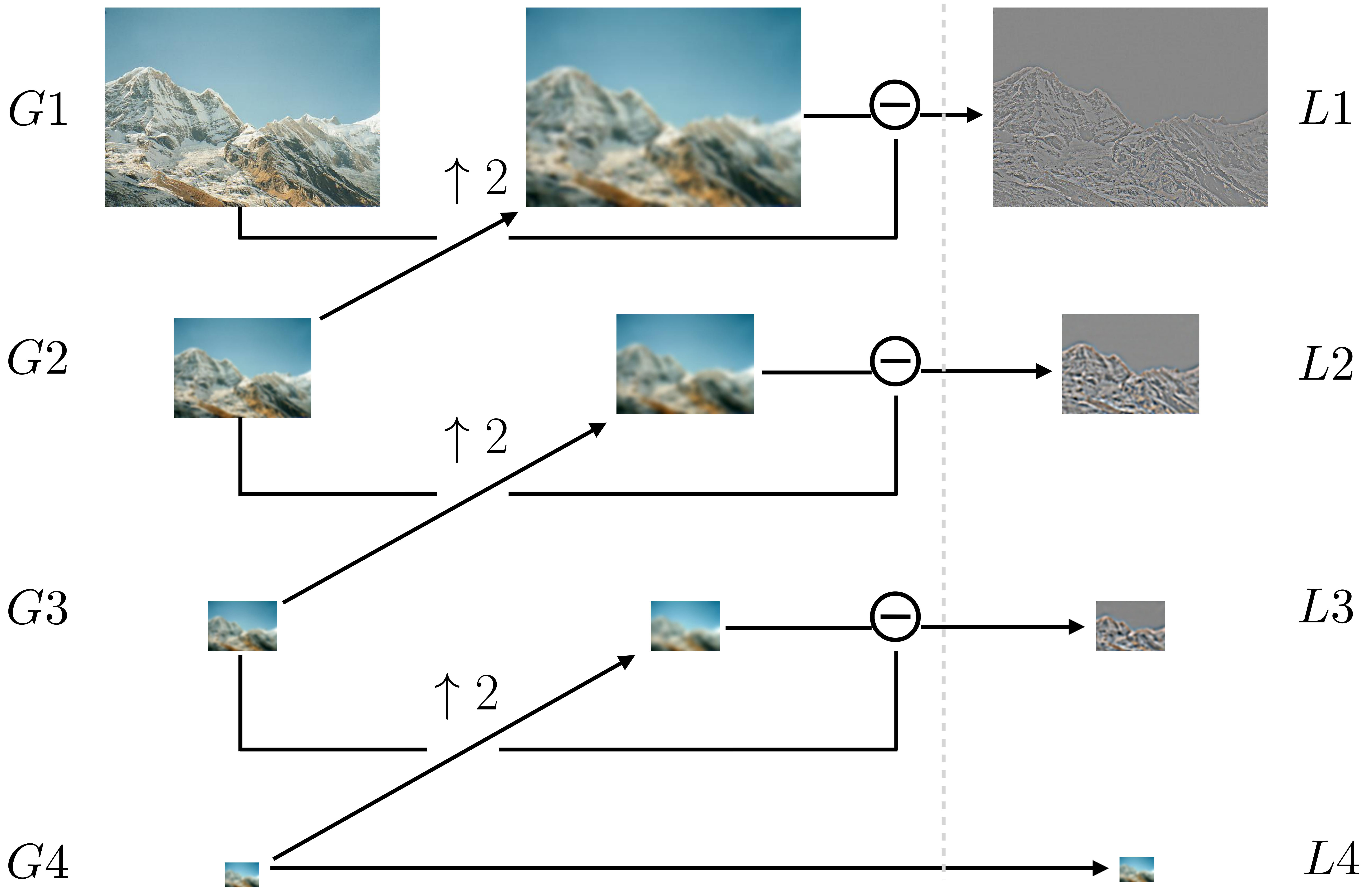
repeat:

upsample

sum with residual

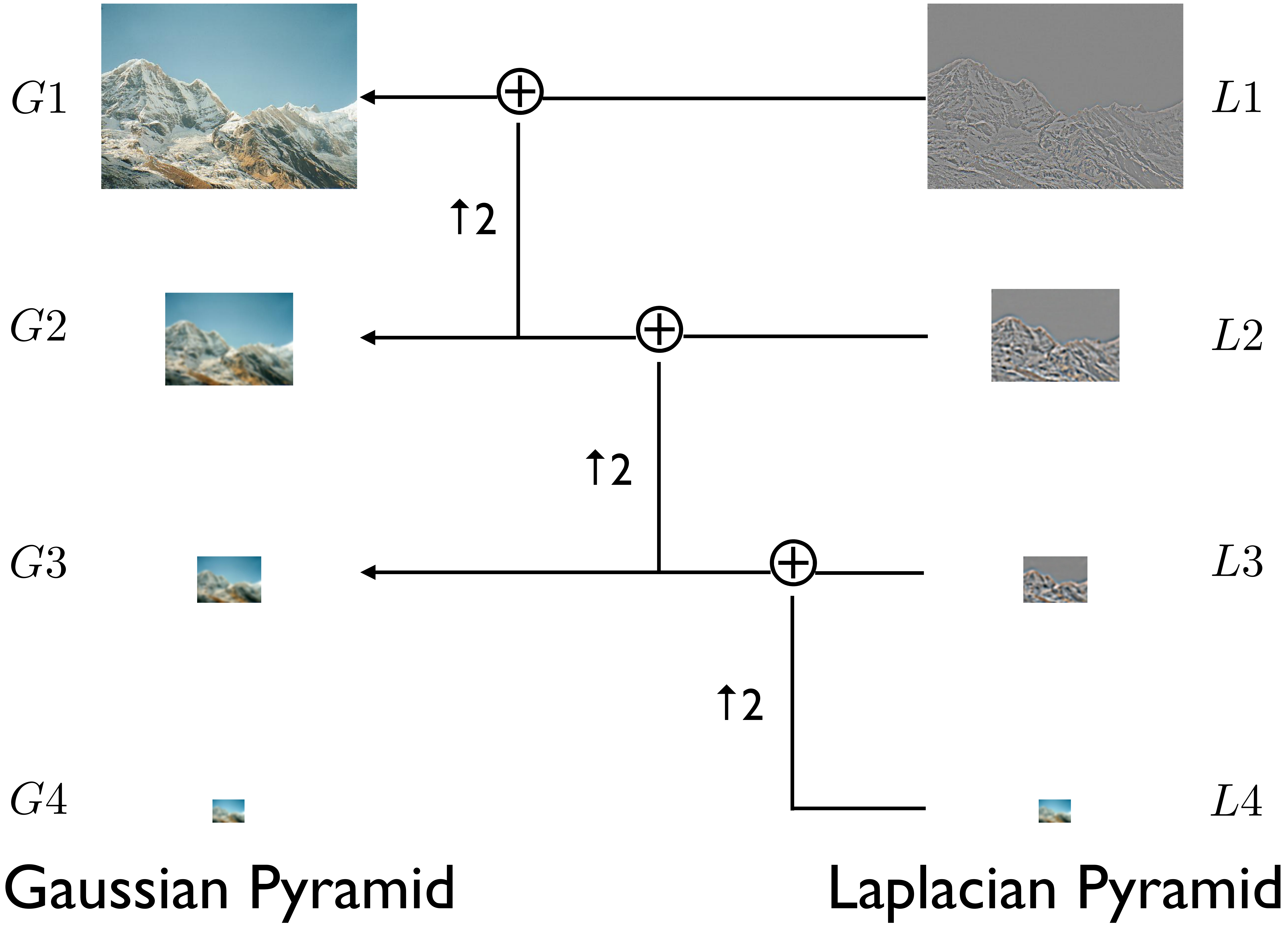
until orig resolution reached

**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

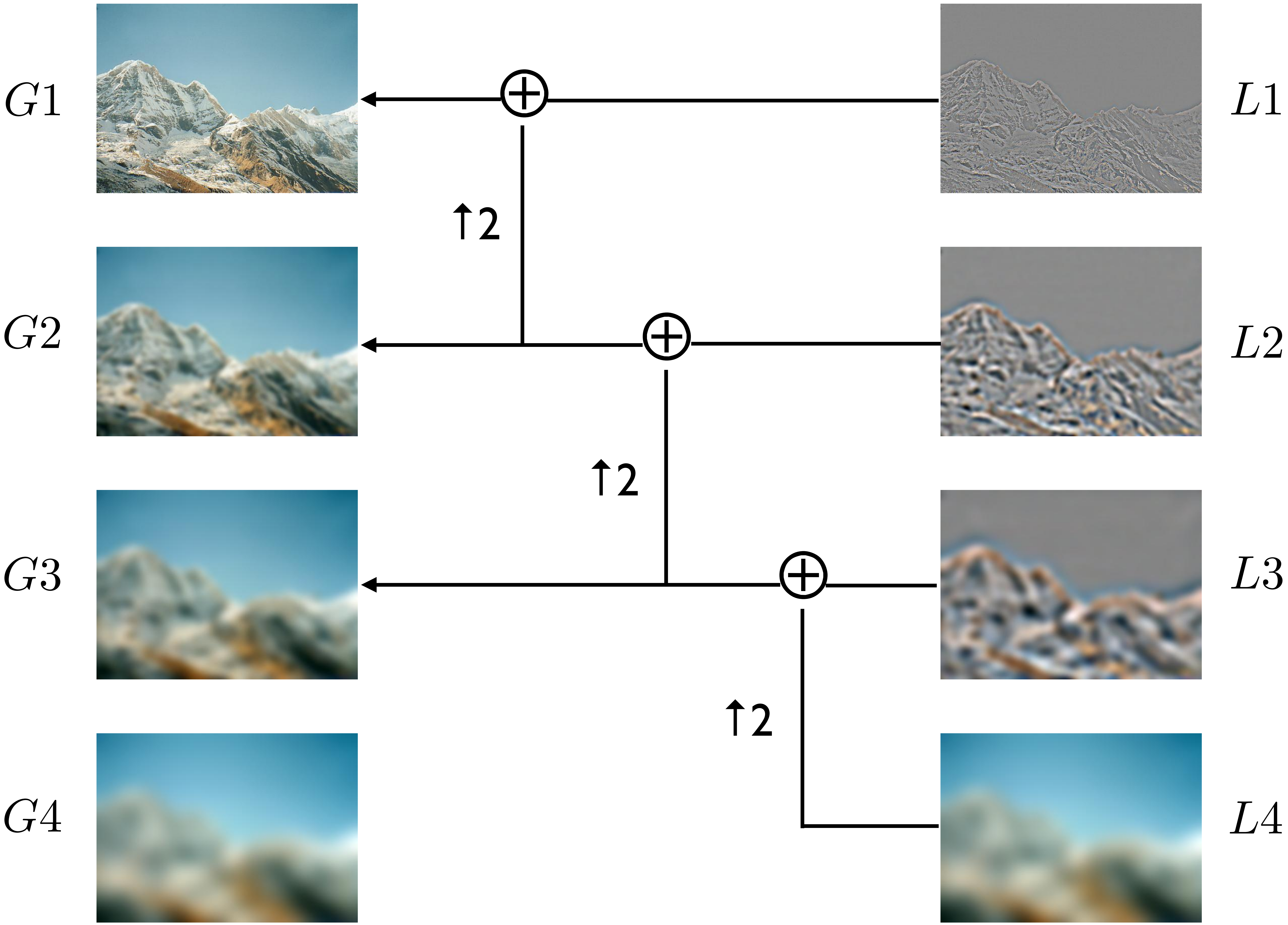



Gaussian Pyramid

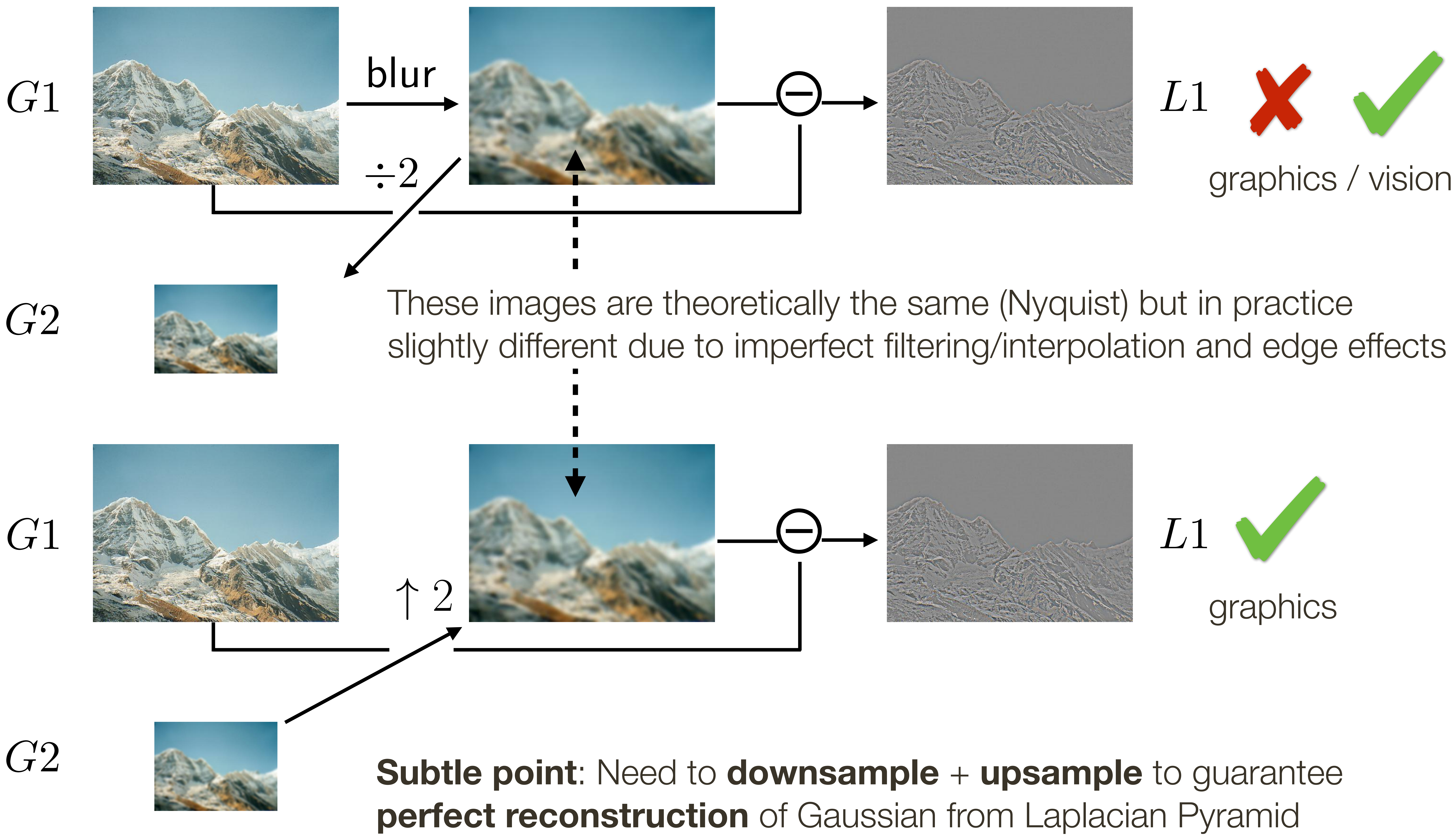
Laplacian Pyramid



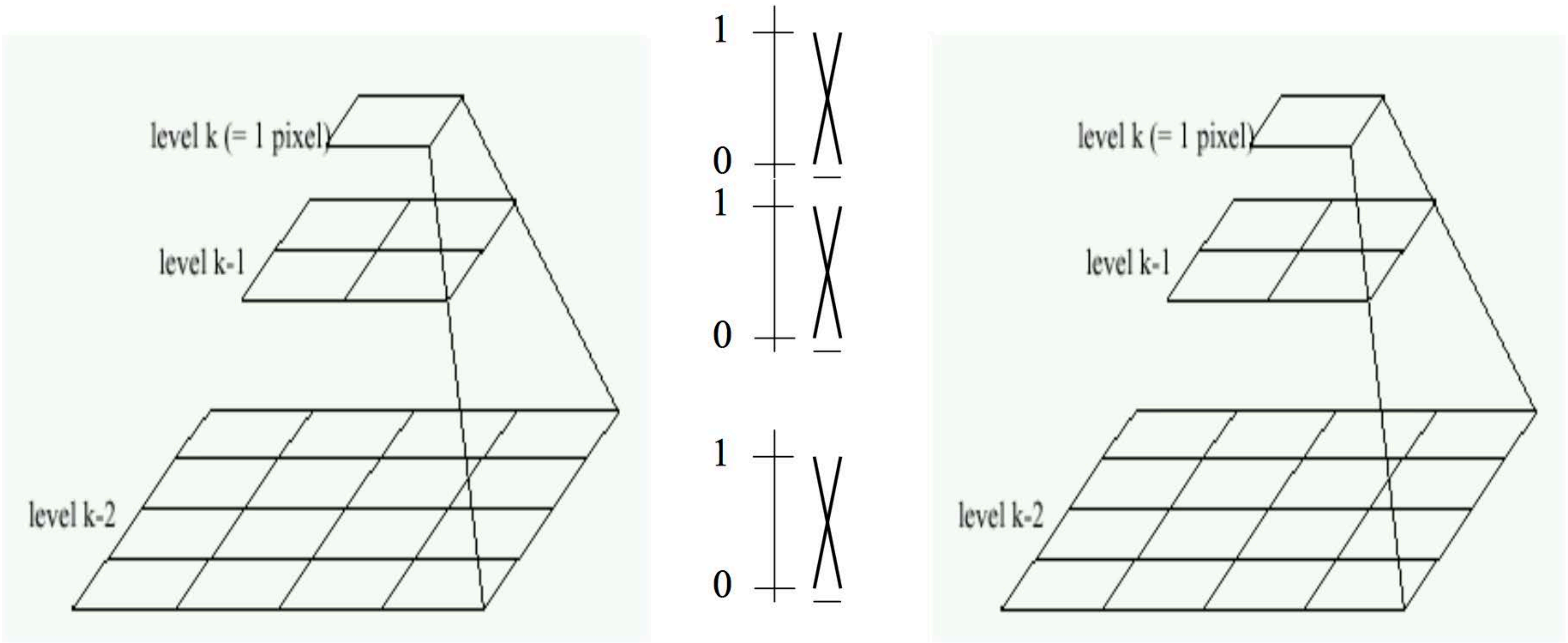
$G1$
$G2$
$G3$
$G4$
↑2
↑2
↑2
$L1$
$L2$
$L3$
$L4$
Gaussian Pyramid
Laplacian Pyramid



G1
G2
G3
G4
L1
L2
L3
L4
↑2
↑2
↑2


G1
blur
L1
graphics / vision
÷2
G2
These images are theoretically the same (Nyquist) but in practice slightly different due to imperfect filtering/interpolation and edge effects
G1
↑ 2
L1
graphics
G2
**Subtle point**: Need to **downsample** + **upsample** to guarantee **perfect reconstruction** of Gaussian from Laplacian Pyramid

# **Application**: Image Blending

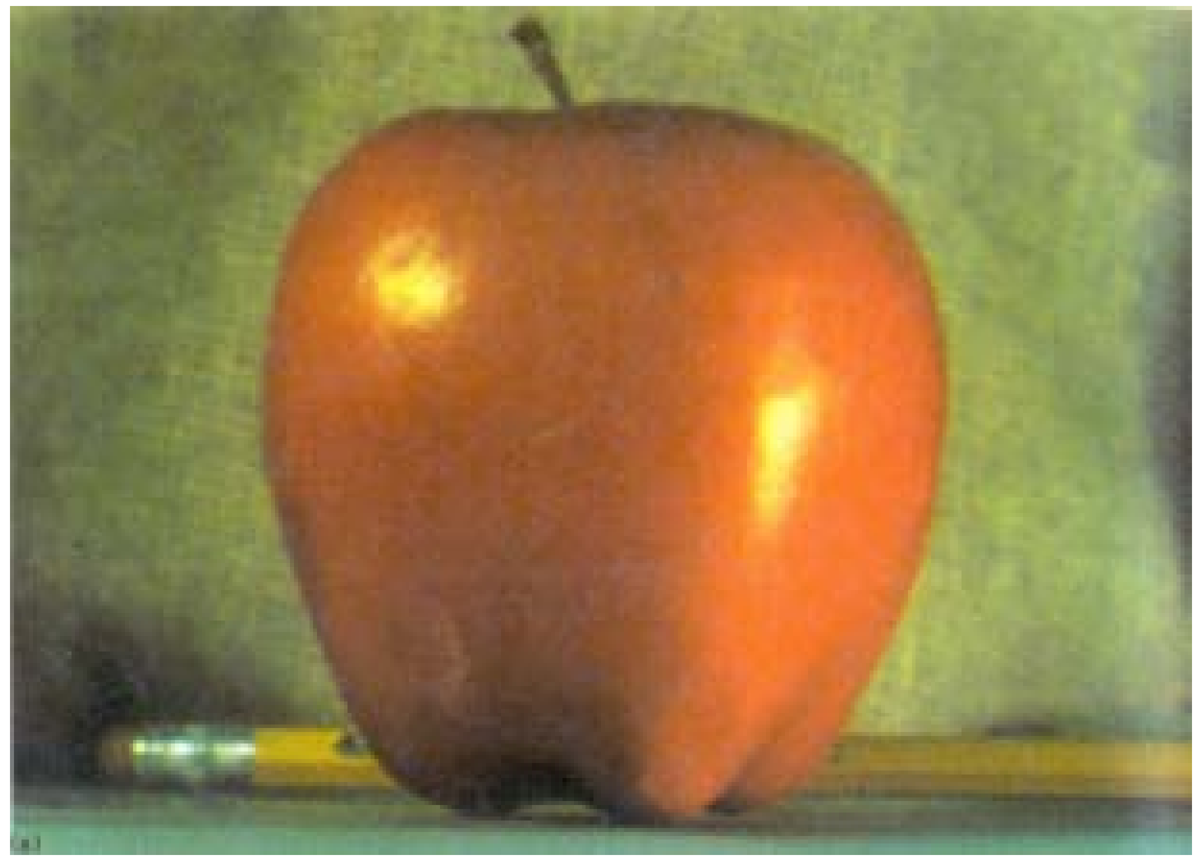



Left pyramid　　blend　　Right pyramid

**Burt and Adelson**, “A multiresolution spline with application to image mosaics,”ACM Transactions on Graphics, 1983, Vol.2, pp.217-236.

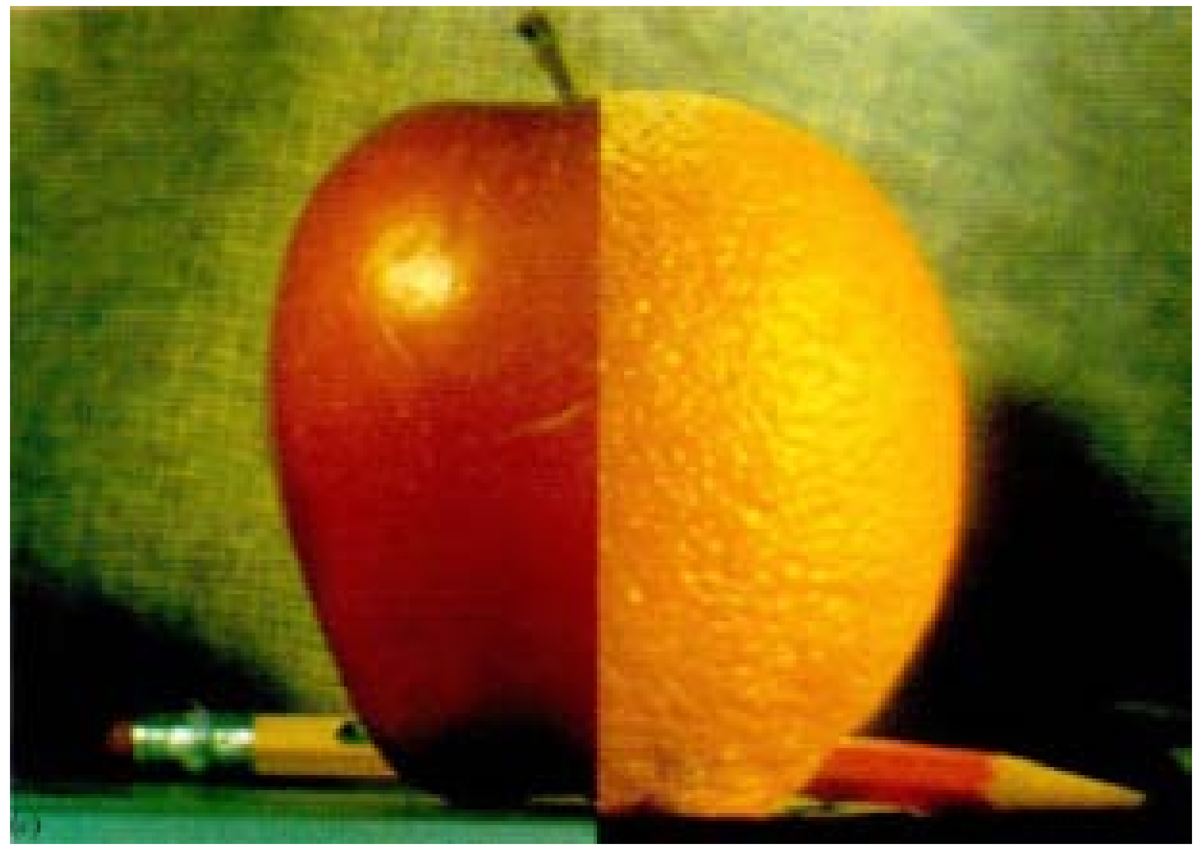
# Pyramid Blending

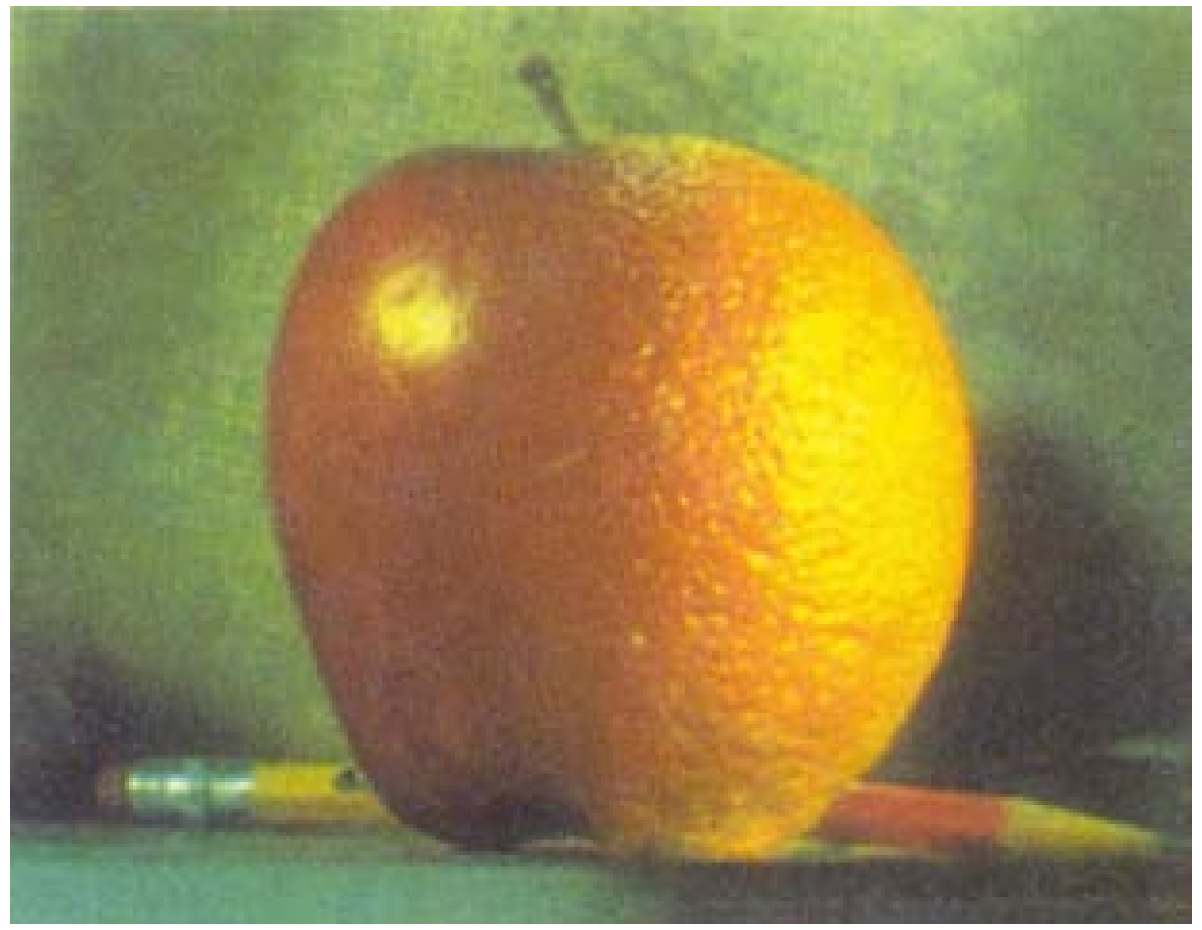
Smooth low frequencies, whilst preserving high frequency detail



(a)

(b)

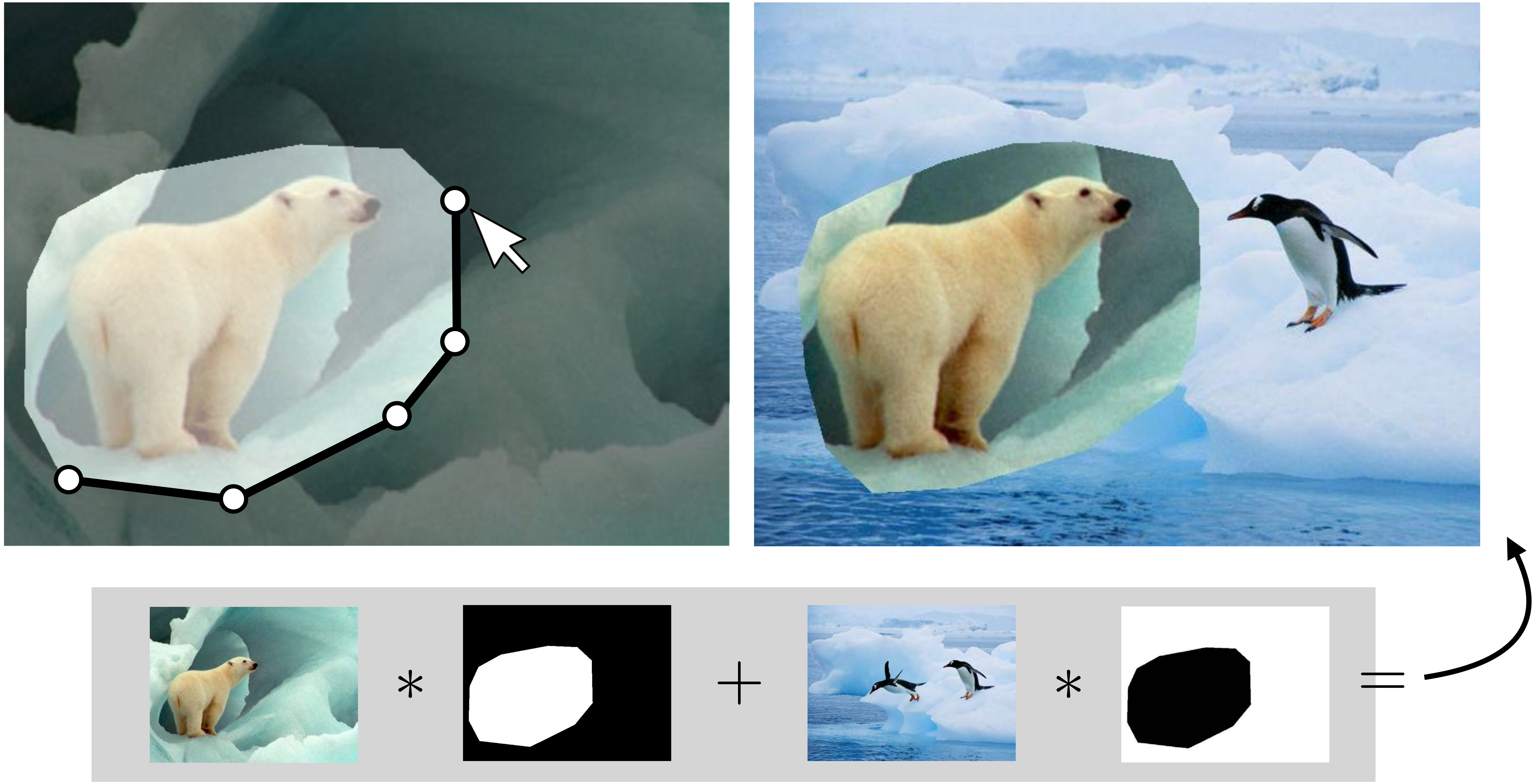




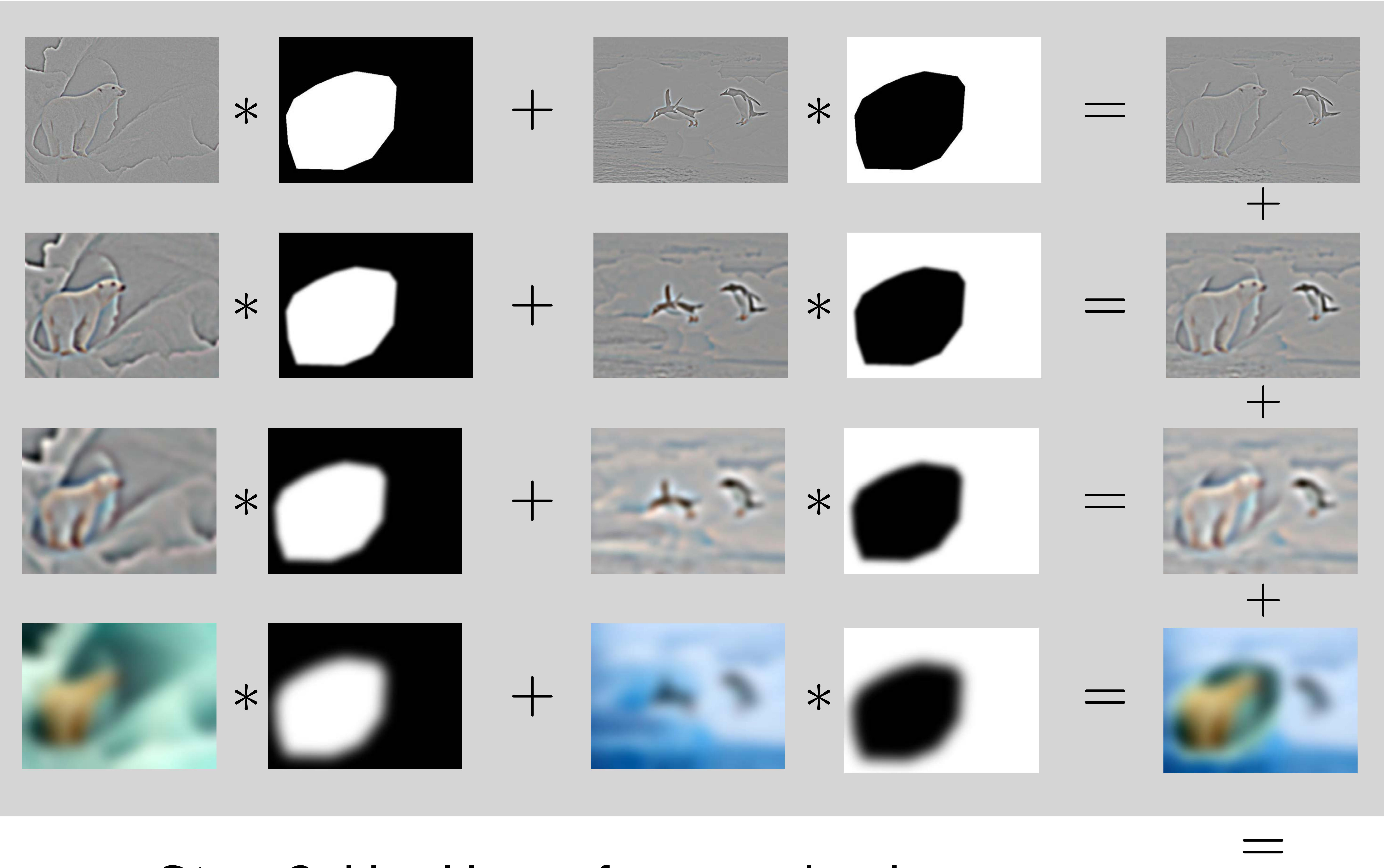
[ Burt Adelson 1983 ]

# Pyramid Blending

# Pyramid Blending




**Step 1**: Specify an Image Mask



**Step 2:** blend lower frequency bands over larger spatial ranges, high frequency bands over small spatial ranges

# **Application**: Image Blending

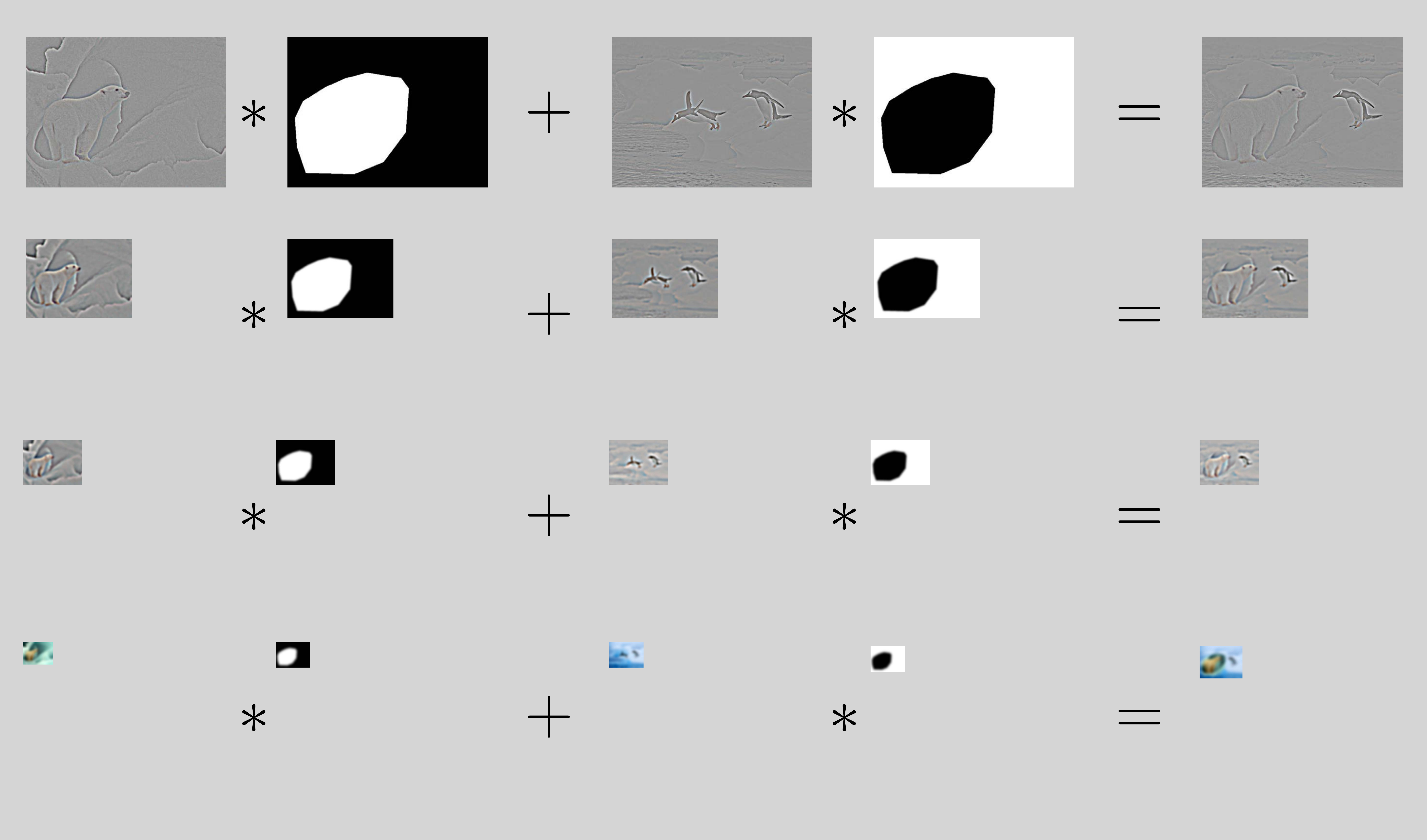
**Algorithm**:

1. Build Laplacian pyramid LA and LB from images A and B

2. Build a Gaussian pyramid GR from mask image R (the mask defines which image pixels should be coming from A or B)

3. From a combined (blended) Laplacian pyramid LS, using nodes of GR as weights: $LS(i,j) = GR(i,j) * LA(i,j) + (1-GR(i,j)) * LB(i,j)$

4. Reconstruct the final blended image from LS



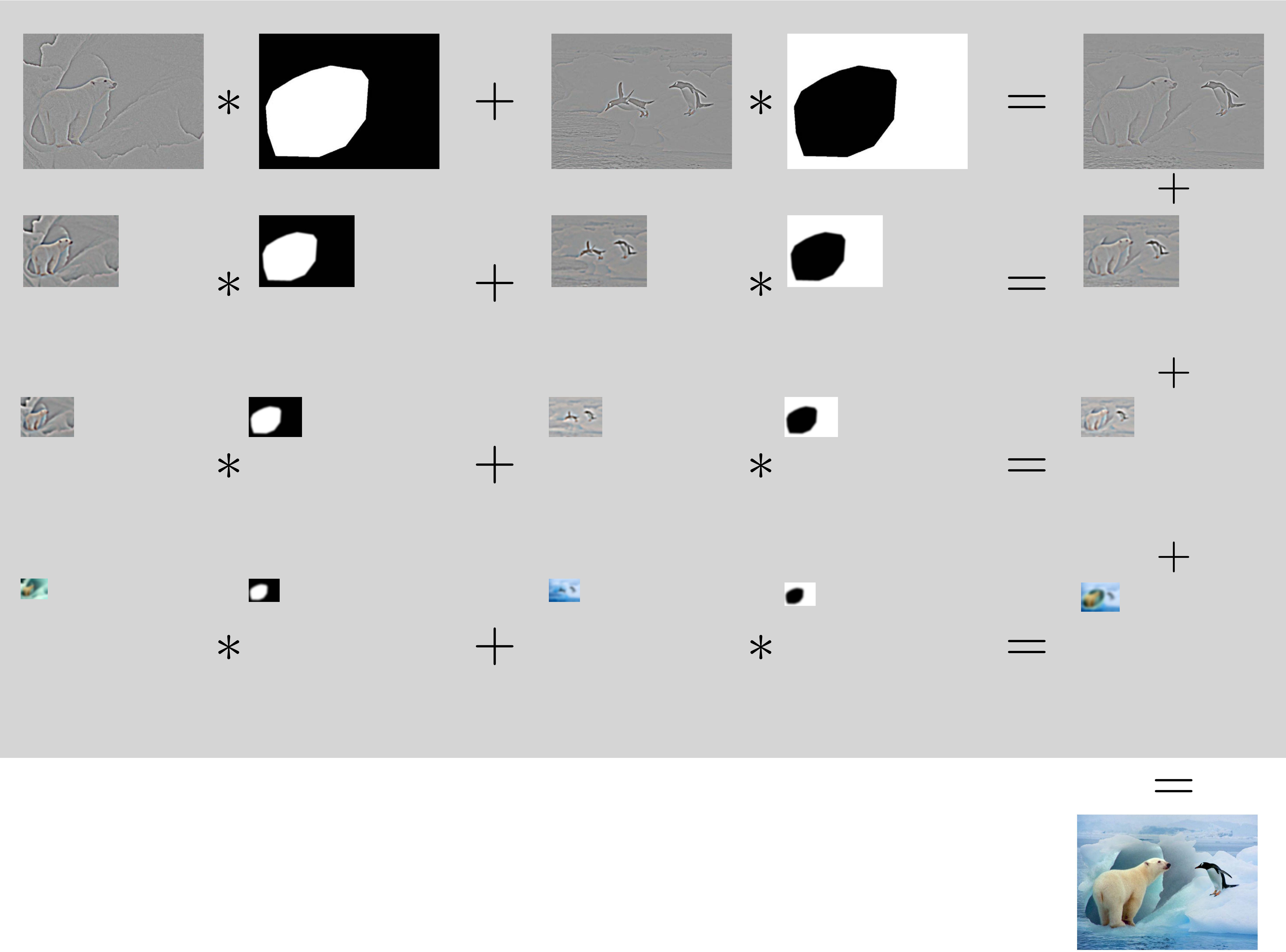
Polar Bear Laplacian Pyramid

Mask Gaussian Pyramid

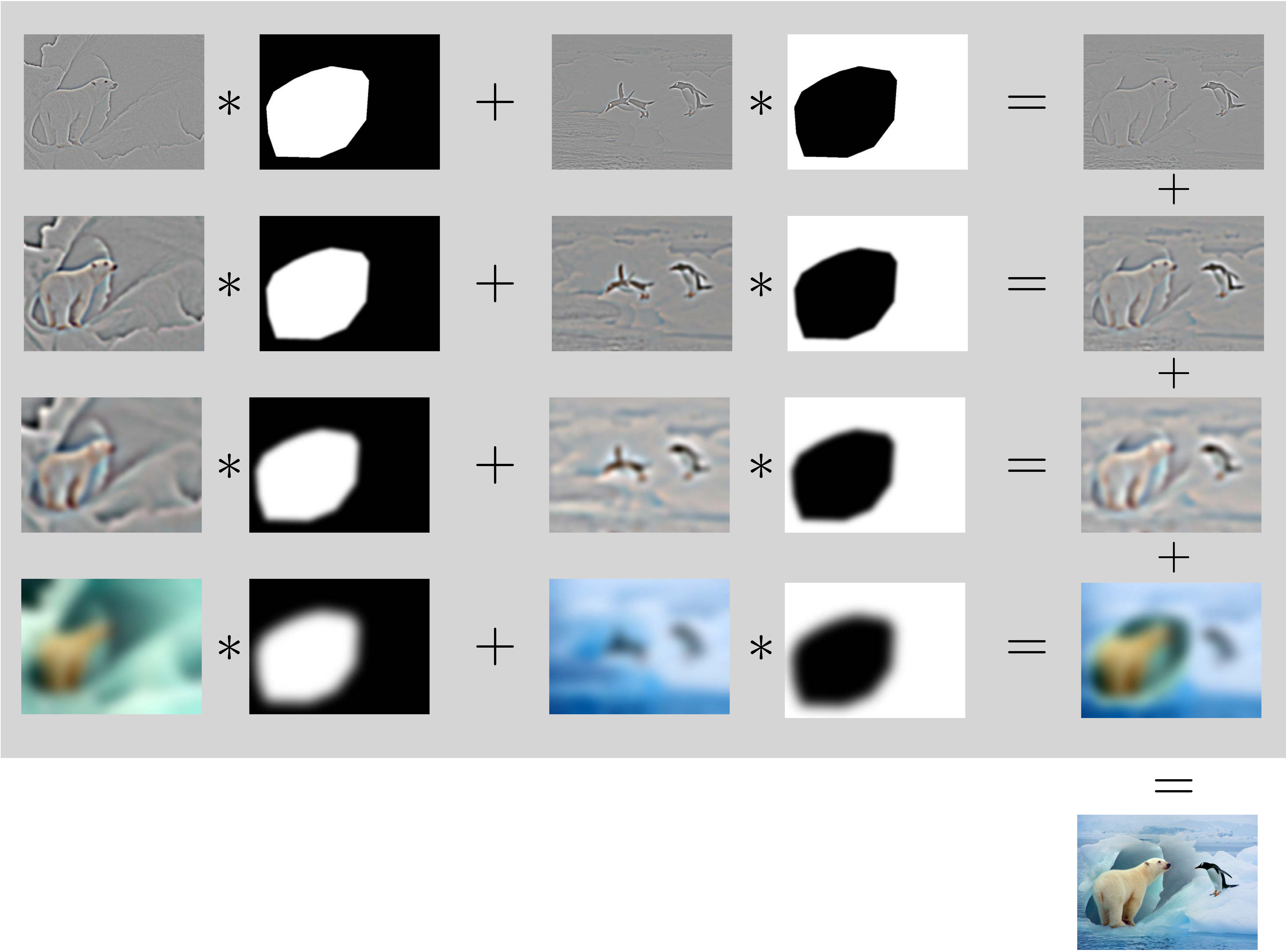
Penguin Laplacian Pyramid

1 - Mask Gaussian Pyramid

Result Pyramid



Reconstruct
Result

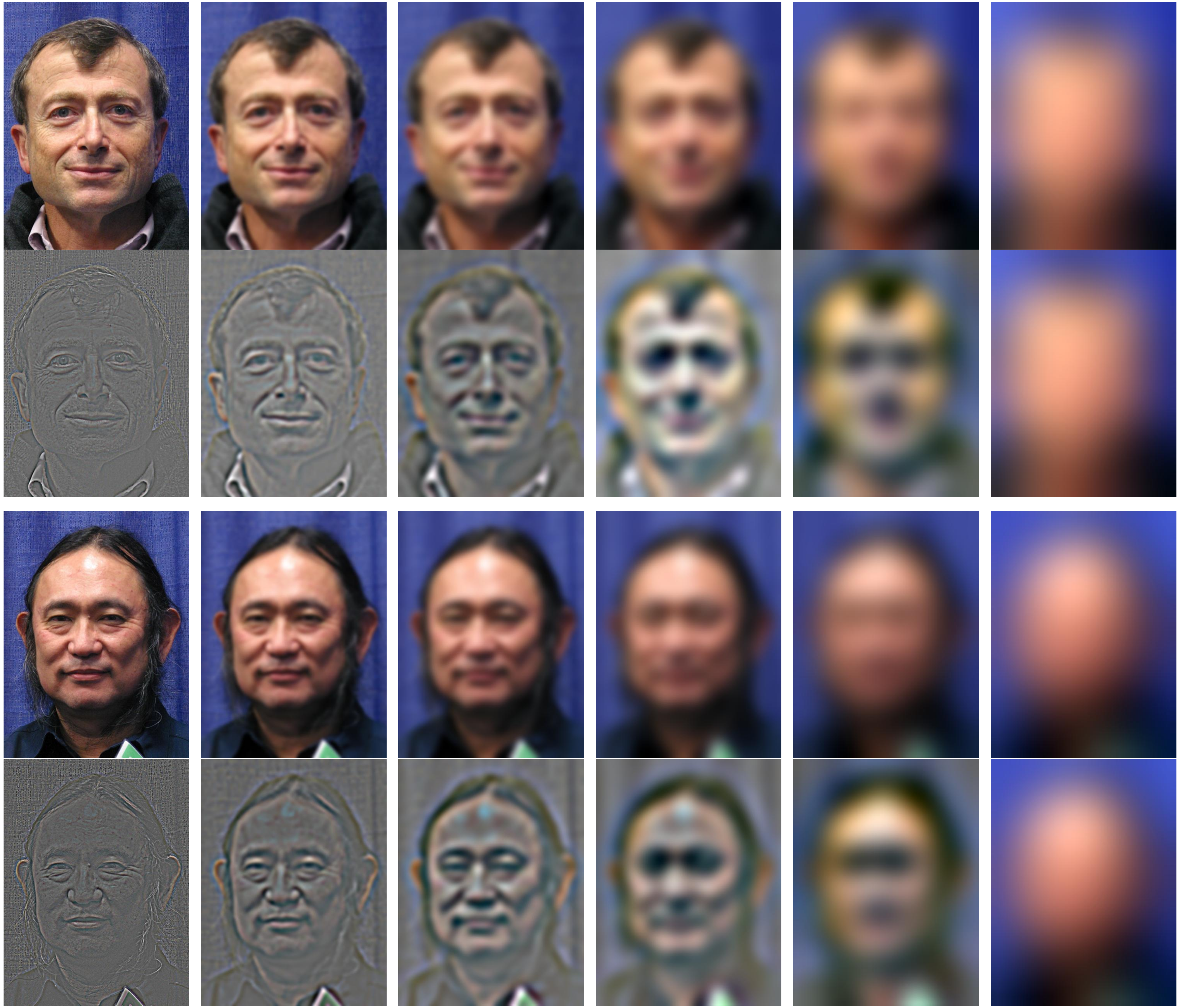


Reconstruct Result

[ Jim Kajiya, Andries van Dam]

[ Jim Kajiya, Andries van Dam]



Alpha blend with sharp fall-off


Alpha blend with gradual fall-off

Pyramid Blend

# Summary: **Scaled Representations**

**Gaussian Pyramid**

— Each level represents a **low-pass** filtered image at a different scale

— Generated by successive Gaussian blurring and downsampling

— Useful for image resizing, sampling

**Laplacian Pyramid**

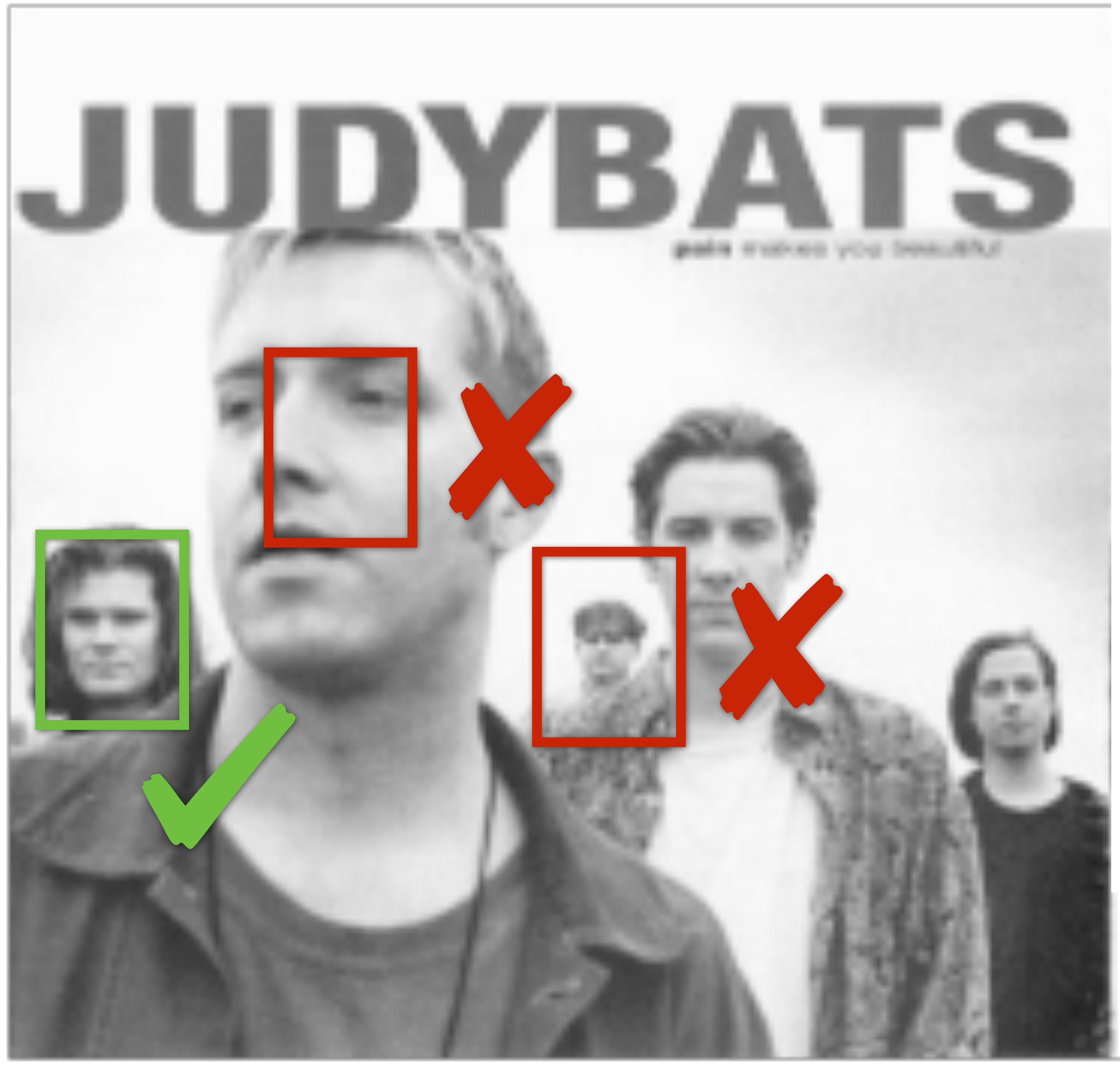
— Each level is a **band-pass** image at a different scale

— Generated by differences between successive levels of a Gaussian Pyramid

— Used for pyramid blending, feature extraction etc.

# Recap: **Multi-Scale** Template Matching

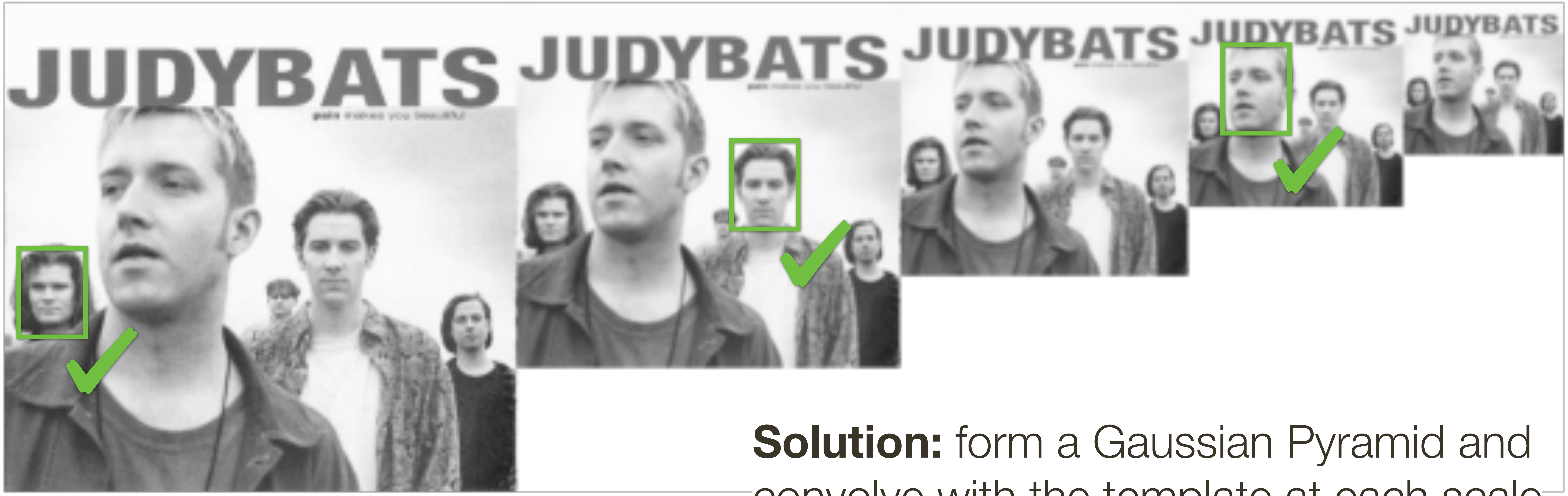
**Correlation** with a **fixed-sized image** only detects faces at **specific scales**




= Template

# Recap: **Multi-Scale** Template Matching

**Correlation** with a **fixed-sized image** only detects faces at **specific scales**




**Solution:** form a Gaussian Pyramid and convolve with the template at each scale

Q. **Why scale** the **image** and not the **template**?

= Template

# **Improving** Template Matching

Consider the problem of finding images of an elephant using a template

An elephant looks different from different viewpoints

— from above (as in an aerial photograph or satellite image)

— head on

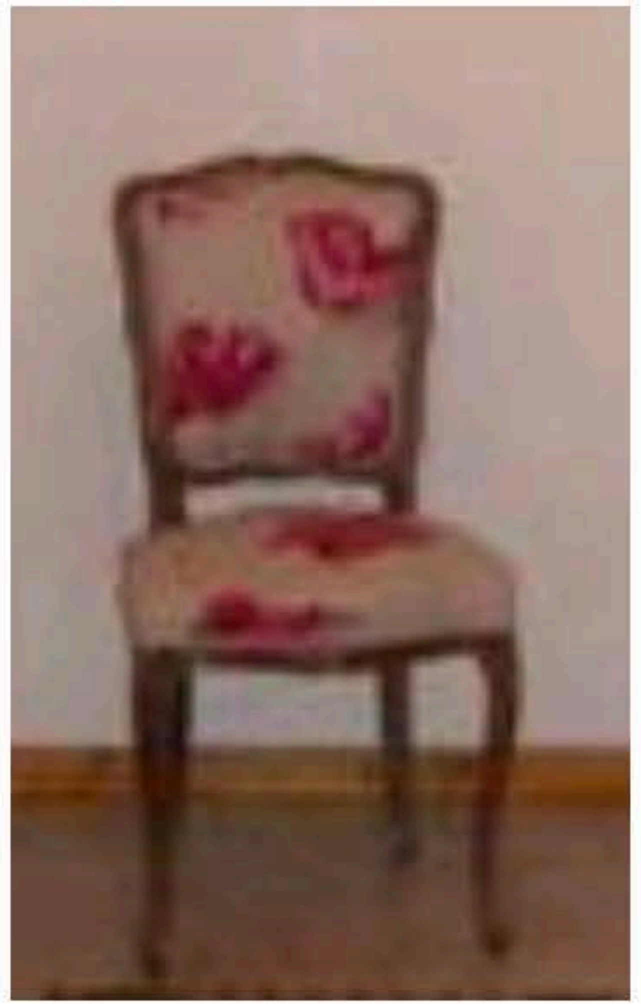
— sideways (i.e., in profile)

— rear on

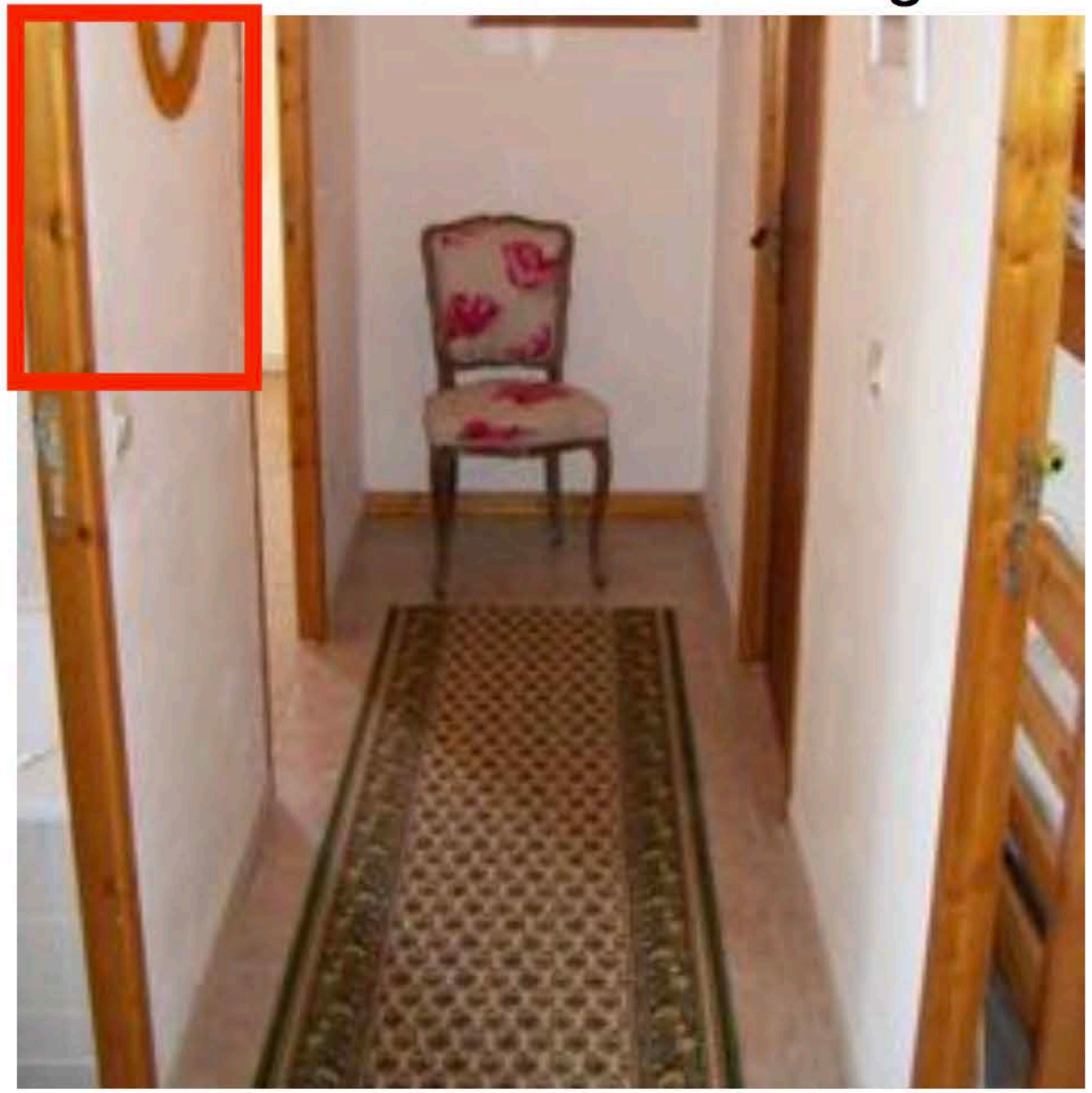
What happens if parts of an elephant are obscured from view by trees, rocks, other elephants?

# **Improving** Template Matching

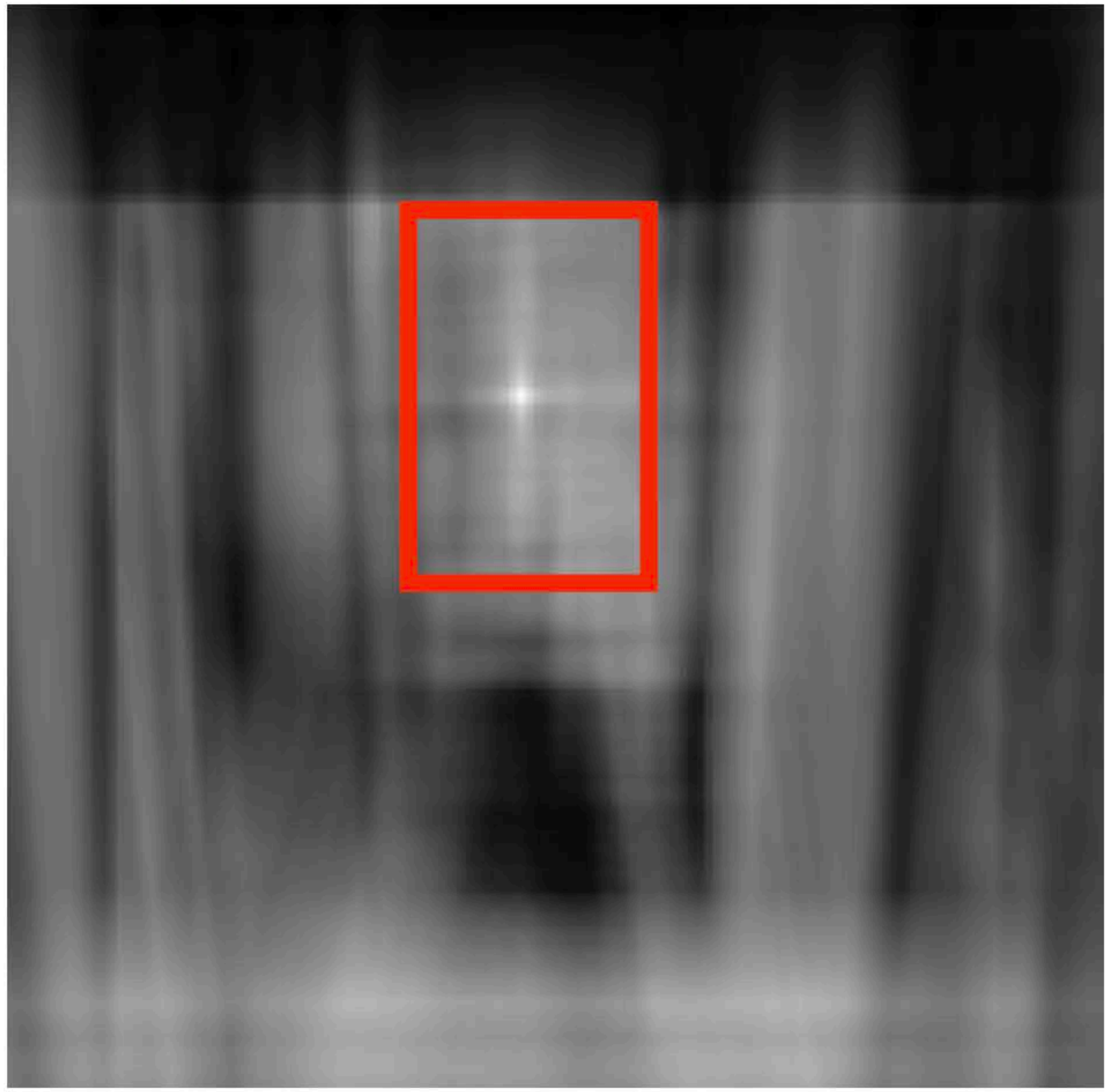
This is a chair



Find the chair in this image

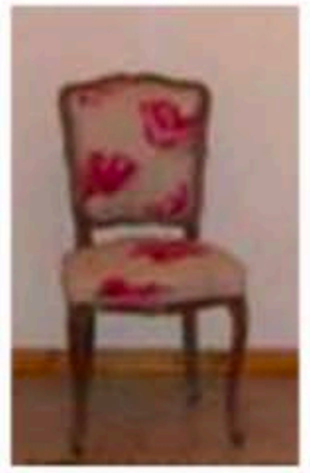


Output of normalized correlation

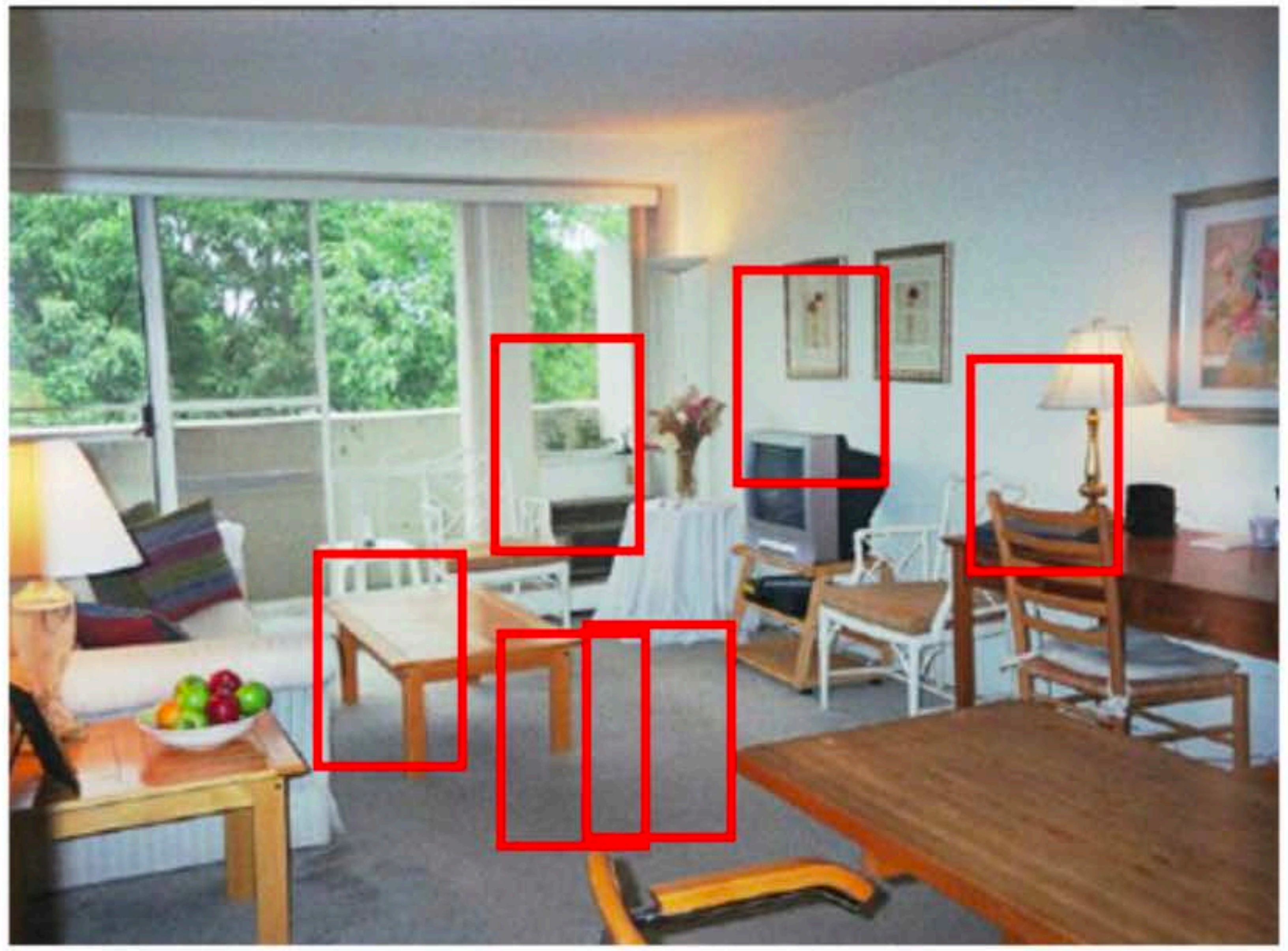


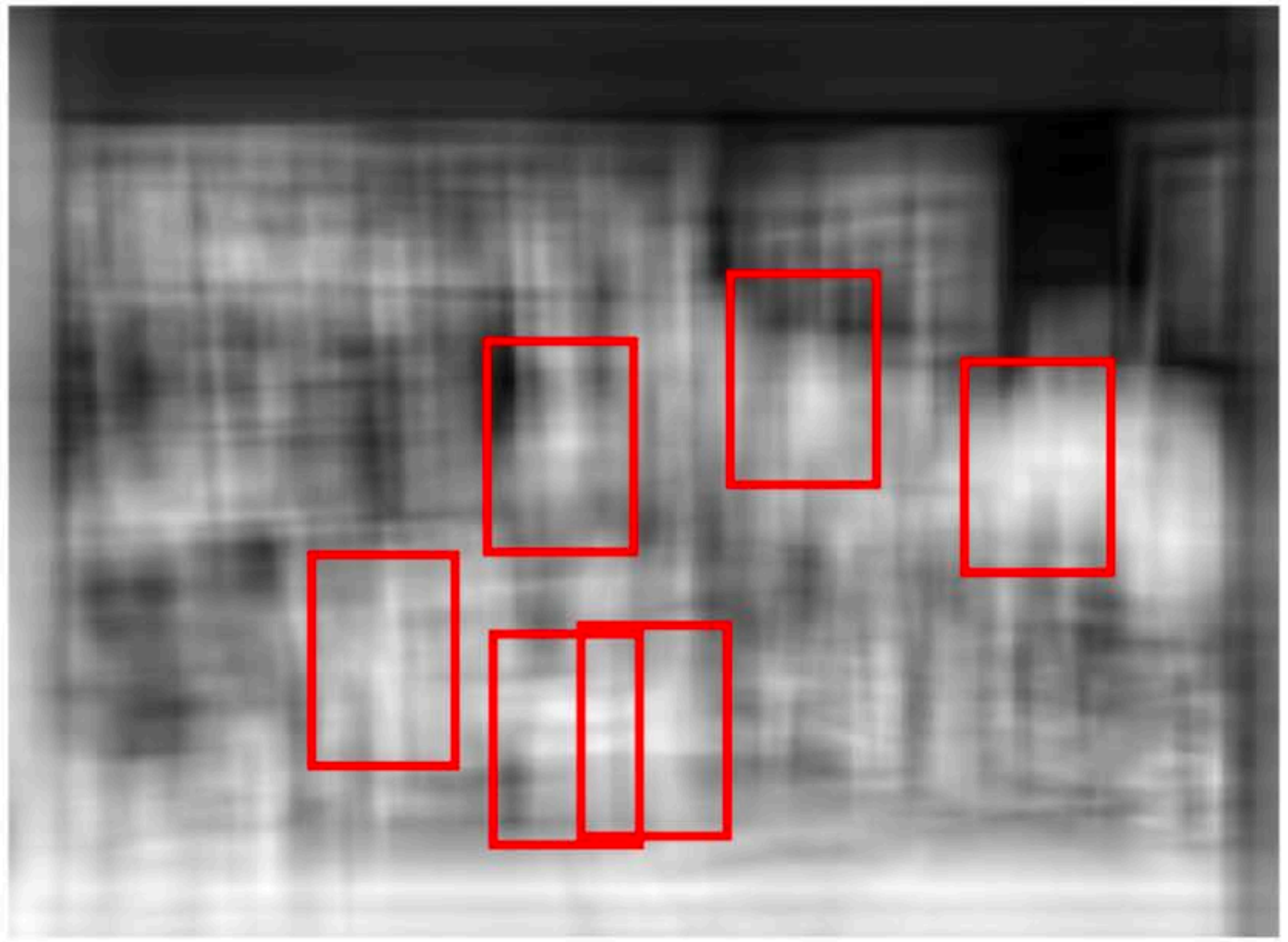
**Slide Credit**: Li Fei-Fei, Rob Fergus, and Antonio Torralba

# **Improving** Template Matching



Find the chair in this image





Pretty much garbage
Simple template matching is not going to make it

**Slide Credit**: Li Fei-Fei, Rob Fergus, and Antonio Torralba

# **Improving** Template Matching

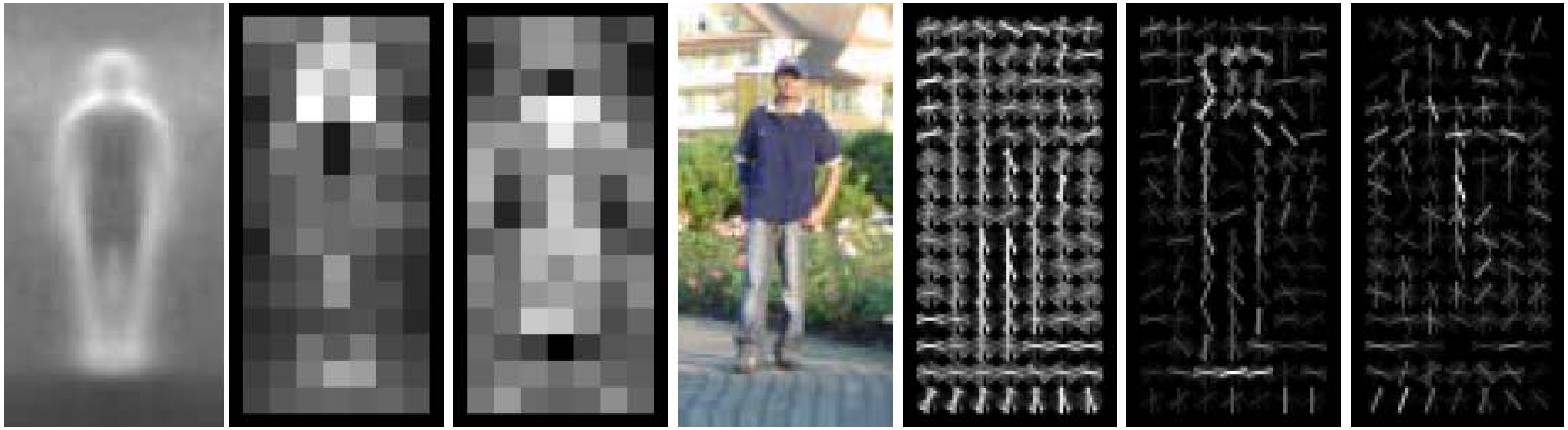
Improved detection algorithms make use of **image features**

These can be **hand coded** or **learned**

# Template Matching with HOG

Template matching can be improved by using better features, e.g., Histograms of Gradients (HOG) [ Dalal Triggs 2005 ]

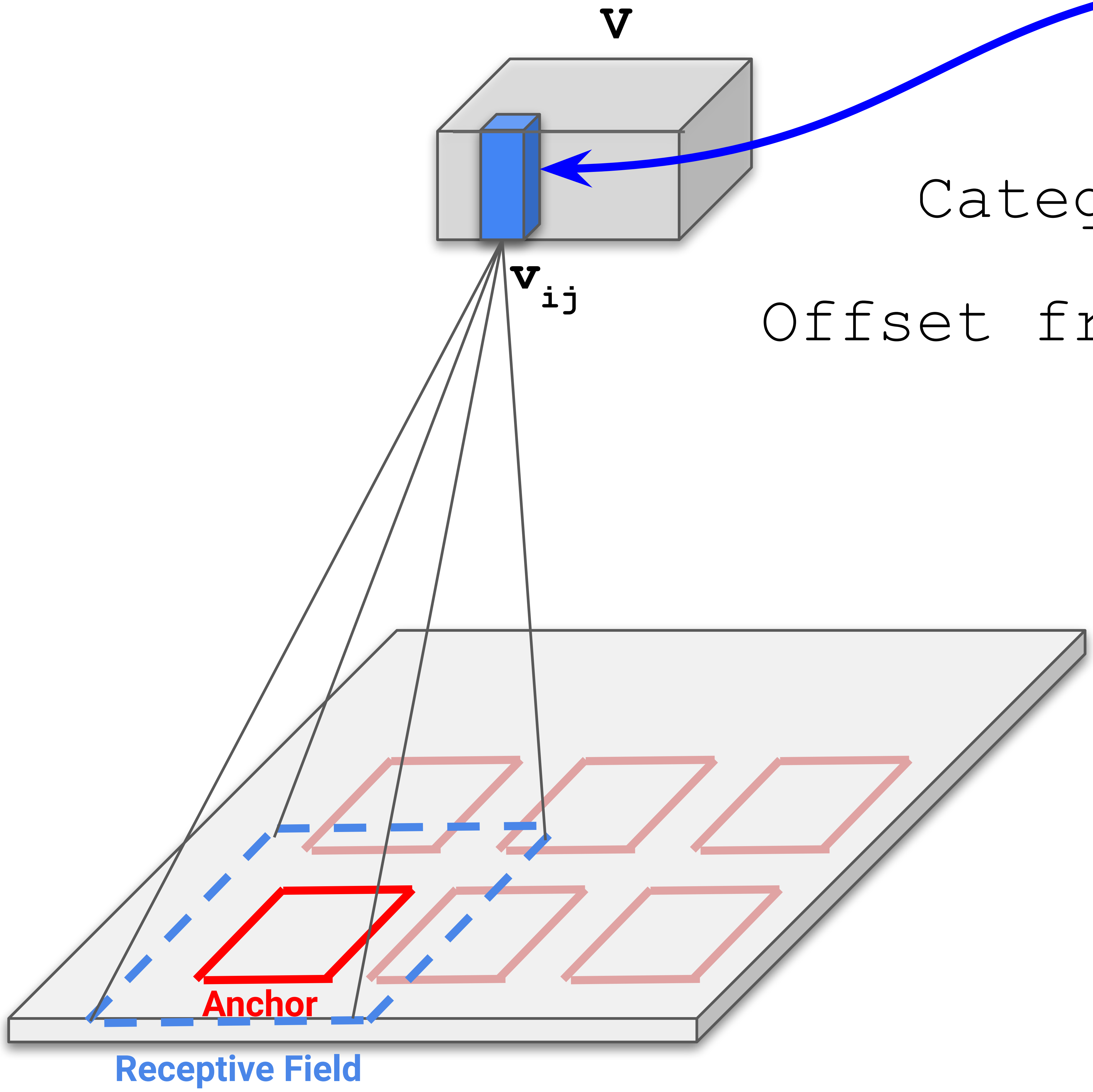
The authors use a Learning-based approach (Support Vector Machine) to find an optimally weighted template



avg grad SVM weights HOG weighted HOG

+ −

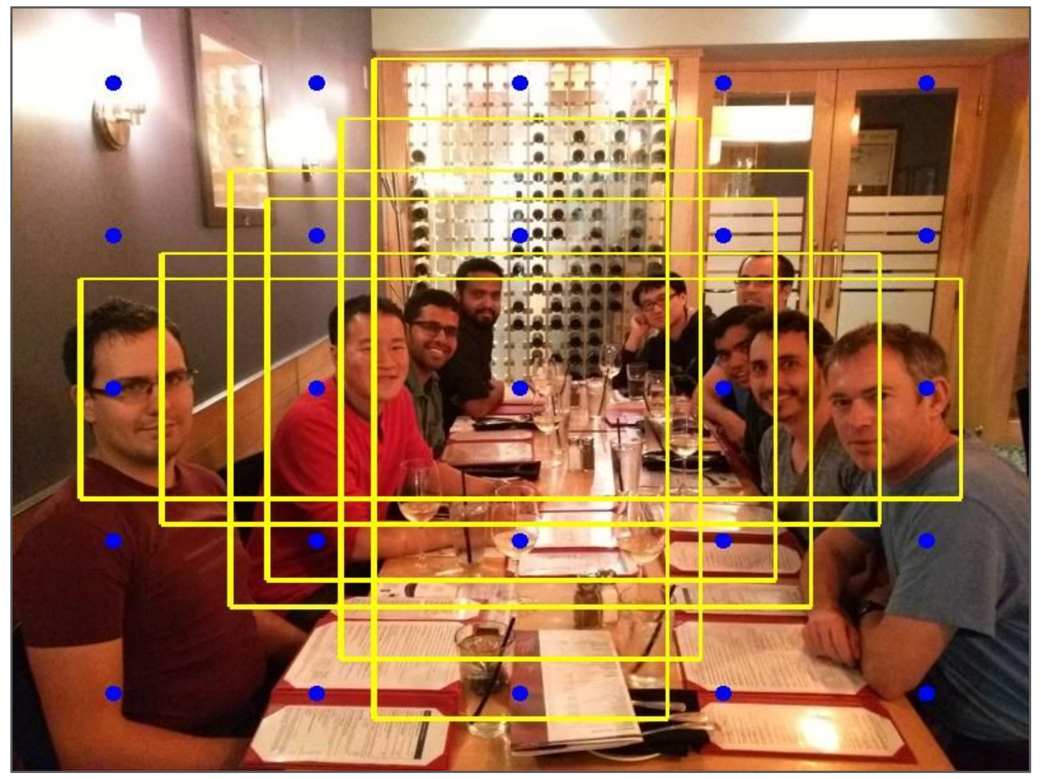
# **Convnet** Object Detection




Think of each feature vector $\mathbf{v}_{ij}$ as corresponding to a sliding window (anchor).

$$\text{Category score} = \text{SoftMax}(W^{cls} \cdot \mathbf{v}_{ij})$$

$$\text{Offset from anchor} = W^{loc} \cdot \mathbf{v}_{ij}$$



— Convnet based object detectors resemble sliding window template matching in feature space

— Architectures typically involve multiple scales and aspect ratios, and regress to a 2D offset in addition to category scores

[ Images: Jonathan Huang ]

# Summary

**Template matching** as (normalized) correlation. Template matching is not robust to changes in:

— 2D spatial scale and 2D orientation

— 3D pose and viewing direction

— illumination

**Scaled representations** facilitate

— template matching at multiple scales

— efficient search for image–to–image correspondences

— image analysis at multiple levels of detail

A **Gaussian pyramid** reduces artifacts introduced when sub-sampling to coarser scales