THE UNIVERSITY OF BRITISH COLUMBIA

# CPSC 425: Computer Vision

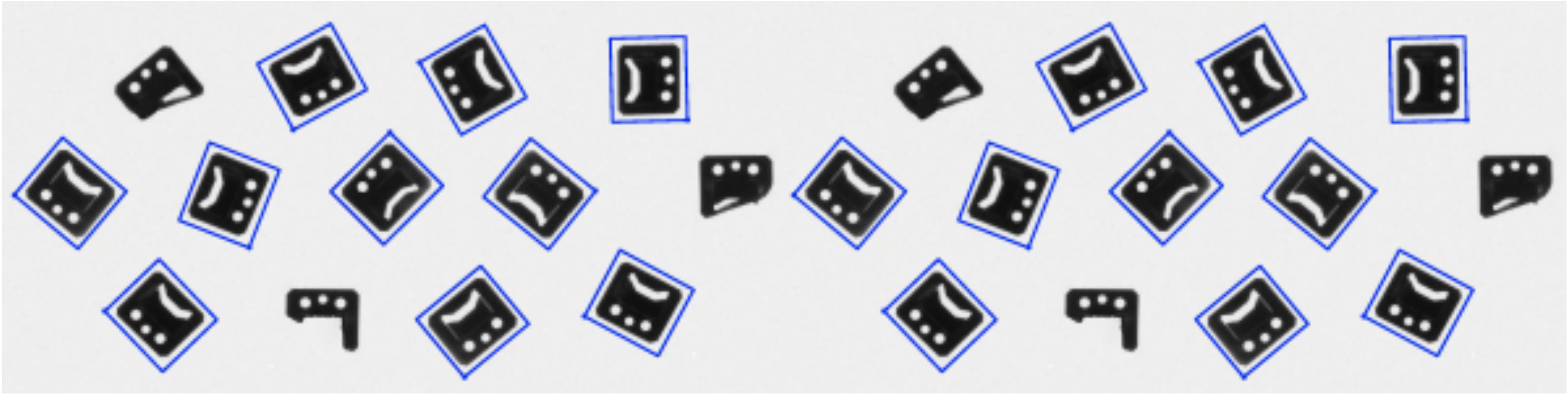


**Image Credit**: https://docs.adaptive-vision.com/4.7/studio/machine_vision_guide/TemplateMatching.html

## **Lecture 10:** Scaled Representations (cont.), Image Gradients

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

# **Menu** for Today (**September 30, 2020**)

## Topics:

— **Scaled** Representations

— Image **Derivatives**

— Edge **Detection**

## Redings:

— **Today's** Lecture: Forsyth & Ponce (2nd ed.) 4.5 - 4.7, 5.1

— **Next** Lecture: Forsyth & Ponce (2nd ed.) 5.1 - 5.2

## Reminders:

— **Assignment 1:** Image Filtering and Hybrid Images is due **today**

— **Assignment 2**: Scaled Representations, Face Detection and Image Blending (likely **tonight**)

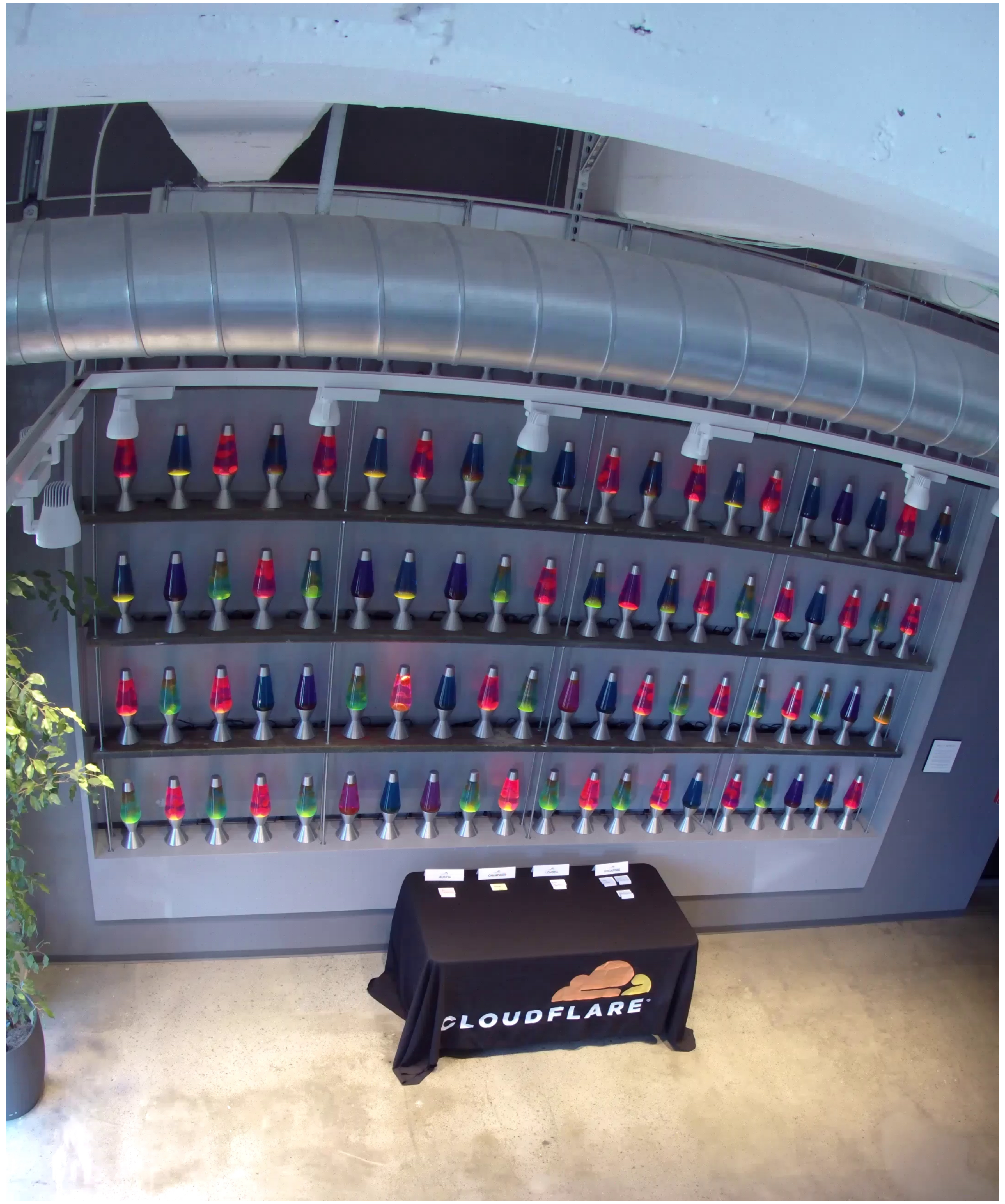
# Today's "**fun**" Example: NCIS

# Today's "**fun**" Example: NCIS

# Today's "**fun**" Example: NCIS

# Today's "**fun**" Example: LavaRAND

# Today’s “**fun**” Example: LavaRAND



# **Lecture 9**: Re-cap **Template** Matching

Linear filtering the entire image computes the entire set of dot products, one for each possible alignment of filter and image

Important **Insight**:

— filters look like the pattern they are intended to find

— filters find patterns they look like

Linear filtering is sometimes referred to as **template matching**

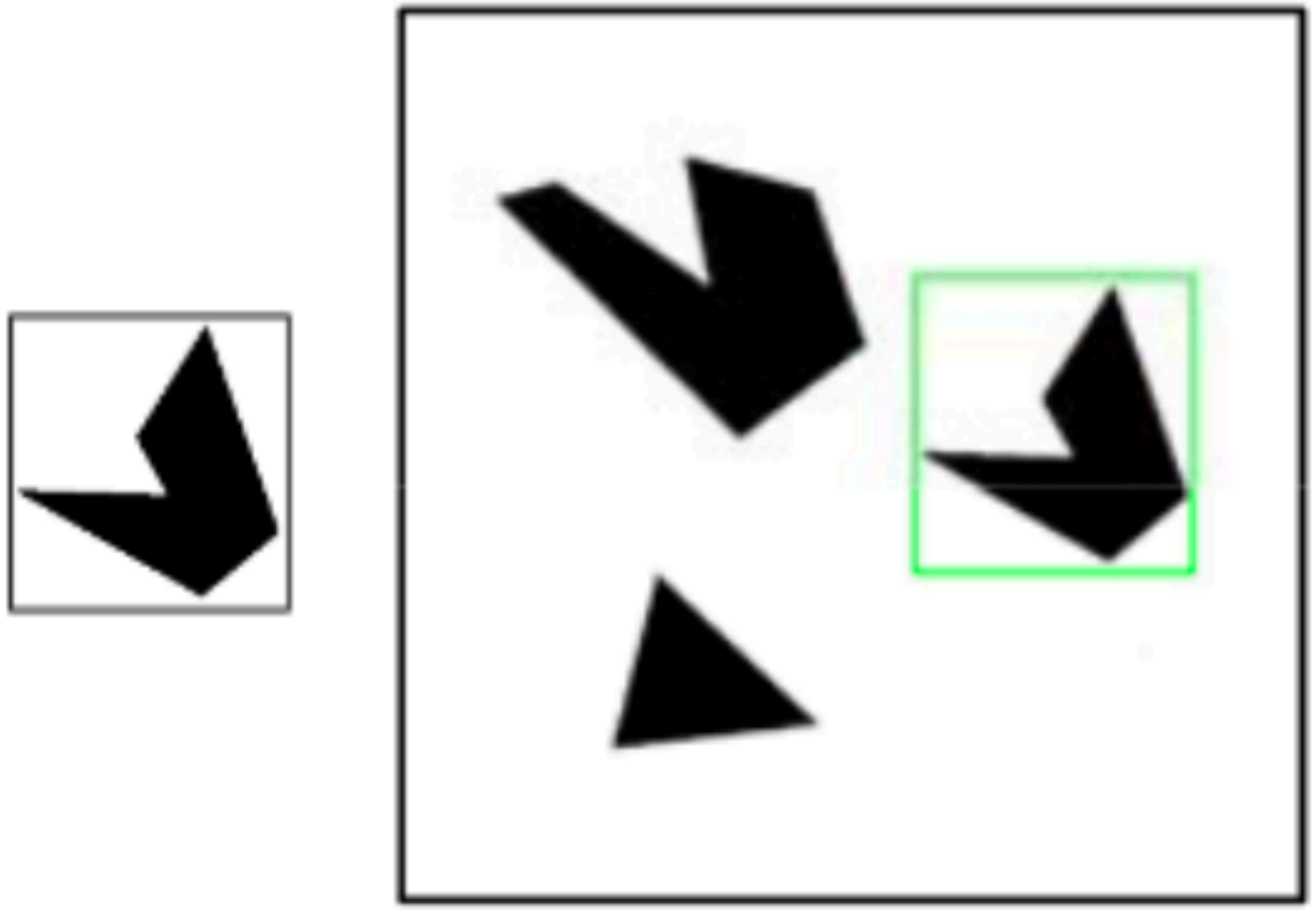
# **Lecture 9**: Re-cap **Template** Matching

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

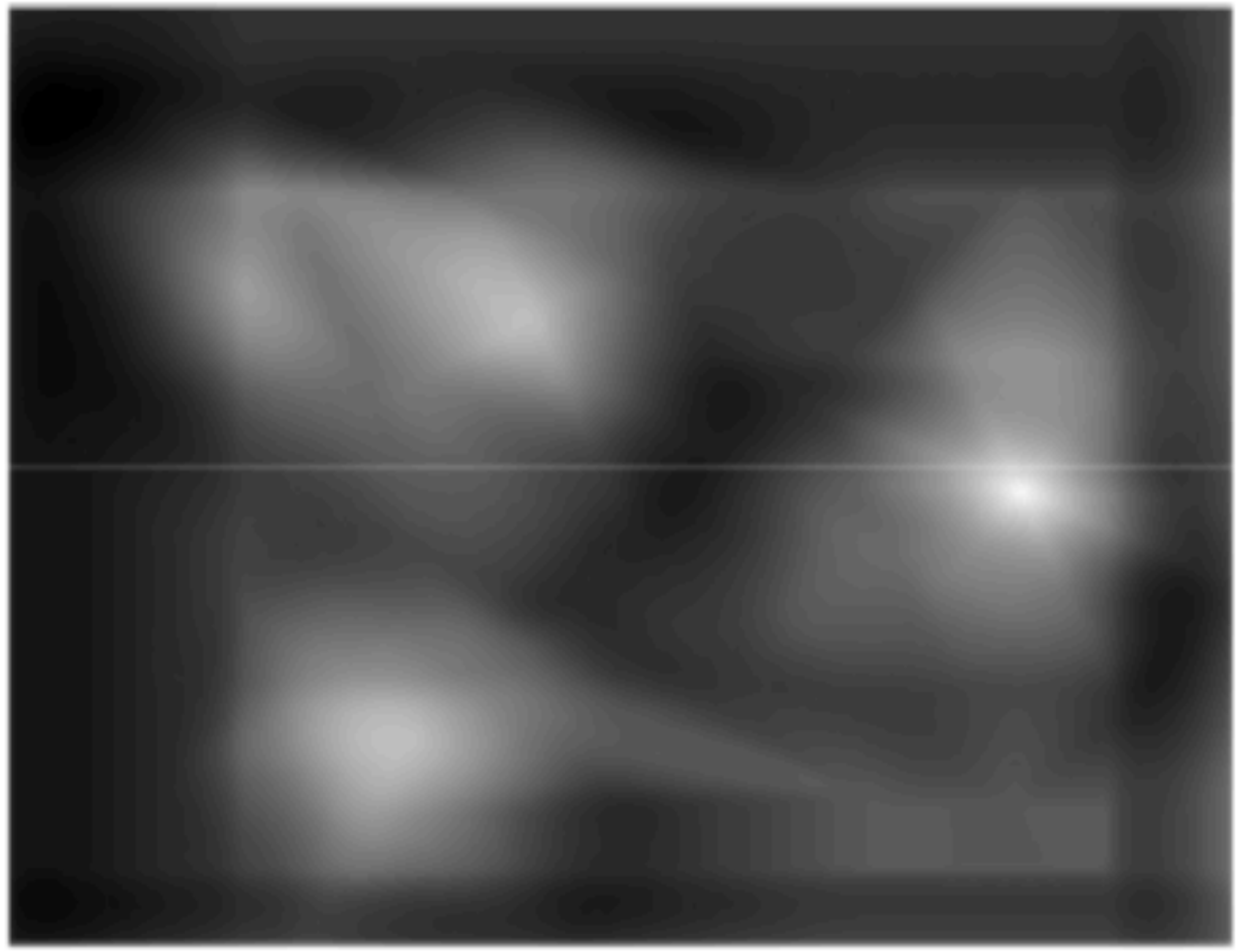
$$\cos\theta = \frac{a \cdot b}{|a||b|} = \frac{a \cdot b}{\sqrt{(a \cdot a)(b \cdot b)}} = \frac{a}{|a|}\frac{b}{|b|}$$

where · is dot product and | | is vector magnitude

# **Lecture 9**: Re-cap **Template** Matching



Detected template



Correlation map

**Slide Credit**: Kristen Grauman

# **Lecture 9**: Re-cap

Template (left), image (middle), normalized correlation (right)

Note peak value at the true position of the hand

**Credit**: W. Freeman et al., “Computer Vision for Interactive Computer Graphics,” IEEE Computer Graphics and Applications, 1998

# **Lecture 9**: Re-cap

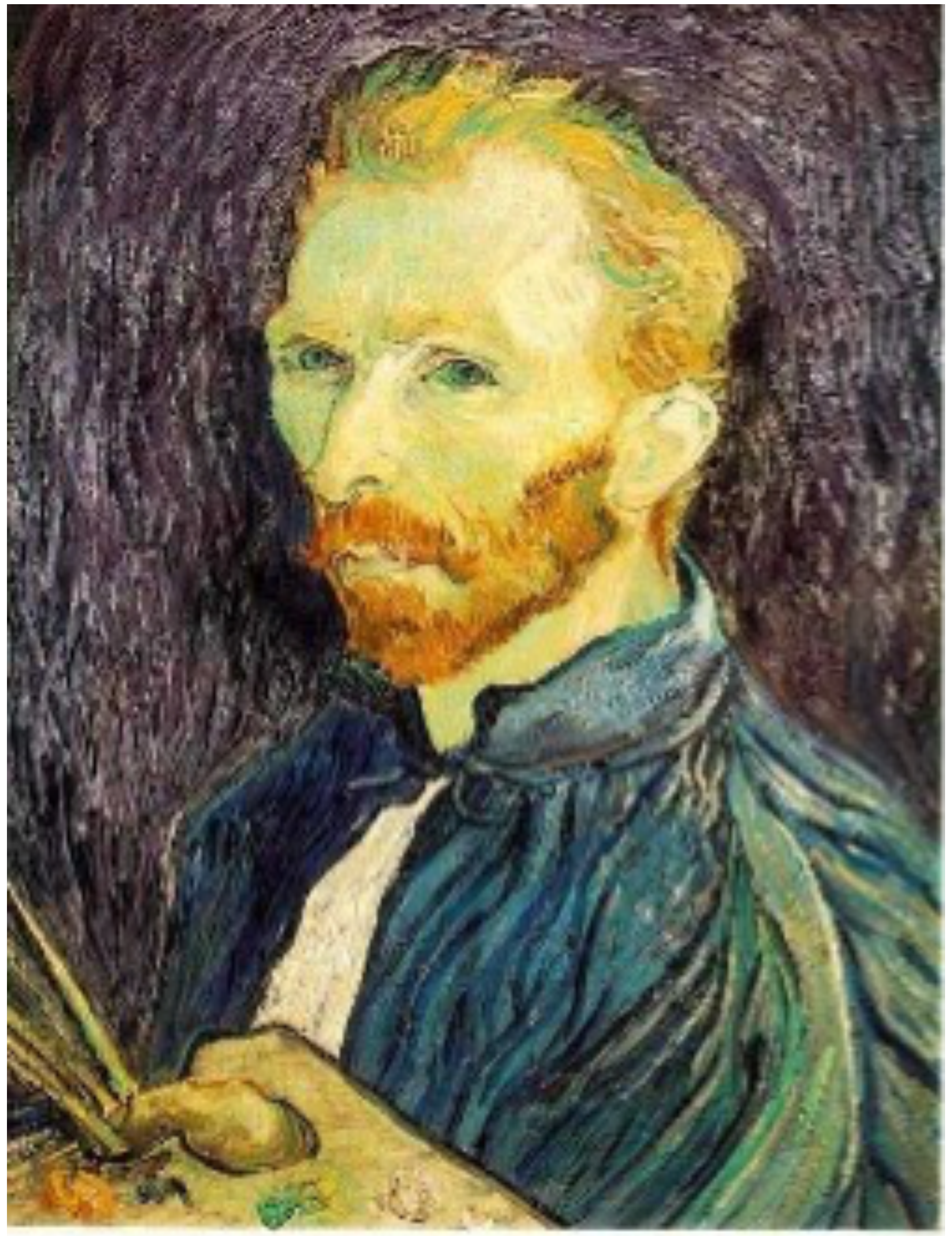
**Template matching** as (normalized) correlation

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

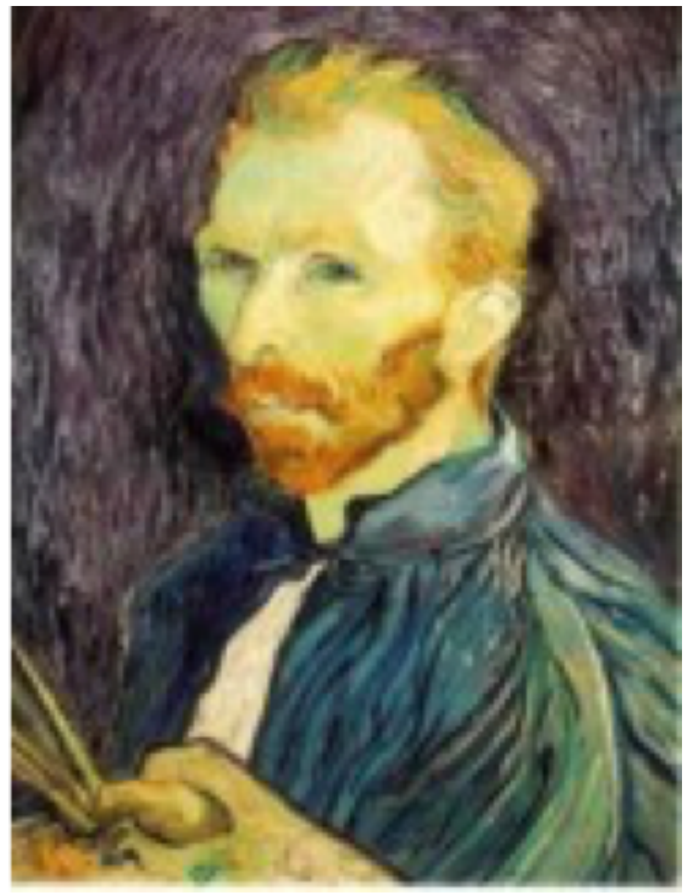
# **Template** Matching: Sub-sample with Gaussian Pre-filtering



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

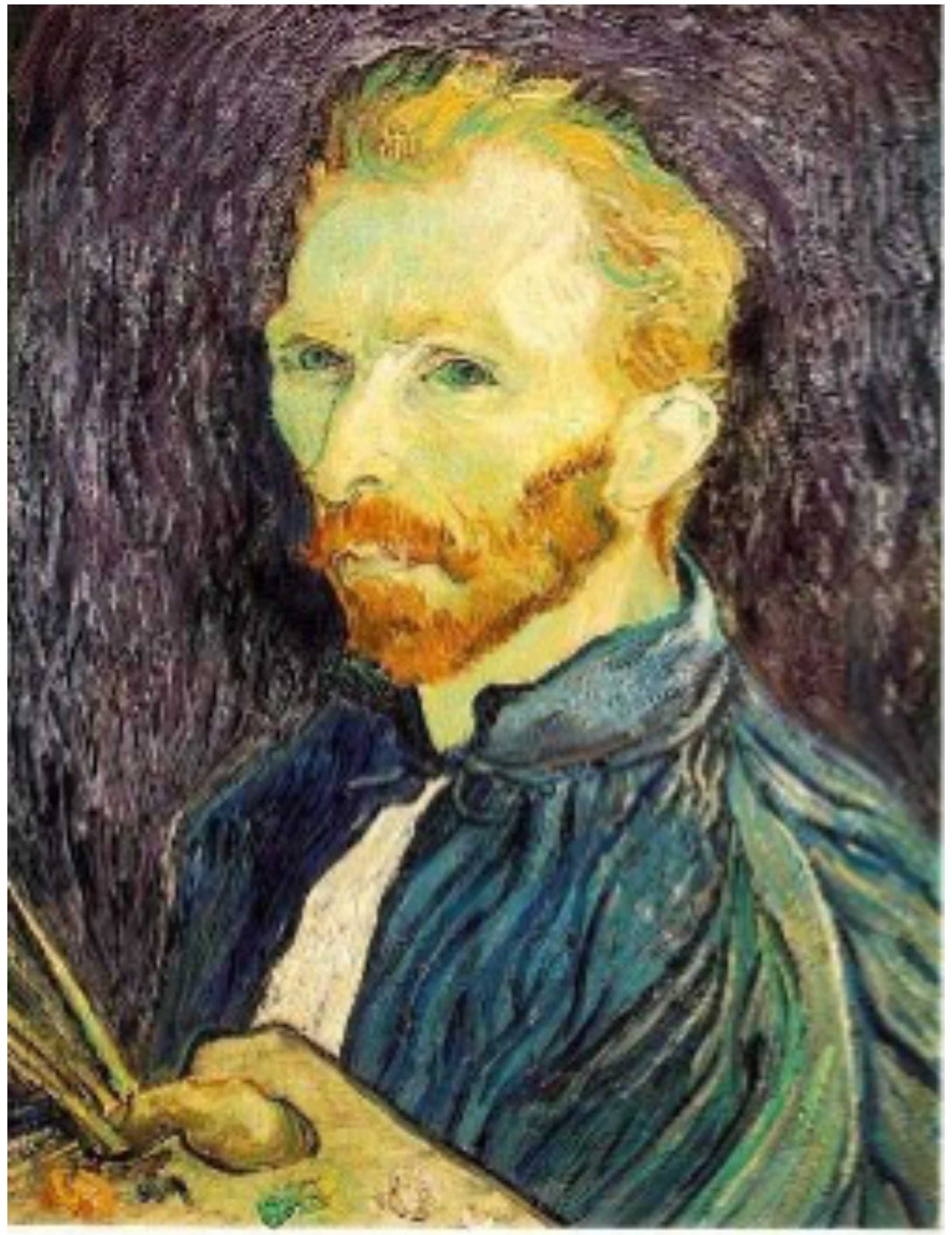
Gaussian filter
delete even rows
delete even columns

1/8

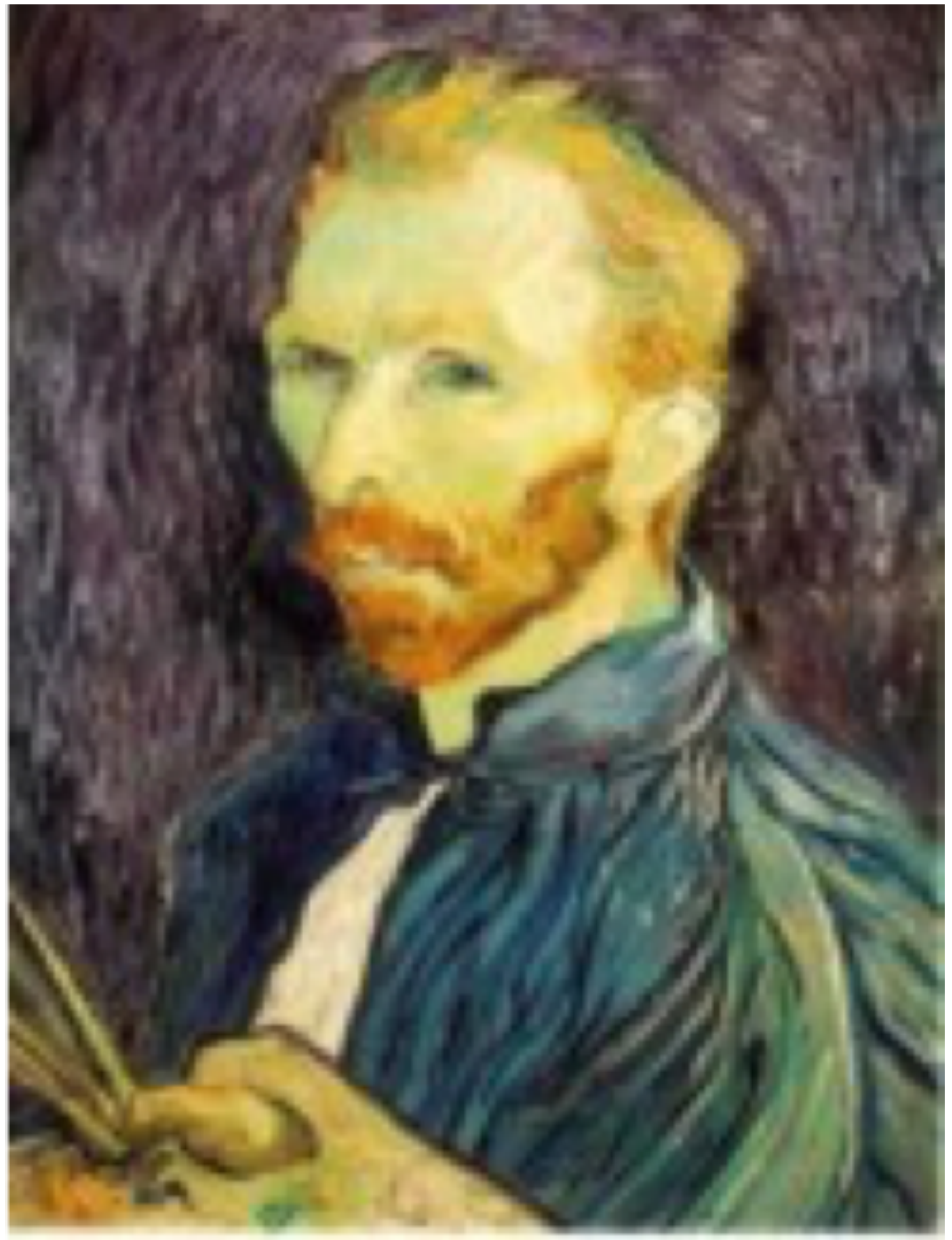


**Slide Credit**: Ioannis (Yannis) Gkioulekas (CMU)

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



1/2



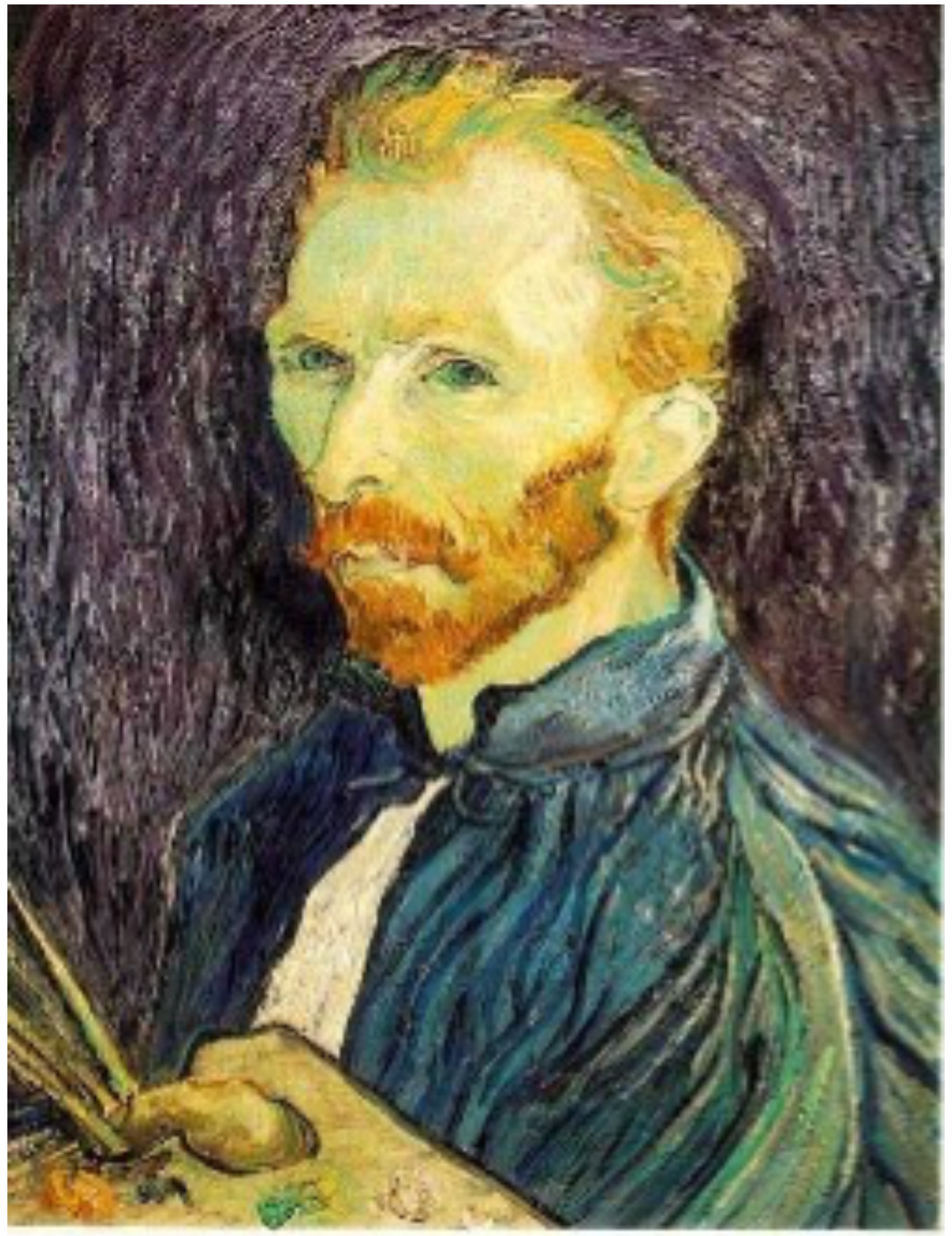
1/4 (2x zoom)

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)

# **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 $\sigma = 1$

**In General:** Sigma inversely proportional to image reduction $\sigma = \frac{1}{2s}$

# Image **Pyramid**

An **image pyramid** is a collection of representations of an image. Typically, each layer of the pyramid is half the width and half the height

of the previous layer.

In a **Gaussian pyramid**, each layer is smoothed by a Gaussian filter and resampled to get the next layer

# **Gaussian** Pyramid

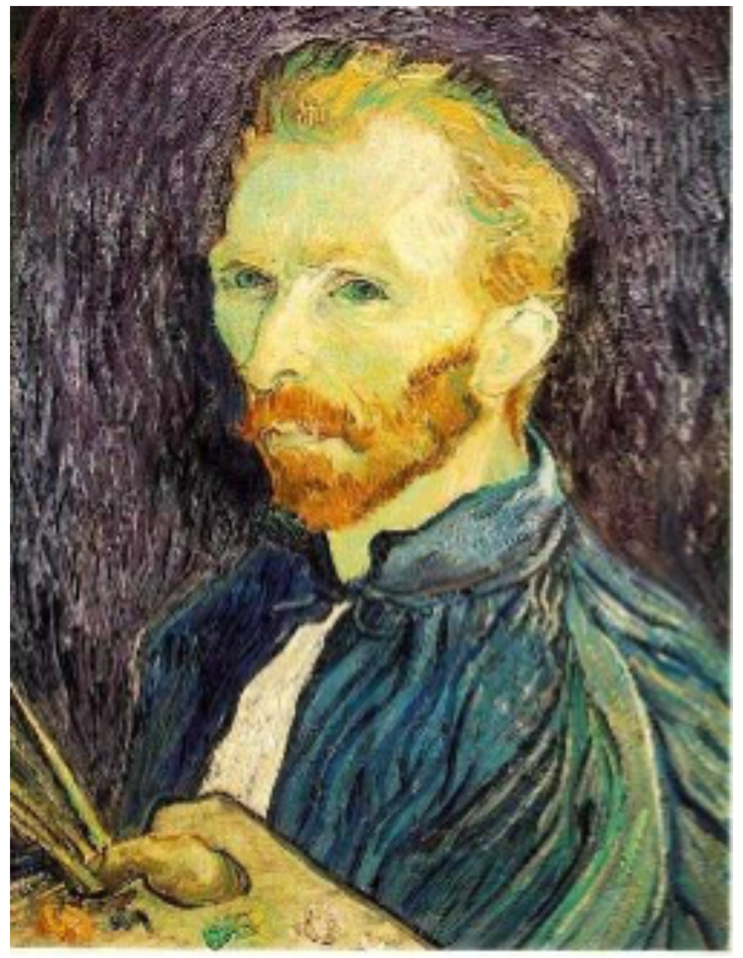
Again, let $\otimes$ denote convolution

Create each level from previous one
— smooth and (re)sample

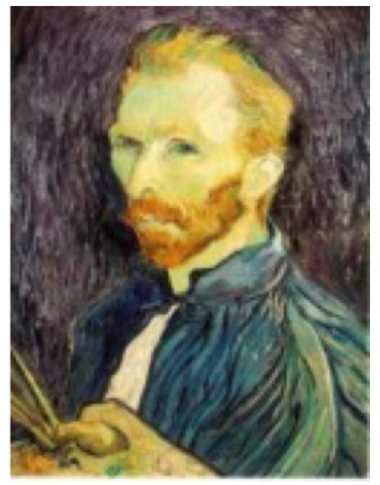
Smooth with Gaussian, taking advantage of the fact that

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

# **Gaussian** Pyramid



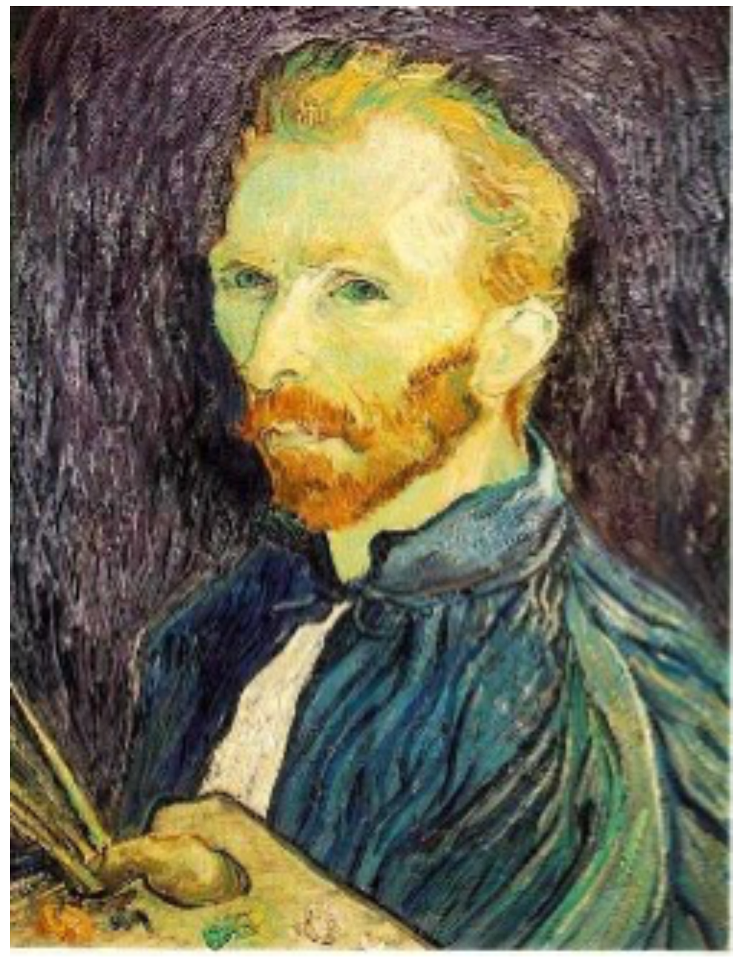
Gaussian filter ( $\sigma = 1$ )
take odd rows
take odd columns



1/2

Gaussian filter ( $\sigma = 1$ )
take odd rows
take odd columns

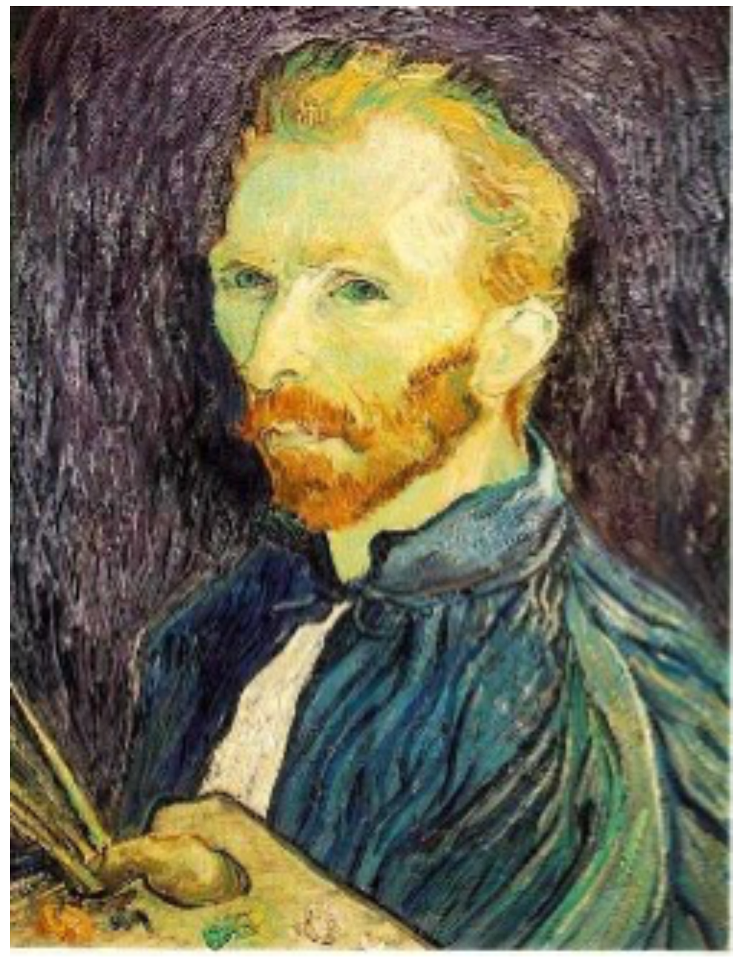
1/4



Gaussian filter ( $\sigma = \sqrt{2}$ )
take every 4th row
take every 4th column

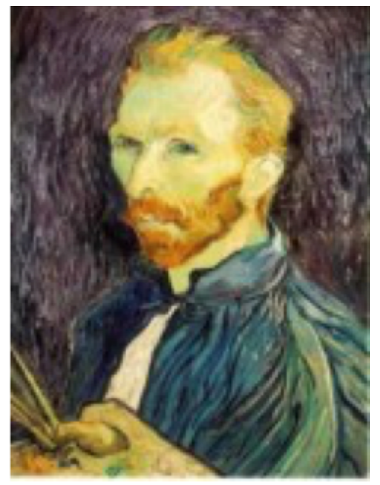
1/4

# **Gaussian** Pyramid



**Filter size**: 7x 7
applied on
**Image** = M x N

**Cost**: 49 x M x N



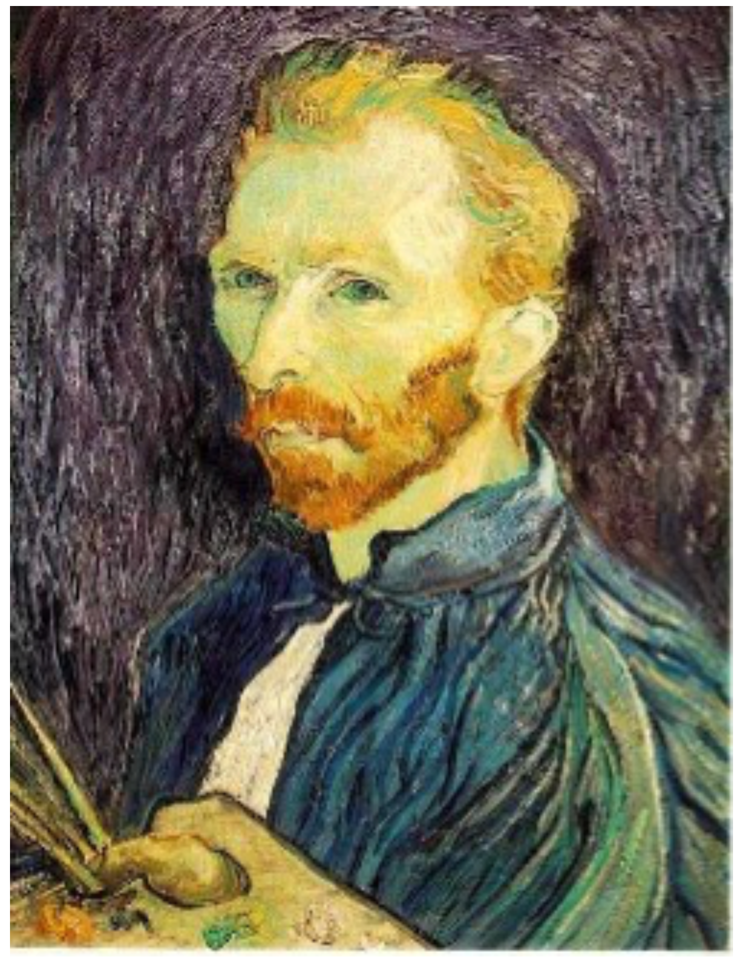
1/2

**Filter size**: 7x 7
applied on
**Image** = M/2 x N/2

**Cost**: ~12 x M x N

1/4

---

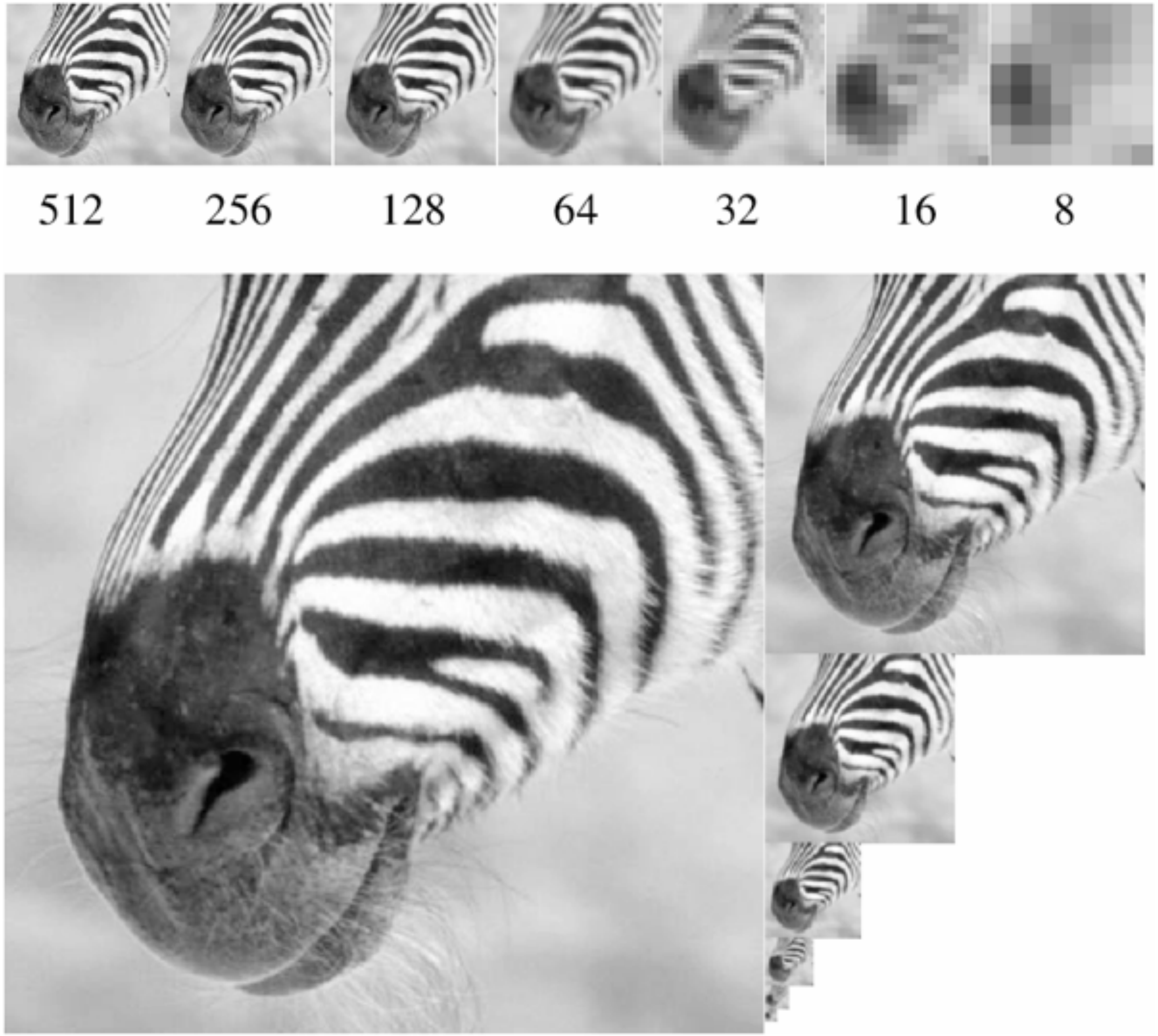


**Filter size**: 9 x 9
applied on
**Image** = M x N

**Cost**: 81 x M x N

1/4

# **Example 2**: Gaussian Pyramid

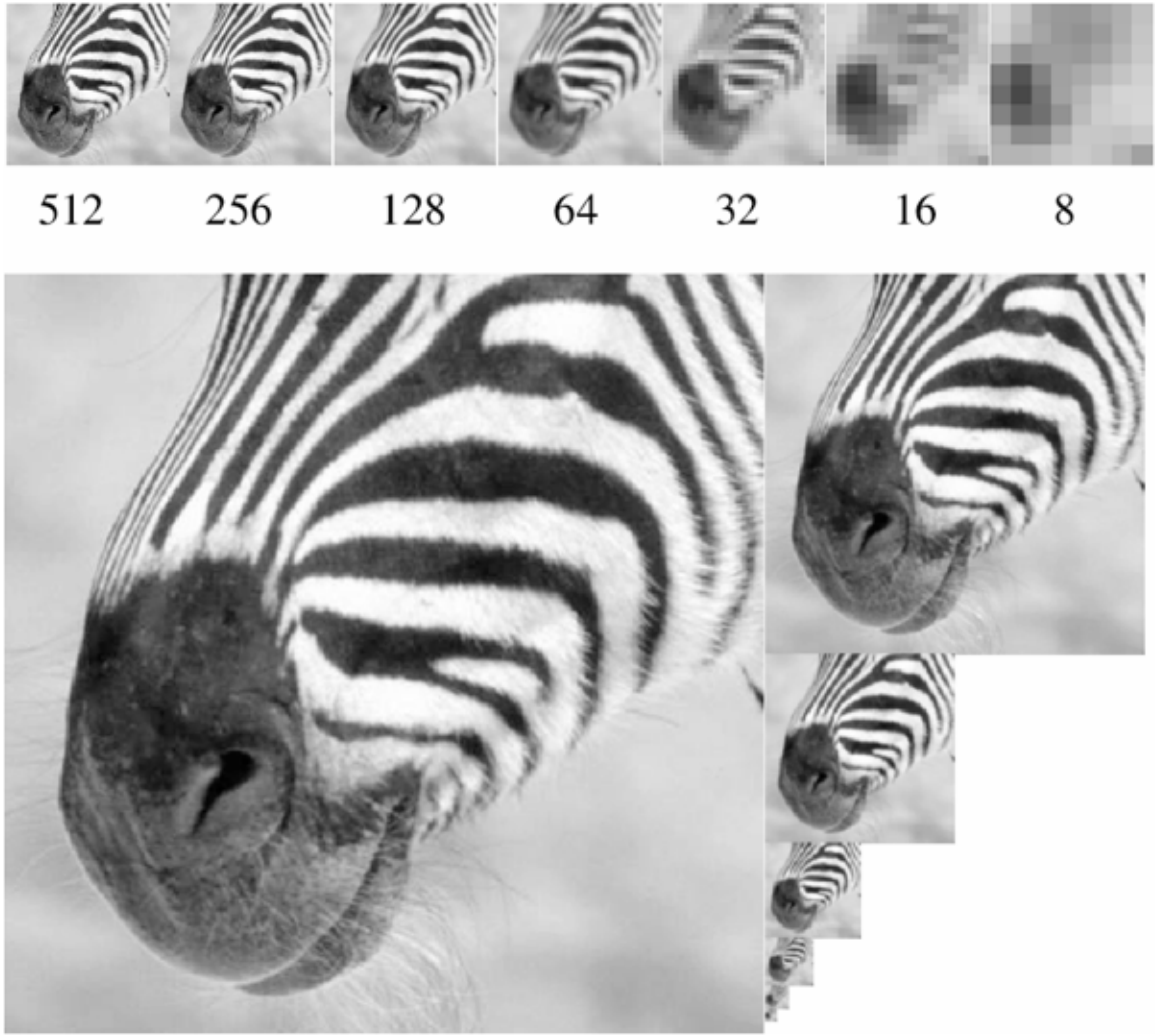



Forsyth & Ponce (2nd ed.) Figure 4.17



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

# **Example 2**: Gaussian Pyramid



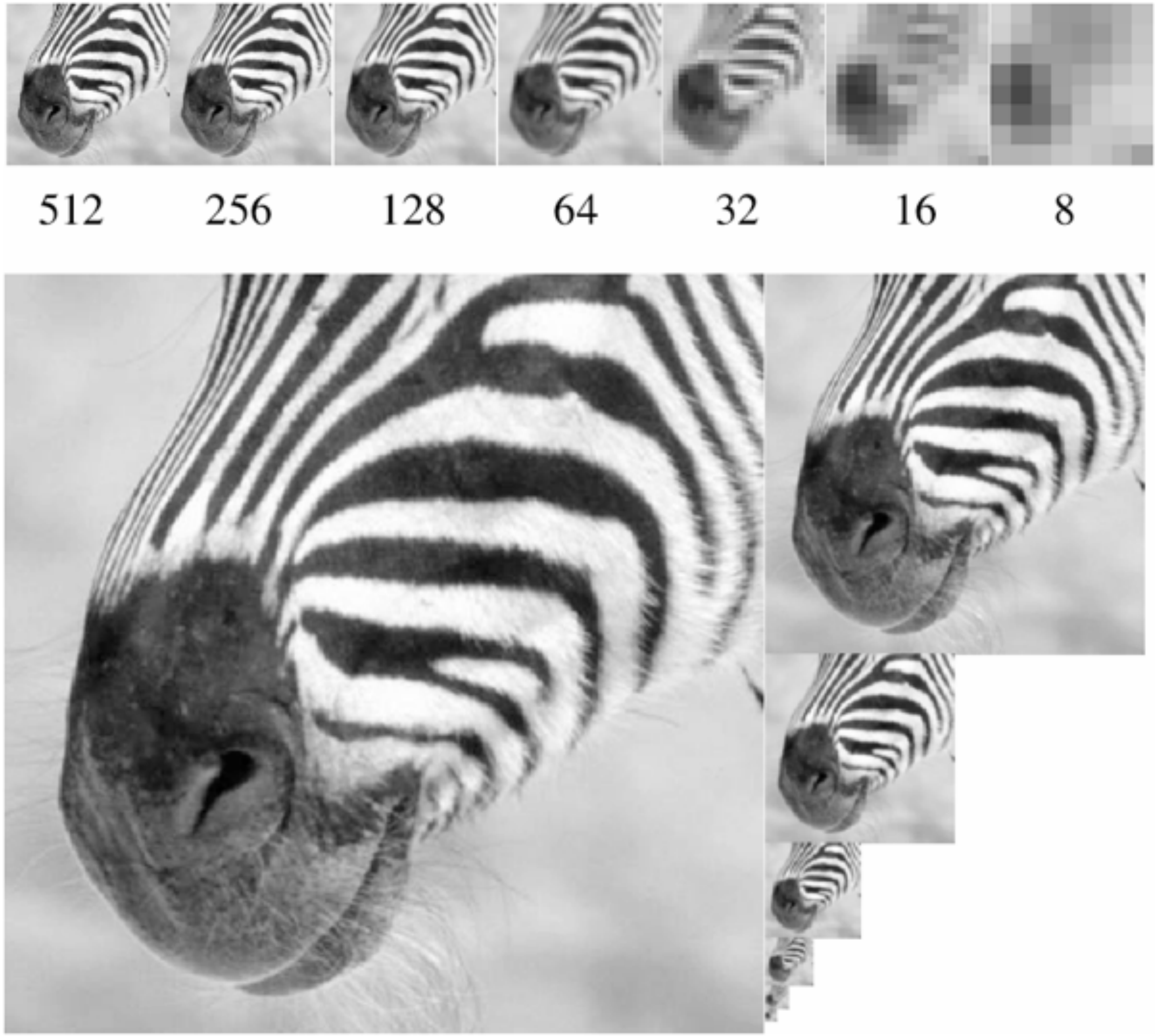

Forsyth & Ponce (2nd ed.) Figure 4.17

What happens to the details?



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

# **Example 2**: Gaussian Pyramid




Forsyth & Ponce (2nd ed.) Figure 4.17

What happens to the details?

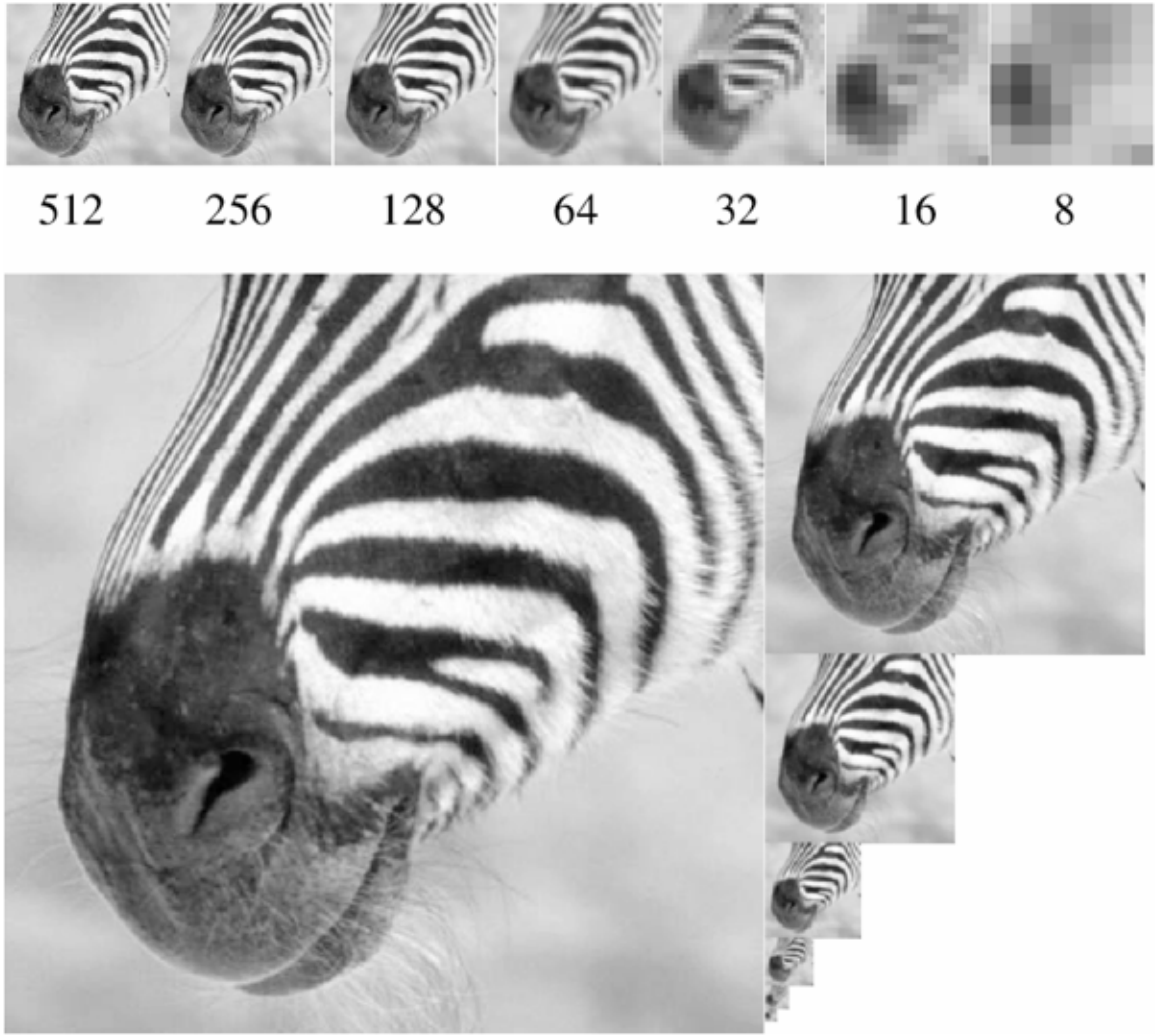
— They get smoothed out as we move to higher levels

What is preserved at the higher levels?



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

# **Example 2**: 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?

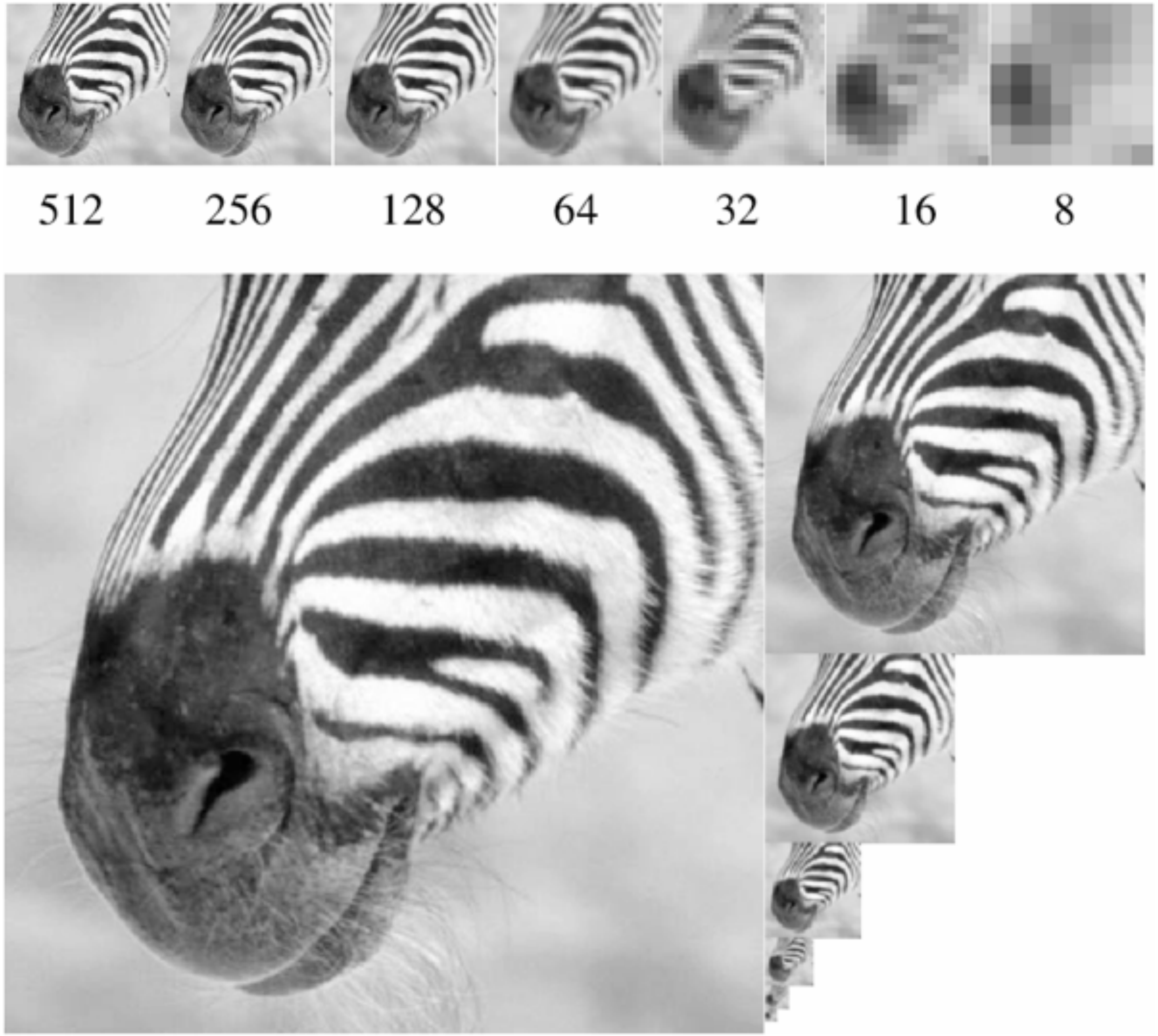
— Mostly large uniform regions in the original image

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



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

# **Example 2**: 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)

# From Template Matching to **Local Feature Detection**

We’ll now shift from global template matching to **local feature detection**

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

# From Template Matching to **Local Feature Detection**

We’ll now shift from global template matching to **local feature detection**

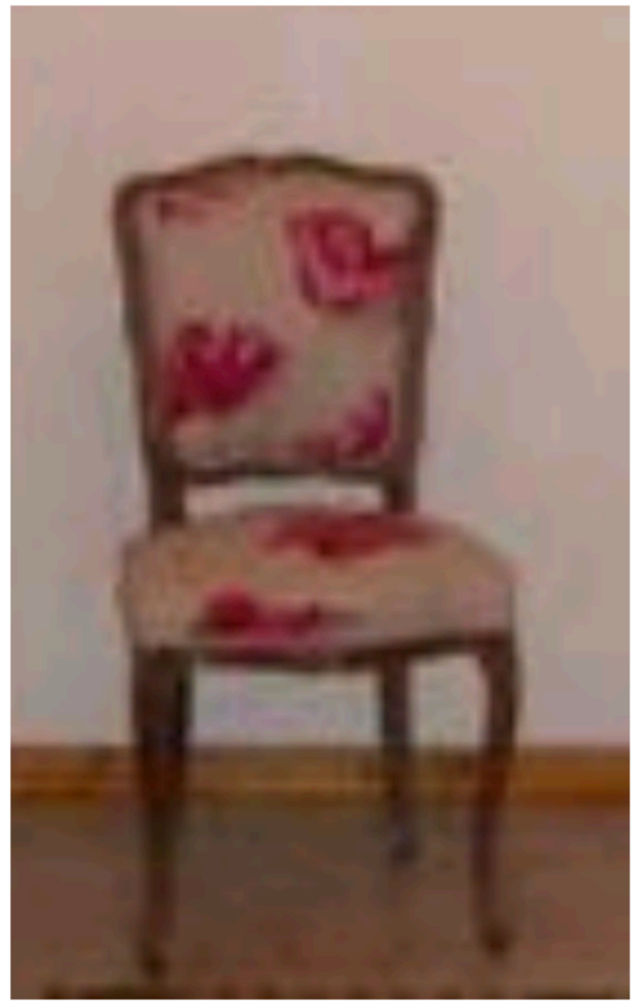
Consider the problem of finding images of an elephant using a template

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

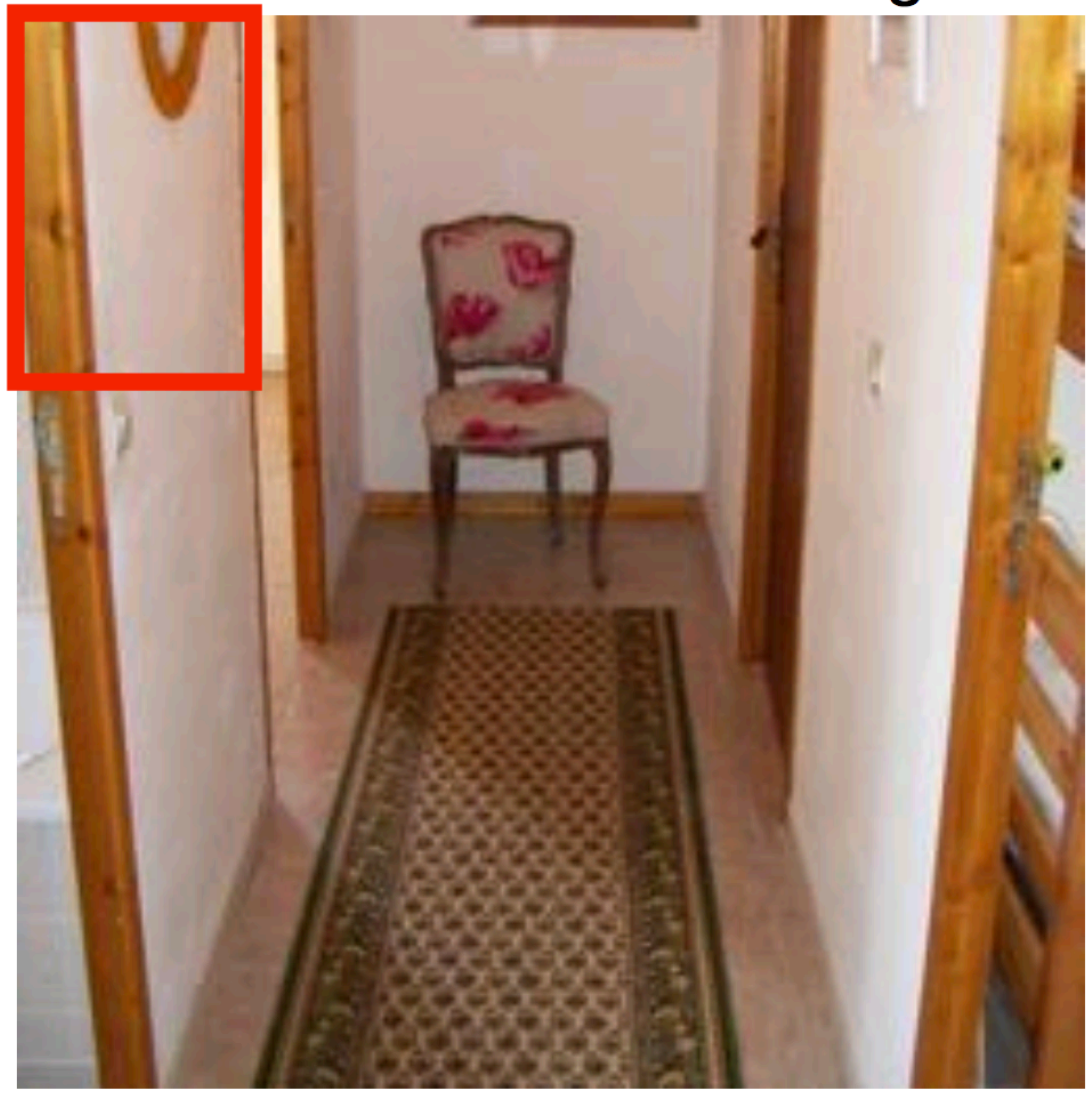
What happens if parts of an elephant are obscured from view by trees, rocks, other elephants?

# From Template Matching to **Local Feature Detection**

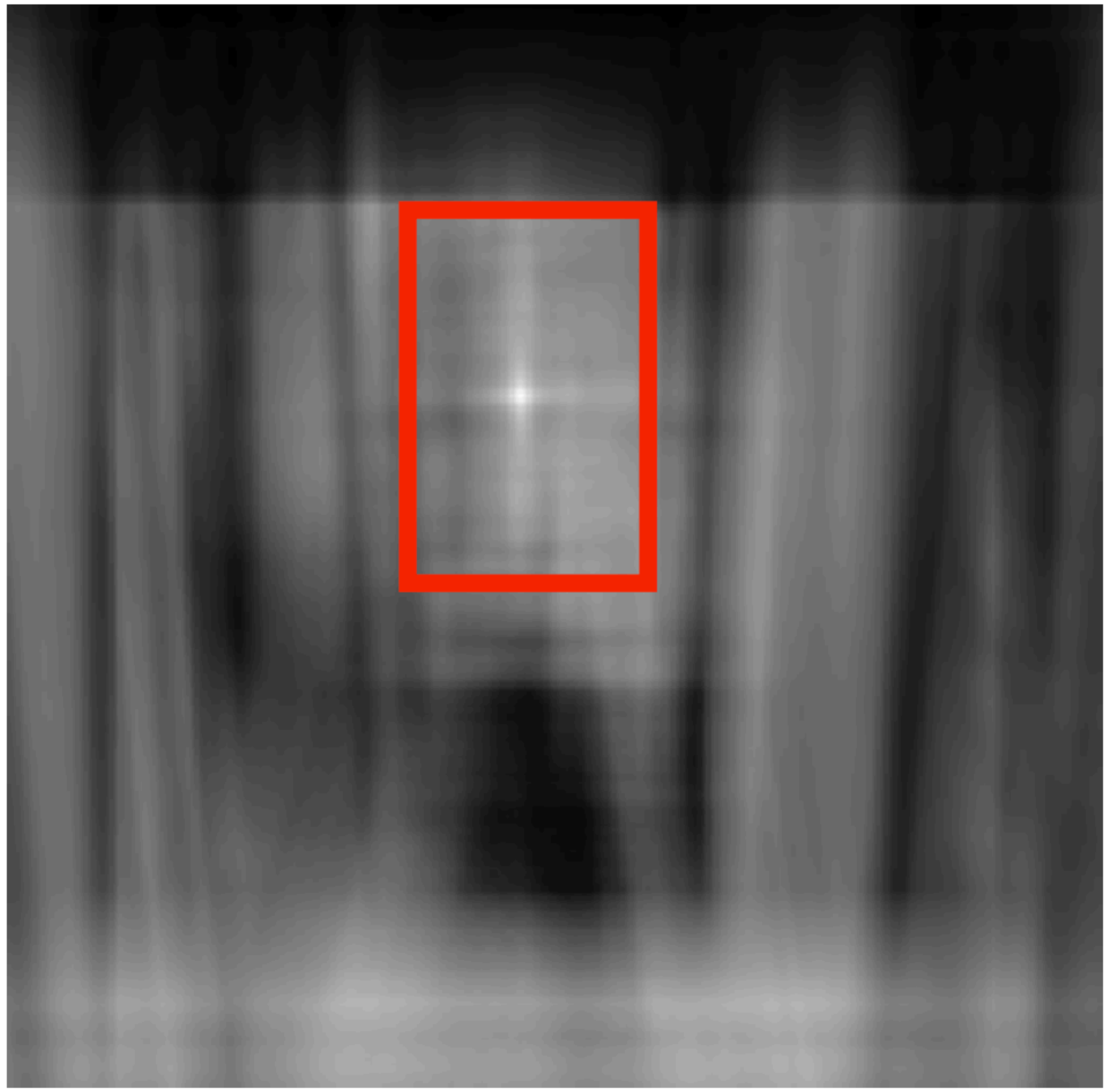
This is a chair



Find the chair in this image

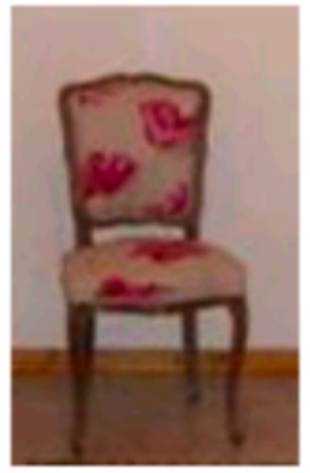


Output of normalized correlation

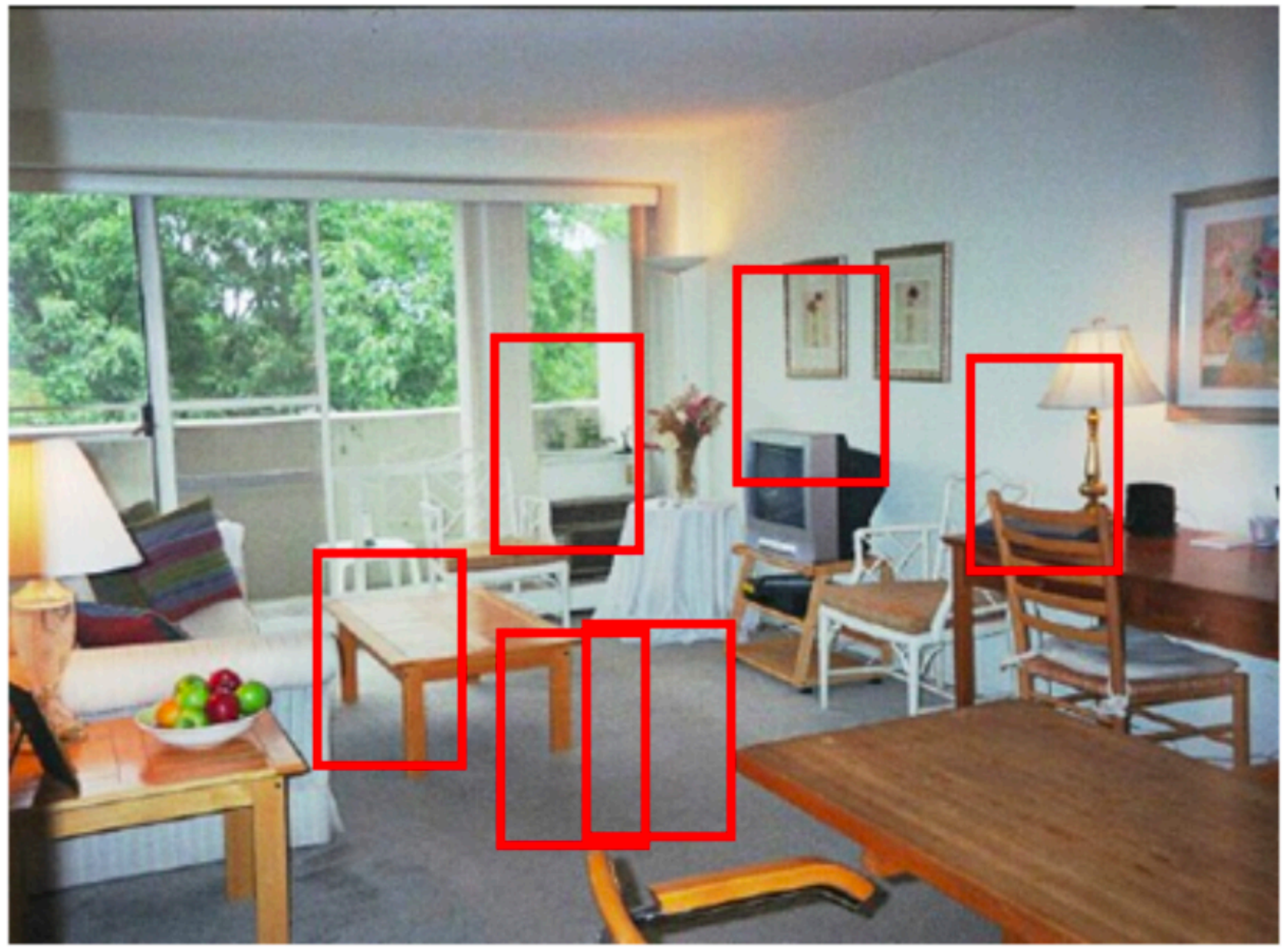


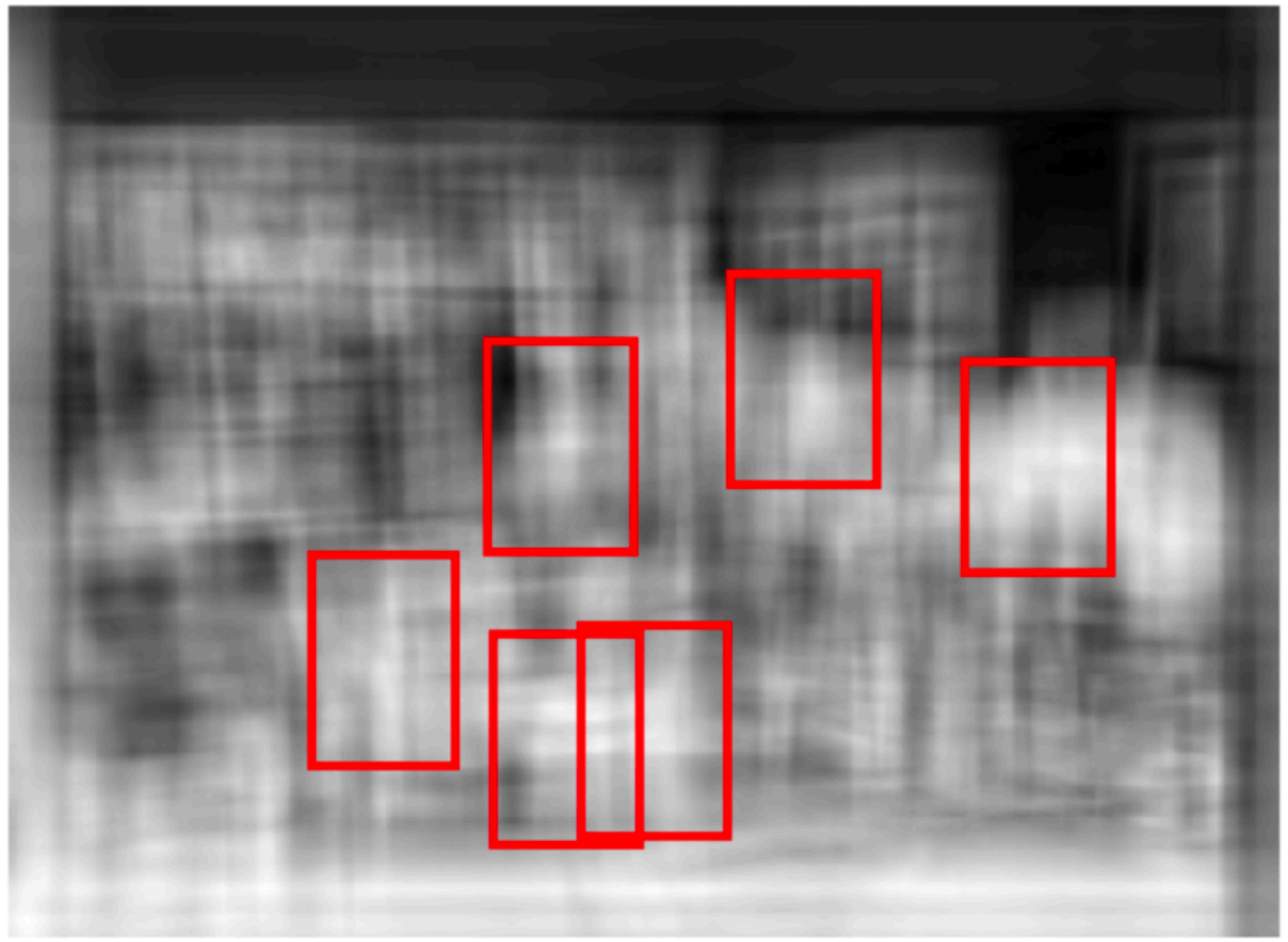
**Slide Credit**: Li Fei-Fei, Rob Fergus, and Antonio Torralba

# From Template Matching to **Local Feature Detection**



Find the chair in this image





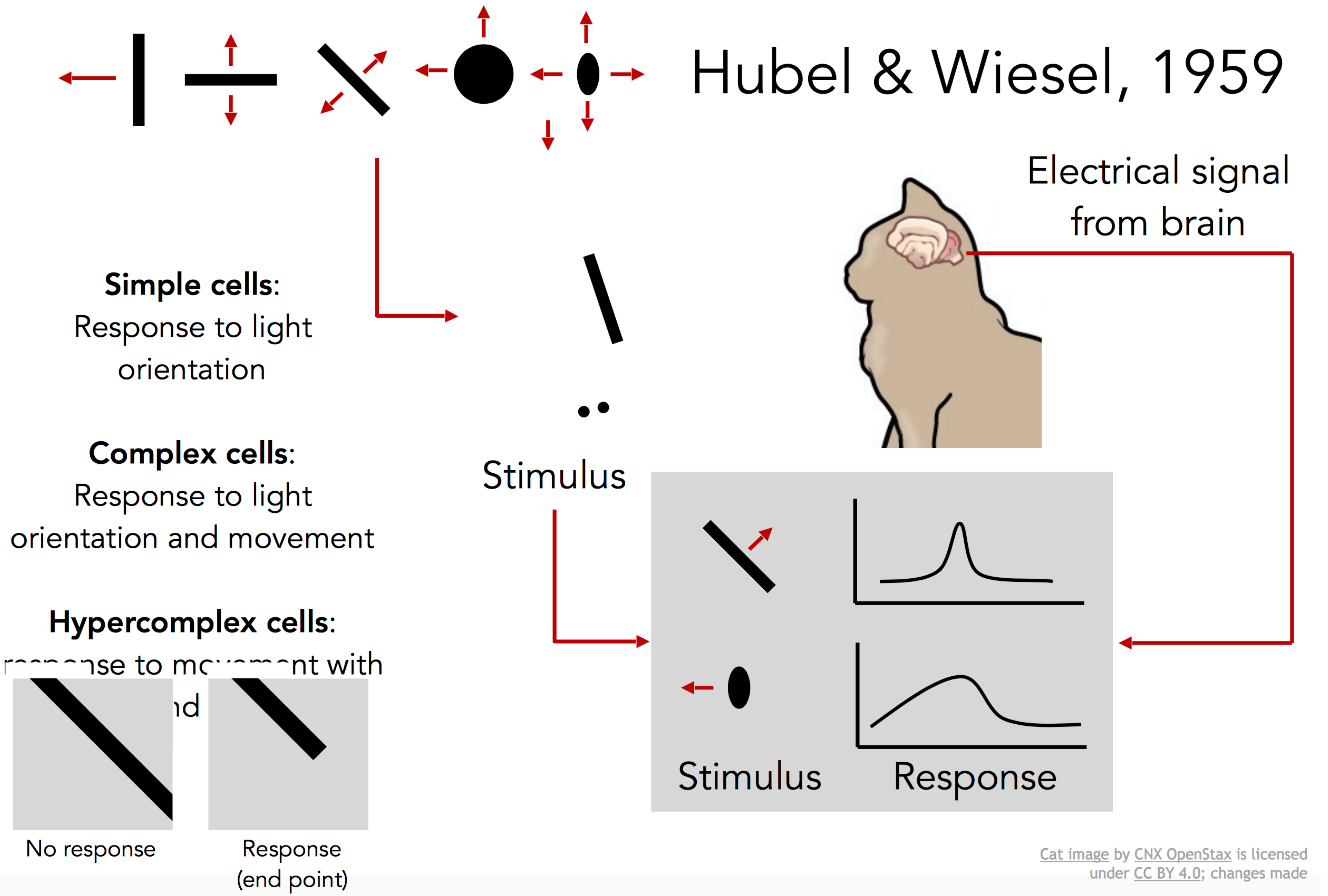
Pretty much garbage
Simple template matching is not going to make it

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

# From Template Matching to **Local Feature Detection**

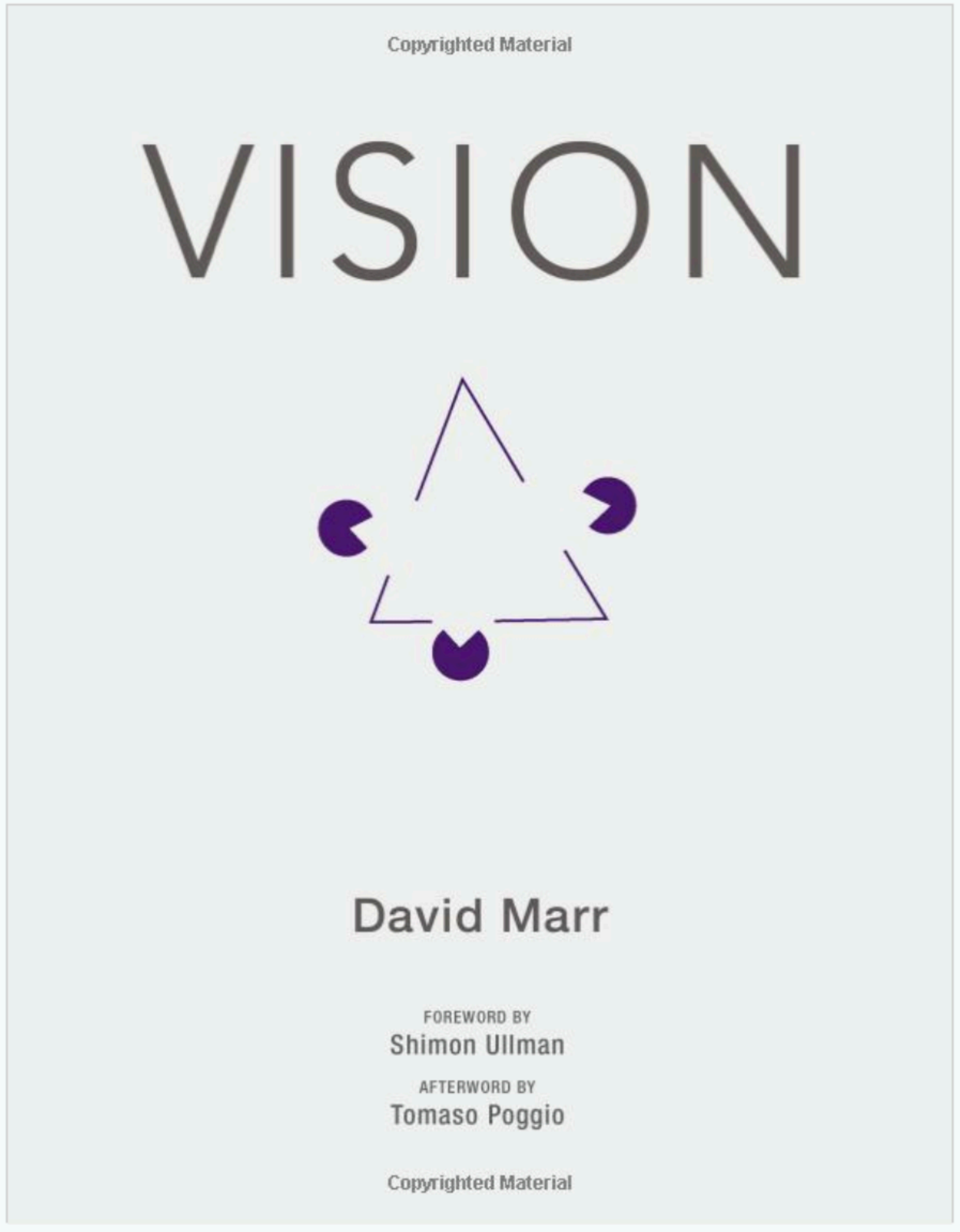
- Move from global template matching to **local template matching**
- Local template matching also called local **feature detection**
- Obvious local features to detect are **edges** and **corners**

# **Human** vision …



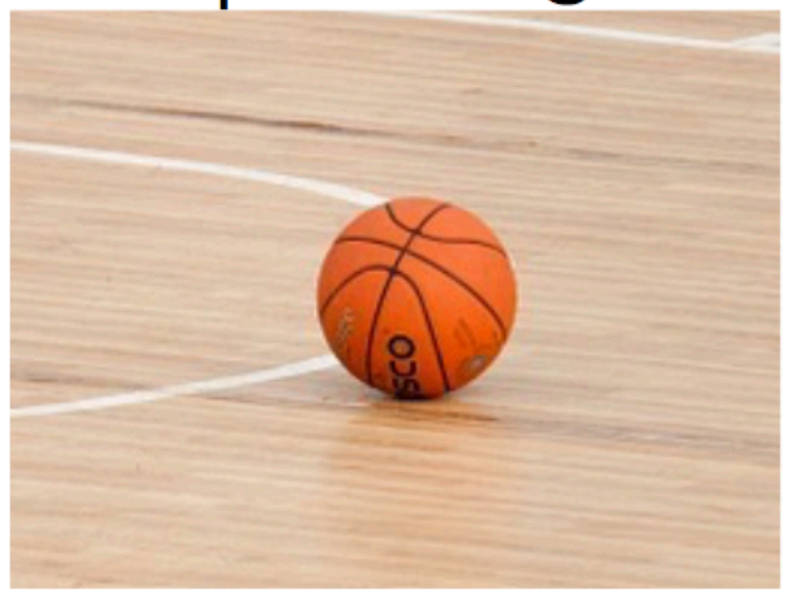

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

# David **Marr**, 1970s



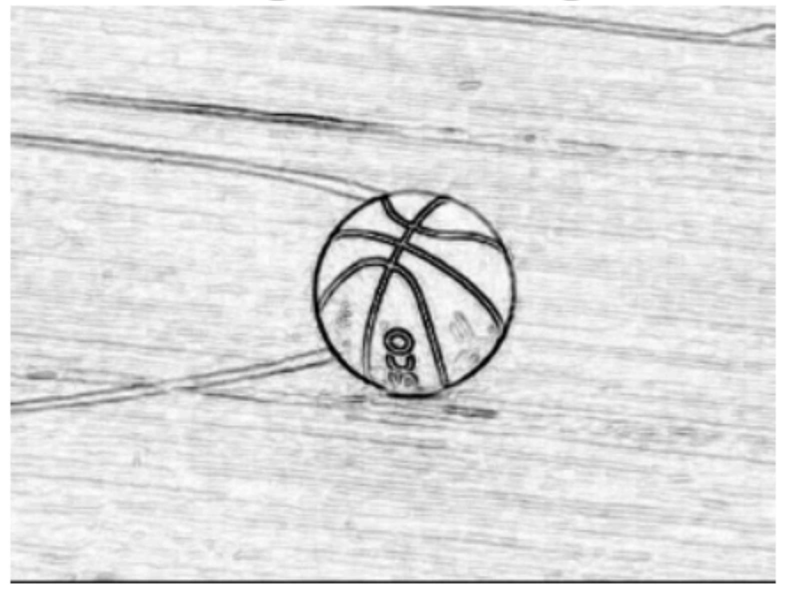
# David **Marr**, 1970s

Input image

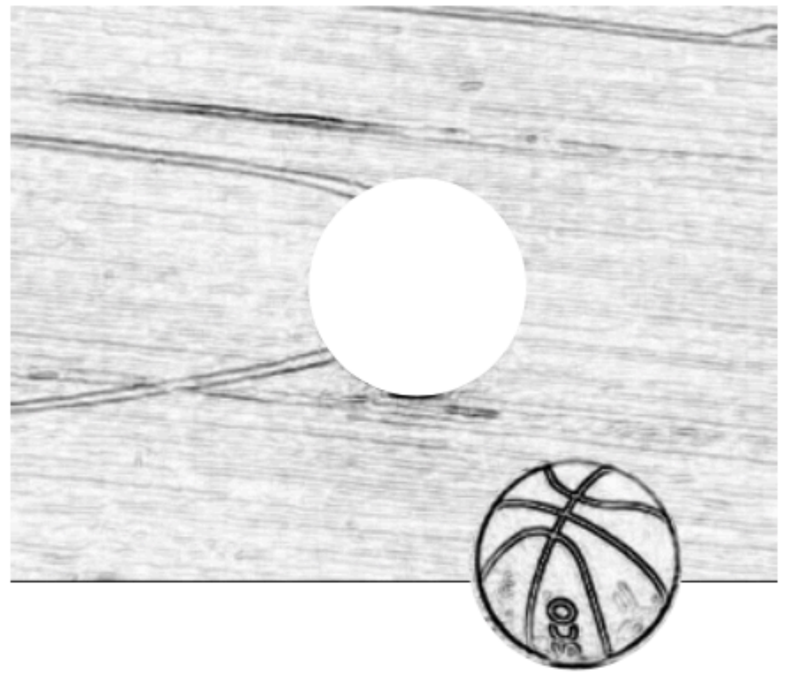


This image is CC0 1.0 public domain

Edge image



2 ½-D sketch



3-D model

This image is CC0 1.0 public domain

| Input Image | → | Primal Sketch | → | 2 ½-D Sketch | → | 3-D Model Representation |
|---|---|---|---|---|---|---|
| Perceived intensities | | Zero crossings, blobs, edges, bars, ends, virtual lines, groups, curves boundaries | | Local surface orientation and discontinuities in depth and in surface orientation | | 3-D models hierarchically organized in terms of surface and volumetric primitives |

[ Stages of Visual Representation, **David Marr** ]

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

# From Template Matching to **Local Feature Detection**

- Move from global template matching to **local template matching**
- Local template matching also called local **feature detection**
- Obvious local features to detect are **edges** and **corners**

# Estimating **Derivatives**

Recall, for a 2D (continuous) function, f(x,y)

$$\frac{\partial f}{\partial x} = \lim_{\epsilon \to 0} \frac{f(x+\epsilon, y) - f(x,y)}{\epsilon}$$

Differentiation is linear and shift invariant, and therefore can be implemented as a convolution

# Estimating **Derivatives**

Recall, for a 2D (continuous) function, f(x,y)

$$\frac{\partial f}{\partial x} = \lim_{\epsilon \to 0} \frac{f(x+\epsilon, y) - f(x,y)}{\epsilon}$$

Differentiation is linear and shift invariant, and therefore can be implemented as a convolution

A (discrete) approximation is

$$\frac{\partial f}{\partial x} \approx \frac{F(X+1, y) - F(x,y)}{\Delta x}$$

# Estimating **Derivatives**

Recall, for a 2D (continuous) function, f(x,y)

$$\frac{\partial f}{\partial x} = \lim_{\epsilon \to 0} \frac{f(x+\epsilon, y) - f(x,y)}{\epsilon}$$

Differentiation is linear and shift invariant, and therefore can be implemented as a convolution

A (discrete) approximation is

$$\frac{\partial f}{\partial x} \approx \frac{F(X+1, y) - F(x,y)}{\Delta x}$$

| | |
|---|---|
| $-1$ | $1$ |

# Estimating **Derivatives**

A (**discrete**) approximation is

$$\frac{\partial f}{\partial x} \approx \frac{F(X+1, y) - F(x, y)}{\Delta x}$$

"**forward** difference" implemented as

correlation

$$\begin{array}{|c|c|}\hline -1 & 1 \\ \hline \end{array}$$

from **left**

convolution

$$\begin{array}{|c|c|}\hline 1 & -1 \\ \hline \end{array}$$

"**backward** difference" implemented as

correlation

$$\begin{array}{|c|c|}\hline -1 & 1 \\ \hline \end{array}$$

from **right**

convolution

$$\begin{array}{|c|c|}\hline 1 & -1 \\ \hline \end{array}$$

# Estimating **Derivatives**

A (**discrete**) approximation is

$$\frac{\partial f}{\partial x} \approx \frac{F(X+1,y) - F(x,y)}{\Delta x}$$

"**forward** difference" implemented as

correlation

| | |
|---|---|
| $-1$ | $1$ |

from **left**

convolution

| | |
|---|---|
| $1$ | $-1$ |

"**backward** difference" implemented as

correlation

| | |
|---|---|
| $-1$ | $1$ |

from **right**

convolution

| | |
|---|---|
| $1$ | $-1$ |

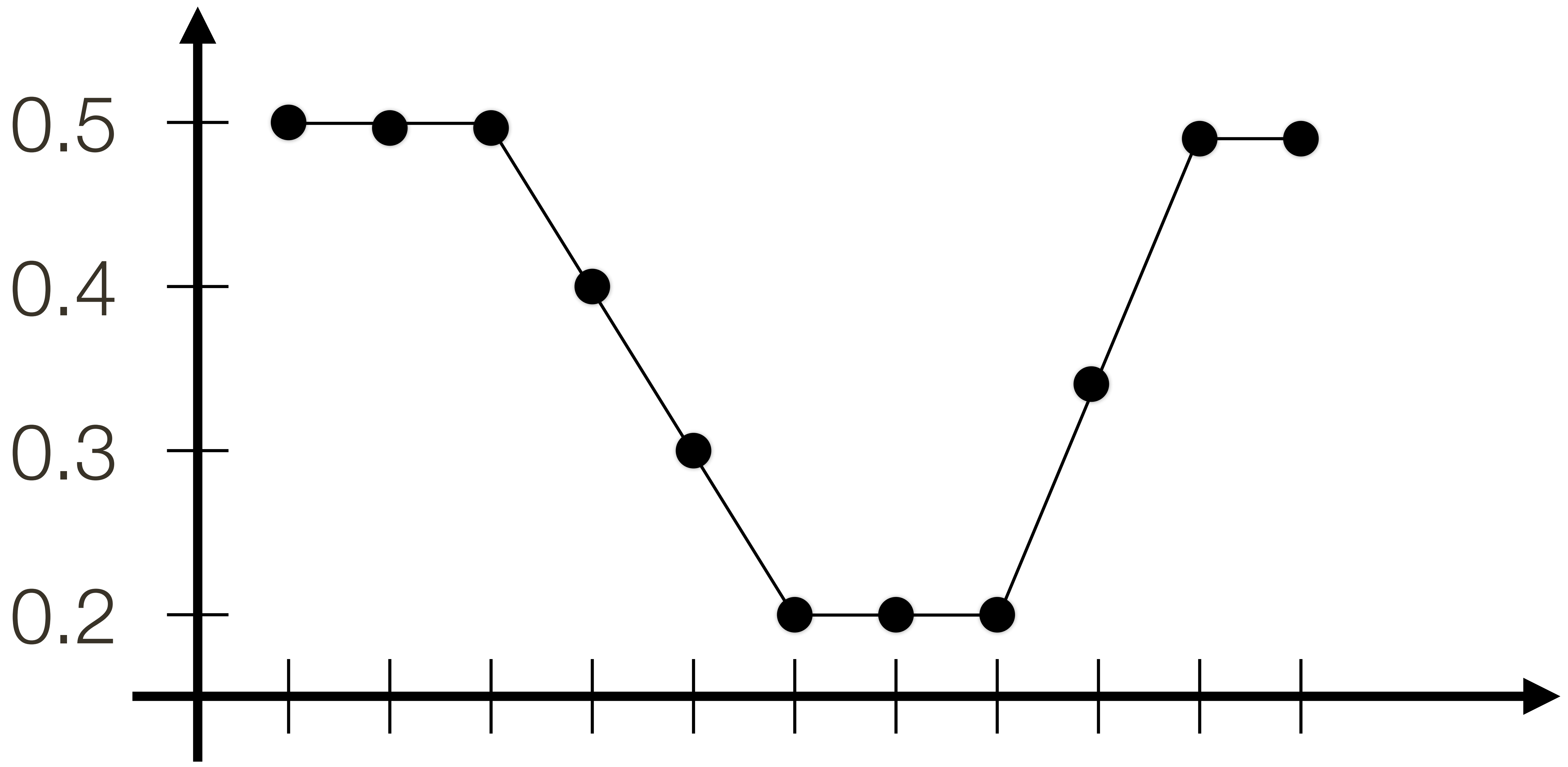
# Estimating **Derivatives**

A similar definition (and approximation) holds for $\frac{\partial f}{\partial y}$

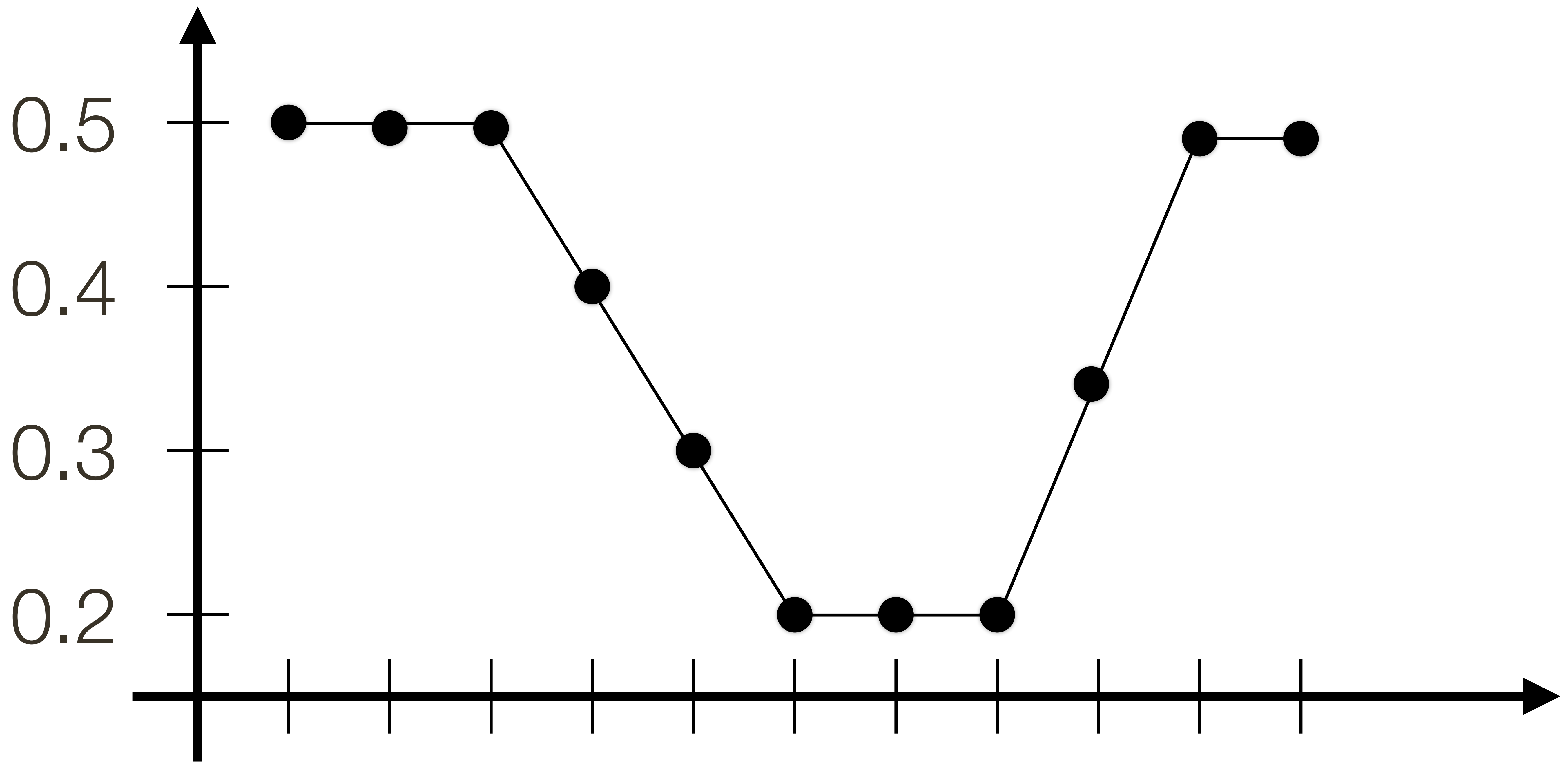
Image **noise** tends to result in pixels not looking exactly like their neighbours, so simple “finite differences” are sensitive to noise.

The usual way to deal with this problem is to **smooth** the image prior to derivative estimation.

# **Example** 1D

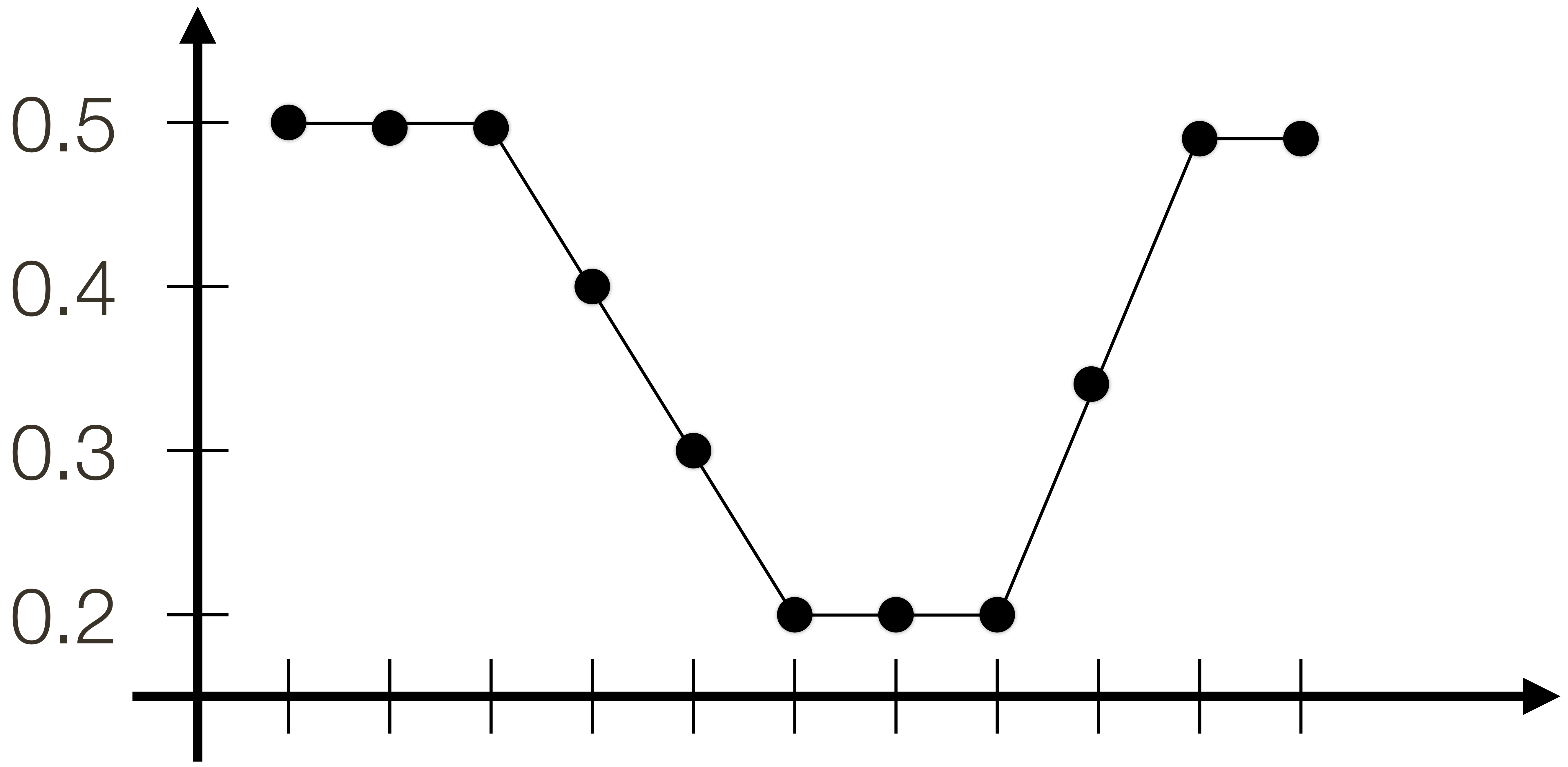


# **Example** 1D



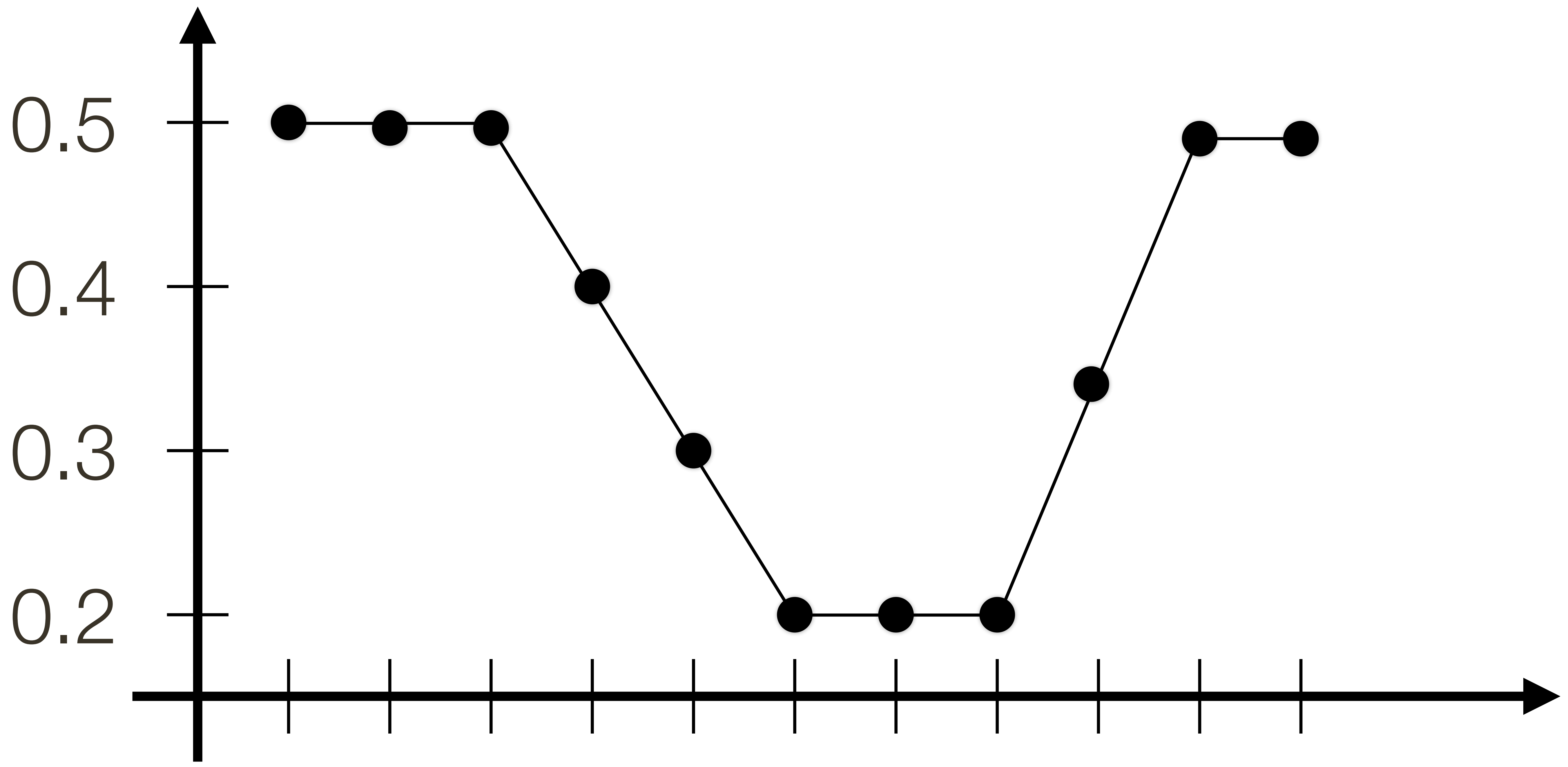

| **Signal** | 0.5 | 0.5 | 0.5 | 0.4 | 0.3 | 0.2 | 0.2 | 0.2 | 0.35 | 0.5 | 0.5 |
|---|---|---|---|---|---|---|---|---|---|---|---|

# **Example** 1D



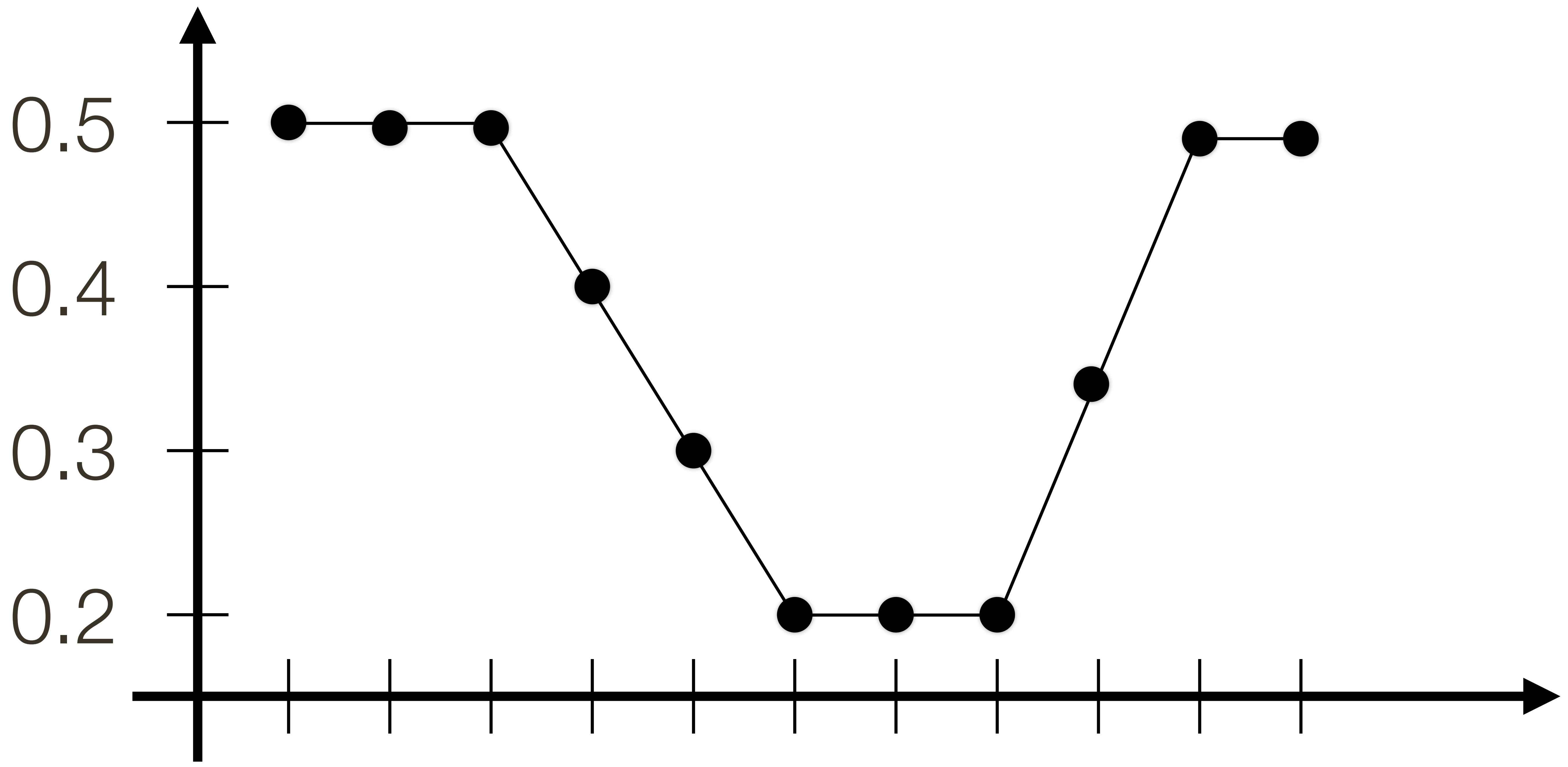

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Signal** | 0.5 | 0.5 | 0.5 | 0.4 | 0.3 | 0.2 | 0.2 | 0.2 | 0.35 | 0.5 | 0.5 |
| **Derivative** | | | | | | | | | | | |

# **Example** 1D



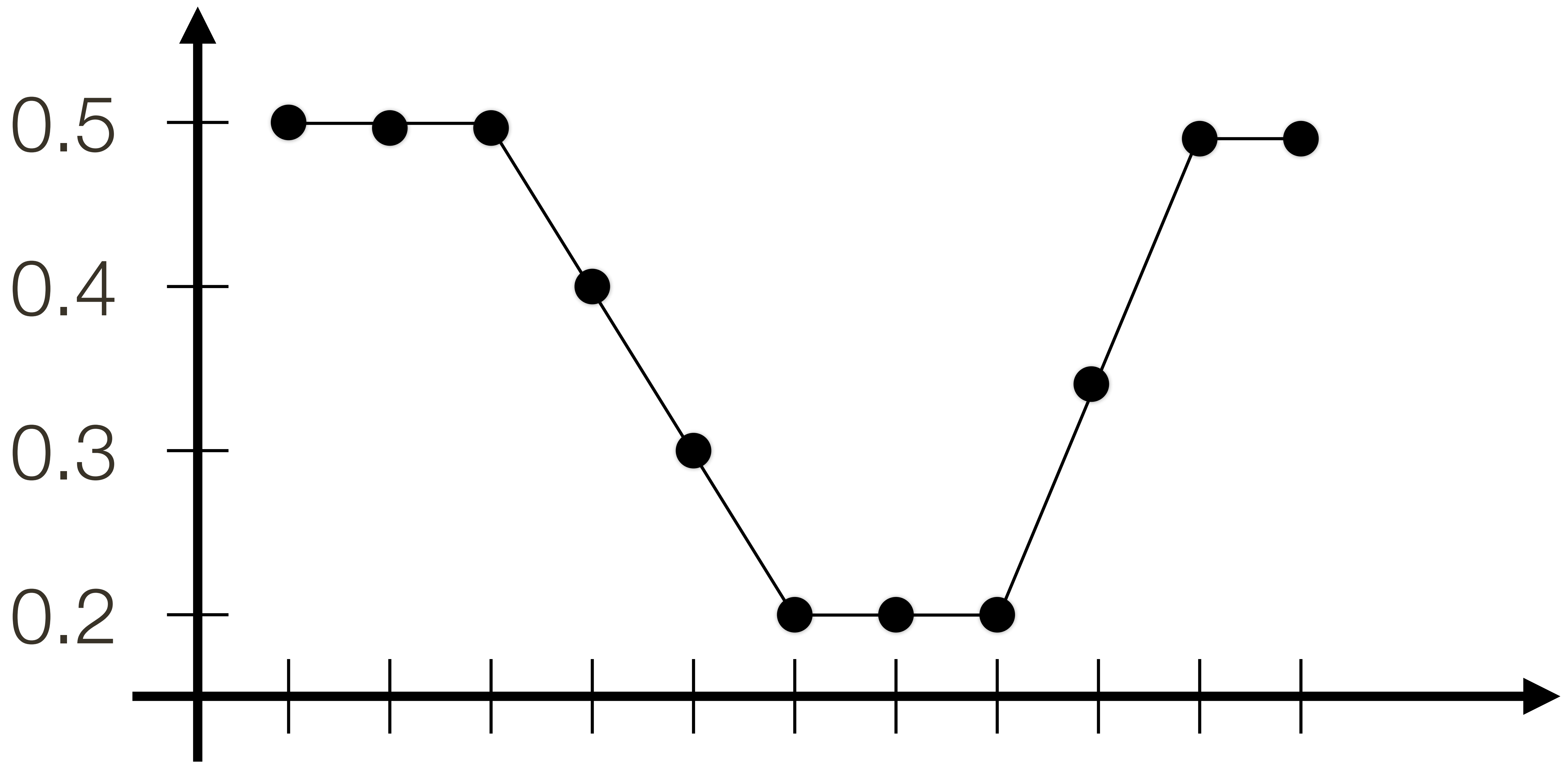

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Signal** | 0.5 | 0.5 | 0.5 | 0.4 | 0.3 | 0.2 | 0.2 | 0.2 | 0.35 | 0.5 | 0.5 |
| **Derivative** | 0.0 | | | | | | | | | | |

# **Example** 1D



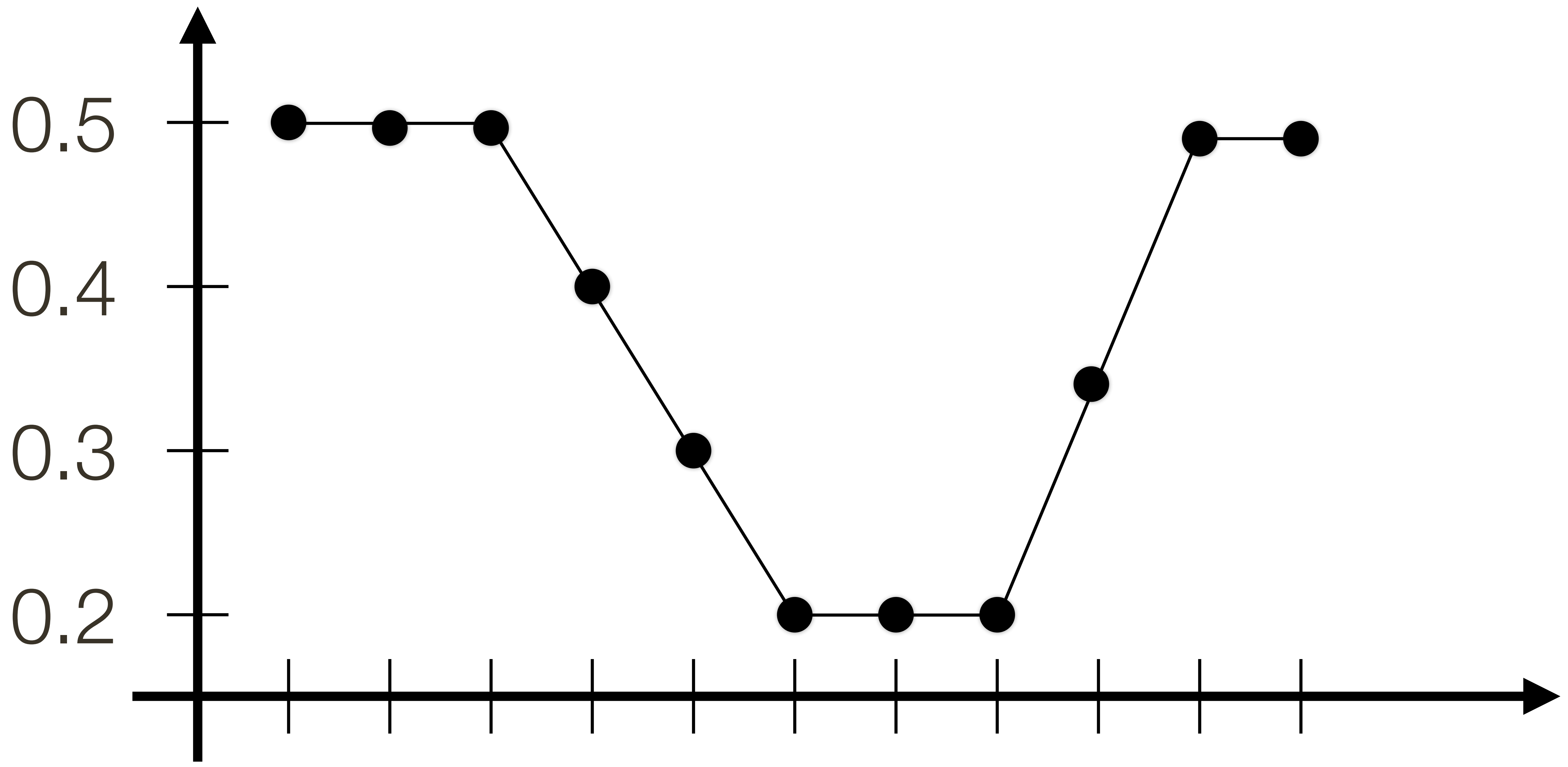

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Signal** | 0.5 | 0.5 | 0.5 | 0.4 | 0.3 | 0.2 | 0.2 | 0.2 | 0.35 | 0.5 | 0.5 |
| **Derivative** | 0.0 | | | | | | | | | | |

# **Example** 1D



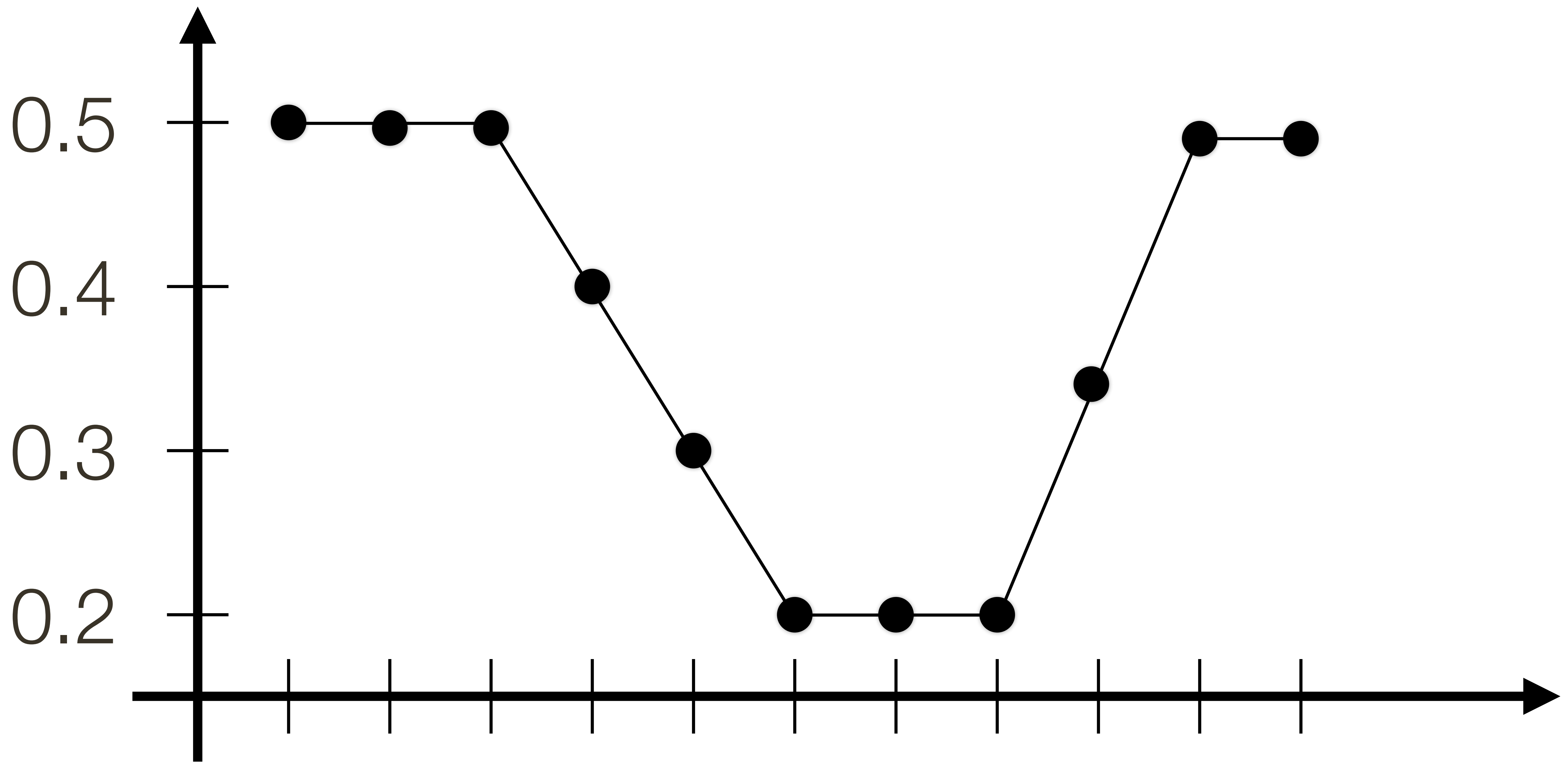

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Signal** | 0.5 | 0.5 | 0.5 | 0.4 | 0.3 | 0.2 | 0.2 | 0.2 | 0.35 | 0.5 | 0.5 |
| **Derivative** | 0.0 | 0.0 | | | | | | | | | |

# **Example** 1D



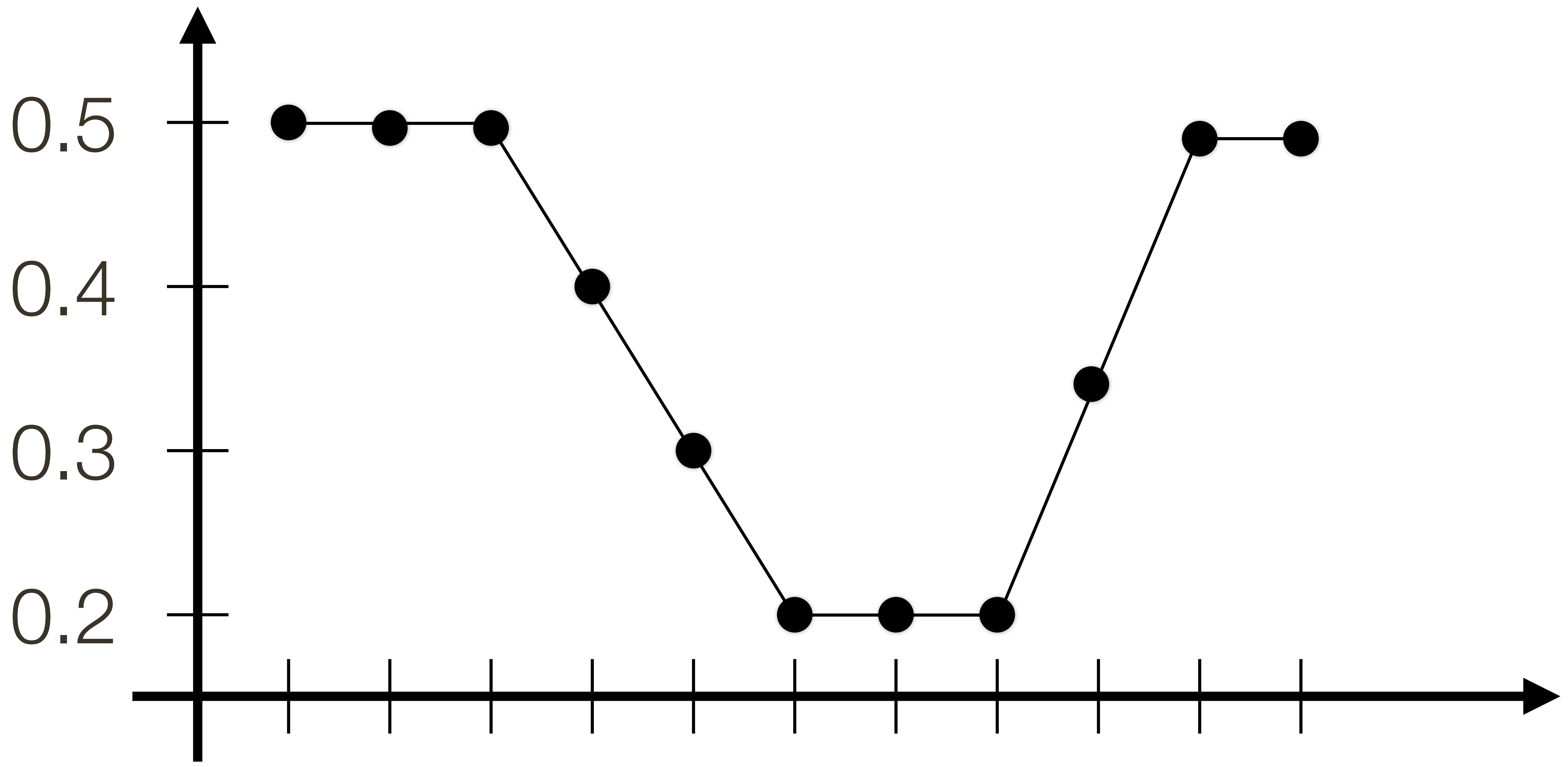

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Signal** | 0.5 | 0.5 | 0.5 | 0.4 | 0.3 | 0.2 | 0.2 | 0.2 | 0.35 | 0.5 | 0.5 |
| **Derivative** | 0.0 | 0.0 | | | | | | | | | |

# **Example** 1D




| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Signal** | 0.5 | 0.5 | 0.5 | 0.4 | 0.3 | 0.2 | 0.2 | 0.2 | 0.35 | 0.5 | 0.5 |
| **Derivative** | 0.0 | 0.0 | -0.1 | | | | | | | | |

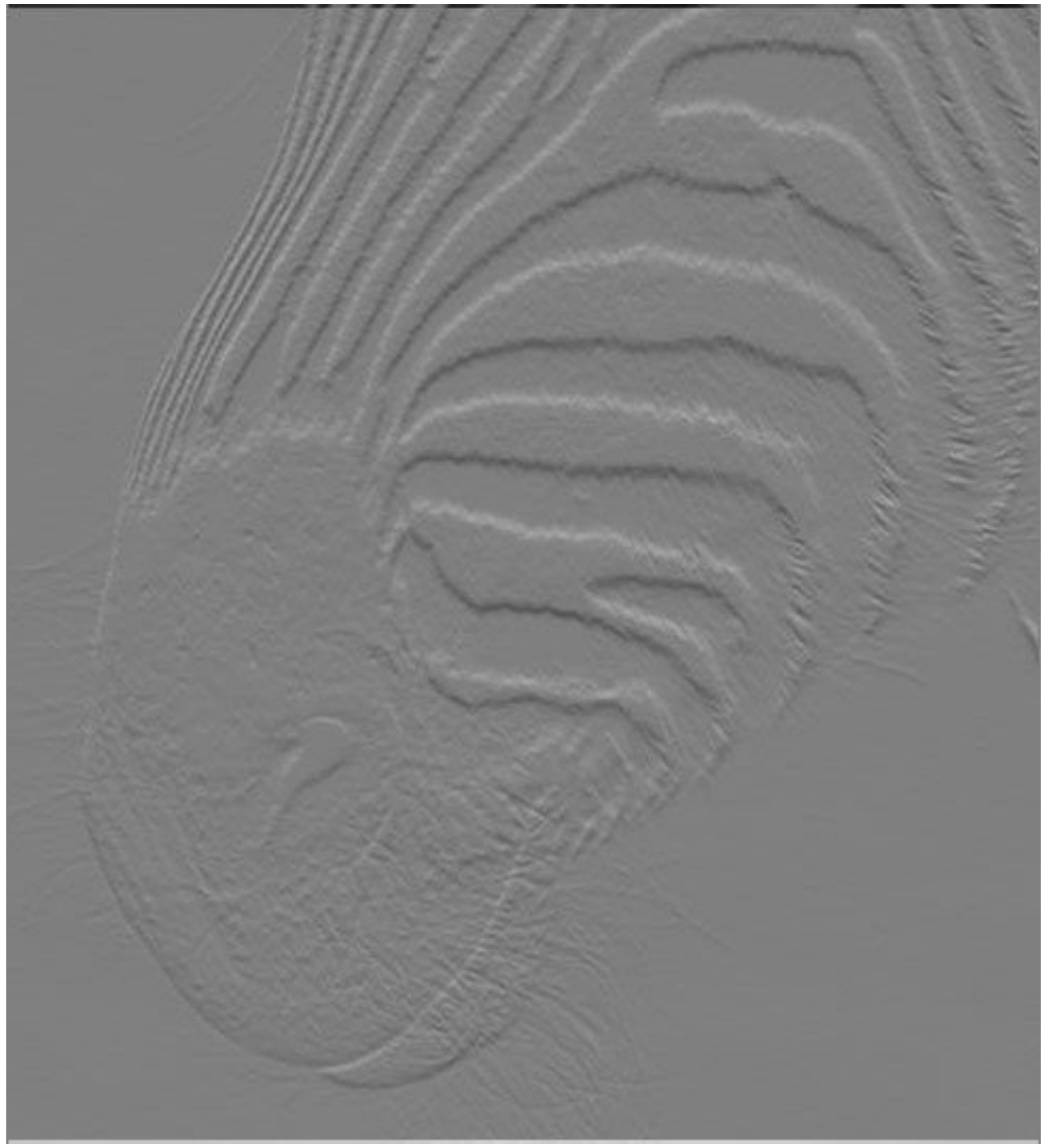
# **Example** 1D




| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Signal** | 0.5 | 0.5 | 0.5 | 0.4 | 0.3 | 0.2 | 0.2 | 0.2 | 0.35 | 0.5 | 0.5 |
| **Derivative** | 0.0 | 0.0 | -0.1 | -0.1 | -0.1 | 0.0 | 0.0 | 0.15 | 0.15 | 0.0 | X |

# Estimating **Derivatives**

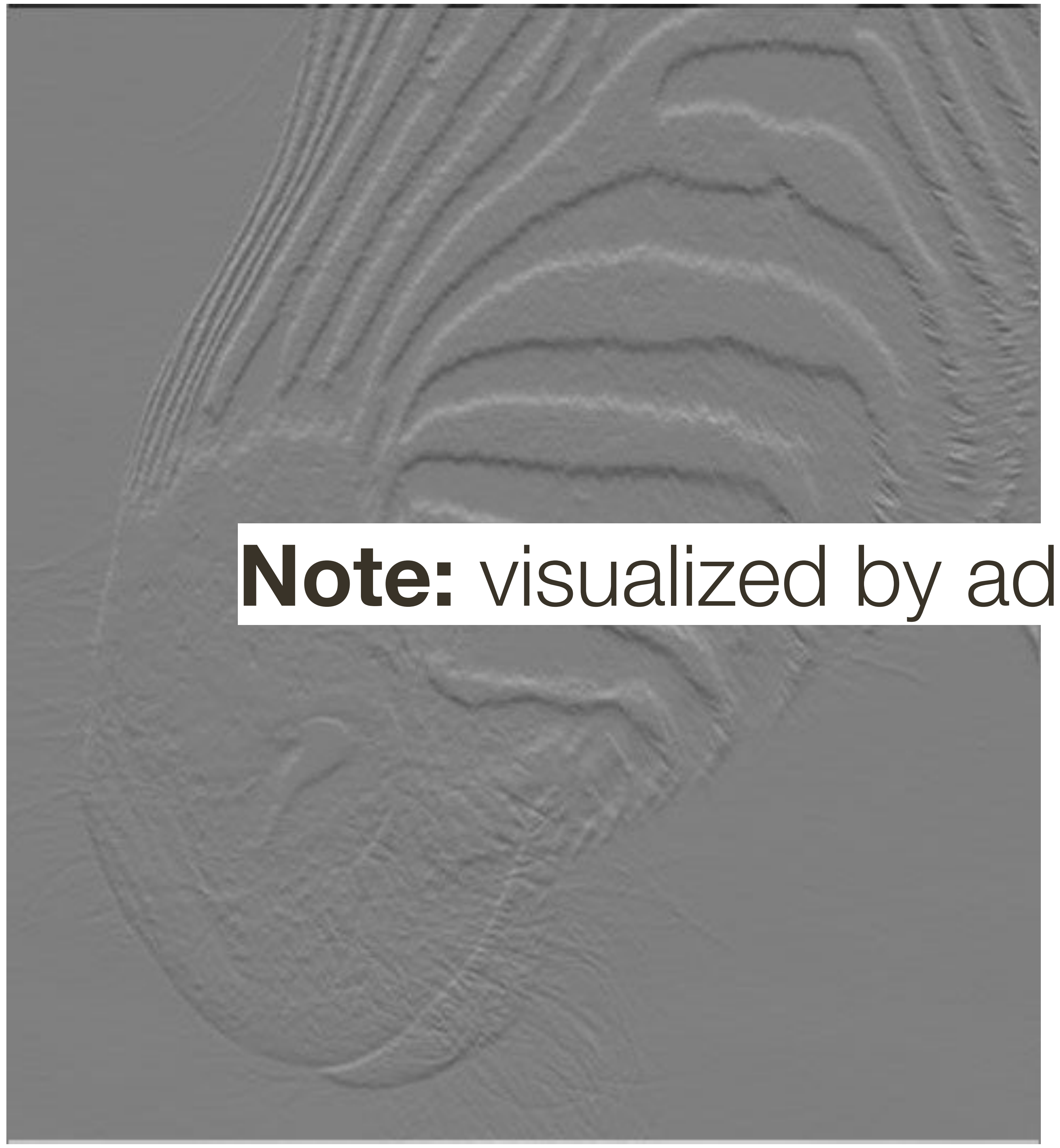
**Derivative** in Y (i.e., vertical) direction



Forsyth & Ponce (1st ed.) Figure 7.4 (top left & top middle)

# Estimating **Derivatives**

**Derivative** in Y (i.e., vertical) direction

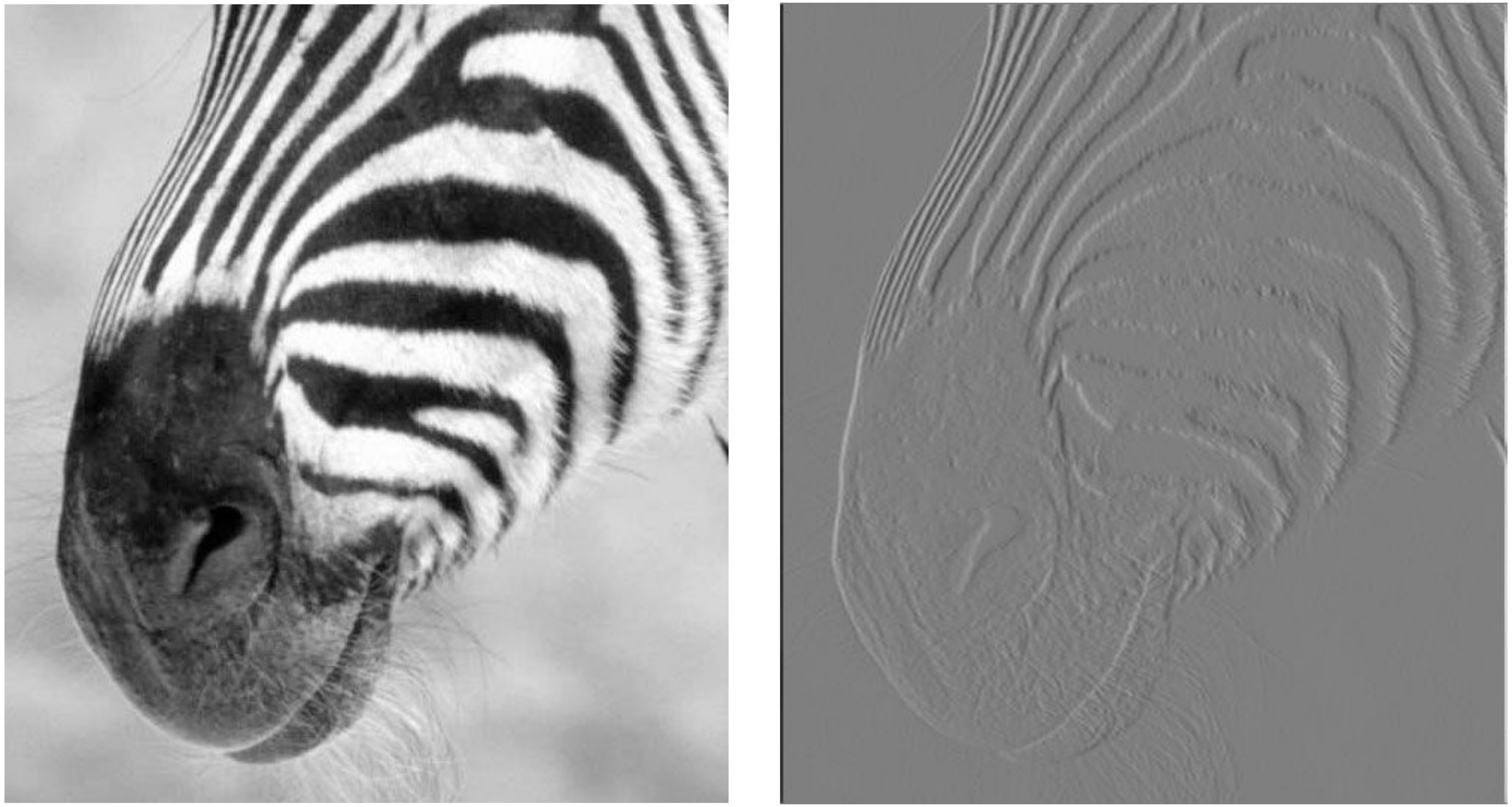


**Note:** visualized by adding 0.5/128

Forsyth & Ponce (1st ed.) Figure 7.4 (top left & top middle)

# Estimating **Derivatives**

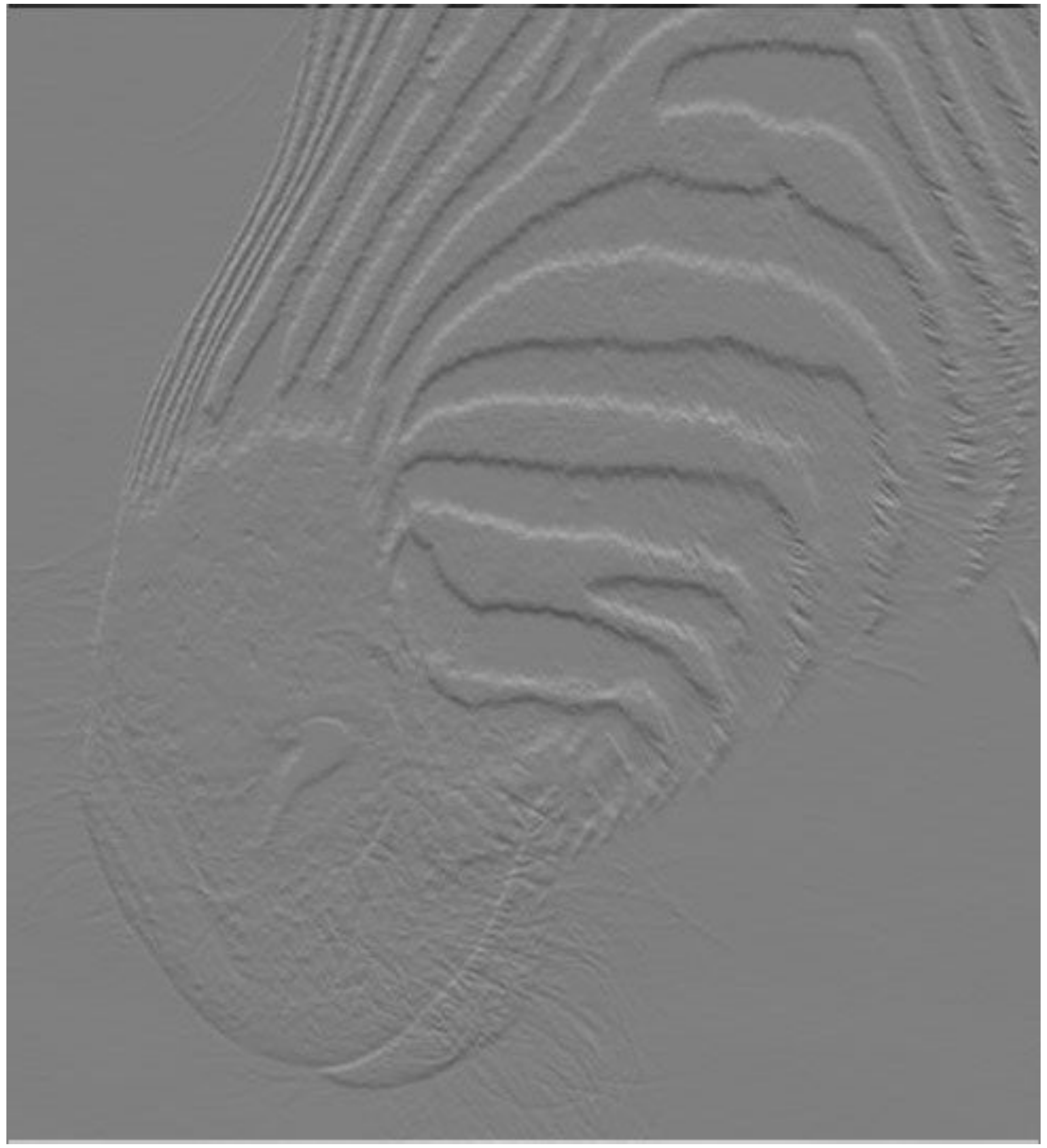
**Derivative** in X (i.e., horizontal) direction



Forsyth & Ponce (1st ed.) Figure 7.4 (top left & top right)

# Estimating **Derivatives**

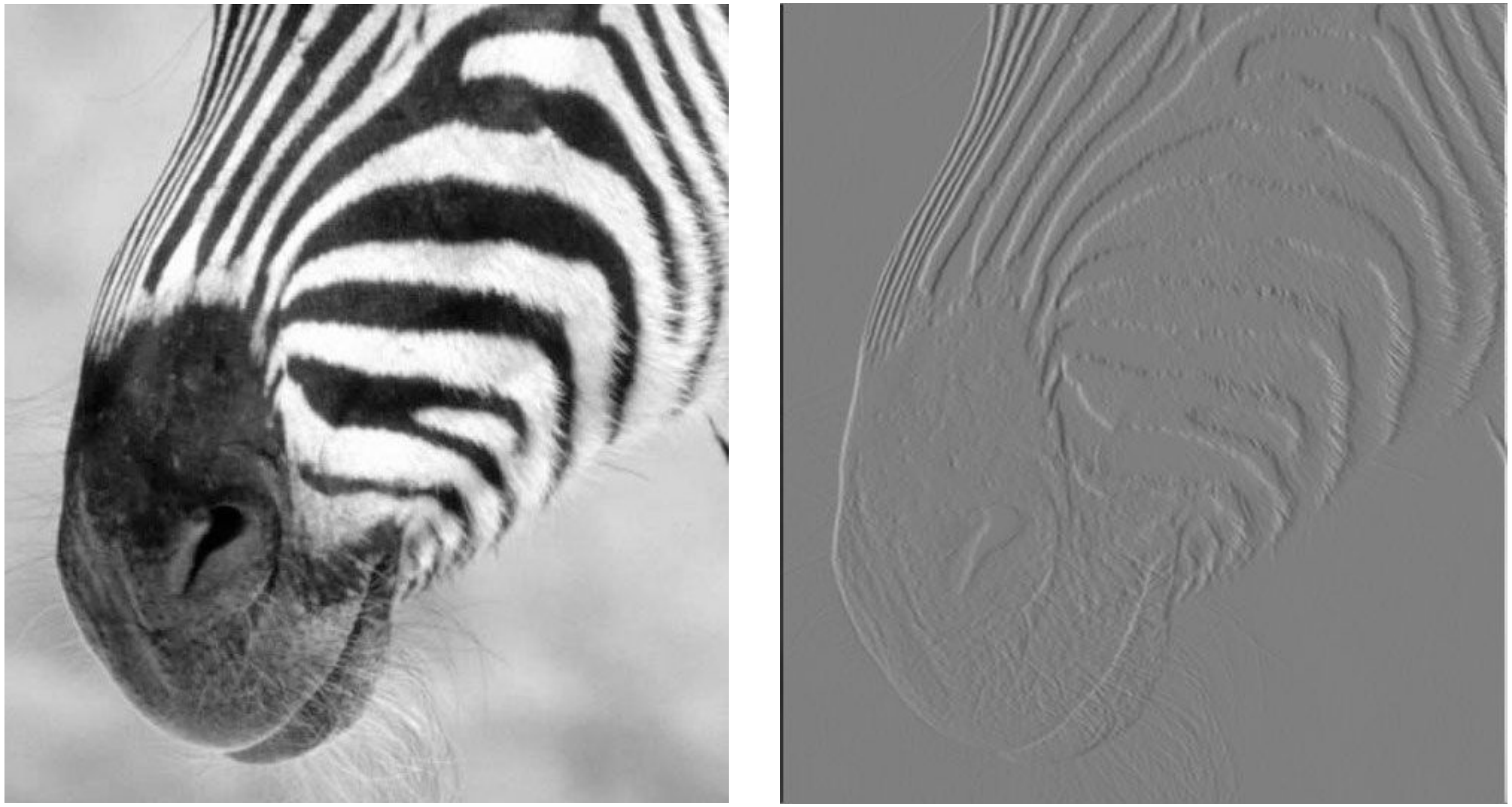
**Derivative** in Y (i.e., vertical) direction



Forsyth & Ponce (1st ed.) Figure 7.4 (top left & top middle)

# Estimating **Derivatives**

**Derivative** in X (i.e., horizontal) direction



Forsyth & Ponce (1st ed.) Figure 7.4 (top left & top right)

# A Sort **Exercise**

Use the “first forward difference" to compute the image derivatives in X and Y directions.

(Compute two arrays, one of $\dfrac{\partial f}{\partial x}$ values and one of $\dfrac{\partial f}{\partial y}$ values.)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 0.6 | 0.3 | 0 | 0 |
| 1 | 1 | 0.6 | 0.3 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

# A Sort **Exercise**: Derivative in X Direction

Use the “first forward difference" to compute the image derivatives in X and Y directions.

(Compute two arrays, one of $\frac{\partial f}{\partial x}$ values and one of $\frac{\partial f}{\partial y}$ values.)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 0.6 | 0.3 | 0 | 0 |
| 1 | 1 | 0.6 | 0.3 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

# A Sort **Exercise**: Derivative in X Direction

Use the “first forward difference" to compute the image derivatives in X and Y directions.

(Compute two arrays, one of $\frac{\partial f}{\partial x}$ values and one of $\frac{\partial f}{\partial y}$ values.)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 0.6 | 0.3 | 0 | 0 |
| 1 | 1 | 0.6 | 0.3 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| 0 | | | | | |

# A Sort **Exercise**: Derivative in X Direction

Use the "first forward difference" to compute the image derivatives in X and Y directions.

(Compute two arrays, one of $\frac{\partial f}{\partial x}$ values and one of $\frac{\partial f}{\partial y}$ values.)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 0.6 | 0.3 | 0 | 0 |
| 1 | 1 | 0.6 | 0.3 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | $\boxed{0}$ | $\boxed{0}$ | 0 | 0 | 0 |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| 0 | | | | | |

# A Sort **Exercise**: Derivative in X Direction

Use the “first forward difference" to compute the image derivatives in X and Y directions.

(Compute two arrays, one of $\dfrac{\partial f}{\partial x}$ values and one of $\dfrac{\partial f}{\partial y}$ values.)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 0.6 | 0.3 | 0 | 0 |
| 1 | 1 | 0.6 | 0.3 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | $\boxed{0}$ | $\boxed{0}$ | 0 | 0 | 0 |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| 0 | 0 | | | | |

# A Sort **Exercise**: Derivative in X Direction

Use the “first forward difference" to compute the image derivatives in X and Y directions.

(Compute two arrays, one of $\dfrac{\partial f}{\partial x}$ values and one of $\dfrac{\partial f}{\partial y}$ values.)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 0.6 | 0.3 | 0 | 0 |
| 1 | 1 | 0.6 | 0.3 | 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 | |

# A Sort **Exercise**: Derivative in X Direction

Use the “first forward difference" to compute the image derivatives in X and Y directions.

(Compute two arrays, one of $\frac{\partial f}{\partial x}$ values and one of $\frac{\partial f}{\partial y}$ values.)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 0.6 | 0.3 | 0 | 0 |
| 1 | $\boxed{1}$ | $\boxed{0.6}$ | 0.3 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

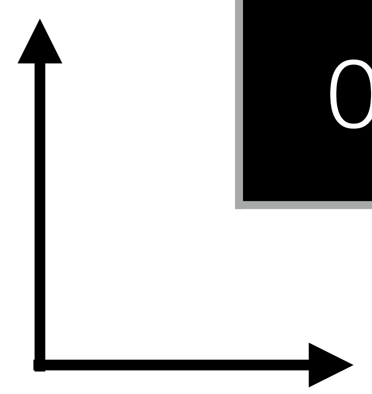
| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| 0 | -0.4 | | | | |
| 0 | 0 | 0 | 0 | 0 | |
| 0 | 0 | 0 | 0 | 0 | |

# A Sort **Exercise**: Derivative in X Direction

Use the “first forward difference" to compute the image derivatives in X and Y directions.

(Compute two arrays, one of $\frac{\partial f}{\partial x}$ values and one of $\frac{\partial f}{\partial y}$ values.)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 0.6 | 0.3 | 0 | 0 |
| 1 | 1 | 0.6 | 0.3 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |



| | | | | | |
|---|---|---|---|---|---|
| 0 | -0.4 | -0.3 | -0.3 | 0 | |
| 0 | -0.4 | -0.3 | -0.3 | 0 | |
| 0 | 0 | 0 | 0 | 0 | |
| 0 | 0 | 0 | 0 | 0 | |

# A Sort **Exercise**: Derivative in Y Direction

Use the “first forward difference" to compute the image derivatives in X and Y directions.

(Compute two arrays, one of $\frac{\partial f}{\partial x}$ values and one of $\frac{\partial f}{\partial y}$ values.)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 0.6 | 0.3 | 0 | 0 |
| 1 | 1 | 0.6 | 0.3 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

# A Sort **Exercise**: Derivative in Y Direction

Use the “first forward difference" to compute the image derivatives in X and Y directions.

(Compute two arrays, one of $\frac{\partial f}{\partial x}$ values and one of $\frac{\partial f}{\partial y}$ values.)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 0.6 | 0.3 | 0 | 0 |
| 1 | 1 | 0.6 | 0.3 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| 0 | | | | | |

# A Sort **Exercise**: Derivative in Y Direction

Use the "first forward difference" to compute the image derivatives in X and Y directions.

(Compute two arrays, one of $\frac{\partial f}{\partial x}$ values and one of $\frac{\partial f}{\partial y}$ values.)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 0.6 | 0.3 | 0 | 0 |
| 1 | 1 | 0.6 | 0.3 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

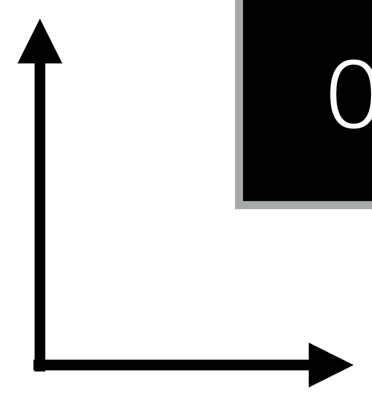
| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 |

# A Sort **Exercise**: Derivative in Y Direction

Use the “first forward difference" to compute the image derivatives in X and Y directions.

(Compute two arrays, one of $\frac{\partial f}{\partial x}$ values and one of $\frac{\partial f}{\partial y}$ values.)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 0.6 | 0.3 | 0 | 0 |
| 1 | 1 | 0.6 | 0.3 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |



| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0.6 | 0.3 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

# Estimating **Derivatives**

**Question**: Why, in general, should the weights of a filter used for differentiation sum to 0?

# Estimating **Derivatives**

**Question**: Why, in general, should the weights of a filter used for differentiation sum to 0?

**Answer**: Think of a constant image, $I(X, Y) = k$. The derivative is 0. Therefore, the weights of any filter used for differentiation need to sum to 0.

# Estimating **Derivatives**

**Question**: Why, in general, should the weights of a filter used for differentiation sum to 0?

**Answer**: Think of a constant image, $I(X, Y) = k$. The derivative is 0. Therefore, the weights of any filter used for differentiation need to sum to 0.

$$\sum_{i=1}^{N} f_i \cdot k = k \sum_{i=1}^{N} f_i = 0 \implies \sum_{i=1}^{N} f_i = 0$$

# **Edge** Detection

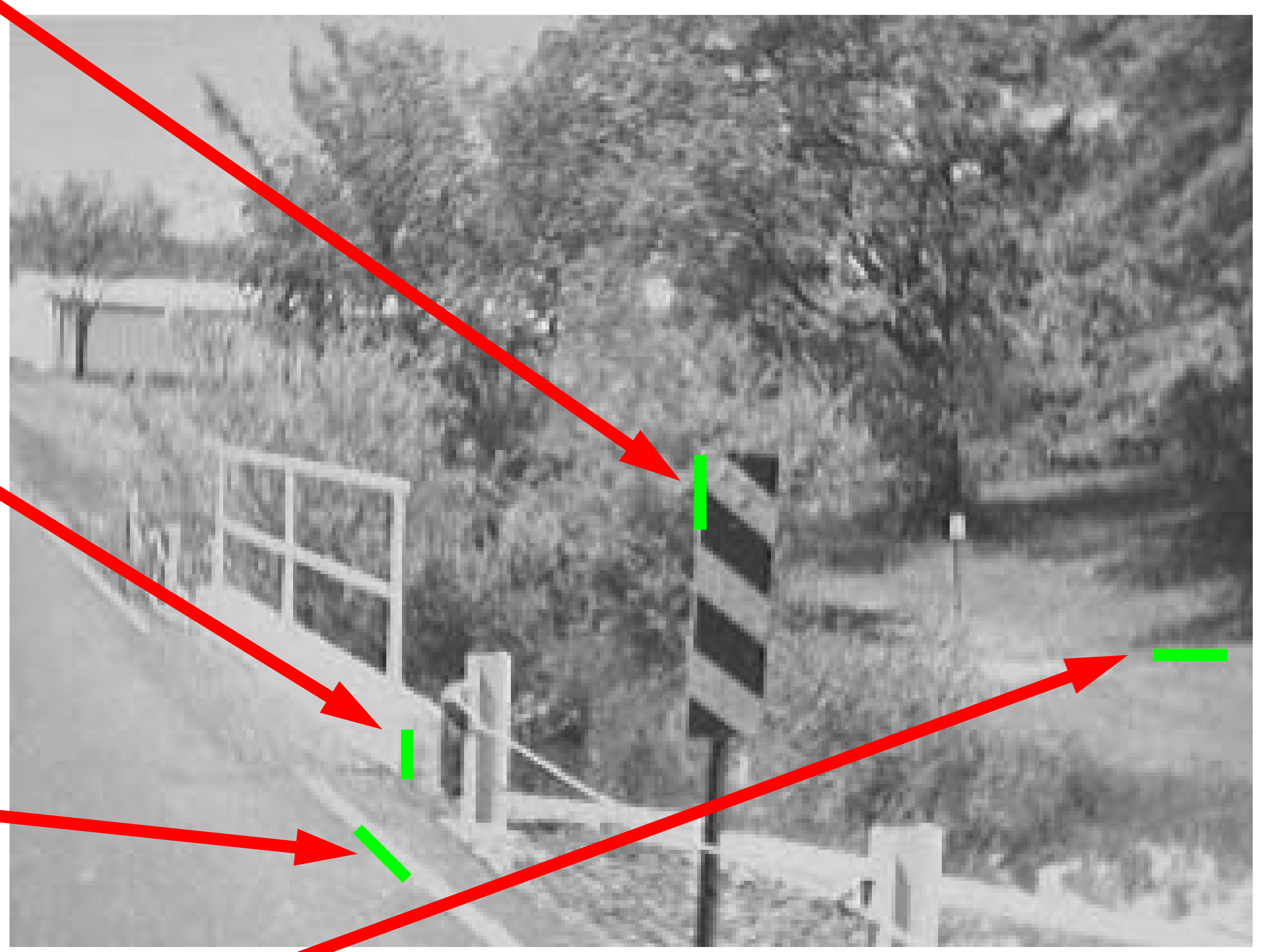
**Goal**: Identify sudden changes in image intensity

This is where most shape information is encoded

**Example**: artist's line drawing (but artist also is using object-level knowledge)

# What Causes **Edges**?

- Depth discontinuity
- Surface orientation discontinuity
- Reflectance discontinuity (i.e., change in surface material properties)
- Illumination discontinuity (e.g., shadow)



**Slide Credit**: Christopher Rasmussen

# **Smoothing** and Differentiation

**Edge**: a location with high gradient (derivative)

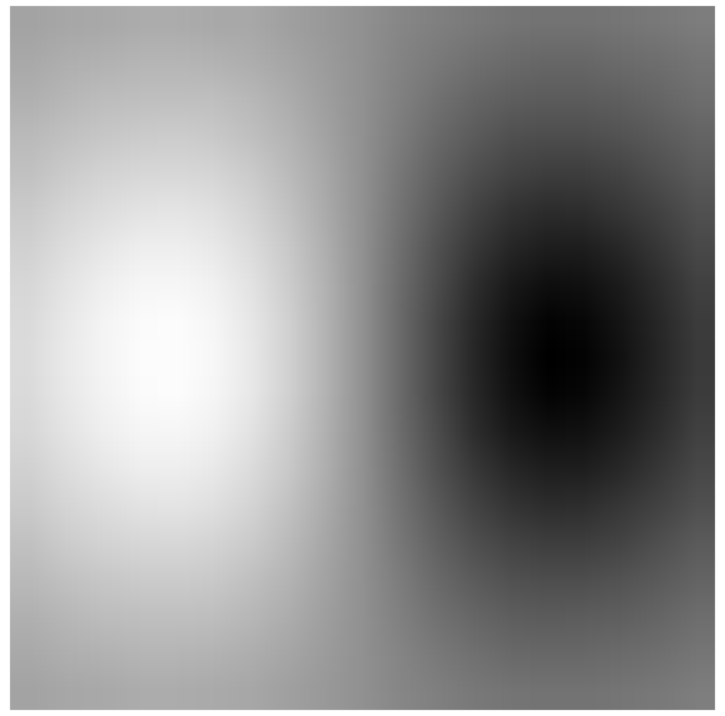
Need smoothing to reduce noise prior to taking derivative

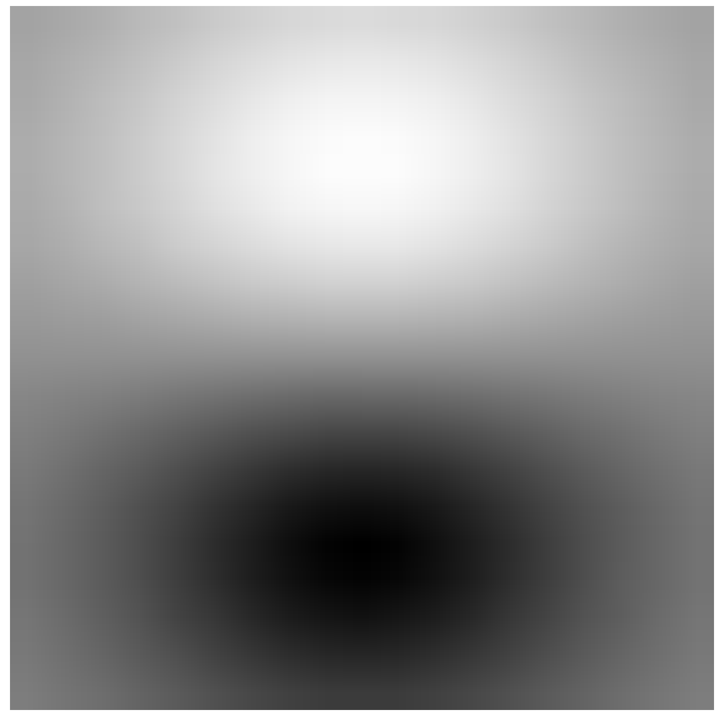
Need two derivatives, in x and y direction

We can use **derivative of Gaussian** filters
— because differentiation is convolution, and
— convolution is associative

Let $\otimes$ denote convolution

$$D \otimes (G \otimes I(X,Y)) = (D \otimes G) \otimes I(X,Y)$$

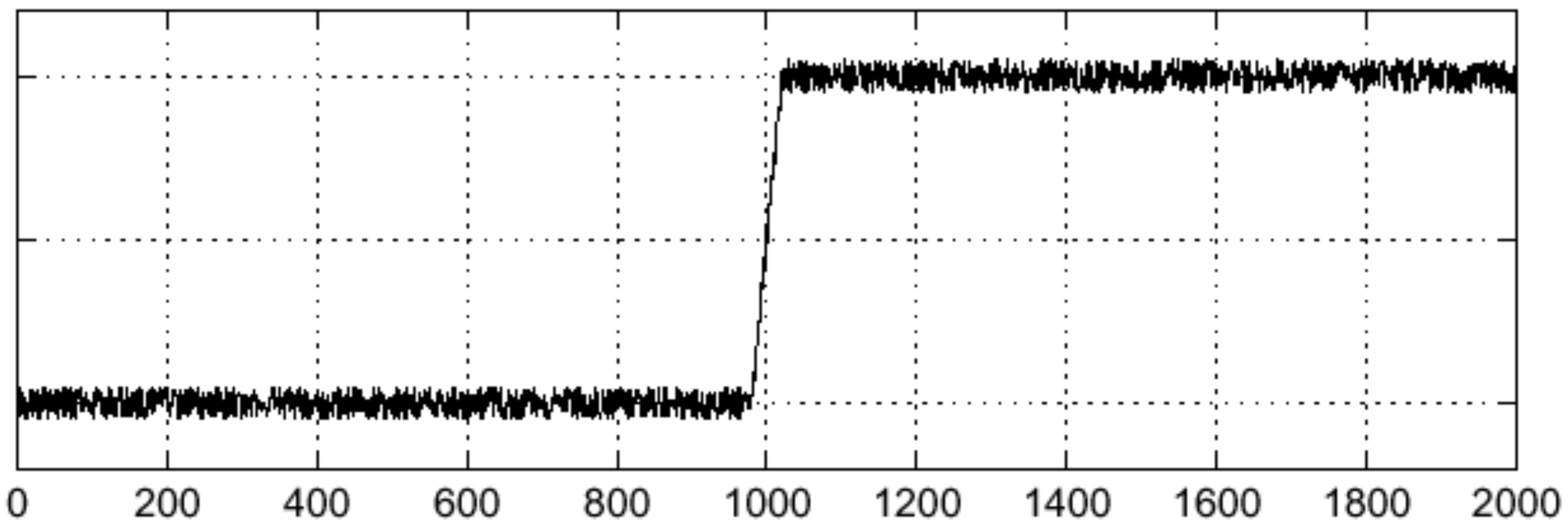




# 1D **Example**

Lets consider a row of pixels in an image:

$I(X, 245)$

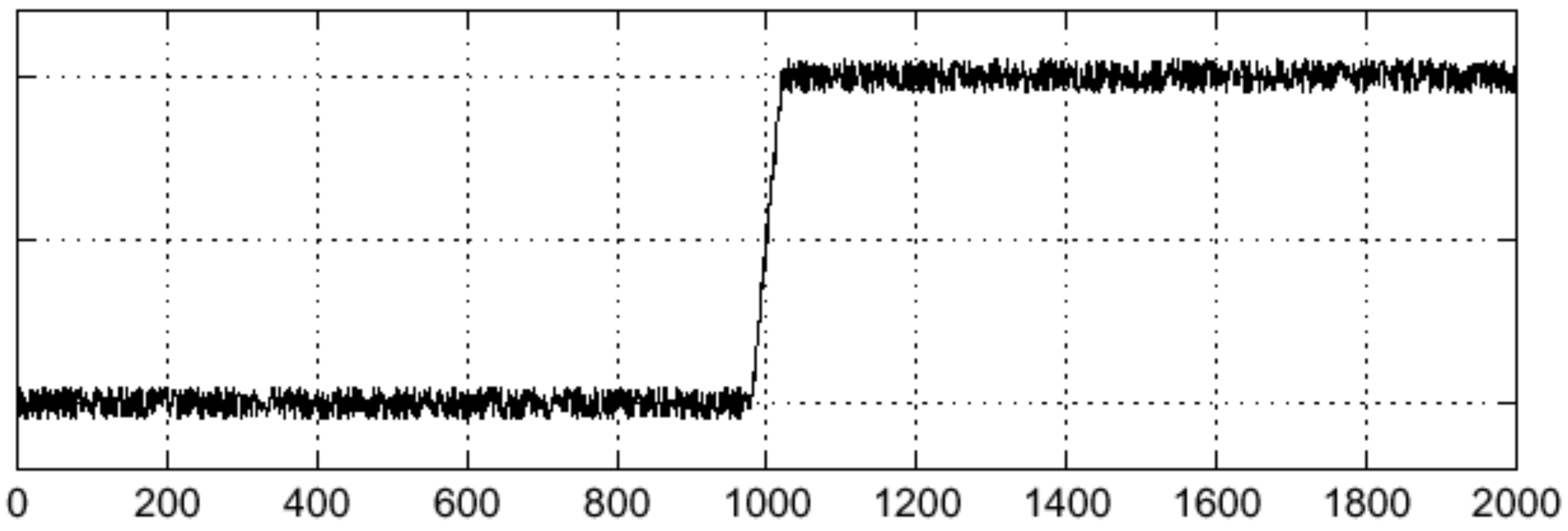



Where is the edge?

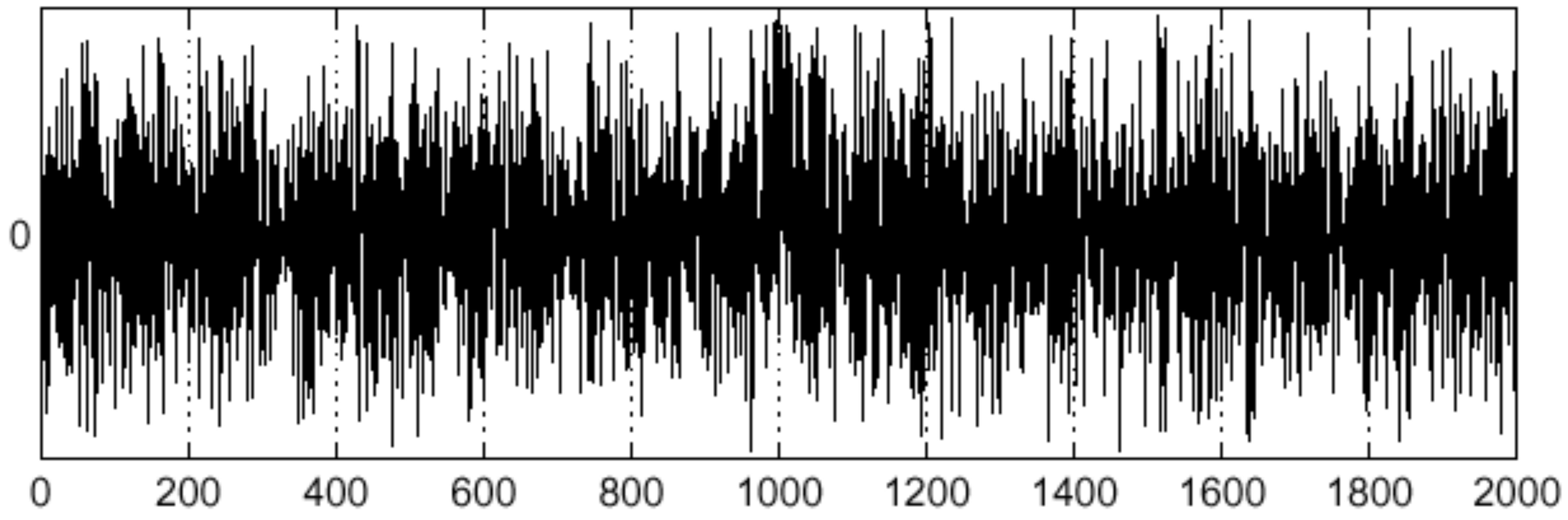
# 1D **Example**: Derivative

Lets consider a row of pixels in an image:

$I(X, 245)$




$$\frac{\partial I(X, 245)}{\partial x}$$




Where is the edge?

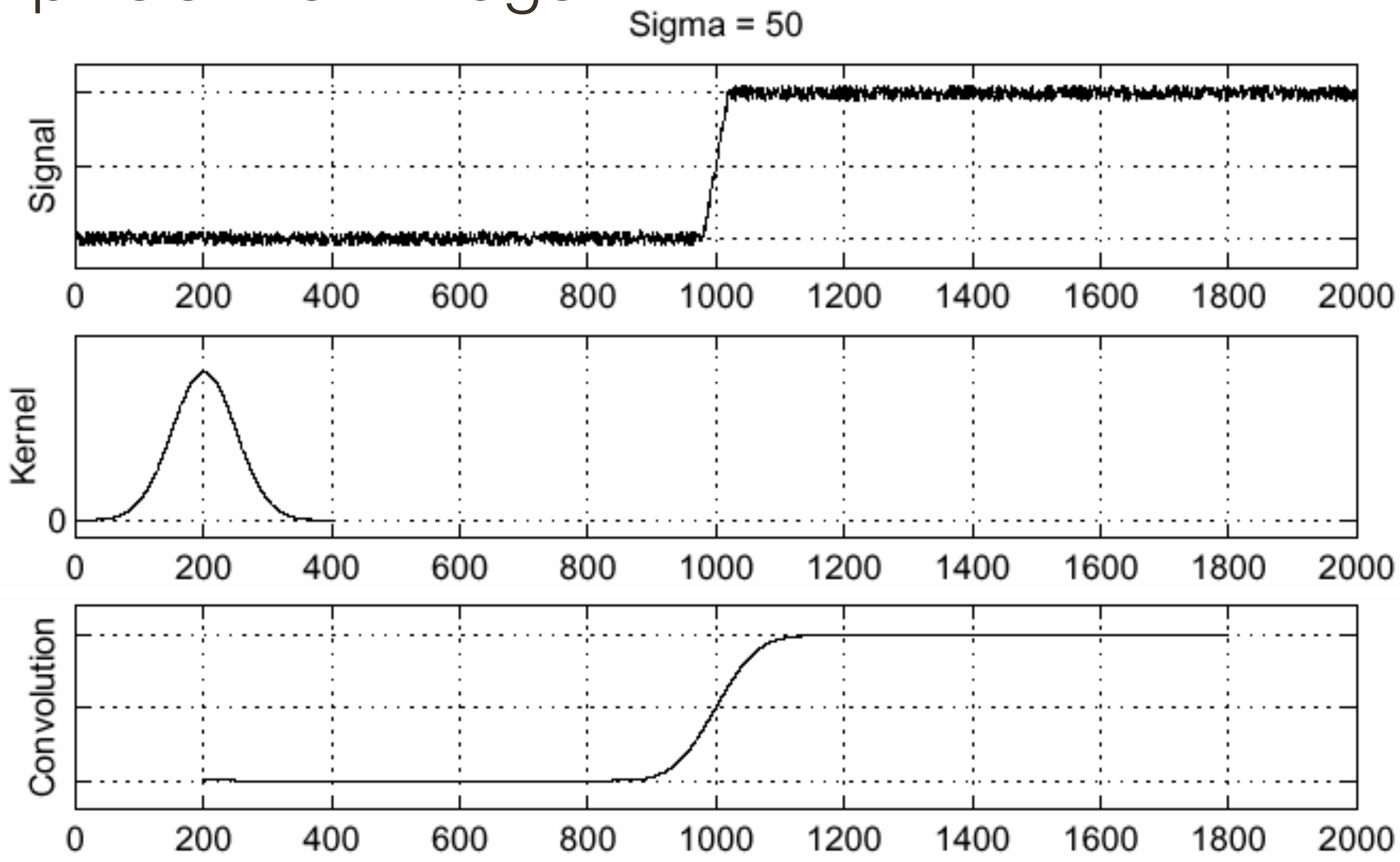
# 1D **Example**: Smoothing + Derivative

Lets consider a row of pixels in an image:

$I(X, 245)$

$G$

$G \otimes I(X,Y)$



# 1D **Example**: Smoothing + Derivative

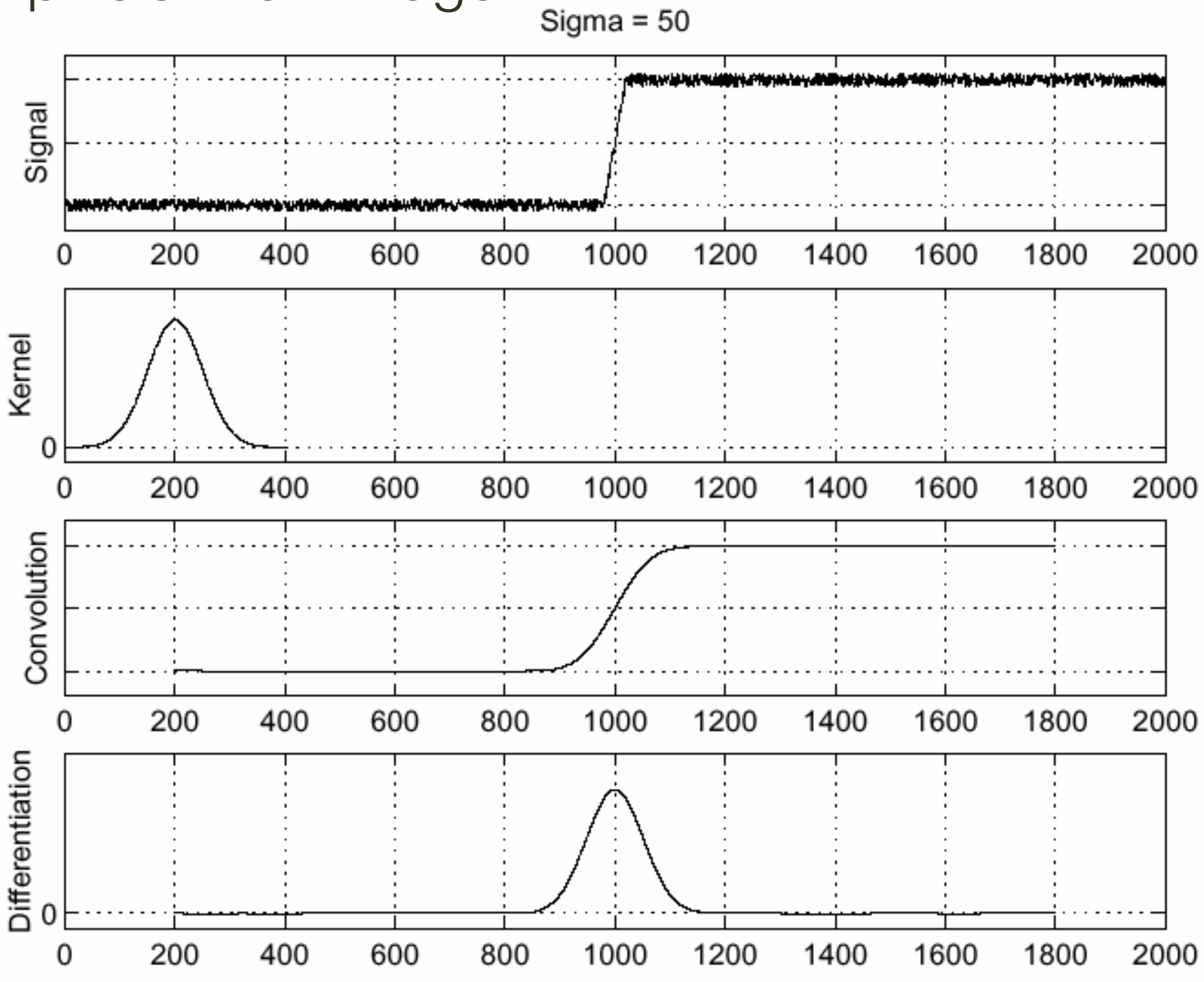
Lets consider a row of pixels in an image:

$$I(X, 245)$$

$$G$$

$$G \otimes I(X, Y)$$

$$\frac{\partial G \otimes I(X, Y)}{\partial x}$$



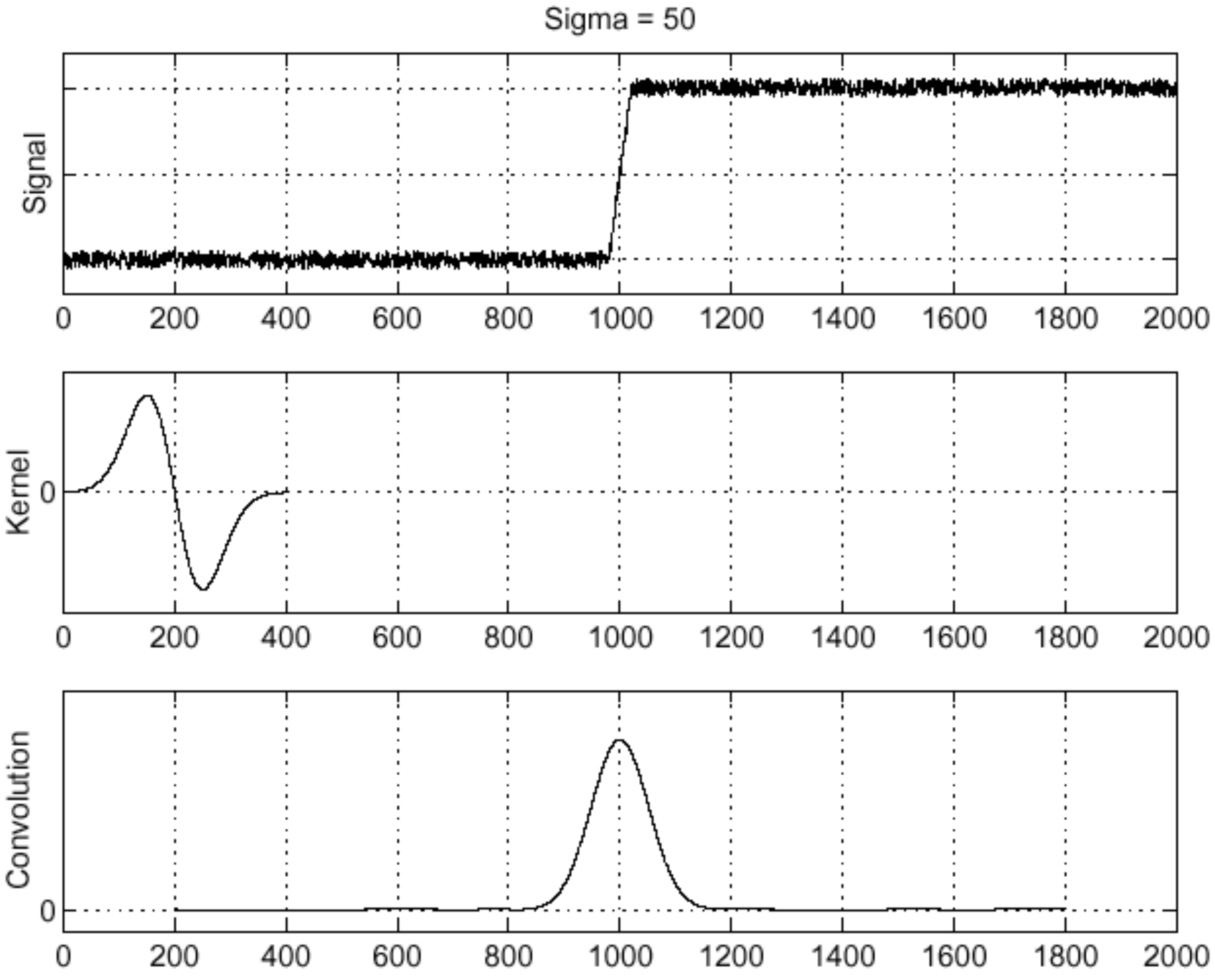
# 1D **Example**: Smoothing + Derivative (efficient)

Lets consider a row of pixels in an image:

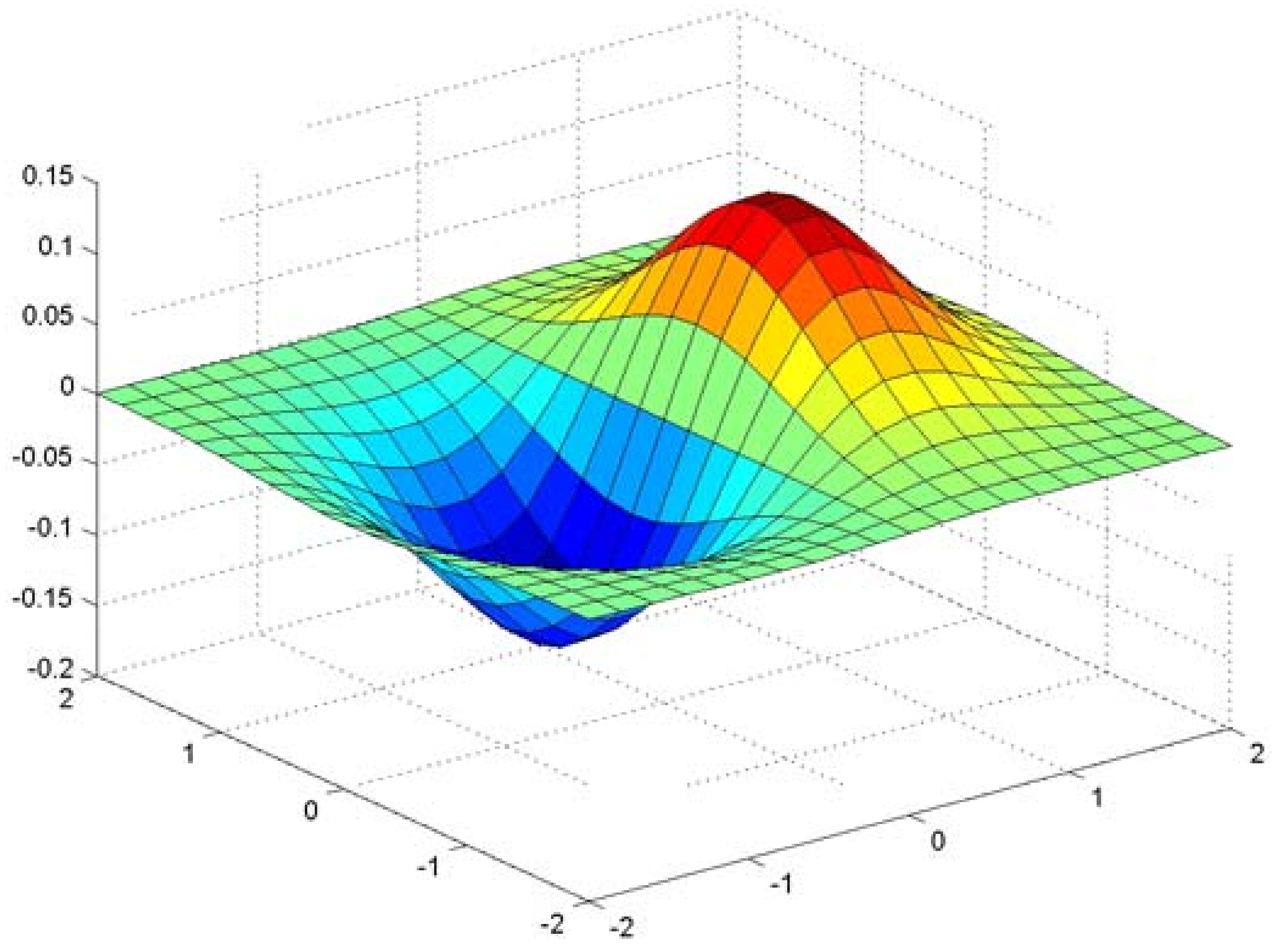
$$I(X, 245)$$

$$\frac{\partial G}{\partial x}$$

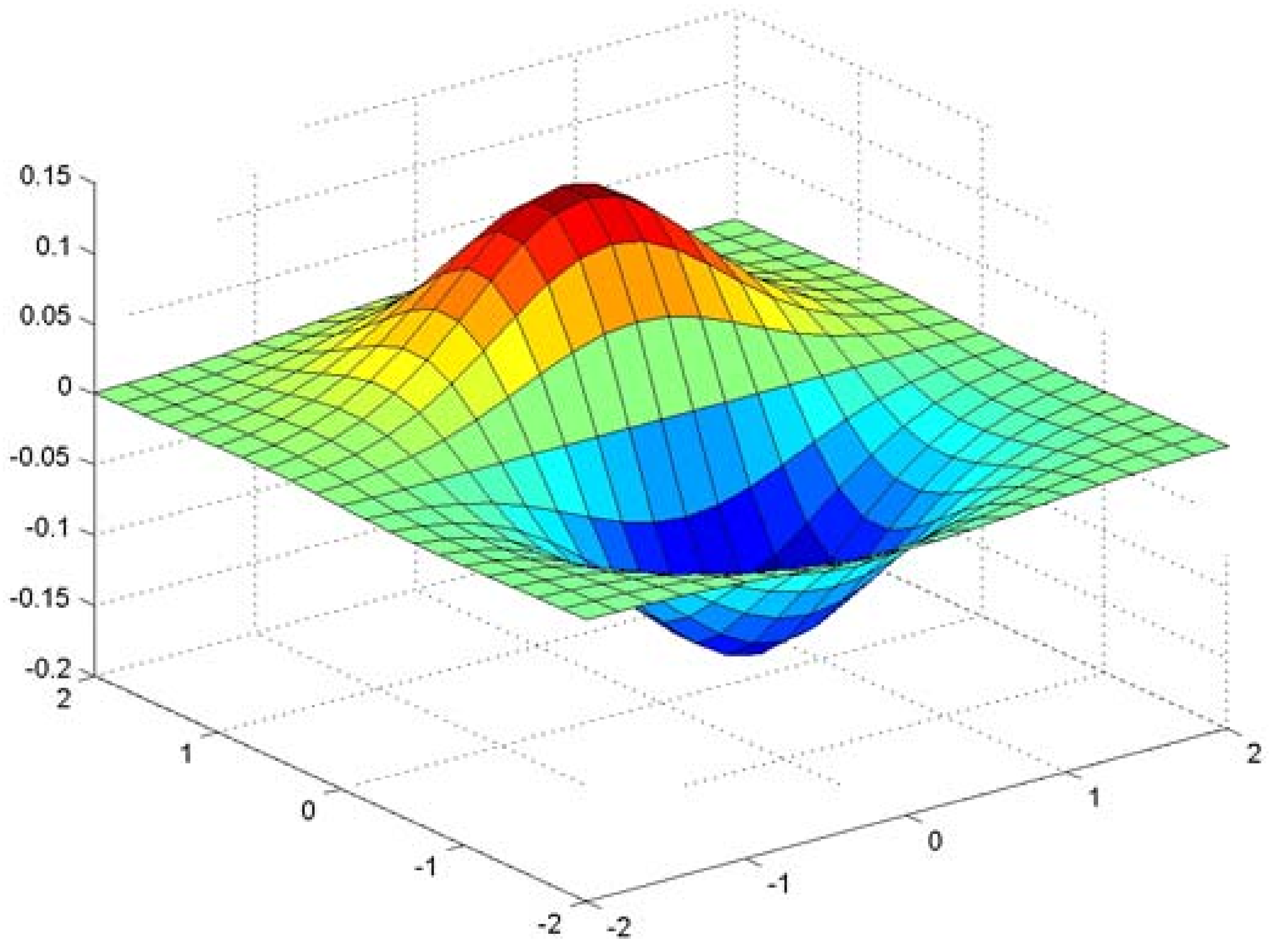
$$\frac{\partial G}{\partial x} \otimes I(X, Y)$$



# Partial **Derivatives** of **Gaussian**




$$\frac{\partial}{\partial x} G_\sigma$$




$$\frac{\partial}{\partial y} G_\sigma$$

**Slide Credit**: Christopher Rasmussen