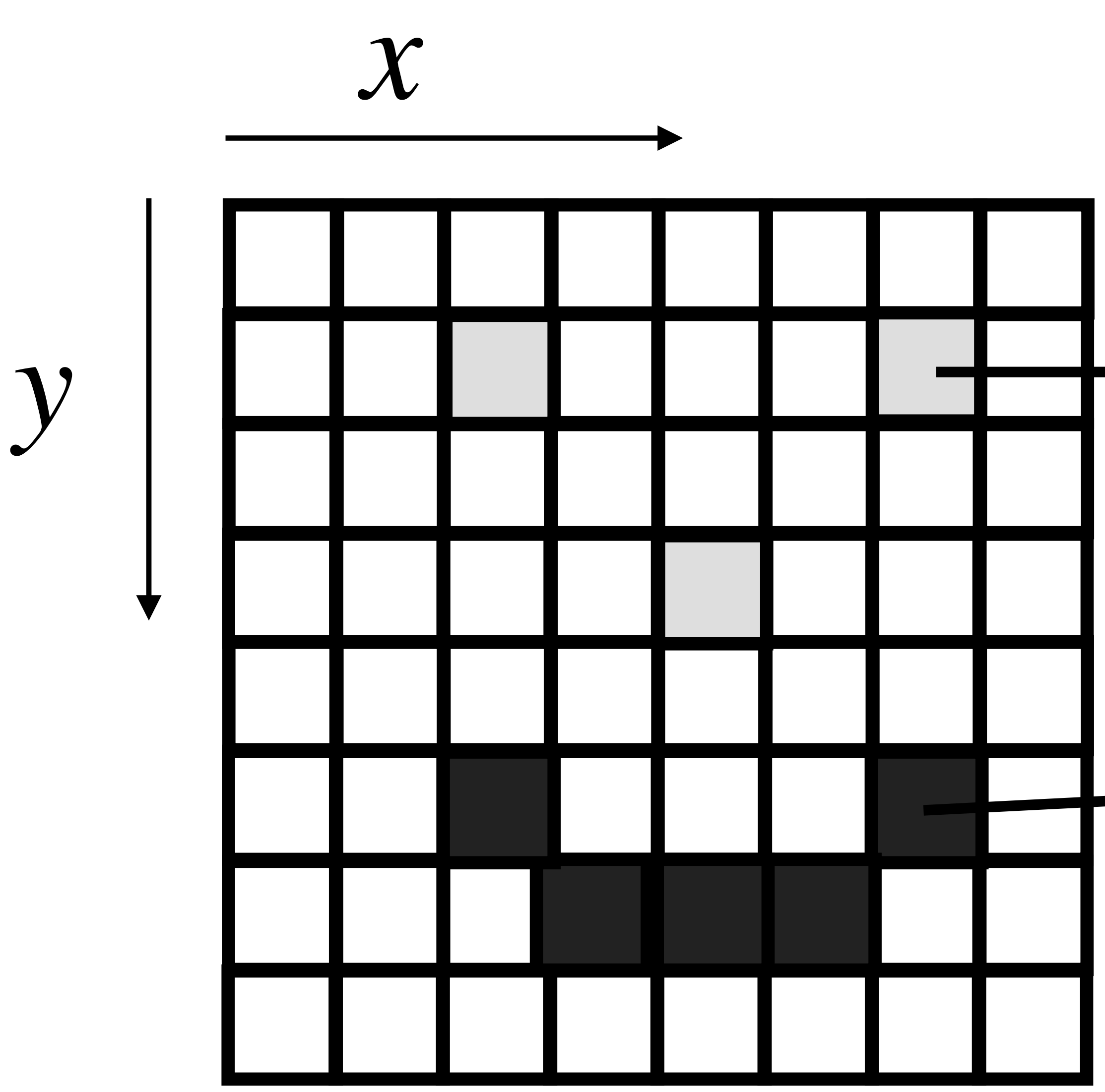
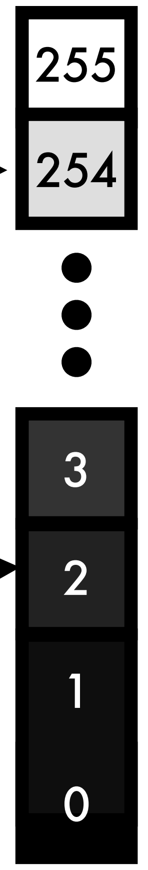
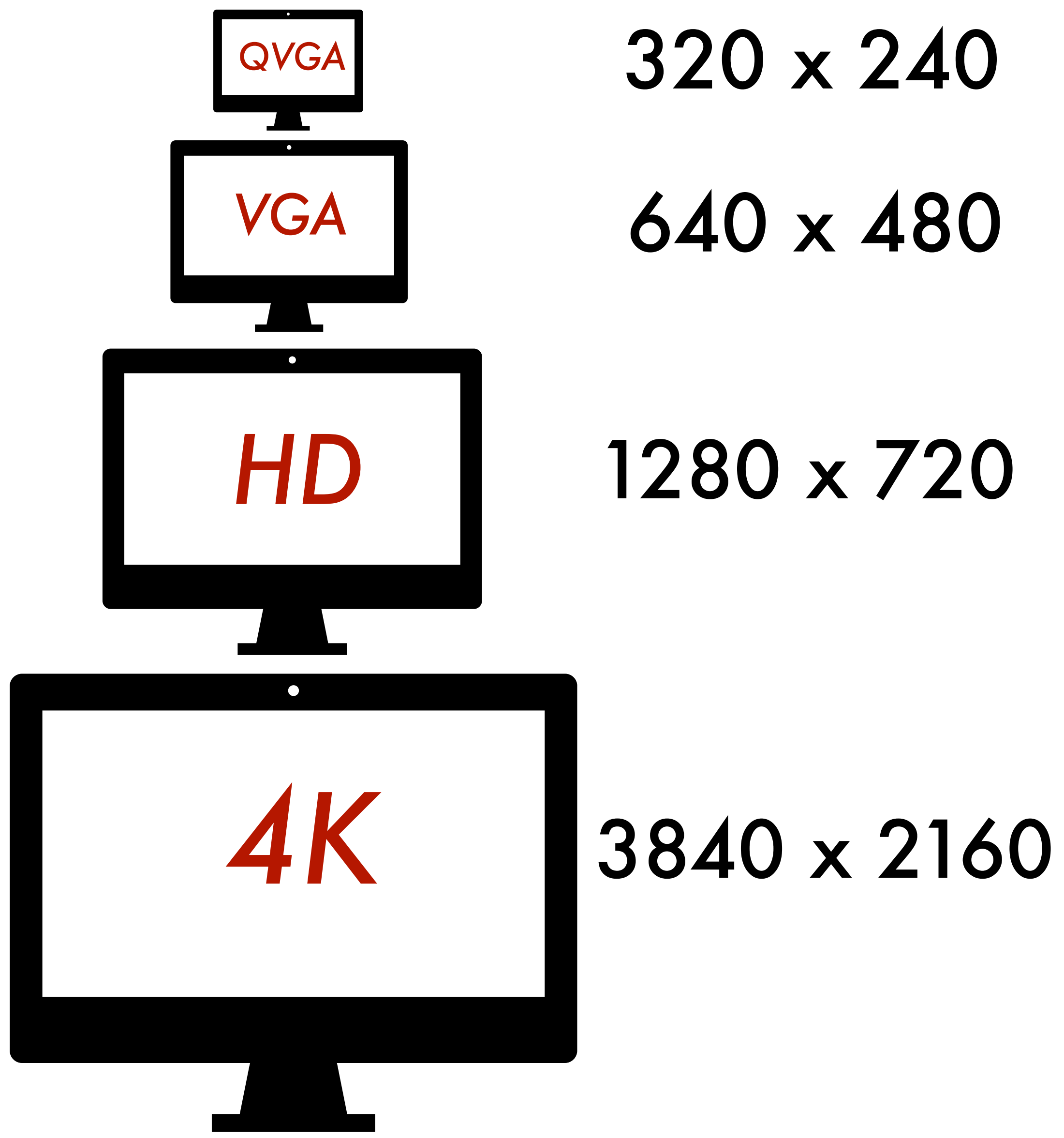
Representing Images via Intensities



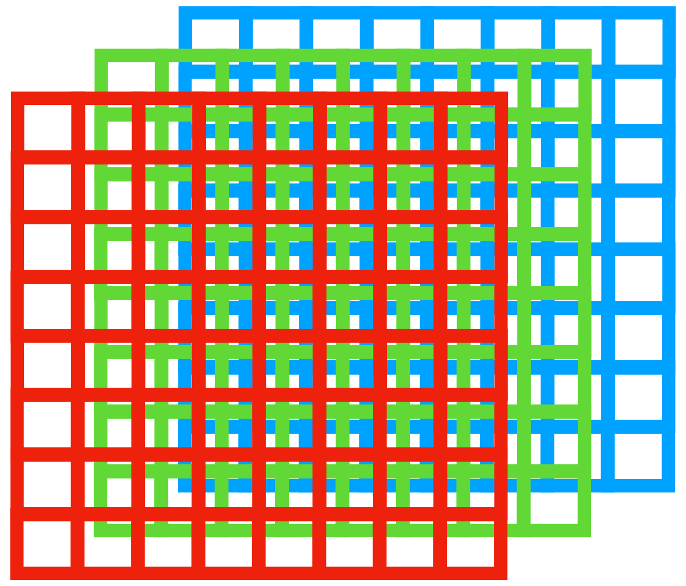
A monochrome image can be represented as a matrix of intensity values $I(x, y)$



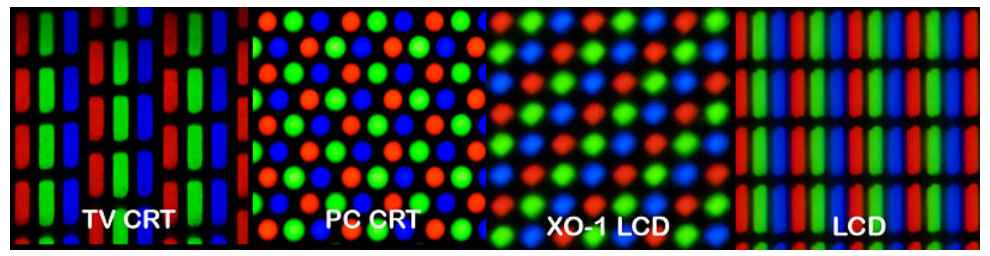
Typically use 256 (2^8) discrete values to represent intensities



Certain matrix sizes (resolutions) common for TVs, monitors etc..



Can represent colour by stacking 3 matrices (Red-Green-Blue)



RGB elements can be packed together to produce any colour **NE**

VE Tag = Very Examinable

NE Tag = Non Examinable

Image credit: "pixel geometry", https://commons.wikimedia.org/wiki/File:Pixel_geometry_01_Pengo.jpg

Challenge: Nuisance factors in image data

If a point on an object (in this case, a *torta di mele*) is visible in view, the intensity $I(x, y)$ is a function of many geometric and photometric variables:

- The position and orientation of the camera
- The geometry of the scene (3D shapes and layout)
- The nature and distribution of light sources
- Reflectance properties of the surfaces: specular-Lambertian ("matte" surface), albedo 0 (black) - 1 (white)
- The properties of the camera lens and sensor array

In practice, the point may only be partially visible, or its appearance may also be affected by occlusion

Challenge: Data reduction

With current computers, we need to discard most data from the camera before an attempt can be made at real-time interpretation

Raw image data e.g. GoPro 11 (CMOS): 5.3K @60 fps

O(10 Gbits/sec)

iSight CCD: 640x480 @30 fps

O(100 Mbits/sec)

Raw Image data → Generic salient features → Interpretation

O(100+ Kbits/sec)

How can we achieve such a dramatic data reduction?

Goals for <u>generic features</u>:

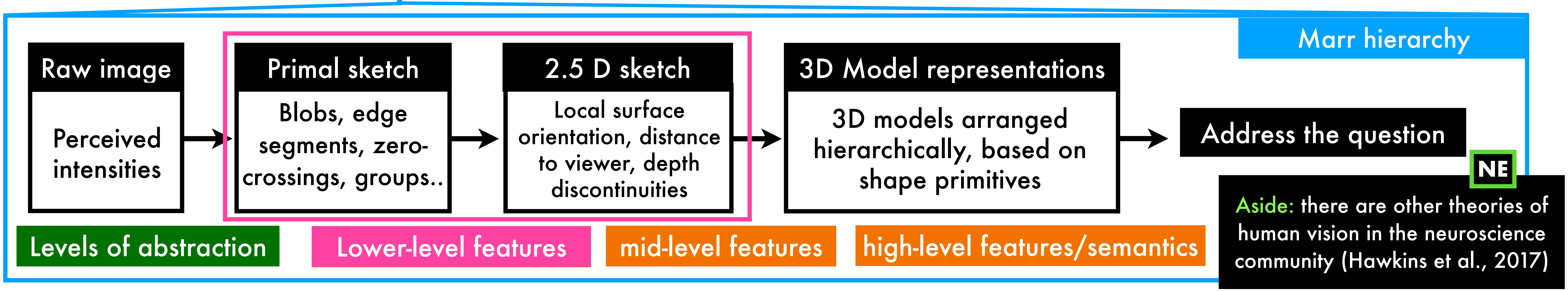
- *Allow image to be discarded - all subsequent processing is done on features themselves (<u>reduce the amount of data</u>)*
- *Preserve useful info in images (e.g. 2D shape of objects in scene)*
- *Discard redundant info in the images (e.g. lighting conditions)*
- *<u>As generic as possible</u> (so same processing will be useful across a wide range of applications)*

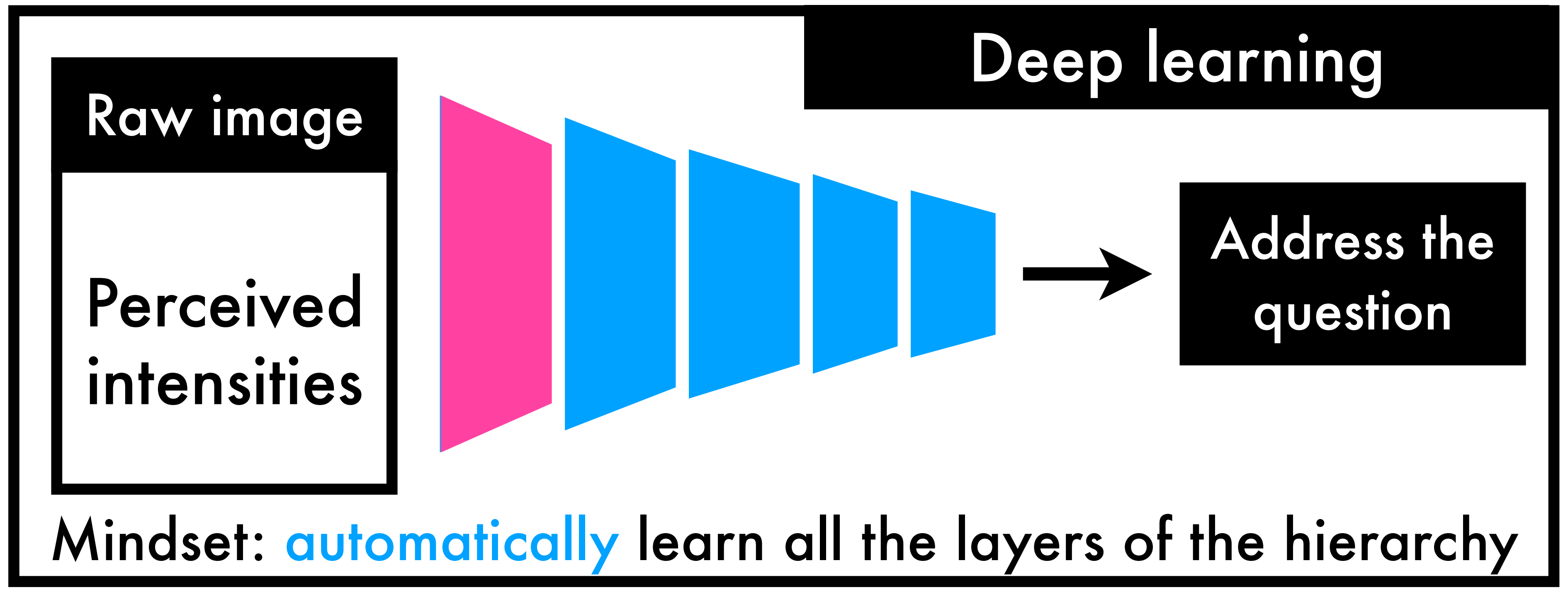
Image credits: https://pixabay.com/photos/camera-go-pro-people-hand-mount-2590899/
https://en.wikipedia.org/wiki/File:Apple_iSight_FireWire_Camera.jpg

Research Trivia — Boyle & Smith won Nobel Prize in Physics in 2009 for invention of CCD technology — NE

Research Trivia — CLIP models (trained on hundreds of GPUs) still only uses up to 448 pixel images (typically fewer) — NE

Computer vision as hierarchical processing





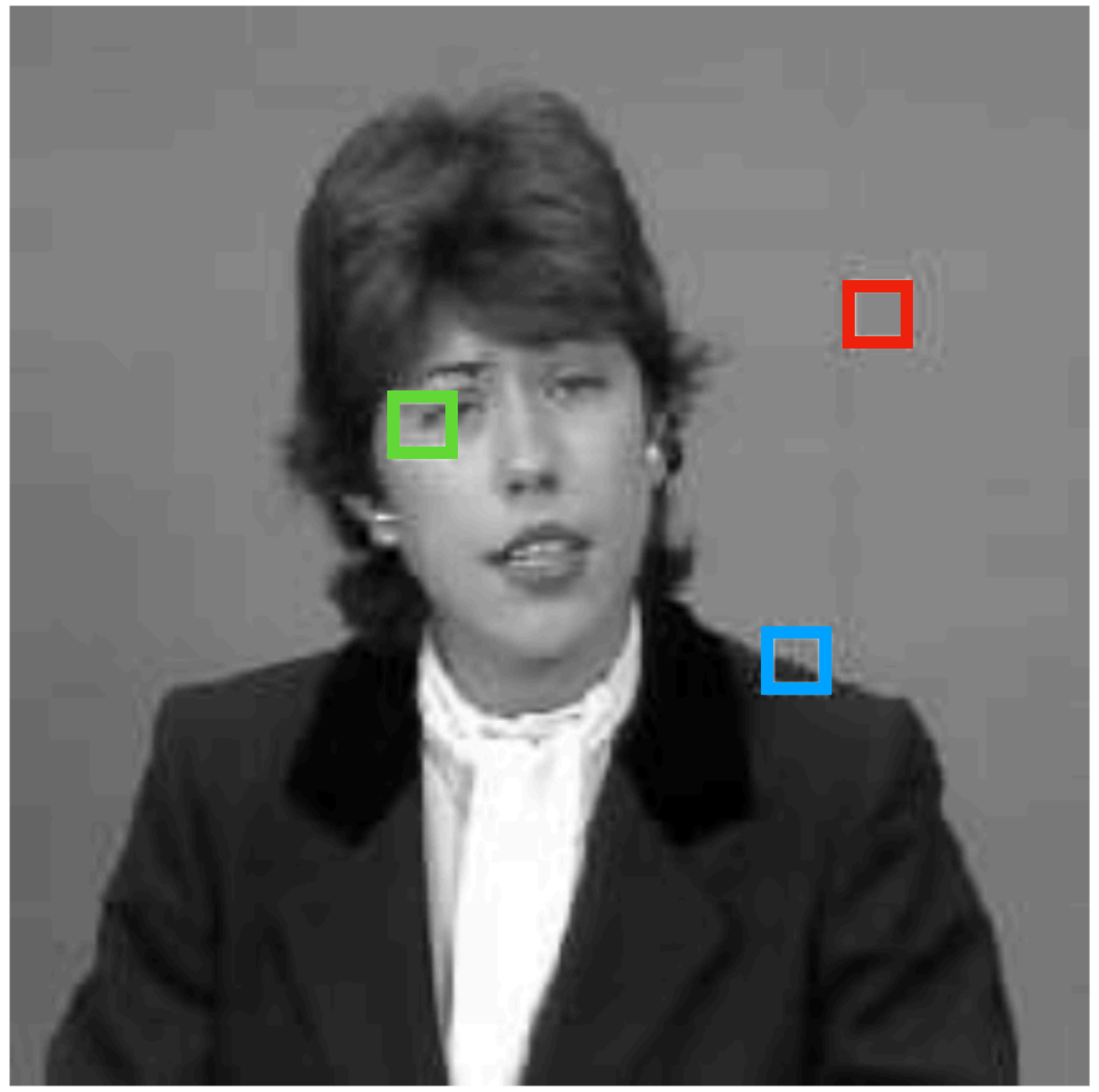
Study of Image Structure

Natural images have properties that enable dramatic compression

We want to understand in detail how to efficiently extract low-level *generic features* ("*primitives*") from image structures as building blocks for image interpretation (*particularly matching*)

Reference: D. Marr, "Vision: A computational investigation into the human representation and processing of visual information" (1982)
J. Hawkins, S. Ahmad, and Y. Cui, "A theory of how columns in the neocortex enable learning the structure of the world." *Frontiers in neural circuits* (2017)

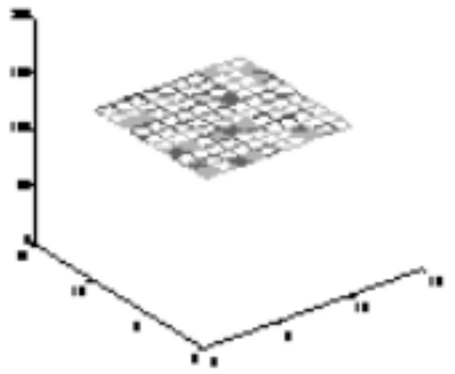
Image structure

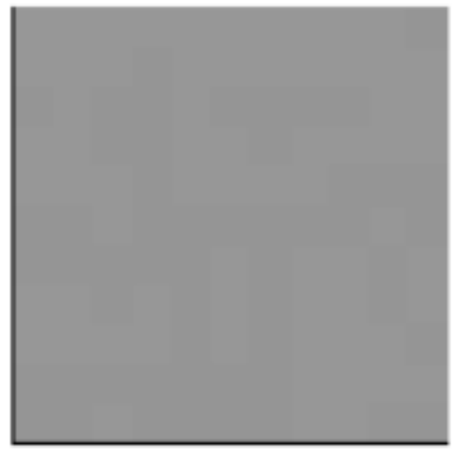


Let's examine pixel values in three patches in this photo of Claire:

- A featureless region
- An edge
- A corner

140	141	141	140	141	141	140	141	141	141	139
140	141	141	139	140	140	140	140	140	140	140
139	140	139	139	140	139	139	138	139	140	140
140	140	139	139	140	140	138	140	140	140	140
140	140	141	139	141	141	140	140	139	139	138
139	139	141	139	139	139	139	139	139	140	139
139	139	139	139	139	140	139	140	140	139	141
140	140	139	140	139	141	139	140	140	139	140
140	140	140	140	139	140	139	140	131	131	139
139	139	138	139	139	139	139	140	141	140	140
139	139	140	139	139	139	139	140	140	139	139

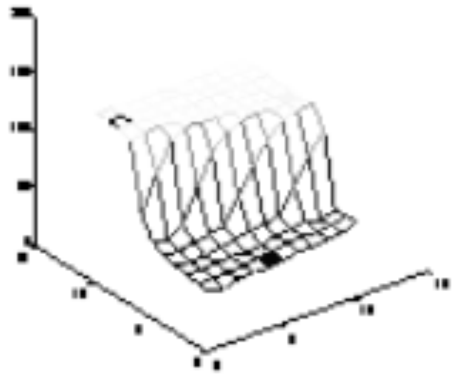




0D

The featureless region is characterised by a smooth variation of intensities

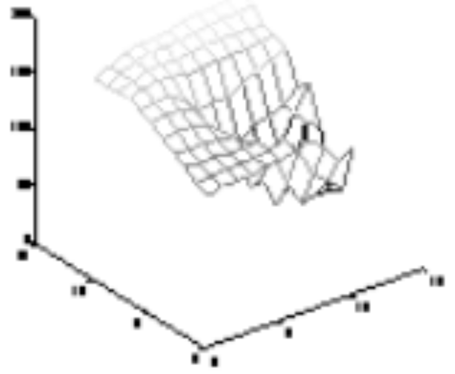
136	136	135	134	135	134	134	135	135	136	135
136	135	134	134	136	135	133	133	135	135	136
136	135	134	134	133	133	134	134	134	134	135
130	133	134	133	132	132	132	133	133	132	133
73	103	127	135	134	133	134	133	132	133	134
54	52	60	88	114	127	133	134	132	133	132
49	48	50	53	56	76	99	117	130	133	135
46	45	45	48	50	53	55	57	77	99	118
44	44	45	46	45	47	50	52	54	49	54
42	43	43	44	46	47	50	51	50	52	55
41	40	44	44	44	46	49	48	50	51	56

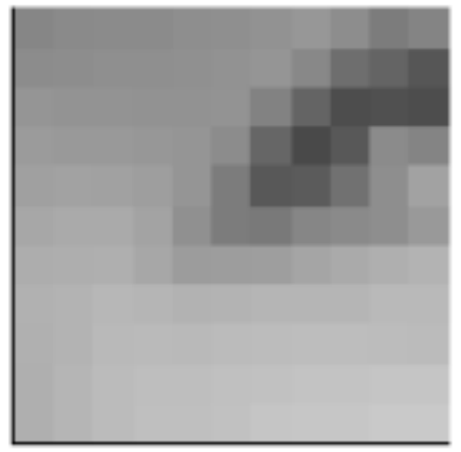


1D

The patch containing the edge reveals an intensity discontinuity in one direction

124	127	128	128	131	312	135	140	130	113	121
129	130	132	132	133	135	139	125	99	89	76
138	136	137	135	135	136	130	89	68	71	69
144	142	143	141	139	129	92	64	78	128	121
150	152	151	148	138	113	79	81	102	131	152
158	160	160	154	133	113	111	123	127	131	143
163	165	166	158	145	146	147	155	160	166	171
168	171	174	173	169	171	172	173	173	176	176
167	171	176	177	176	178	180	181	181	180	179
166	173	179	182	182	184	185	187	188	189	190
166	173	179	183	183	185	189	191	191	194	194





2D

The patch containing the corner reveals an intensity discontinuity in two directions

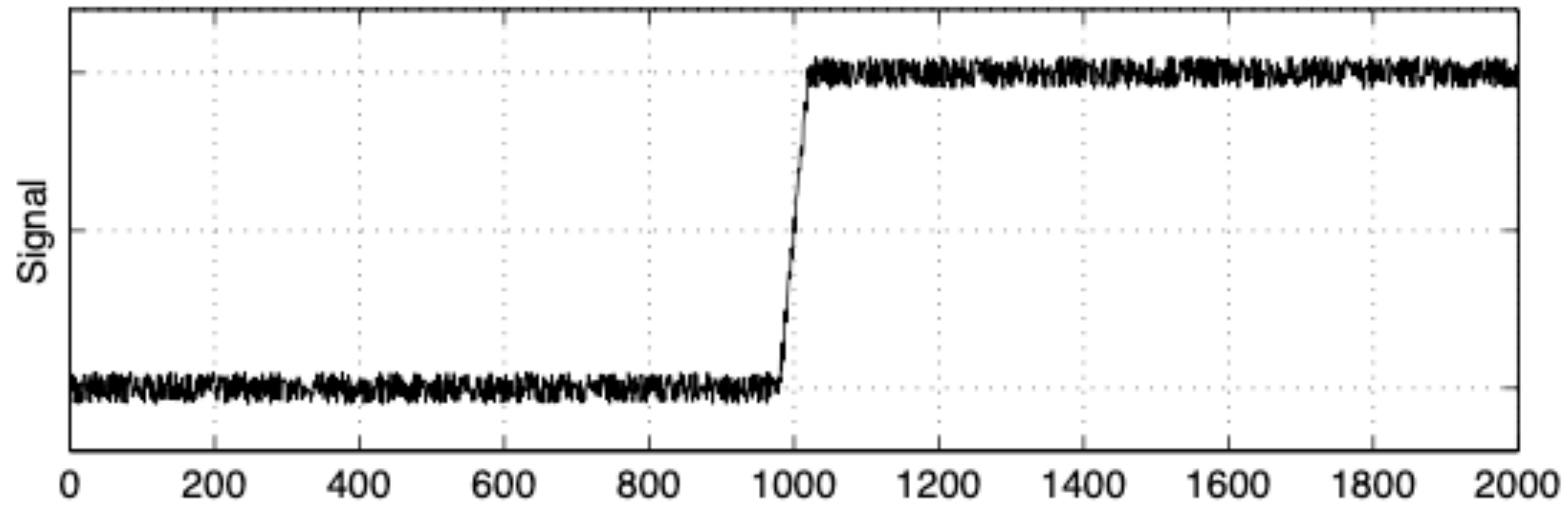
Note that an edge or corner representation imparts a desirable invariance to lighting: *the intensity discontinuities are likely to be prominent, whatever the lighting conditions*

Computational question: *How can we find these structures efficiently?*

1D Edge Detection

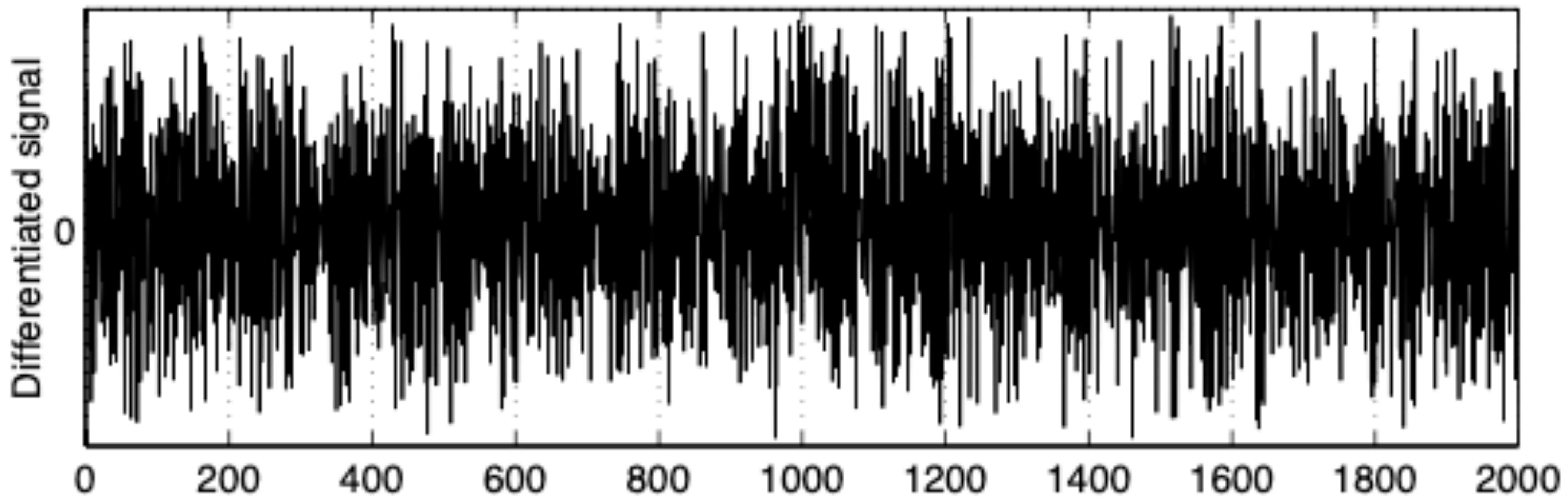
When developing an edge detection algorithm, it is important to bear in mind the invariable presence of image noise

Consider this signal I(x) with an obvious edge:



An intuitive approach to edge detection might be to look for maxima and minima in I'(x)

The derivative of the signal looks like this:



Oh dear: it's hard to spot the edge in this signal!

Our simple strategy was defeated by high-frequency noise (which is amplified by differentiation)

For this reason, all edge detectors start by smoothing the signal to suppress noise

The most common approach is to use a Gaussian filter *(a low-pass filter that suppresses high frequencies)*

Figure credits: S. Seitz

1D Edge Detection (with smoothing)

1D Edge Detection algorithm

A broad overview of 1D edge detection is:

1. First, convolve the signal $I(x)$ with a Gaussian kernel $g_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{x^2}{2\sigma^2}\right)$ to produce smooth $s(x)$.
2. Compute $s'(x)$, the derivative of $s(x)$.
3. Find maxima and minima of $s'(x)$.
4. Use thresholding on the magnitude of the extrema to mark edges.

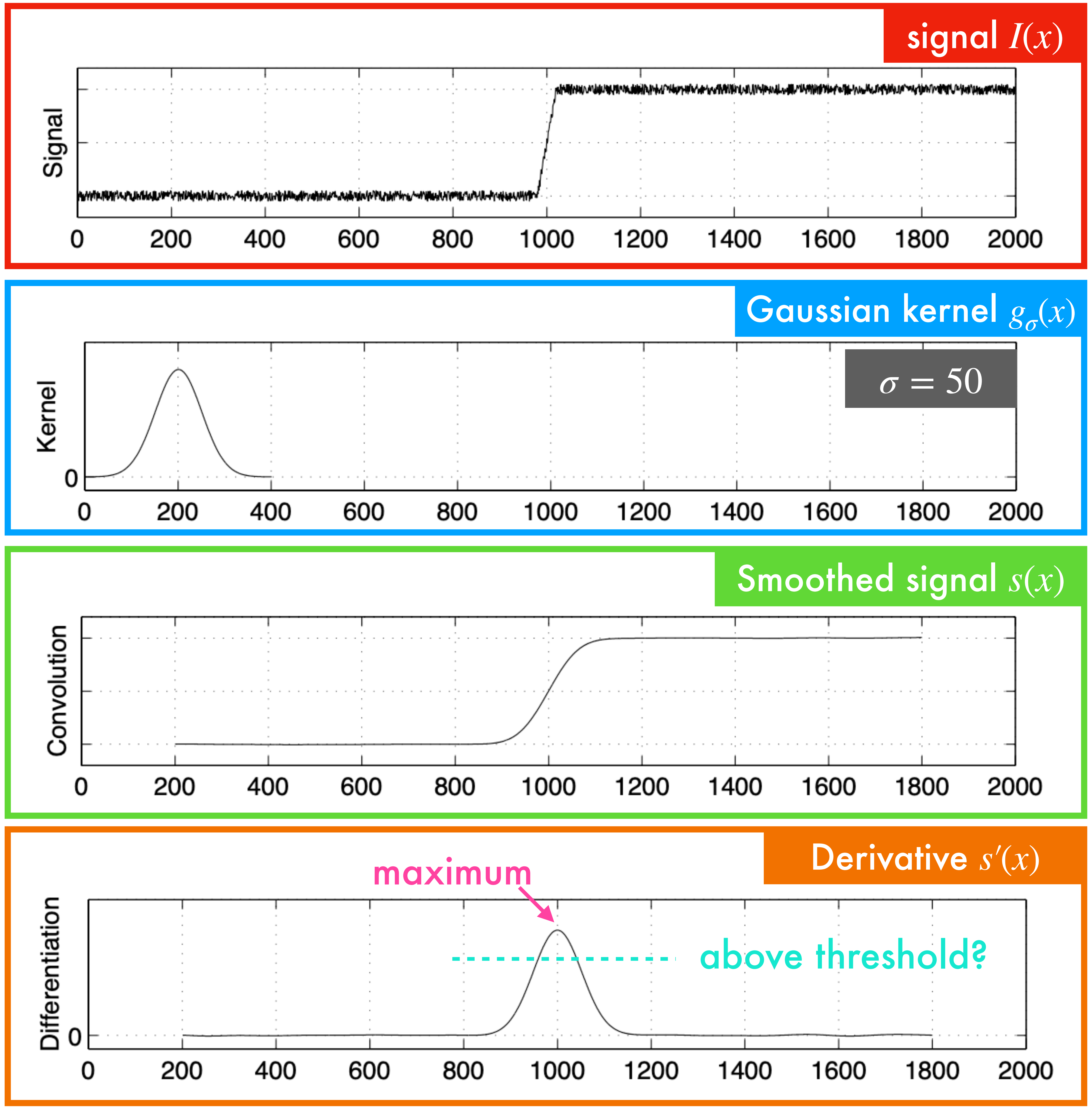


Figure credits: S. Seitz

1D Edge Detection: a computational trick

The smoothing in step 1 was performed by a 1D convolution with a Gaussian:

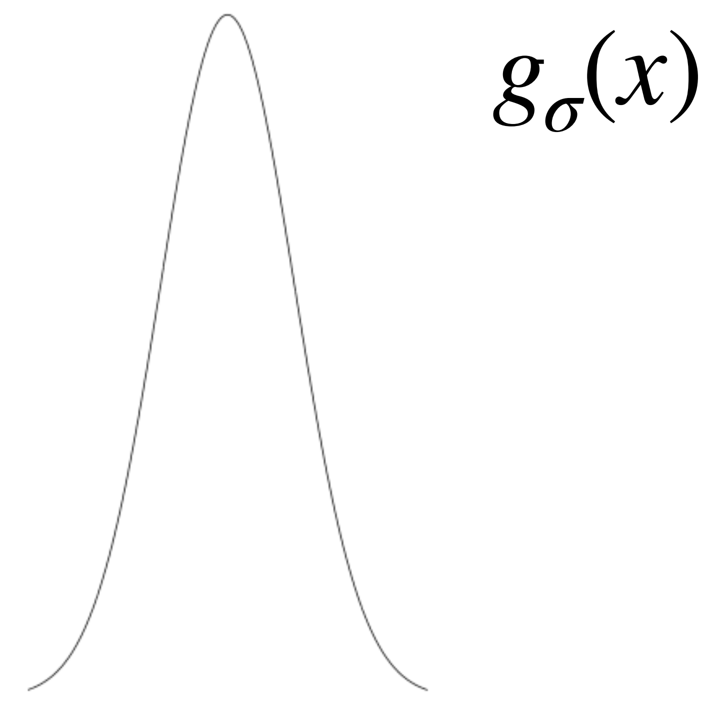
$$s(x) = I(x) \circledast g_\sigma(x) = \int_{-\infty}^{\infty} g_\sigma(u) I(x-u) du = \int_{-\infty}^{\infty} g_\sigma(x-u) I(u) du$$

Differentiation in step 2 is *also* performed by a 1D convolution - it seems edge detection requires two computationally expensive convolutions

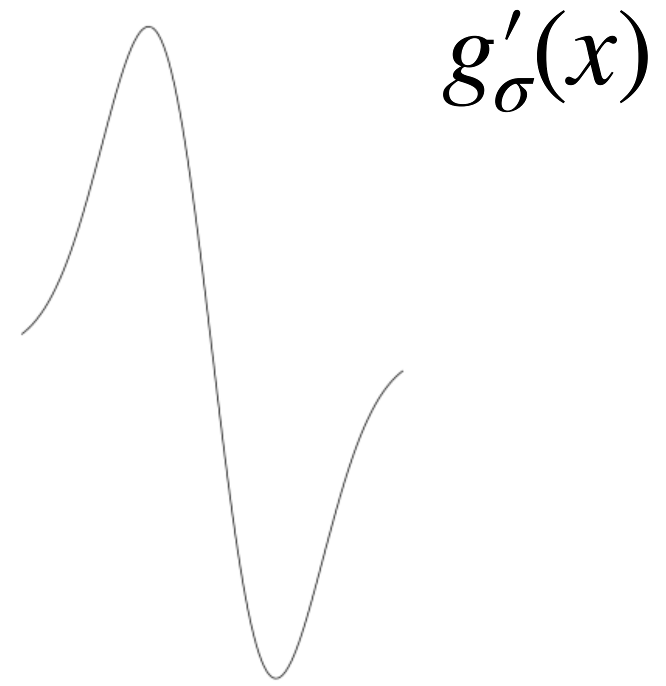
However, the derivative theorem of convolution comes to the rescue!

$$s'(x) = \frac{d}{dx}[g_\sigma(x) \circledast I(x)] = g'_\sigma(x) \circledast I(x)$$

So we can compute $s'(x)$ with just a single convolution - a ***major saving!***



The Gaussian Kernel



The Derivative of Gaussian Kernel

1D Edge Detection (faster)

Faster 1D Edge Detection algorithm

The fast variant of the edge detection algorithm becomes:

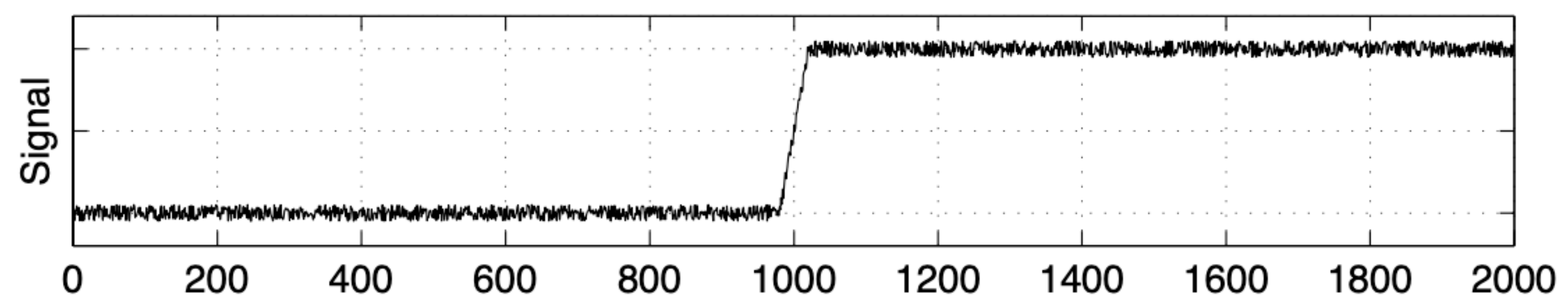
1. Convolve the signal $I(x)$ with a derivative of Gaussian Kernel $g'_\sigma(x)$ to produce $s'(x)$ directly.

2. Find maxima and minima of $s'(x)$.

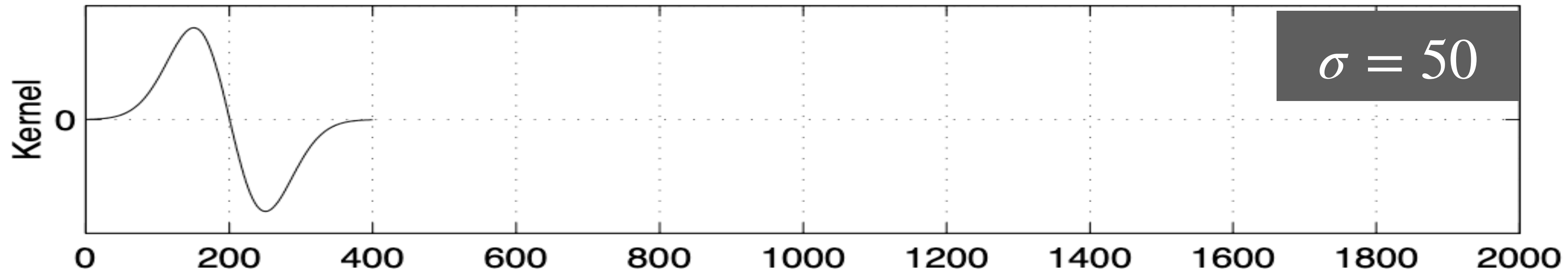
3. Use thresholding on the magnitude of the extrema to mark edges.

VE

signal $I(x)$



Derivative of Gaussian kernel $g'_\sigma(x)$



Derivative $s'(x)$

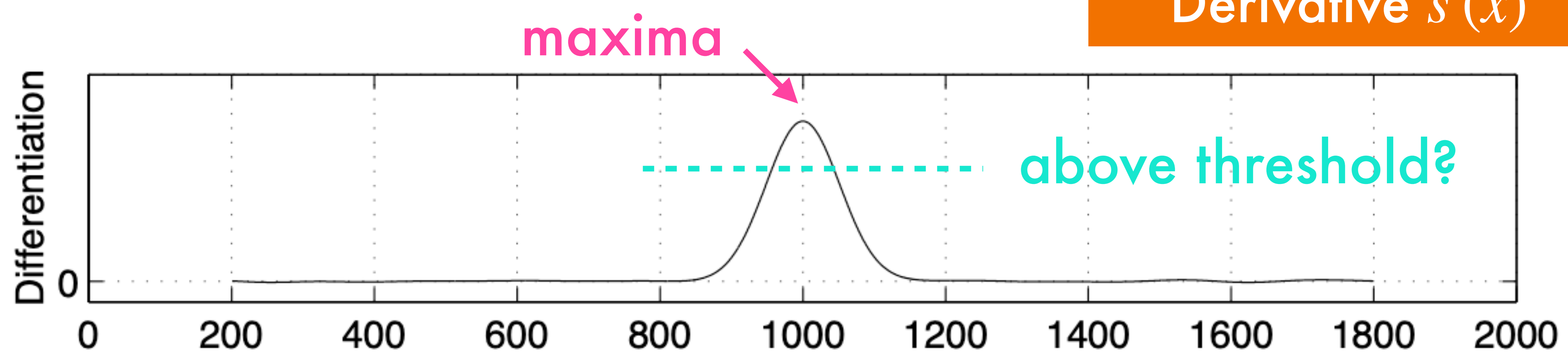


Figure credits: S. Seitz

1D Edge Detection: zero-crossings

Fastest 1D Edge Detection algorithm

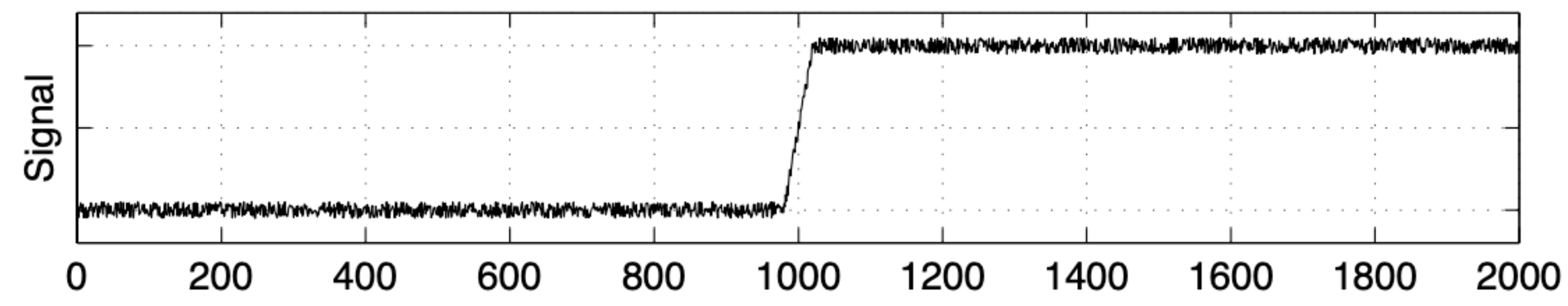
Finding maxima and minima of $s'(x)$ is the same as looking for zero-crossings of $s''(x)$!

In many implementations of edge detection algorithms, the signal is convolved with the Laplacian of a Gaussian ("LoG" kernel), $g''_\sigma(x)$, by applying the derivative theorem of convolution a second time:

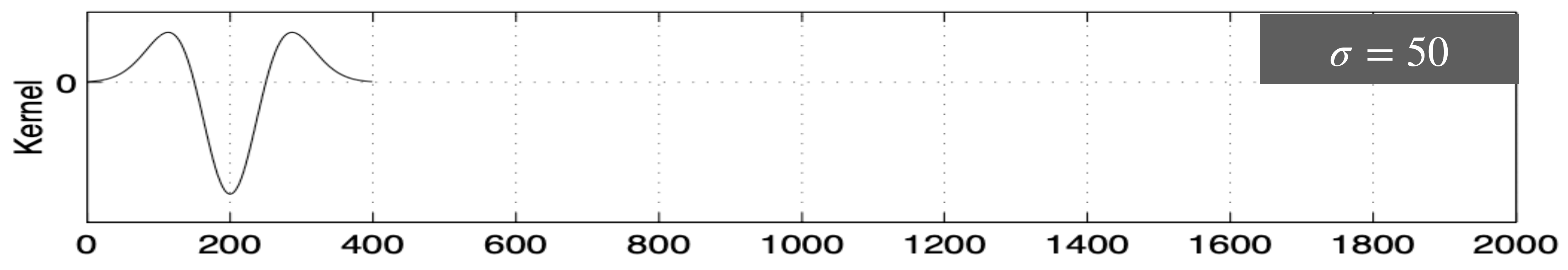
$$s''(x) = g''_\sigma(x) \circledast I(x)$$

The zero crossings of $s''(x)$ mark possible edges

signal $I(x)$



Laplacian of Gaussian kernel $g''_\sigma(x)$



Second Derivative $s''(x)$

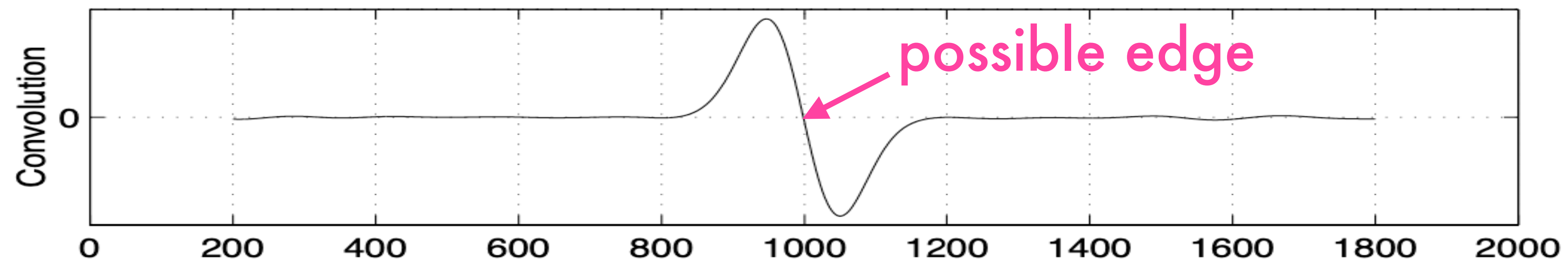
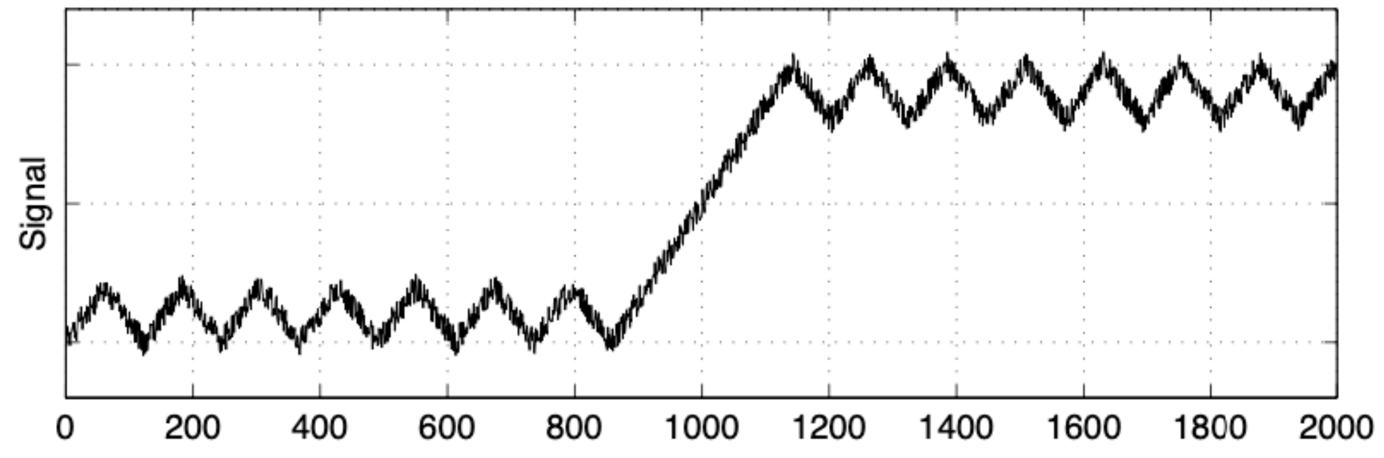


Figure credits: S. Seitz

1D Edge Detection: scale

We have not yet addressed the critical issue of what value of σ to use

Consider this signal:



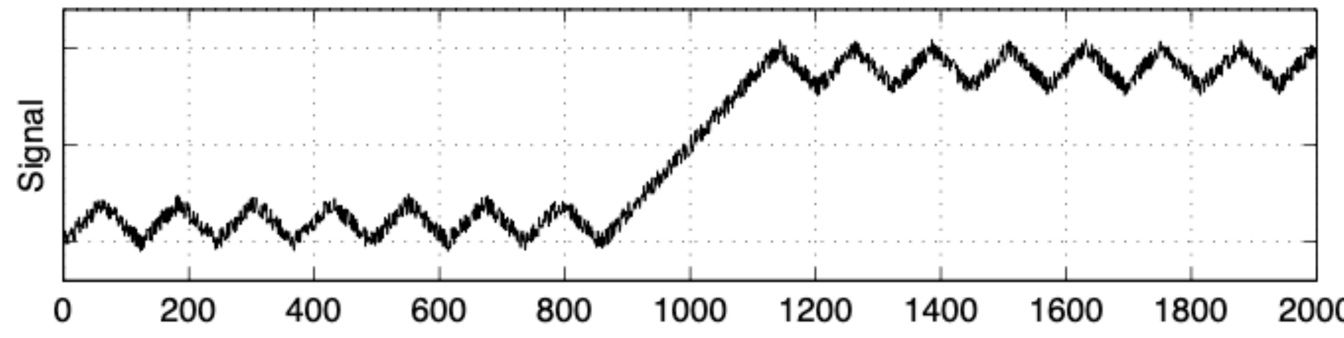
Does the signal have one "positive" edge or a number of "positive" and "negative" edges?

It's up to you to choose!

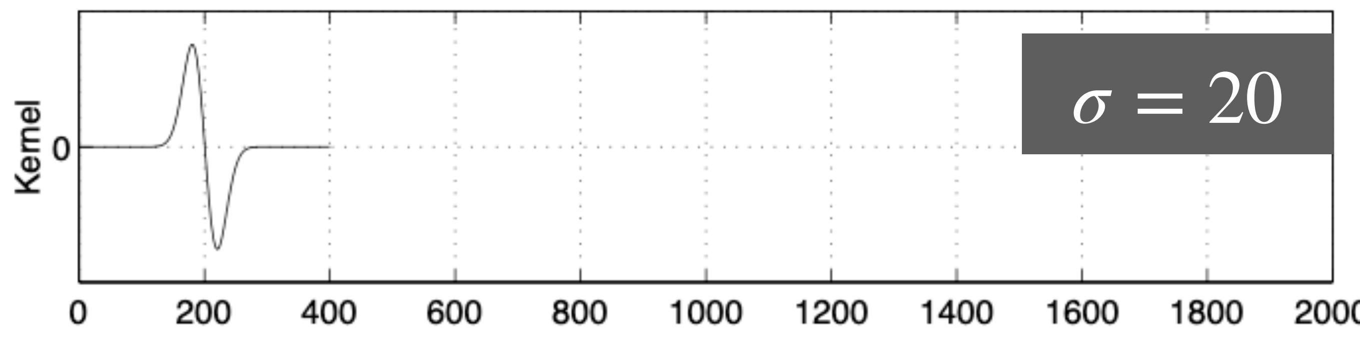
Fine detail edges

Using a small σ brings out many edges

signal $I(x)$

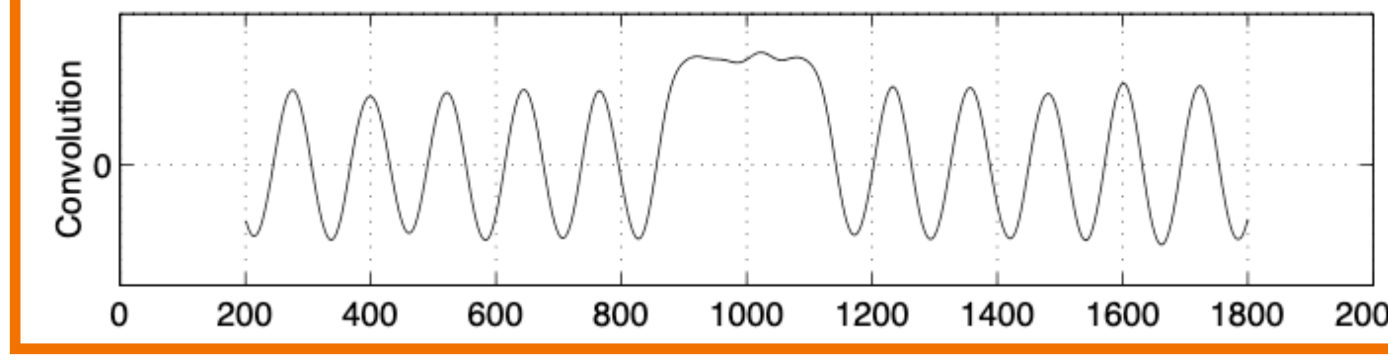


Derivative of Gaussian kernel $g'_\sigma(x)$



$\sigma = 20$

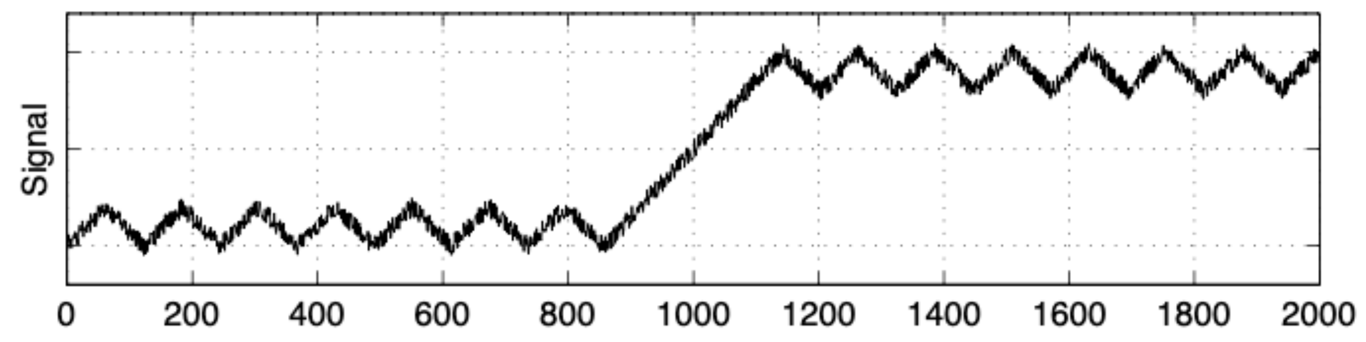
Derivative $s'(x)$



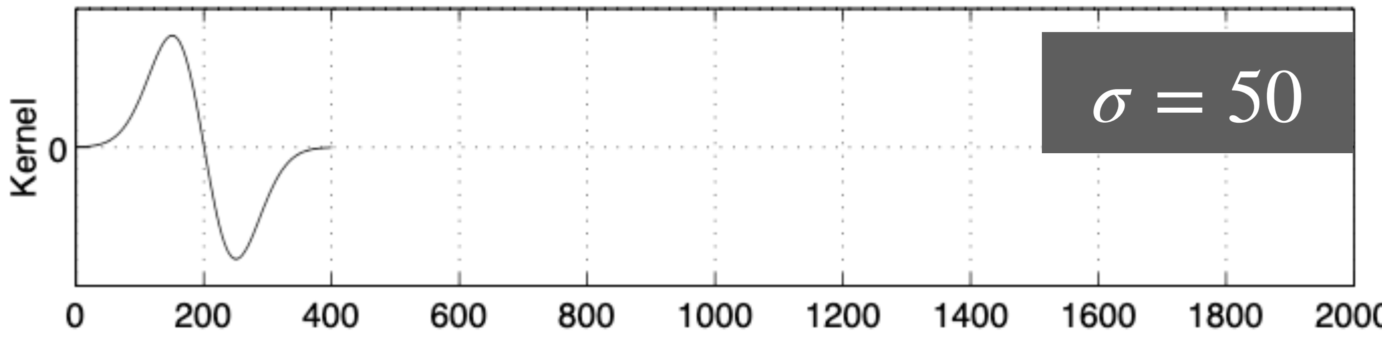
Coarse detail edges

As σ increases, the signal is smoothed more and more, and only the central edge survives

signal $I(x)$



Derivative of Gaussian kernel $g'_\sigma(x)$



$\sigma = 50$

Derivative $s'(x)$

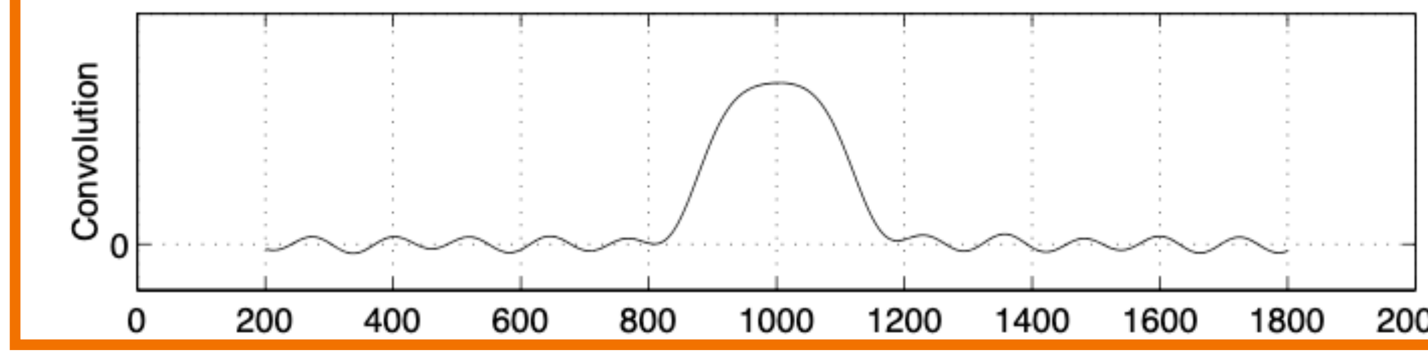


Figure credits: S. Seitz

1D Edge Detection: multi-scale

The link between smoothing and scale

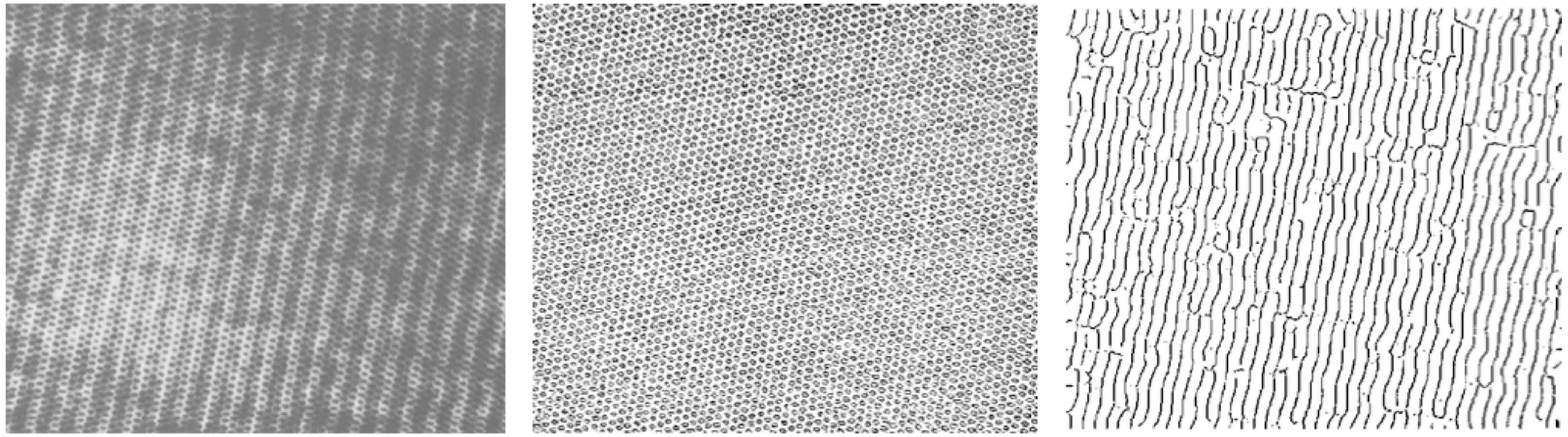
The amount of smoothing controls the scale at which we analyse the image

There is no right or wrong size for the Gaussian kernel: it all depends on the scale we're interested in.

Modest smoothing (Gaussian kernel with small σ) brings out edges at fine scale

More smoothing (larger σ) finds edges at larger scales, suppressing finer detail

Example: a dish cloth



Original image $\sigma = 1$ $\sigma = 5$

<u>Note</u>: Fine scale edge detection is particularly sensitive to noise (less of an issue at coarse scales)

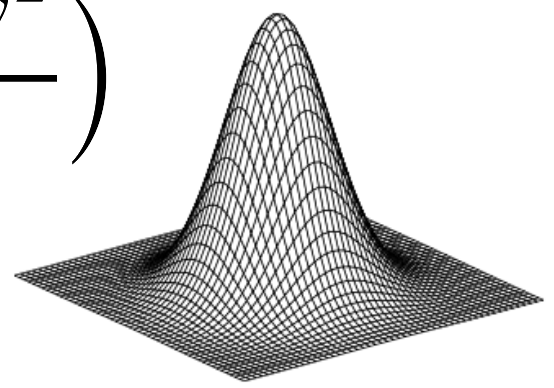
2D Edge Detection (step 1: smoothing)

Step 1: smoothing

The 1D edge detection scheme can be extended to work in two dimensions

First we smooth the image $I(x, y)$ by convolving with a 2D Gaussian $G_\sigma(x, y)$:

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(- \frac{x^2 + y^2}{2\sigma^2} \right)$$



$$S(x, y) = G_\sigma(x, y) \circledast I(x, y)$$

$$= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} G_\sigma(u, v) I(x - u, y - v) du dv$$

Effects of Gaussian smoothing

Original image

$\sigma = 3$ pixels

$\sigma = 4$ pixels

2D Edge Detection (step 2: gradients)

Step 2: gradients

Second step: compute the gradient of the smoothed image $S(x, y)$ at every pixel:

$$\nabla S = \nabla(G_\sigma \circledast I)$$

$$= \begin{bmatrix} \frac{\partial(G_\sigma \circledast I)}{\partial x} \\ \frac{\partial(G_\sigma \circledast I)}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial G_\sigma}{\partial x} \circledast I \\ \frac{\partial G_\sigma}{\partial y} \circledast I \end{bmatrix}$$

Example: fruity gradients

Original image

Edge strength $|\nabla S|$

2D Edge Detection (step 3: NMS & step 4: thresholding)

Step 3: Non-Maxima Suppression

The third stage of the edge detection algorithm is Non-Maxima Suppression (NMS)

Edge elements, or *edgels*, are placed at locations where $|\nabla S|$ is greater than local values of $|\nabla S|$ in the directions $\pm\nabla S$

This aims to ensure that all *edgels* are located at ridge-points of the surface $|\nabla S|$

Edge strength $|\nabla S|$ after NMS

Step 4: Thresholding

In the fourth and final step, the *edgels* are thresholded, so that only those with $|\nabla S|$ above a certain value are retained

Edge strength $|\nabla S|$ after NMS and thresholding

2D Edge Detection (variations)

Canny Edge Detection

The edge detection algorithm we have been describing is due to Canny (1986)

The output is a list of *edgel* positions, each with a strength $|\nabla S|$ and an orientation $\nabla S / |\nabla S|$

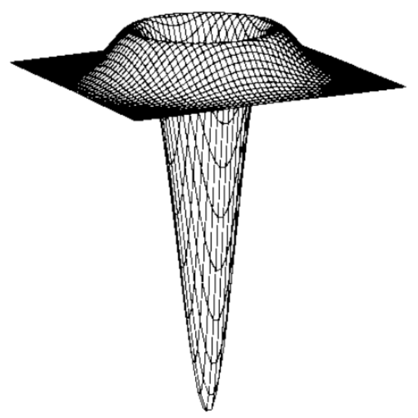
The Canny detector is a directional edge finder (both the gradient magnitude and direction are computed)

Marr-Hildreth Edge Detection

An alternative approach to edge detection was developed by Marr and Hildreth (1980)

Unlike the directional Canny edge detector, the Marr-Hildreth operator is isotropic

It finds zero-crossings of $\nabla^2 G_\sigma \circledast I$, where $\nabla^2 G_\sigma$ is the Laplacian of G_σ (recall the Laplacian operator $\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$)



$\nabla^2 G_\sigma$

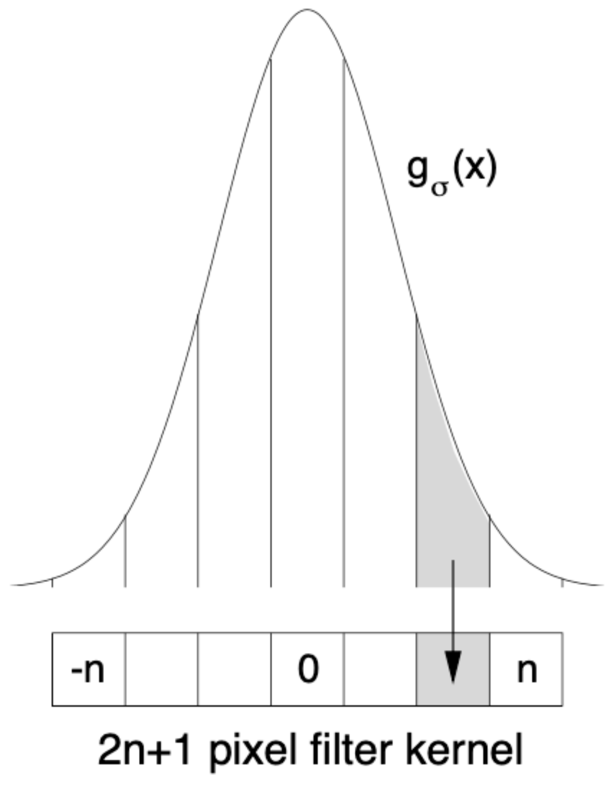
References:
Canny, J. A computational approach to edge detection. *TPAMI 1986*
D. Marr and E. Hildreth, "Theory of edge detection." Proc. of the Royal Society of London. Series B. Biological Sciences, 1980

Edge Detection: Implementation details

Truncated summations

In practice, the image and filter kernels are discrete quantities and the convolutions are performed as truncated summations:

$$S(x,y) = \sum_{u=-n}^{n} \sum_{v=-n}^{n} G_\sigma(u,v) I(x-u, y-v)$$



Truncation: how much is acceptable?

For acceptable accuracy, kernels are generally truncated so that the discarded samples are less than 1/1000 of the peak value.

σ	1.0	1.5	3	6
$2n+1$	7	11	23	45

Another computational trick!

The 2D convolutions would appear to be computationally expensive.

However, they can be decomposed into two 1D convolutions:

$$G_\sigma(x,y) \circledast I(x,y) = g_\sigma(x) \circledast [g_\sigma(y) \circledast I(x,y)]$$

The computational saving is:

$$\frac{(2n+1)^2}{2(2n+1)}$$

Edge Detection: Implementation details

Differentiation via convolution

Differentiation of the smoothed image is also implemented with a discrete convolution

By considering the Taylor-series expansion of $S(x,y)$ one can show that a simple finite-difference approximation to the first-order spatial derivative of $S(x,y)$ with respect to x is given by:

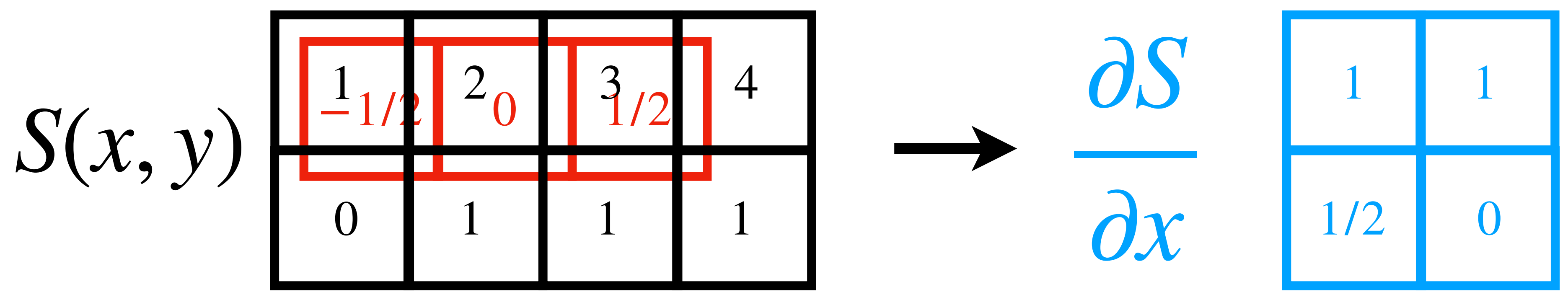
$$\frac{\partial S}{\partial x} = \frac{S(x+1,y) - S(x-1,y)}{2}$$

Implementing first order derivatives

We can calculate the finite-difference approximation to $\partial S / \partial x$ by convolving the rows of smoothed image samples, $S(x,y)$, with the 3-element kernel:

$1/2$	0	$-1/2$

Recall that when convolving, we flip the kernel and sum the element-wise products under each kernel position:



Note: this is often called "*valid*" convolution (the kernel is not allowed to run off the edges of $S(x,y)$).