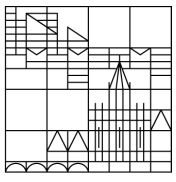
Chapter 2
Image Features



Lecture “Image Analysis and Computer Vision”
Winter semester 2014/15

Bastian Goldlücke

Overview

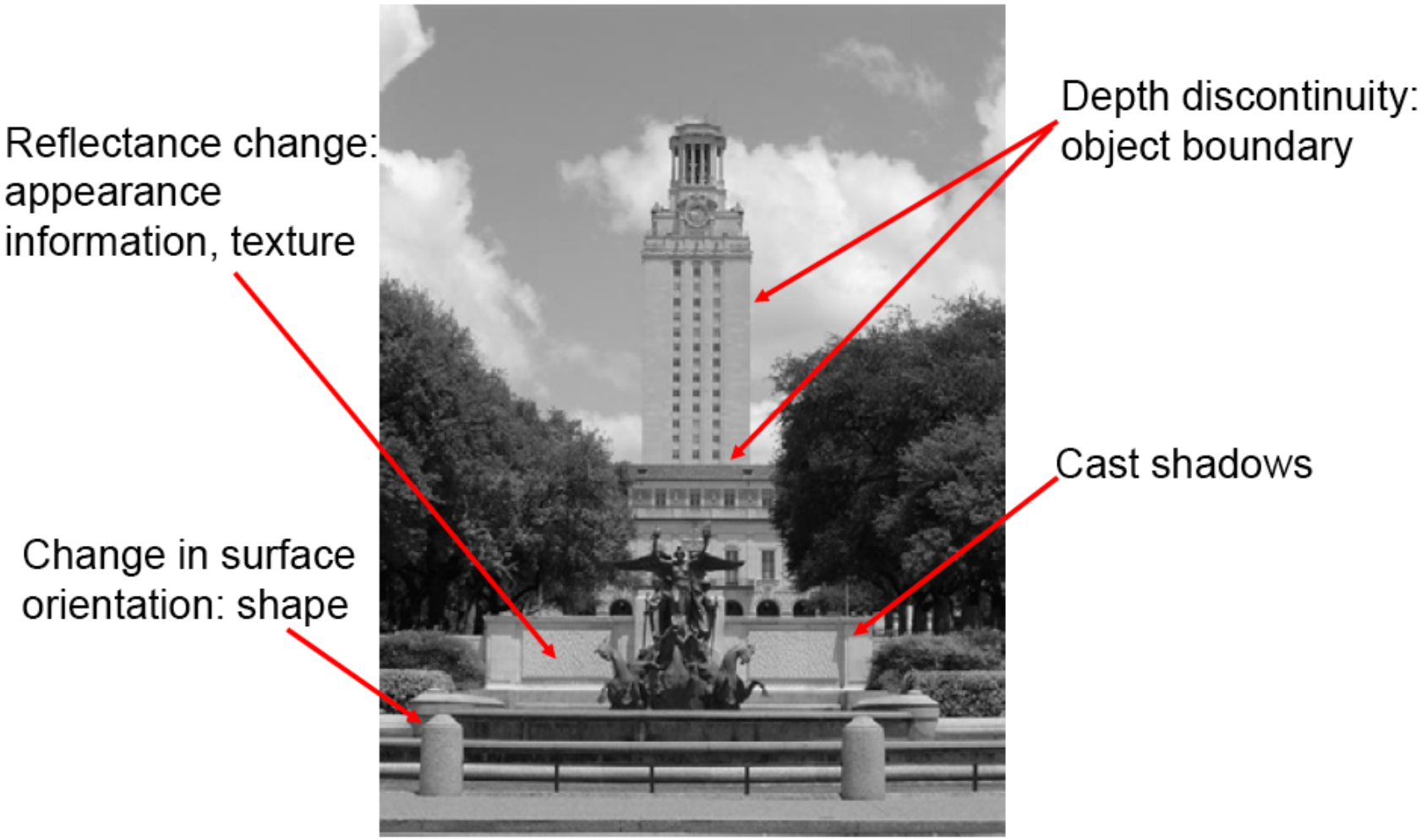
1. **Image edges and image derivatives**
   - Convolutions and derivatives
   - The “Derivative of Gaussian” filter
   - The Canny edge detector
   - Scale space
2. **Image features**
   - Introduction to image features
   - Corner detection
3. **Mathematical background: SVD and PCA**

Overview

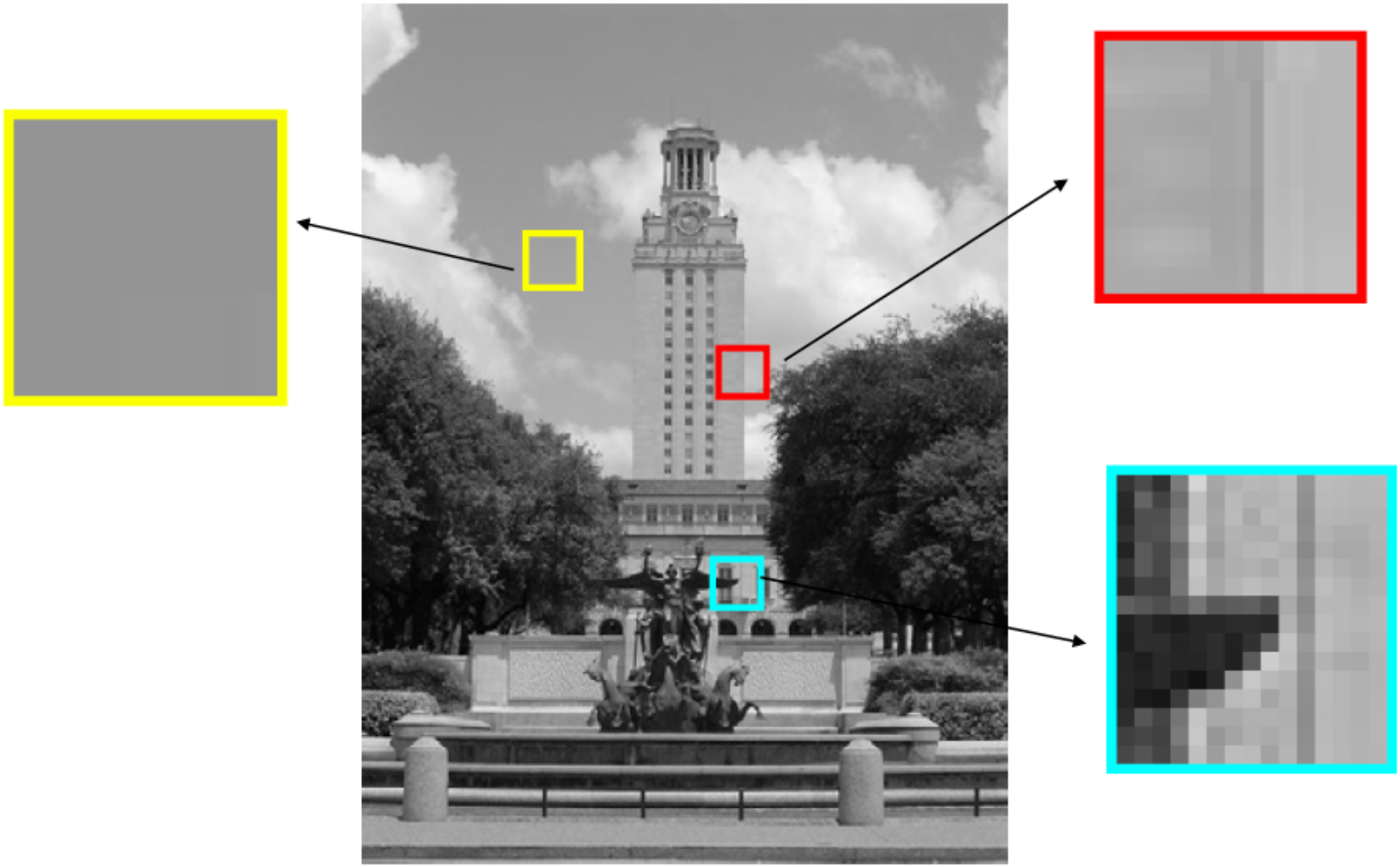
Image Edges

- **Goal:** identify sudden changes (discontinuities) in an image
  - Intuitively, most semantic and shape information from the image is encoded in the edges
  - More compact than pixels
- **Ideal:** artist's line drawing
  - Note: human artist is always using object-level knowledge

What causes an edge?

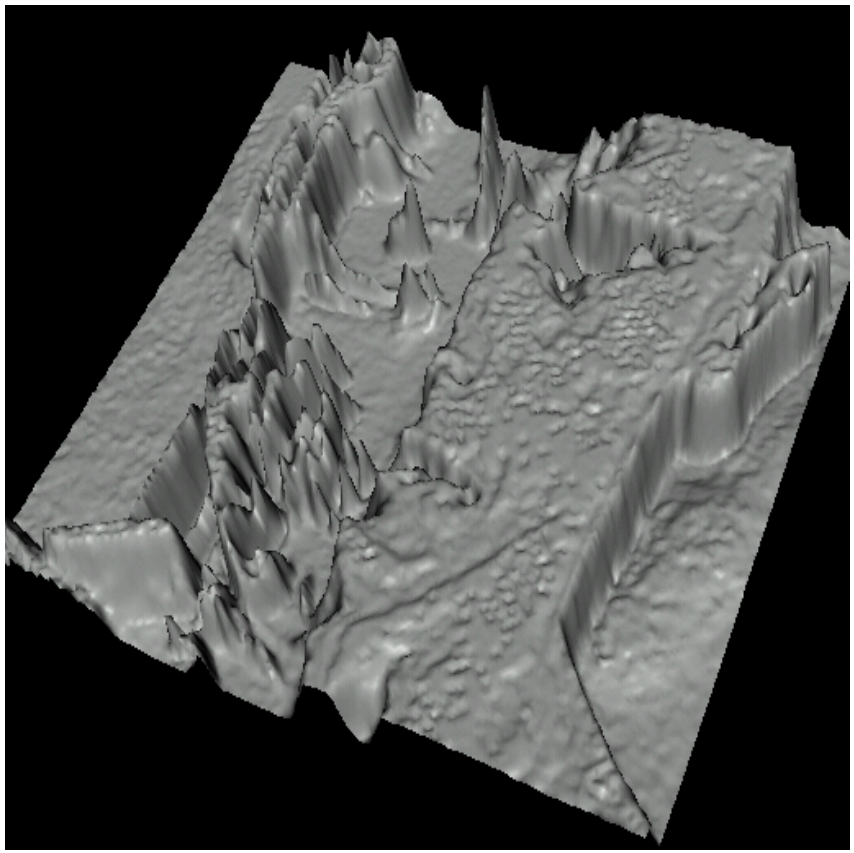


Looking at it more locally ...



Hints for edges in the image function?

Image of a cyborg

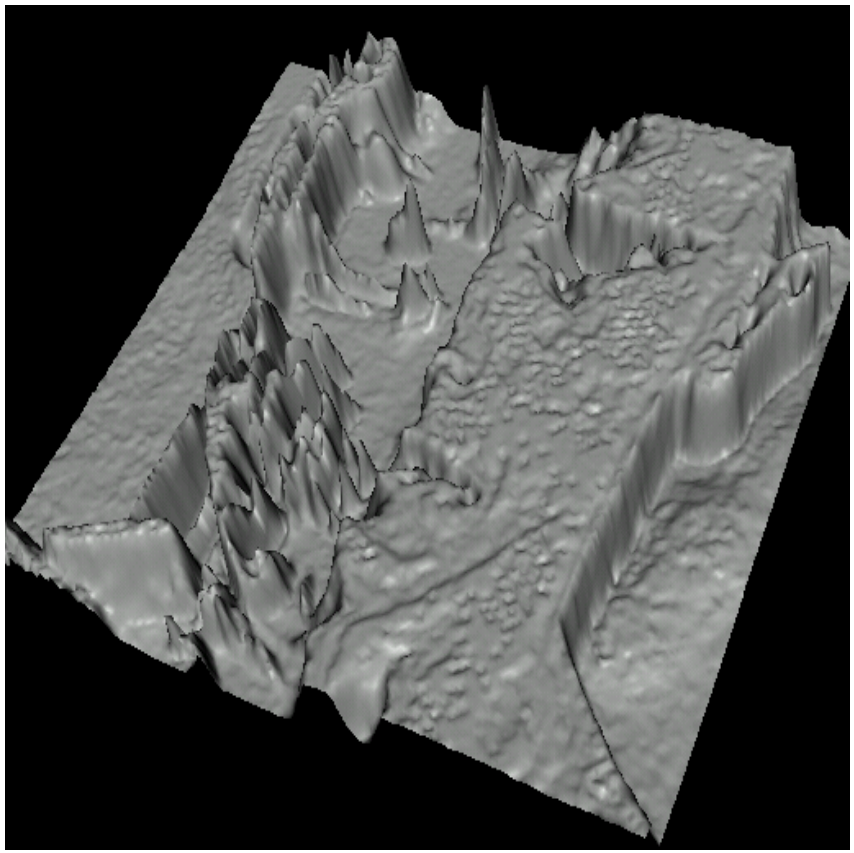


Intensity plotted as a height field

- Edges look like steep cliffs

Hints for edges in the image function?

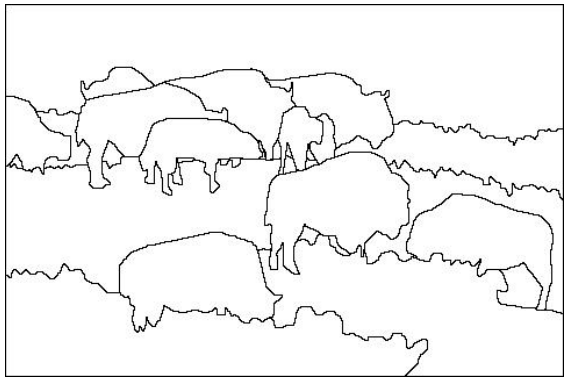
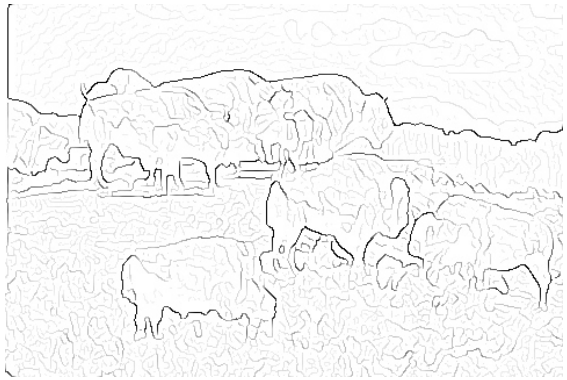
Image of a cyborg

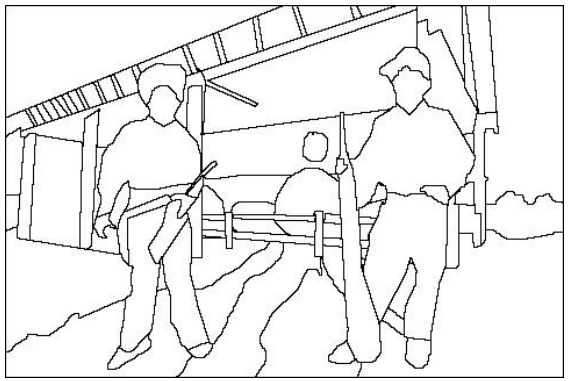


Intensity plotted as a height field

- Edges look like steep cliffs
- Intuition: characterize low-level edges via the gradient magnitude

Gradient magnitude vs. human edge perception


Part of the Berkeley segmentation database

See http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/

More causes of headaches ...

The problem of edge detection

- Image intensity changes, i.e. edges on a low level, are frequently not linked to the actual object contours:
  - possibly strong intensity changes in the object's interior (texture, shadows ...)
  - possibly no intensity change at the boundary (background texture similar, too dark ...)

The problem of edge detection

- Image intensity changes, i.e. edges on a low level, are frequently not linked to the actual object contours:
  - possibly strong intensity changes in the object's interior (texture, shadows ...)
  - possibly no intensity change at the boundary (background texture similar, too dark ...)

- Human edge annotation usually works on the object level, and involves complex mechanism like completion of invisible contours and shadow removal.

- Detecting high-level edges on the level of human perception is still an unsolved problem.

The problem of edge detection

- Image intensity changes, i.e. edges on a low level, are frequently not linked to the actual object contours:
  - possibly strong intensity changes in the object's interior (texture, shadows ...)
  - possibly no intensity change at the boundary (background texture similar, too dark ...)
- Human edge annotation usually works on the object level, and involves complex mechanism like completion of invisible contours and shadow removal.
- Detecting high-level edges on the level of human perception is still an unsolved problem.
- **We will for now be content with gradient-based edge detection.**

digital images are already sampled and quantized - how do we **approximate the gradient** of the original continuous signal?

Image derivatives

- **Variant 1:** Interpolate the image as a function on $\mathbb{R}^2$ (polynomial, spline, ...), differentiate that one, resample - e.g. derivative with respect to x:

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

Image derivatives

- **Variant 1:** Interpolate the image as a function on $\mathbb{R}^2$ (polynomial, spline, ...), differentiate that one, resample - e.g. derivative with respect to x:

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

- **Variant 2:** Use a finite difference approximation of the derivative, e.g.

Forward differences: $$(\partial_x f)_{i,j} \approx \frac{f(i+1,j) - f(i,j)}{h_x}$$

Backward differences: $$(\partial_x f)_{i,j} \approx \frac{f(i,j) - f(i-1,j)}{h_x}$$

Central differences: $$(\partial_x f)_{i,j} \approx \frac{f(i+1,j) - f(i-1,j)}{2h_x}$$

where h_x is the pixel size in x-direction (often normalized to 1).

Image derivatives

- **Variant 1:** Interpolate the image as a function on $\mathbb{R}^2$ (polynomial, spline, ...), differentiate that one, resample - e.g. derivative with respect to x:

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

- **Variant 2:** Use a finite difference approximation of the derivative, e.g.

Forward differences: $$(\partial_x f)_{i,j} \approx \frac{f(i+1,j) - f(i,j)}{h_x}$$

Backward differences: $$(\partial_x f)_{i,j} \approx \frac{f(i,j) - f(i-1,j)}{h_x}$$

Central differences: $$(\partial_x f)_{i,j} \approx \frac{f(i+1,j) - f(i-1,j)}{2h_x}$$

  where h_x is the pixel size in x-direction (often normalized to 1).

- **For now, we only take a closer look at variant 2 ...**
  we'll get back to interpolation when we discuss image transformations.

Finite differences can be written as convolutions !

- **Forward differences:**

$$(\partial_x f)_{i,j} = \frac{f(i+1,j) - f(i,j)}{h_x}, \quad \partial_x f = \boxed{1 \mid -1 \mid 0} * f$$

- **Backward differences:**

$$(\partial_x f)_{i,j} = \frac{f(i,j) - f(i-1,j)}{h_x}, \quad \partial_x f = \boxed{0 \mid 1 \mid -1} * f$$

- **Central differences:**

$$(\partial_x f)_{i,j} = \frac{f(i+1,j) - f(i-1,j)}{2h_x}, \quad \partial_x f = \boxed{1/2 \mid 0 \mid -1/2} * f$$

These are the convolution kernels, not correlation !

The same for derivatives in y-direction

- **Forward differences:**

$$(\partial_y f)_{i,j} = \frac{f(i,j+1) - f(i,j)}{h_y}, \quad \partial_y f = \begin{array}{|c|}\hline 1 \\ \hline -1 \\ \hline 0 \\ \hline \end{array} * f$$

- **Backward differences:**

$$(\partial_y f)_{i,j} = \frac{f(i,j) - f(i,j-1)}{h_y}, \quad \partial_y f = \begin{array}{|c|}\hline 0 \\ \hline 1 \\ \hline -1 \\ \hline \end{array} * f$$

- **Central differences:**

$$(\partial_y f)_{i,j} = \frac{f(i,j+1) - f(i,j-1)}{2h_y}, \quad \partial_y f = \begin{array}{|c|}\hline 1/2 \\ \hline 0 \\ \hline -1/2 \\ \hline \end{array} * f$$

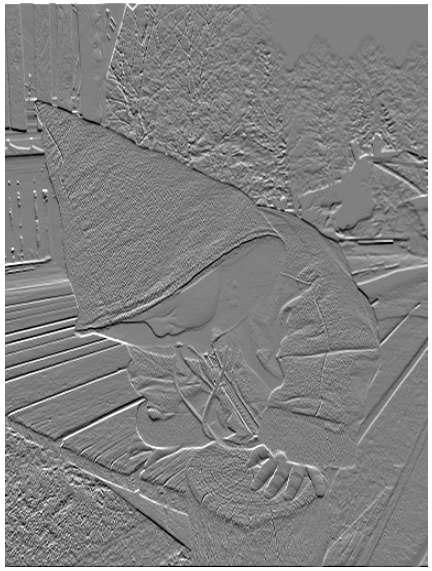
These are the convolution kernels, not correlation !

Example: simple edge detection with central differences

f

$1 - \|\nabla f\|$

$\partial_x f$



$\partial_y f$

Note: images scaled to range $[0, 1]$ for better visibility.

Reminder: gradient and gradient magnitude

- For a greyscale image f, the gradient ∇f is a vector field consisting of the partial derivatives in the different coordinate directions:

$$\nabla f = \begin{bmatrix} \partial_x f \\ \partial_y f \end{bmatrix}.$$

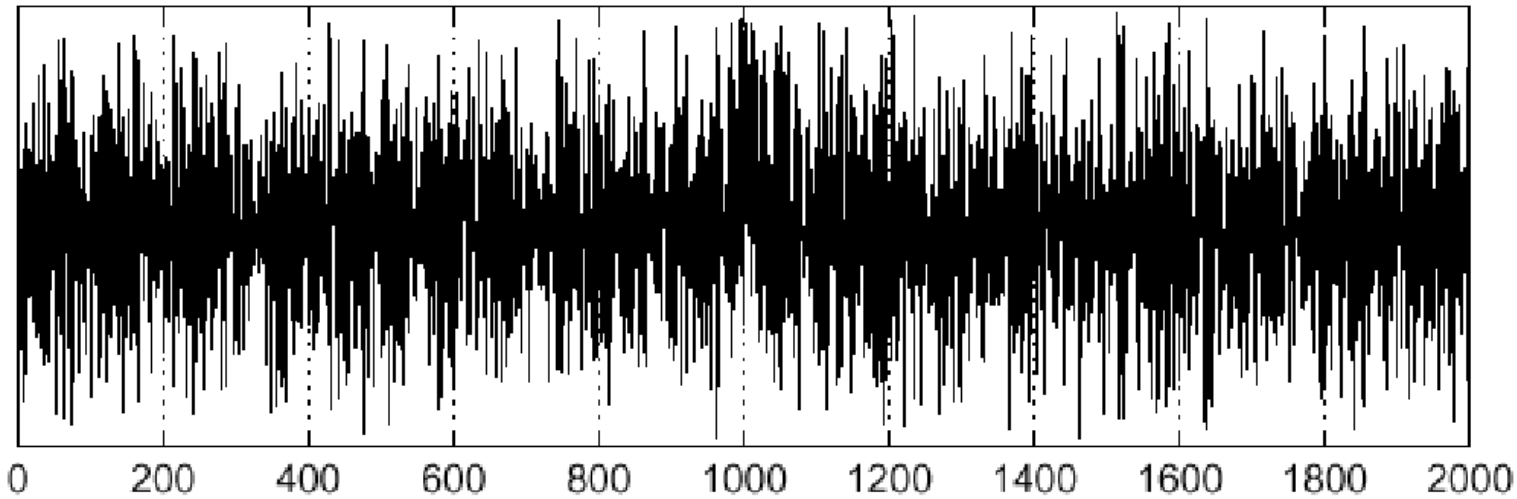
- The gradient can thus be interpreted as a vector-valued image with $n = m = 2$. At each pixel, the gradient points in the direction of the strongest change.

- The magnitude of the gradient is just the length of the vector in every point. It is a scalar image, defined point-wise as

$$\|\nabla f\| = \sqrt{(\partial_x f)^2 + (\partial_y f)^2}.$$

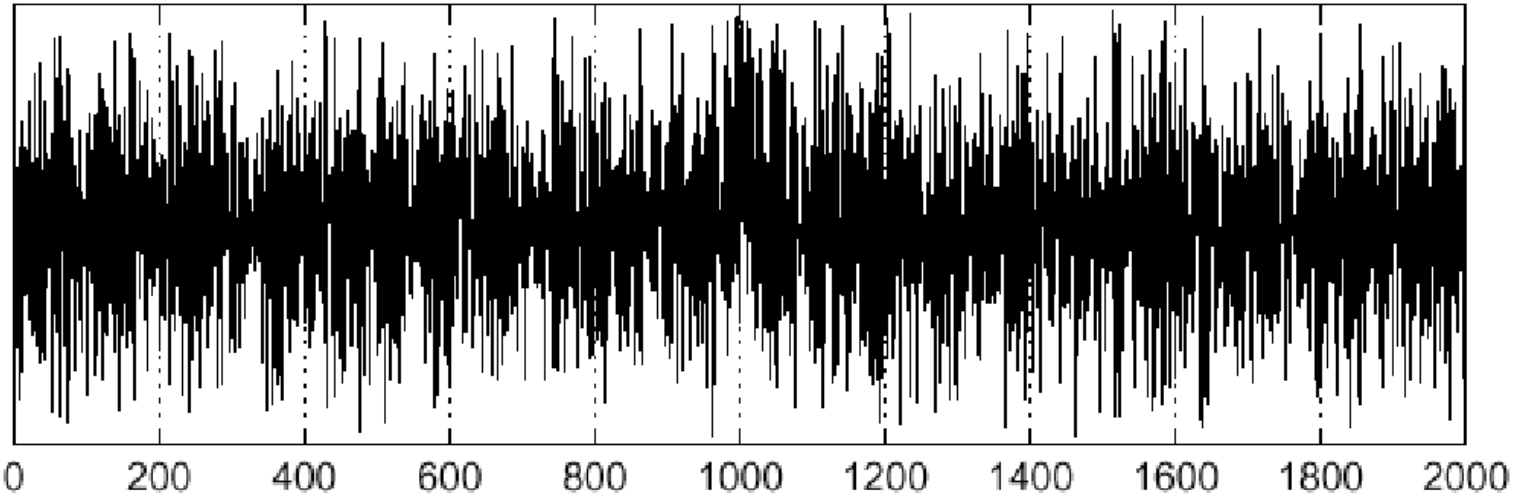
- The larger the magnitude, the stronger the change in gradient direction.

The problem when using finite differences ... noise

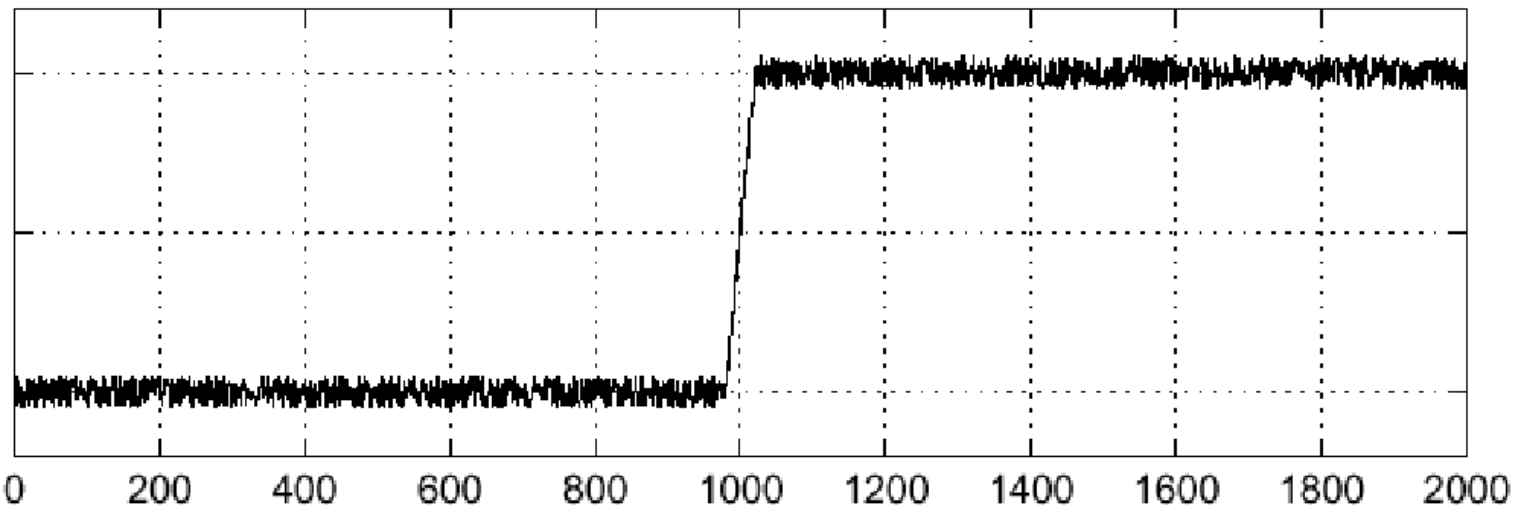


this is the derivative of a noisy 1D signal *f* - but where is the edge?

The problem when using finite differences ... noise



this is the derivative of a noisy 1D signal *f* - but where is the edge?



the original signal *f* does not even look too bad.

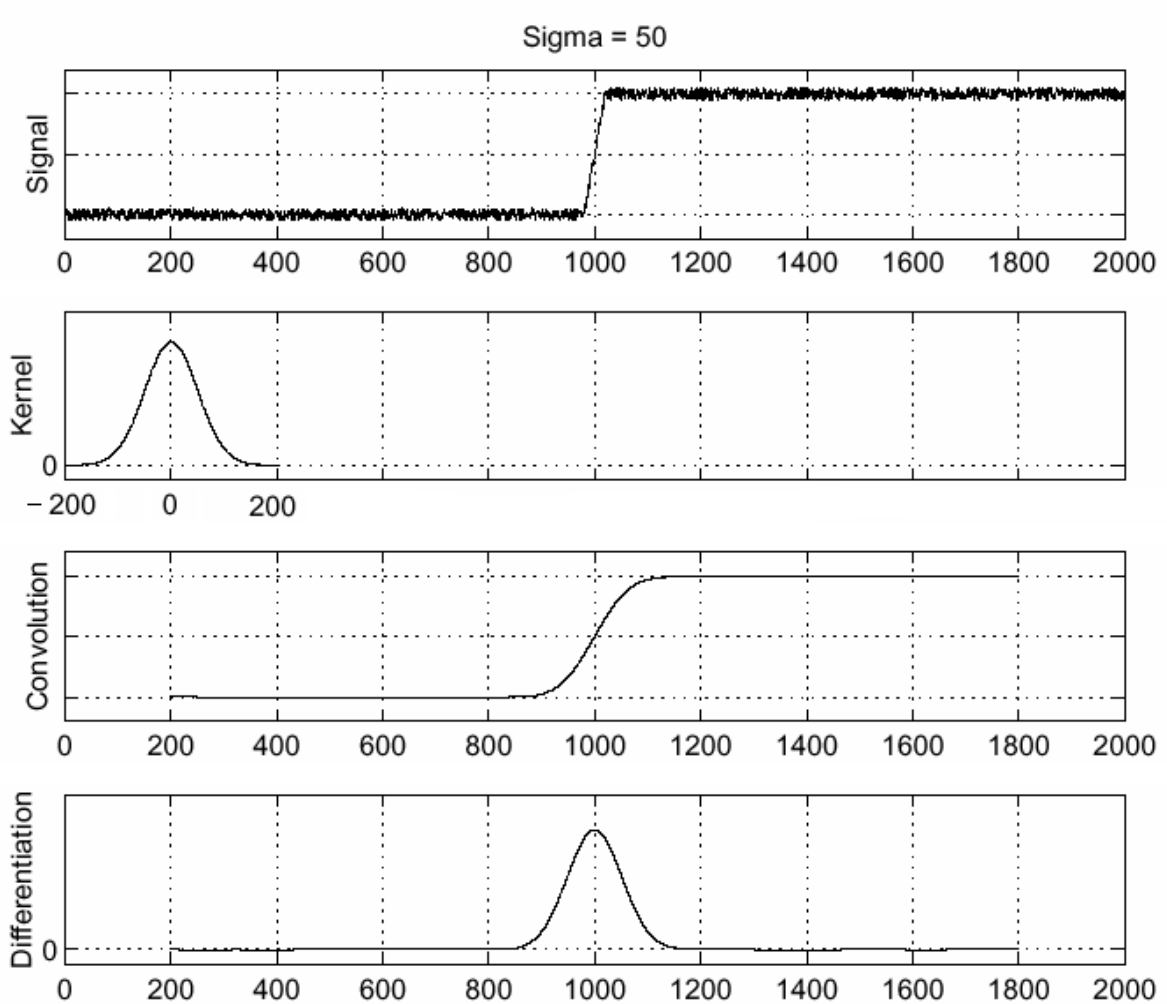
Idea: smooth the signal, then compute derivative

f

g

$f * g$

$\partial_x(f * g)$



Can this be done more efficiently?

The continuous form of the convolution

Let f, g be (real- or complex-valued) functions on $\mathbb{R}^n$ which satisfy some technical constraints (integrable and becoming small fast enough). Then their convolution $f * g$ is defined for each $x \in \mathbb{R}^n$ as

$$(f * g)(x) = \int_{\mathbb{R}^n} f(x - u)\, g(u)\, \mathrm{d}u.$$

Note: Basically, this is the same as the discrete formula with integrals instead of sums. All algebraic rules for discrete convolution (i.e. commutative, associative, ...) still hold.

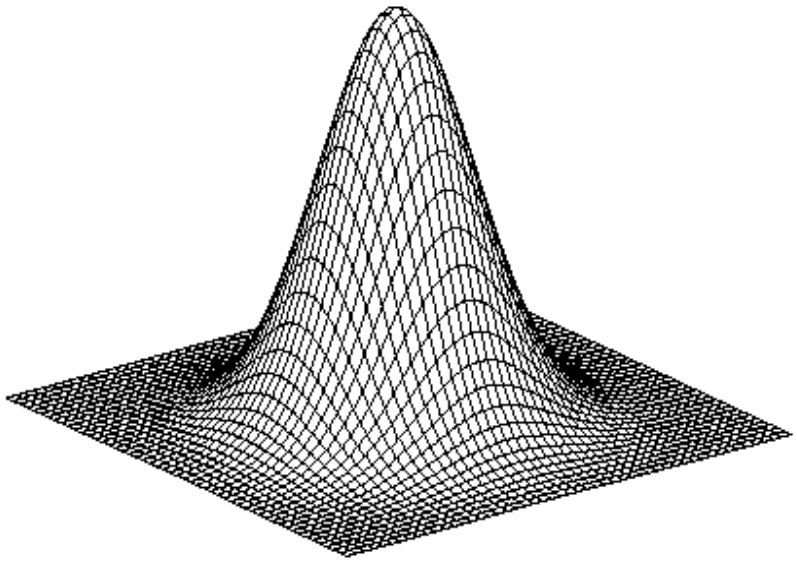
Derivative theorem for the continuous convolution

Under some technical conditions on f and g, the partial derivatives with respect to the ith variable satisfy

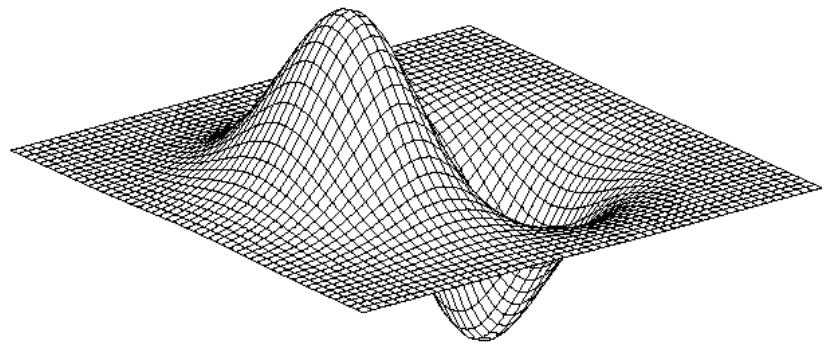
$$\partial_i(f * g) = (\partial_i f) * g = f * (\partial_i g).$$

Taking a partial derivative and computing the continuous convolution commutes.

The “Derivative of Gaussian” convolution filters



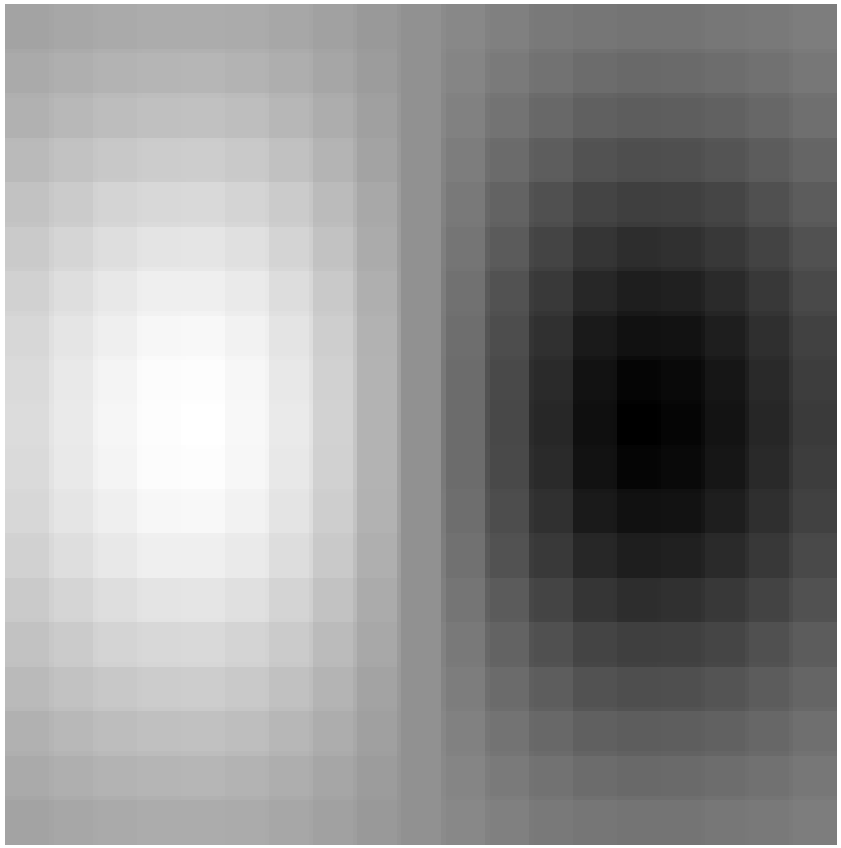
Gaussian $h_\sigma(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$



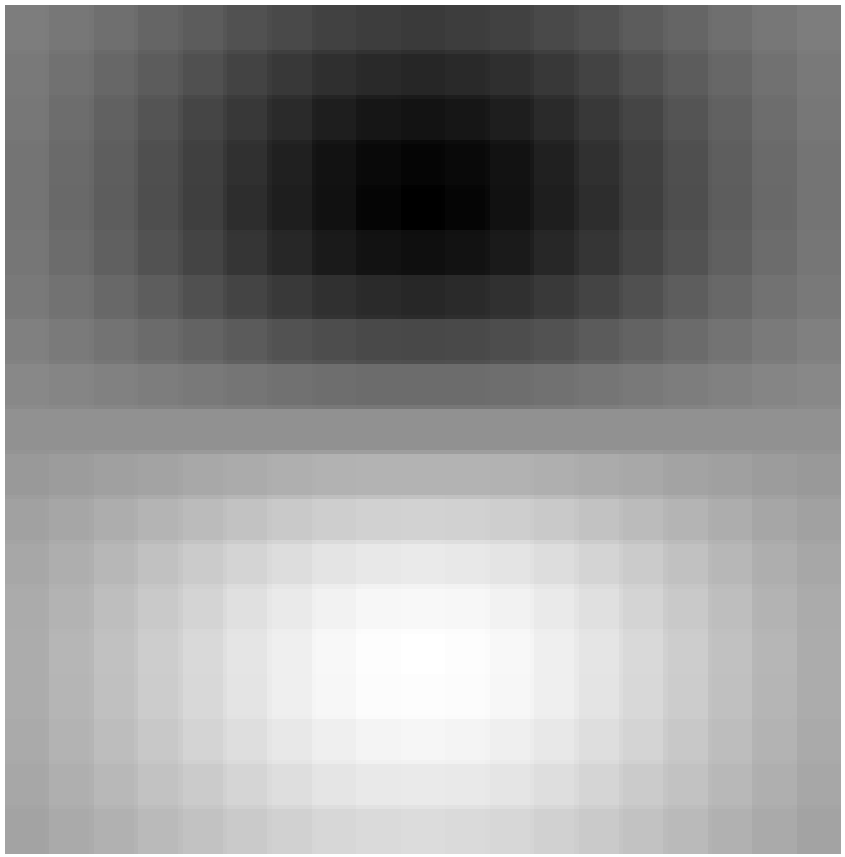
Derivative $\partial_x h_\sigma(x, y)$

Images: N. Snavelny

The “Derivative of Gaussian” convolution filters



x direction



y direction

When convolved with an image, they approximate a derivative of the image pre-filtered with a Gaussian of standard deviation σ.

Advantages of taking the derivative of the filter

- **Efficiency:** Can be pre-computed and thus removes the need for a second convolution operation.
- **Accuracy:** If filter function is known, derivatives can be computed analytically.

Advantages of taking the derivative of the filter

- **Efficiency:** Can be pre-computed and thus removes the need for a second convolution operation.
- **Accuracy:** If filter function is known, derivatives can be computed analytically.

Implementation note: make sure the filter is normalized to zero after discretization, i.e. all elements sum up to zero - for a constant signal, the derivative should be zero.

Remainder of section: one of the most well-known edge detectors based on gradient magnitude: the “Canny” edge detector.

[J. Canny, A Computational Approach To Edge Detection*, TPAMI 1986.]*

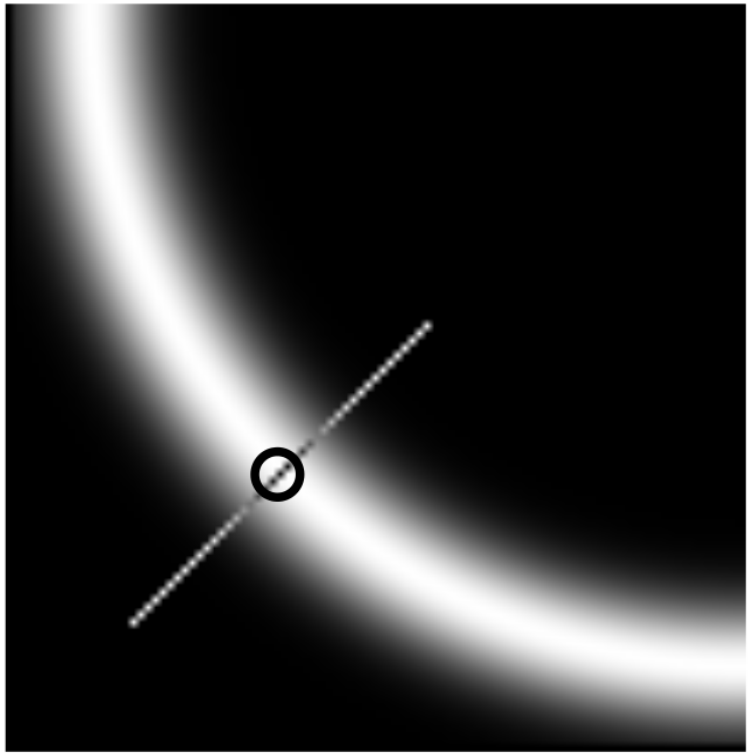
An example image

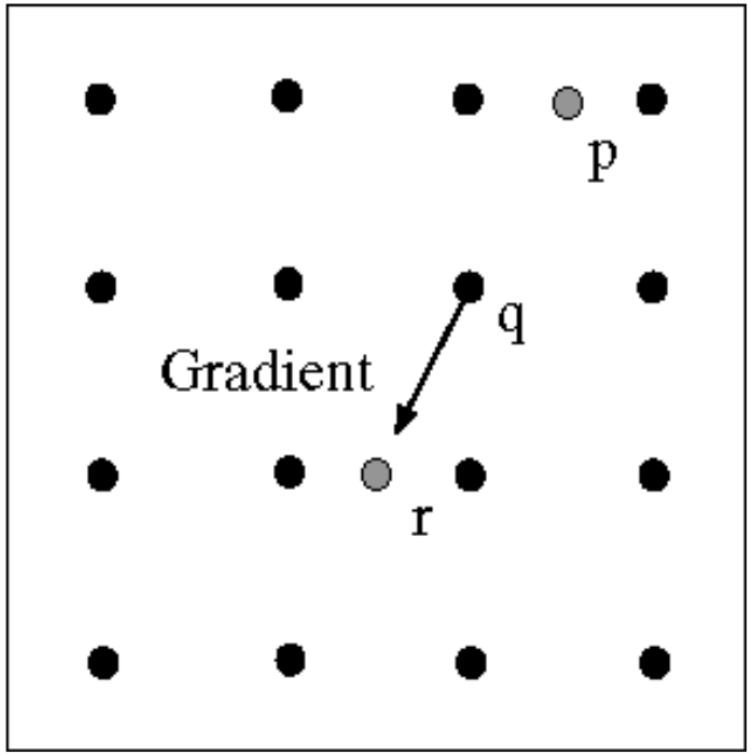
Gradient magnitude at certain scale σ

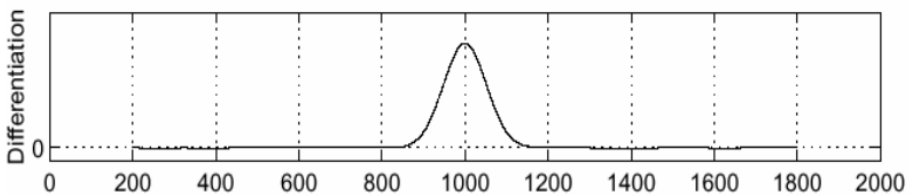
Derivatives computed with “Derivative of Gaussian” filters.

Gradient magnitude at certain scale σ

Non-maxima suppression idea







Keep only pixels which are on a local maximum along the gradient direction (interpolation required).

After non-maxima suppression (thinning)

Summary of algorithm: Canny edge detector

Matlab:

```
edge(image,'canny')
edge(image,'canny', [], sigma)
edge(image,'canny', [tmin, tmax], sigma)
```

- Filter image with derivative of Gaussian at scale σ
- Find magnitude and orientation of gradient
- Non-maximum suppression
- Linking and thresholding (hysteresis):
  - Define two thresholds: low (`tmin`) and high (`tmax`)
  - Use the high threshold to start edge curves and the low threshold to continue them

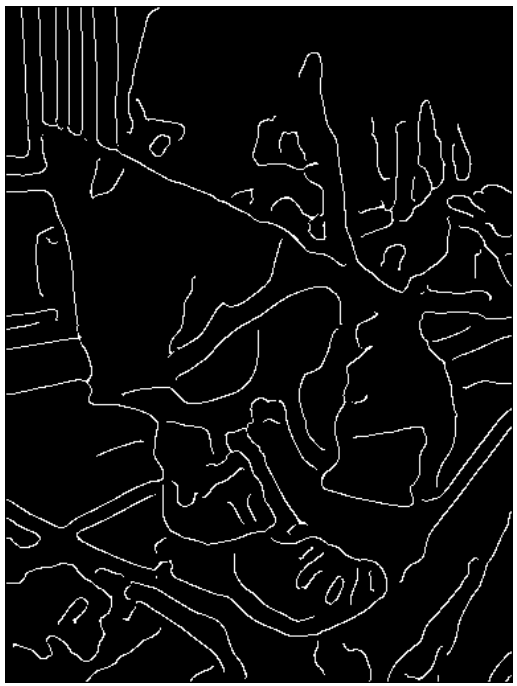
We did not discuss the last step, but it's something of a hack and not very interesting all things considered.

Examples: Canny edges

Computed with `edge( image, 'canny', [], sigma)`

`image`

$\sigma = 1.0$



$\sigma = 5.0$

The scale space idea

Consider Gaussian filtered image, different standard deviations σ ...

$\sigma = 0$

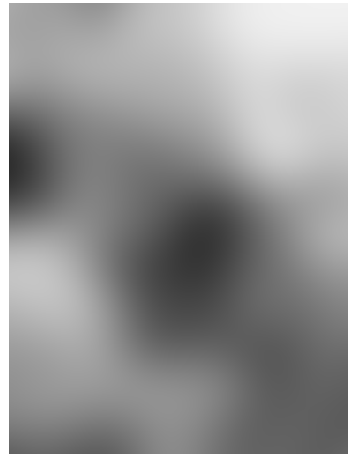
$\sigma = 1.0$

$\sigma = 2.0$

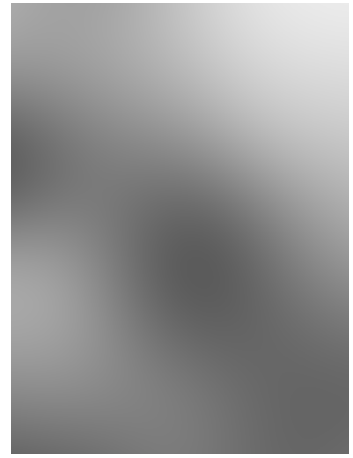
$\sigma = 4.0$

$\sigma = 8.0$

$\sigma = 16.0$



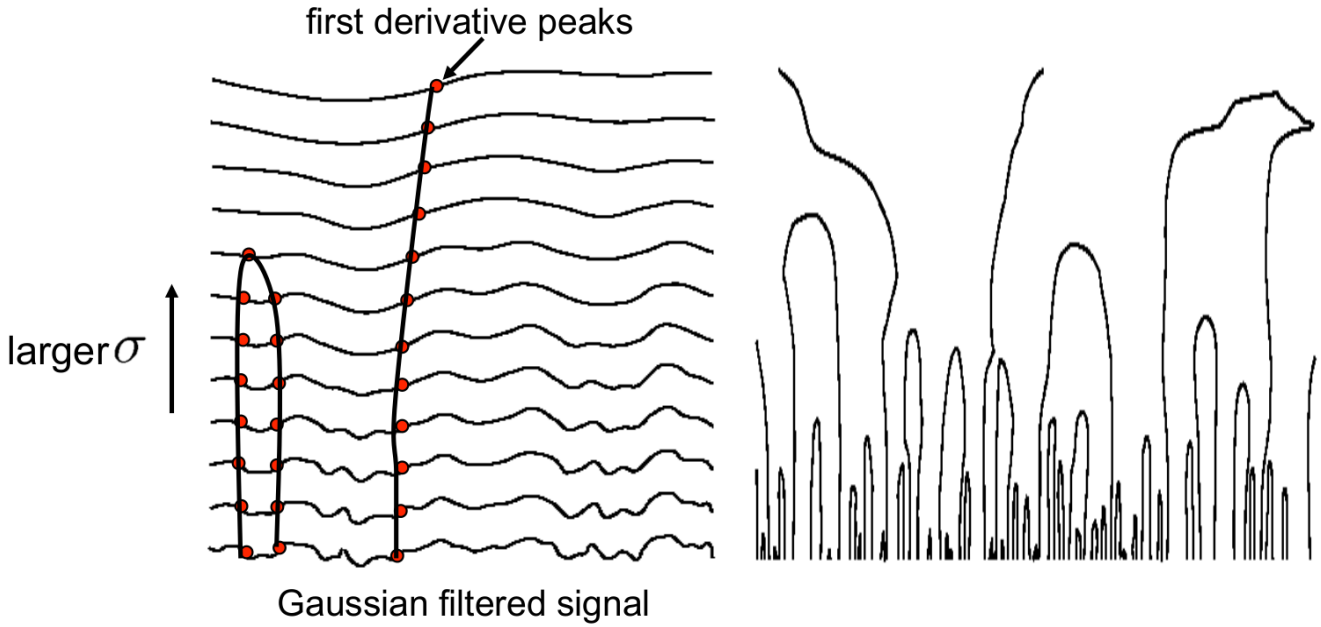
$\sigma = 32.0$



$\sigma = 64.0$

Increasing σ gradually removes detail ...

The scale space idea

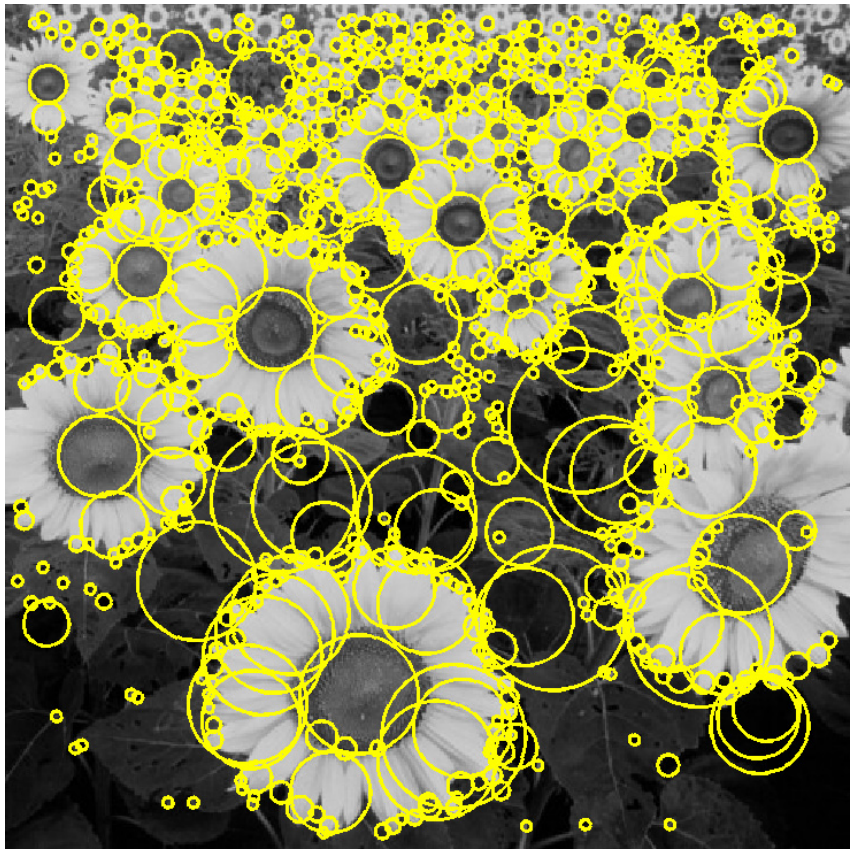


- edge position may shift with increasing scale σ
- two edges may merge with increasing scale
- an edge may not split into two with increasing scale
- note: there are other ways to construct scale spaces, but the Gaussian is the “canonical” one in a certain sense.
- the idea is very powerful, and sparked the conference series *Scale Space and Variational Methods (SSVM)*, popular among mathematicians who are into image analysis.

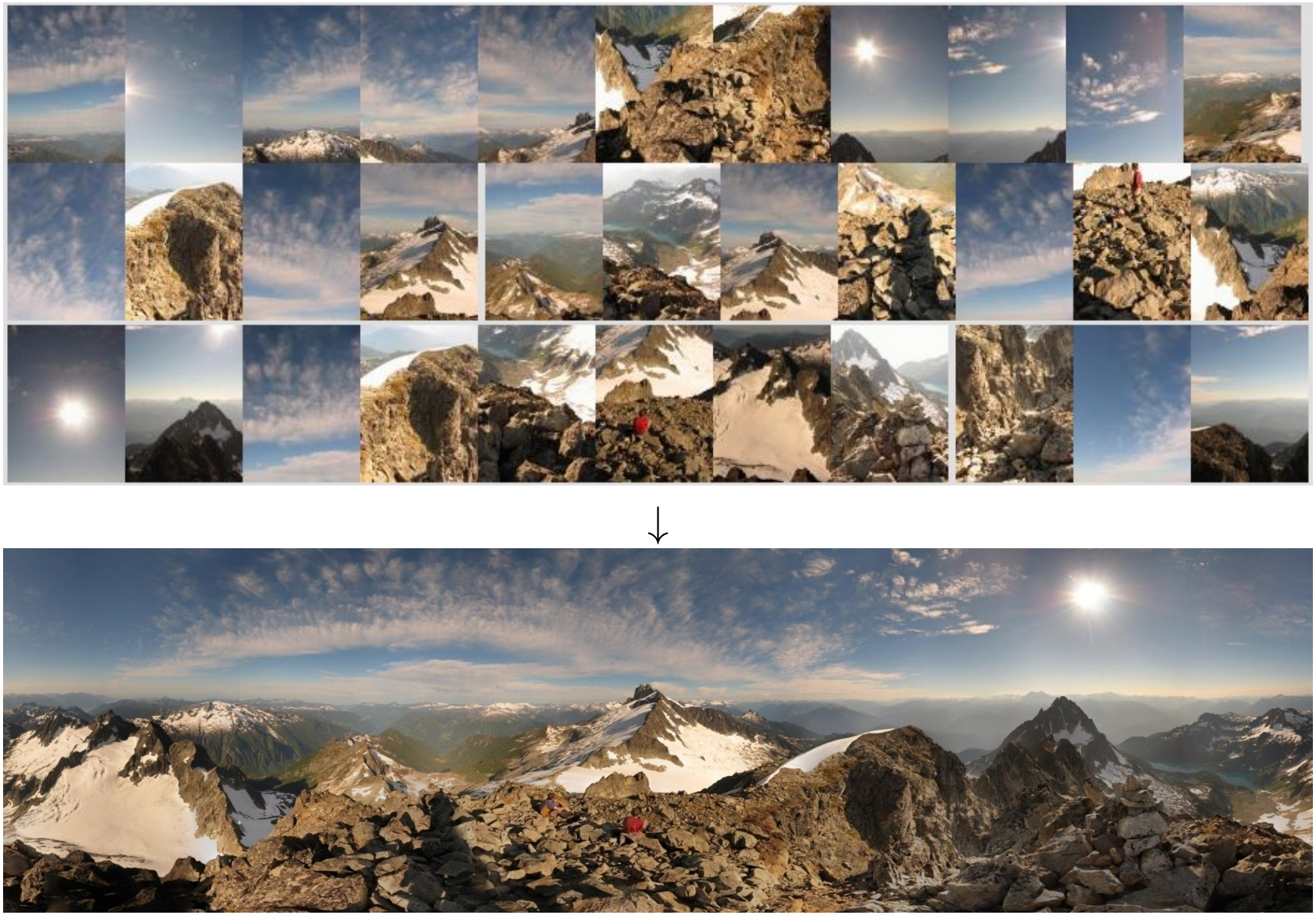
Maybe first paper on scale space: [Witkin 1983]

Overview

Feature extraction: corners and blobs



Motivation: automatic panoramas



Slide: R. Urtasun, Image: Matt Brown

Motivation: automatic panoramas

- HDPan (Microsoft Research)

  `http://research.microsoft.com/en-us/um/redmond/groups/ivm/HDView/`

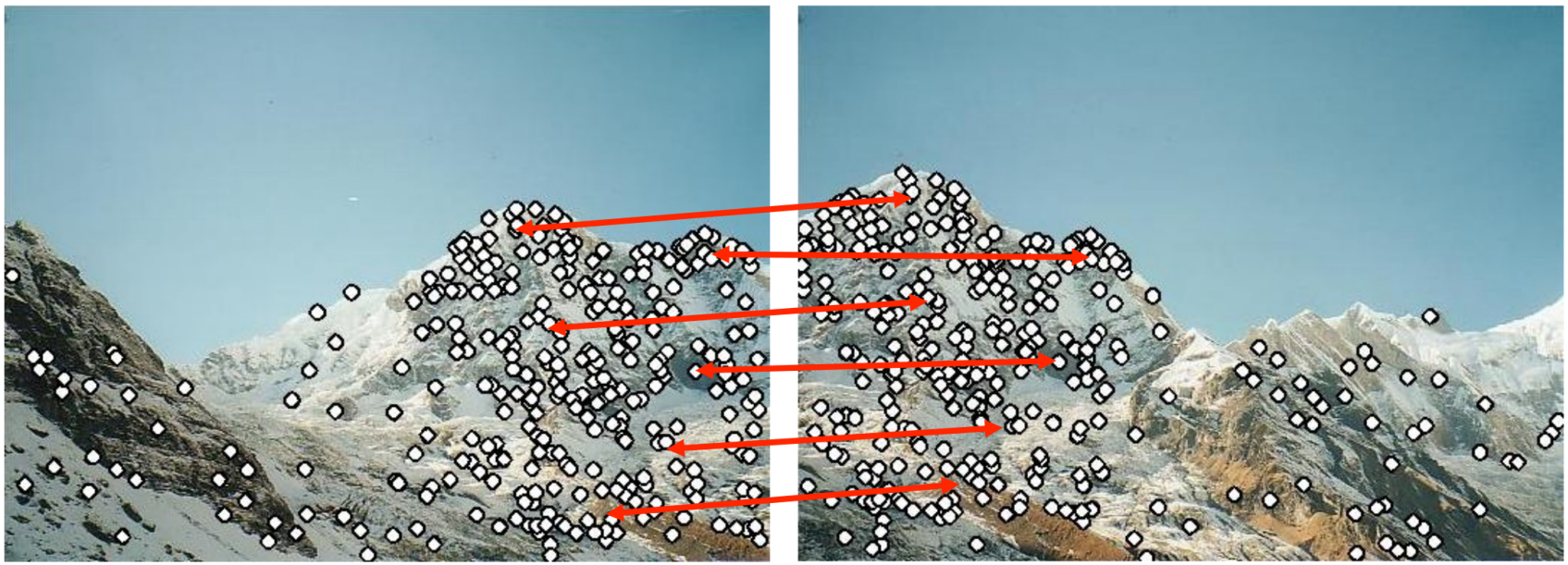
- GigaPan

  `http://gigapan.org`

How to combine two images to form a panorama?

Input: two images

How to combine two images to form a panorama?



Step 1: feature detection and matching

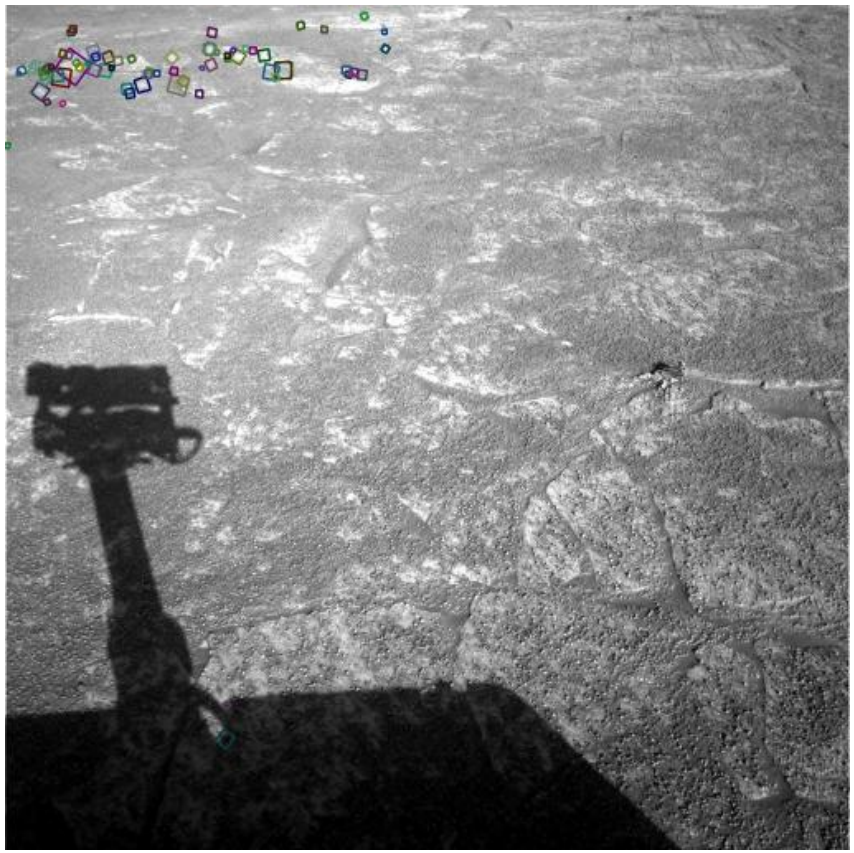
How to combine two images to form a panorama?

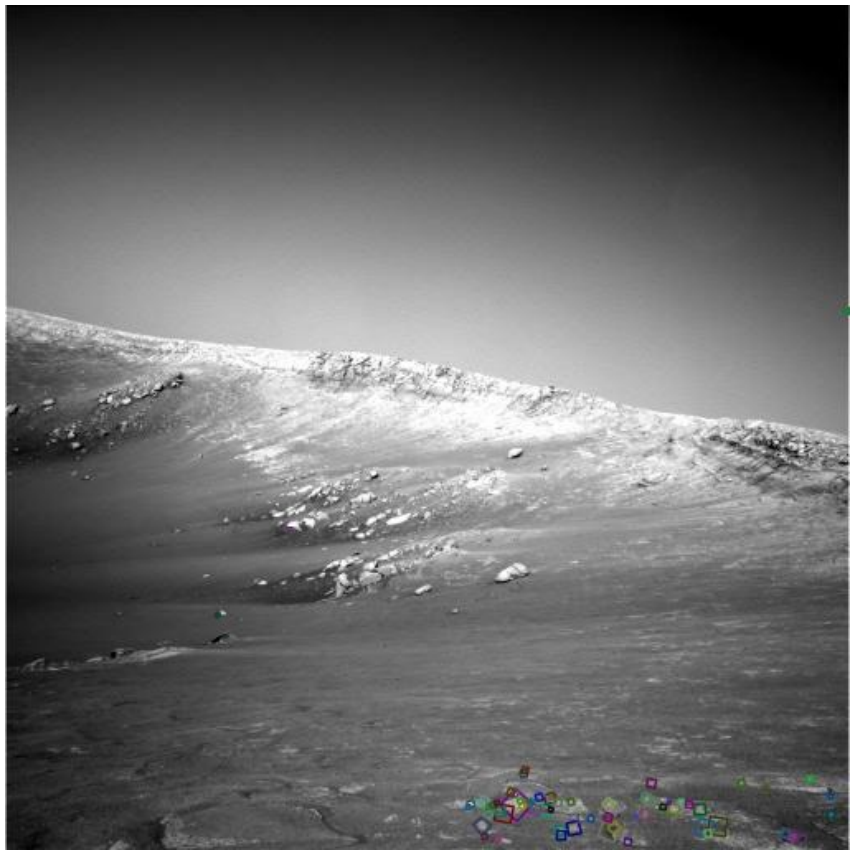
Step 2: image alignment

A difficult case ...

NASA Mars rover images

A difficult case ...





NASA Mars rover images with SIFT features

Three steps to matching ...

- **Detection:** Identify the interest points.
- **Description:** Extract vector feature descriptor around each interest point.
- **Matching:** Determine correspondence between descriptors in two views.

Note: We are still at step 1 ;).

Just a few goals for detectors ...

- **Locality:** features are local, so robust to occlusion and clutter
- **Quantity:** hundreds or thousands in a single image
- **Distinctiveness:** can differentiate a large database of objects
- **Efficiency:** real-time performance achievable
- **Geometric invariance:** translation, rotation, scale, ...
- **Photometric invariance:** brightness, exposure, ...

Yes, that's a lot to ask for ...

... but it's very worthwhile to think hard about it

Feature points are used for:

- Image alignment (e.g. mosaics, panoramas)
- 3D reconstruction
- Motion tracking
- Object recognition
- Indexing and database retrieval
- Robot navigation
- ...

What points to choose?

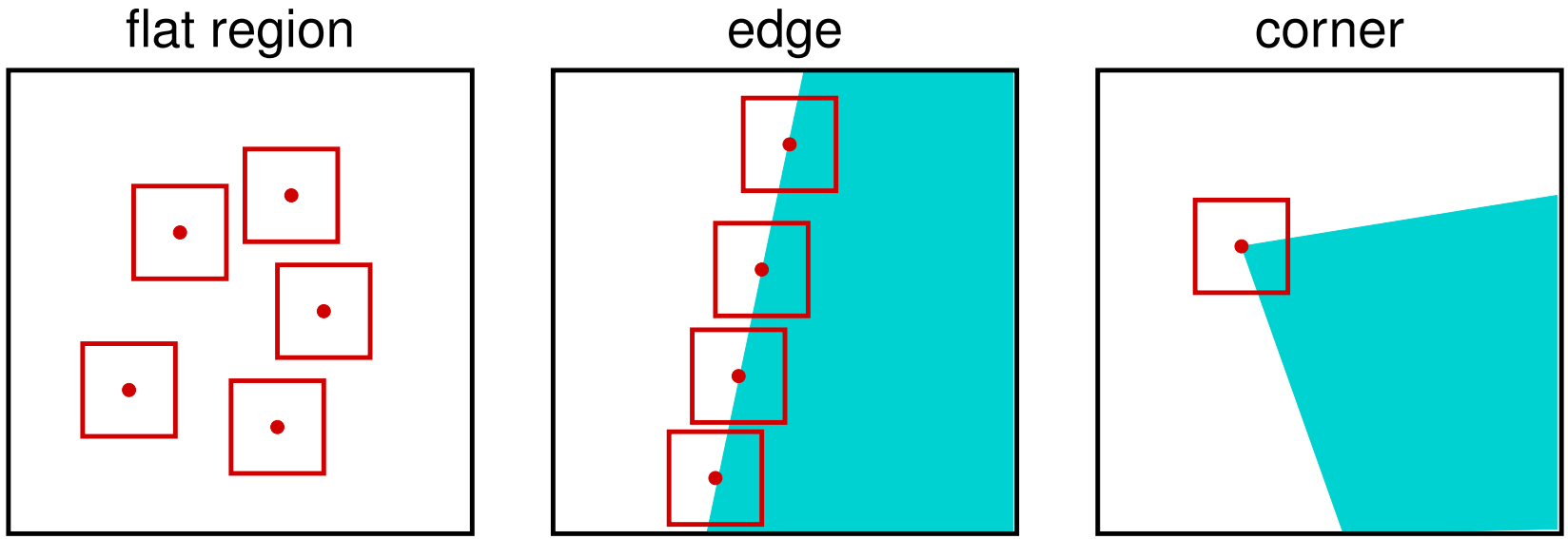
What points to choose?

- Look for image regions that are **unusual**: lead to unambiguous matches in other images
- How to define “unusual”?

What points to choose?

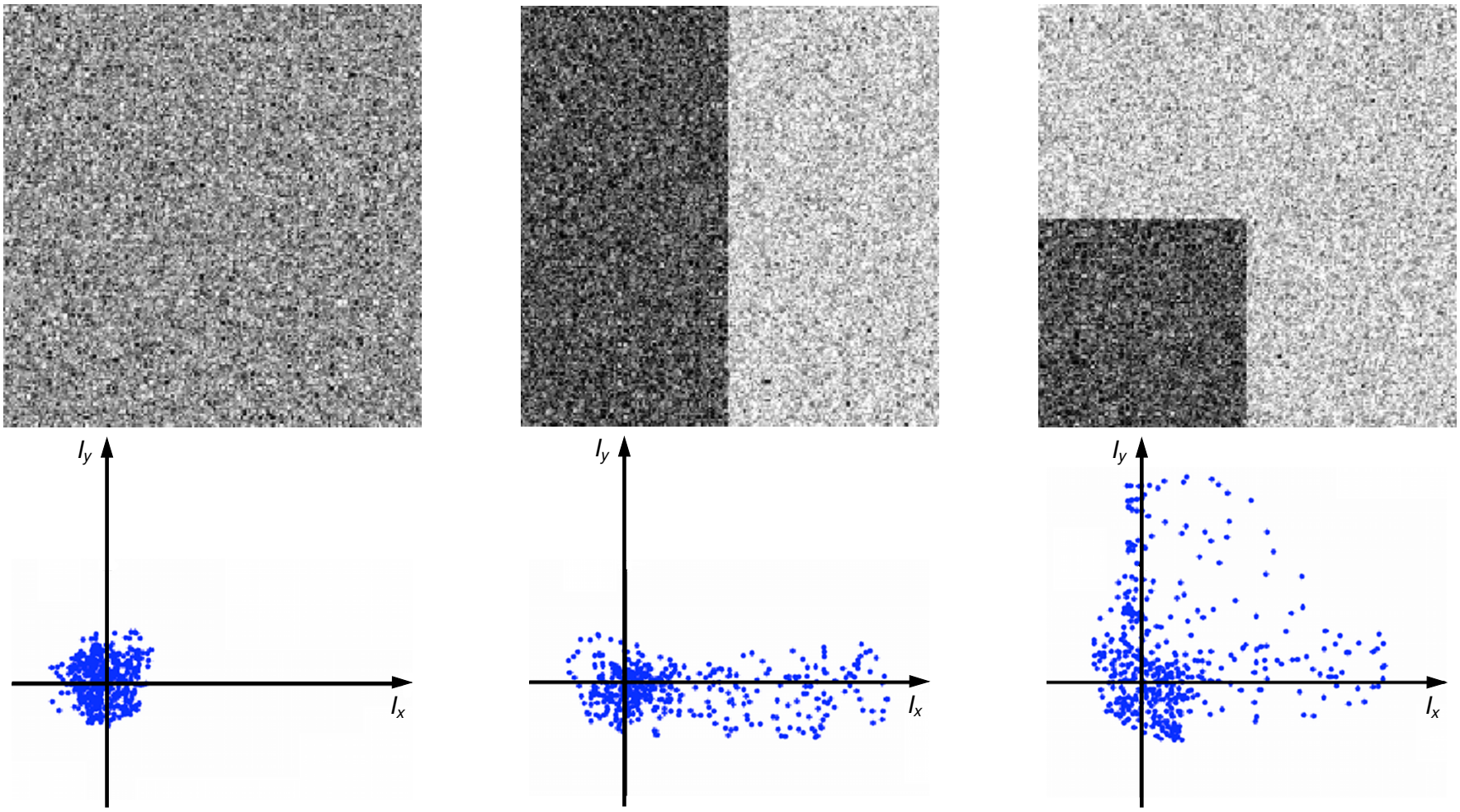
- Textureless patches are nearly impossible to localize.
- Patches with large contrast changes (gradients) are easier to localize.
- But straight line segments at a single orientation suffer from the **aperture problem**, i.e. it is only possible to align the patches along the direction normal to the edge direction.
- Gradients in at least two (significantly) different orientations are the easiest, these can be found at image **corners**.

Image features must be well localized



no unique positioning on flat regions or edges,
so features should lie on corners

Mathematical characterization of edges and corners



The distribution of the partial derivatives is characteristic
$\Rightarrow$ we need to consider statistics again.

Overview

Note: this section being about mathematics, much will be going on on the blackboard. I strongly recommended that you take some notes, otherwise it might be hard to understand the slides once you are back home.

The singular value decomposition (SVD)

Let $A \in \mathbb{R}^{m \times n}$ be a $m \times n$ matrix (wlog $m \geq n$). Then there exists a factorization of the form

$$A = U \Sigma V^T,$$

where

- $U \in \mathbb{R}^{m \times m}$ is an *orthogonal matrix*, i.e. $UU^T = U^T U = I_m$.
- $V \in \mathbb{R}^{n \times n}$ is also orthogonal, i.e. $VV^T = V^T V = I_n$.
- $\Sigma \in \mathbb{R}^{m \times n}$ is a *diagonal matrix* whose diagonal consists of sorted real numbers $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n$.

- *In the above factorization, the σ_i are called the* singular values *of A. They are uniquely determined by A, but the matrices U and V are not.*
- *Interpretation: if A is viewed as a linear map from $\mathbb{R}^n$ to $\mathbb{R}^m$, there always exists a change of basis of the domain and co-domain (given by V and U, respectively) in which A becomes diagonal.*

SVD and Eigendecomposition

- The columns of V are Eigenvectors of $A^T A$.
- The columns of U are Eigenvectors of AA^T.
- The non-zero singular values are the square roots of the non-negative Eigenvalues.

To be continued ...