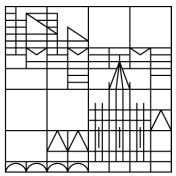
Chapter 2
Detecting Features and Patterns



University of
Konstanz

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

Bastian Goldlücke

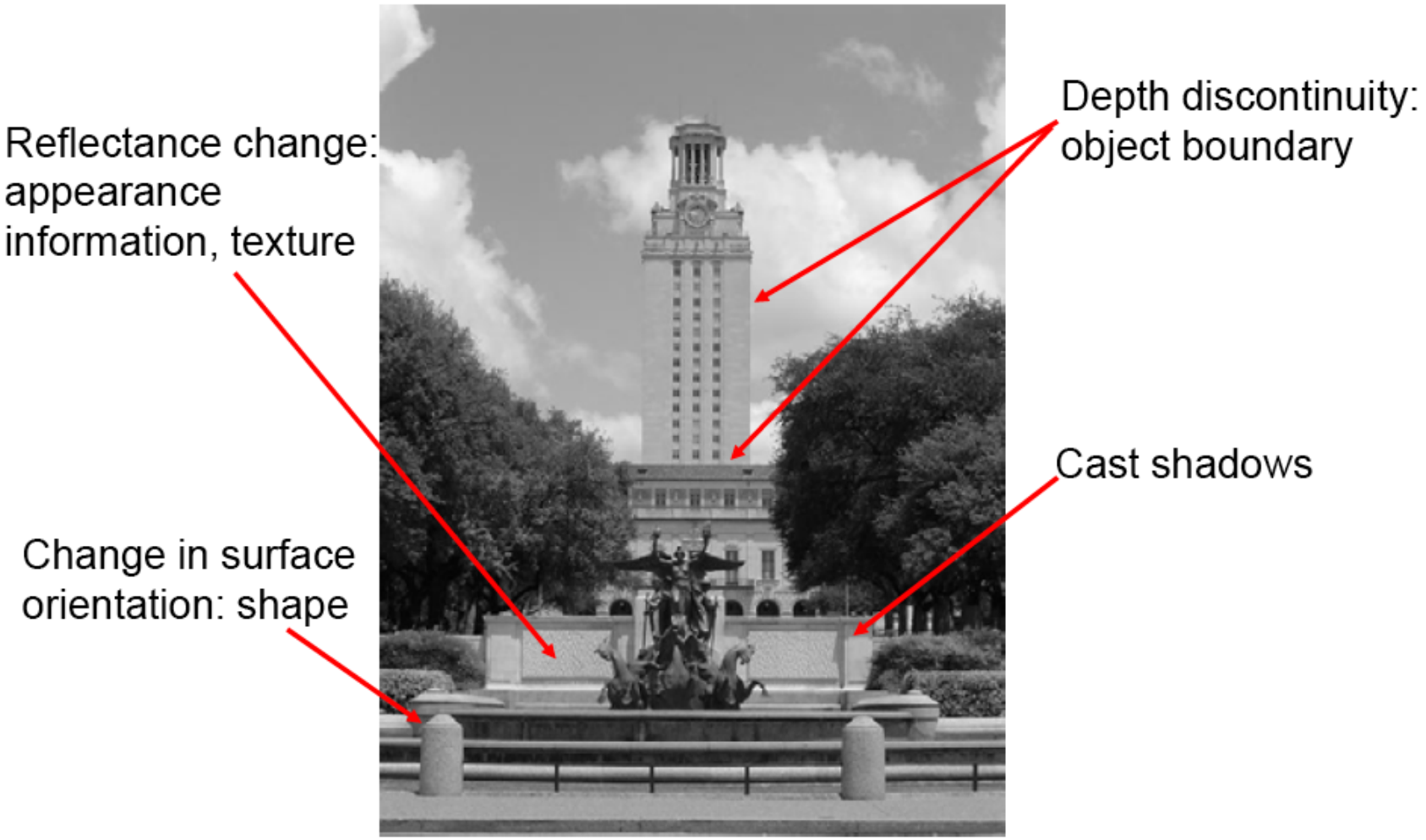
Overview

Overview

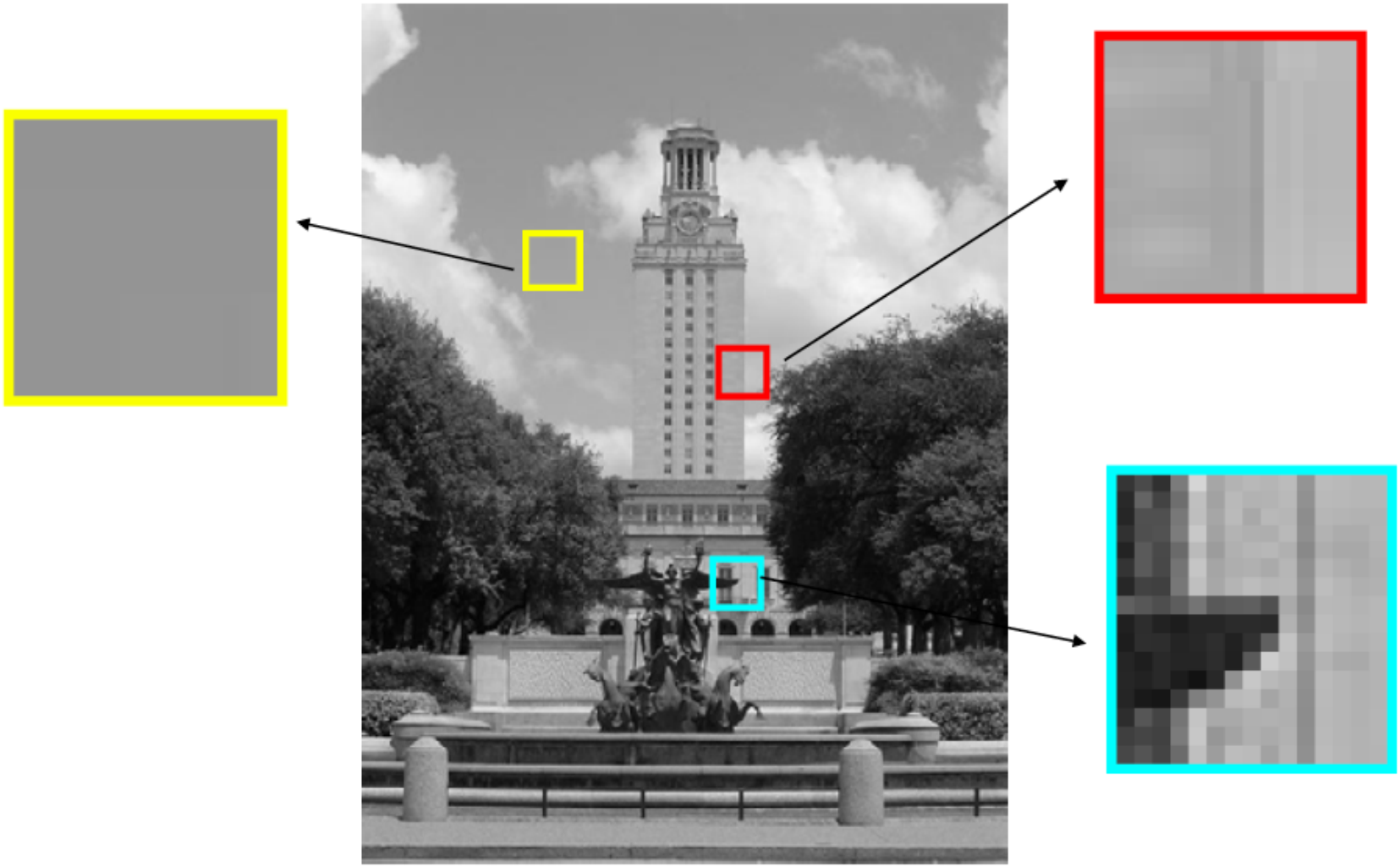
Image Edges

- **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?

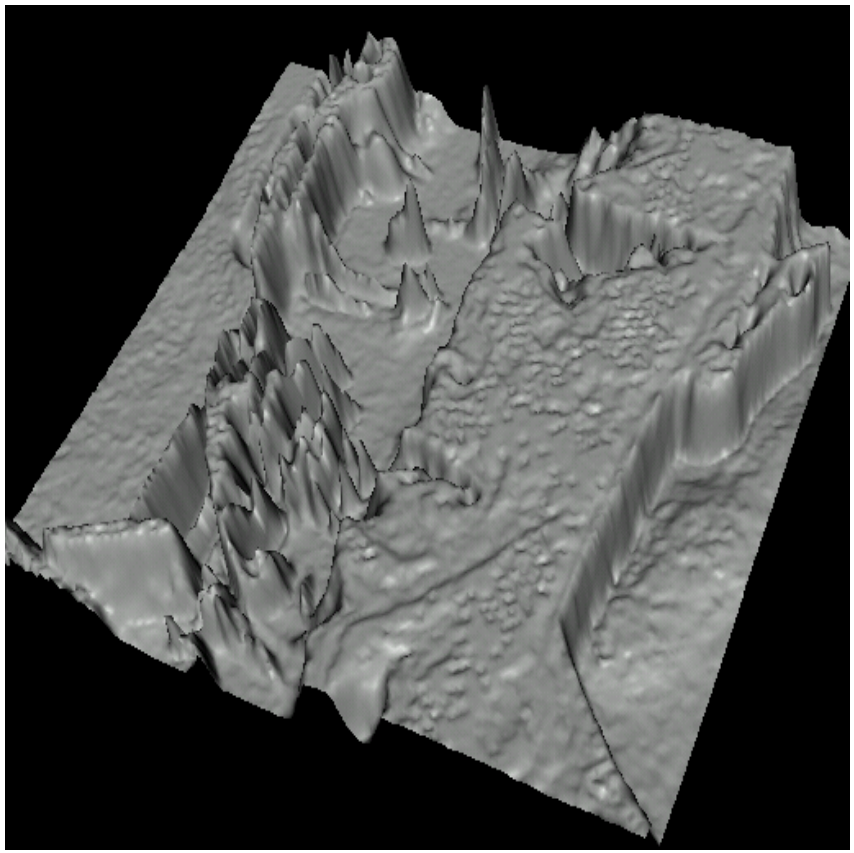


Looking at it more locally ...



Hints for edges in the image function?

Image of a cyborg

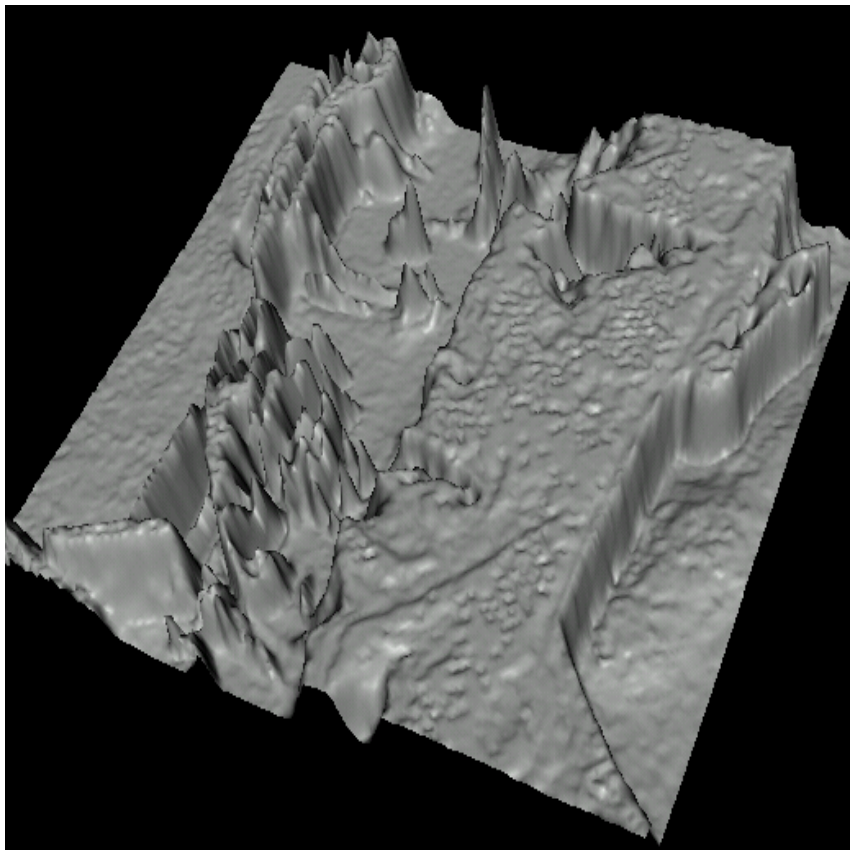


Intensity plotted as a height field

- 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

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:**

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

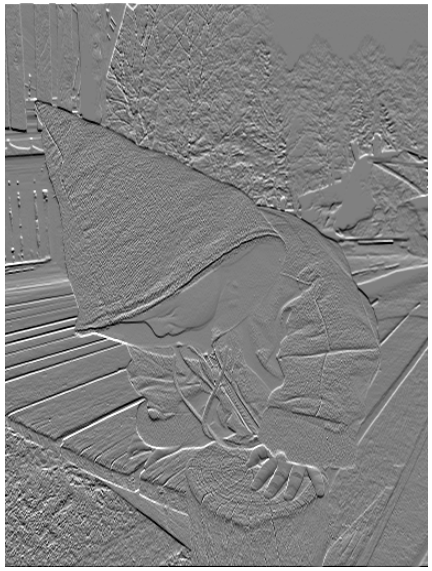
These are the convolution kernels, not correlation !

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:

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

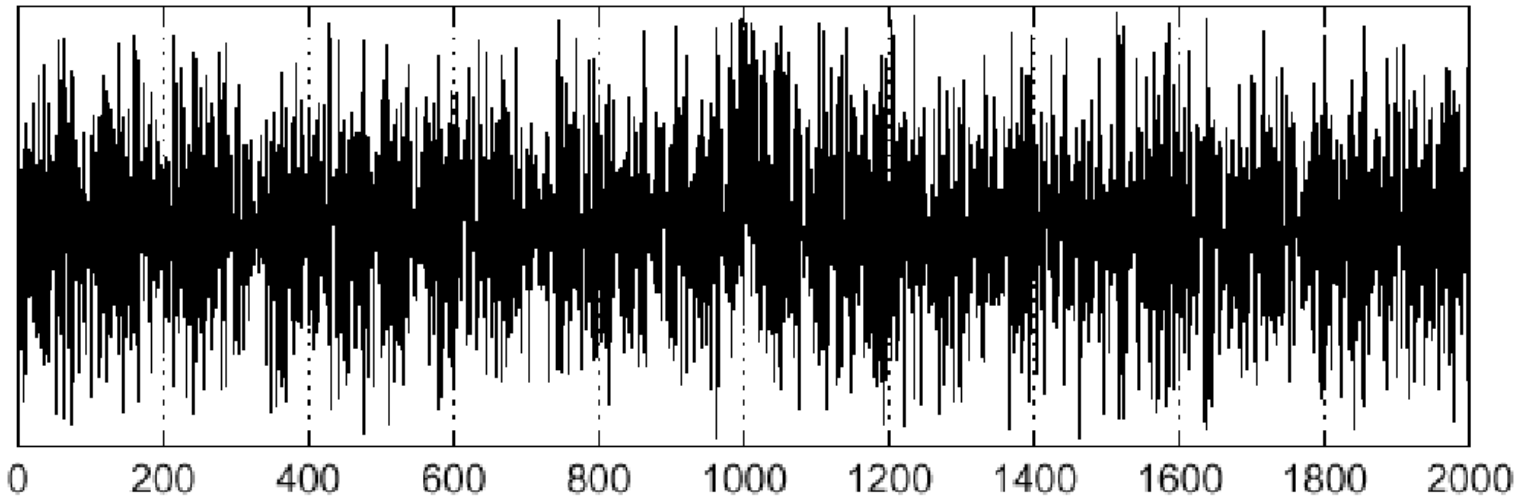
- The gradient can thus be interpreted as a vector-valued image with $n = m = 2$. At each pixel, the gradient points in the direction of the strongest change.

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

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

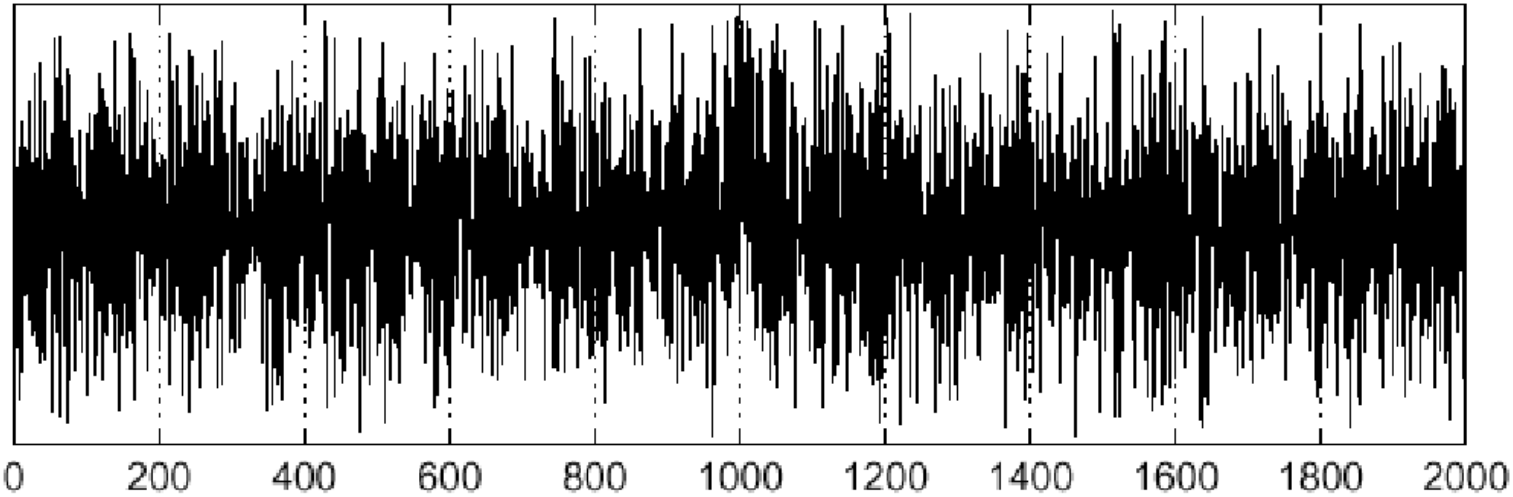
- The larger the magnitude, the stronger the change in gradient direction.

The problem when using finite differences ... noise

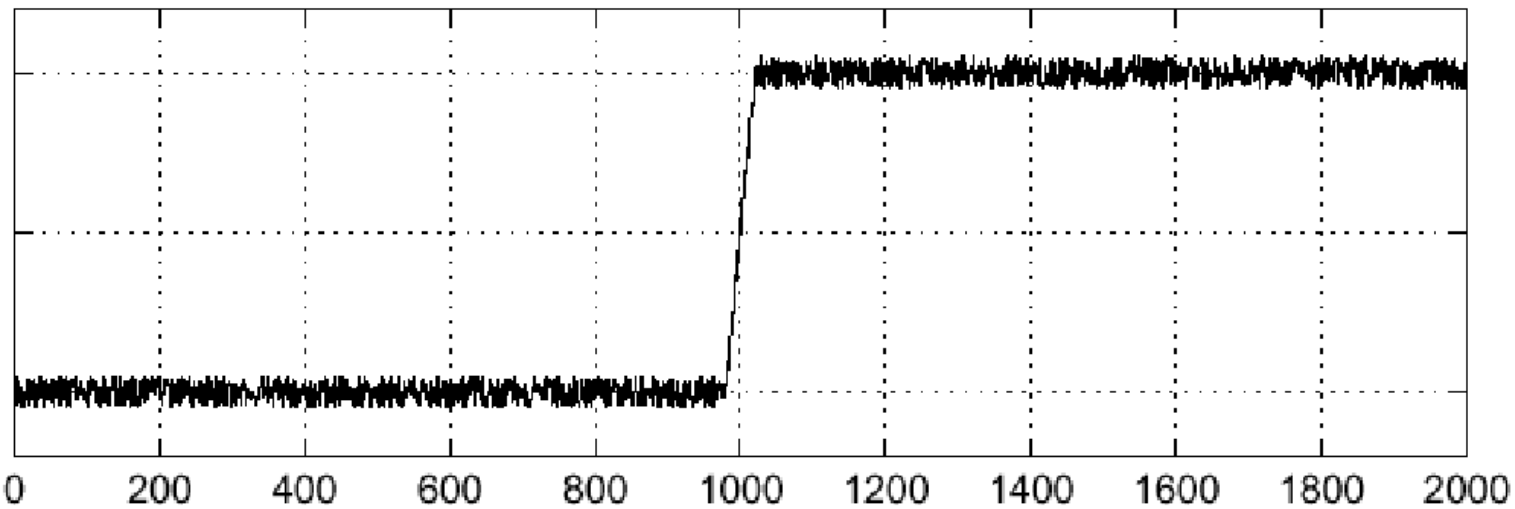


this is the derivative of a noisy 1D signal *f* - but where is the edge?

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.

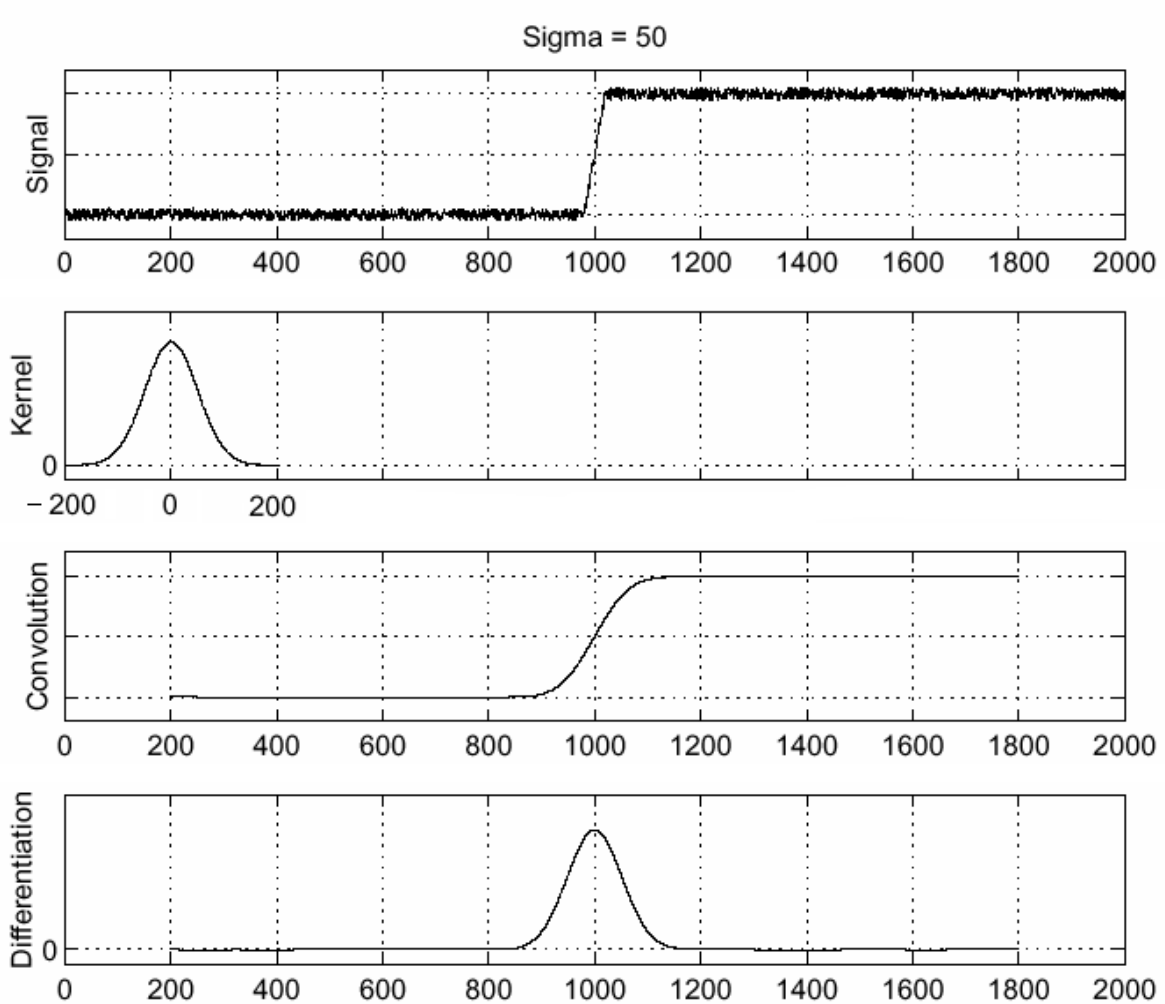
Idea: smooth the signal, then compute derivative

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.

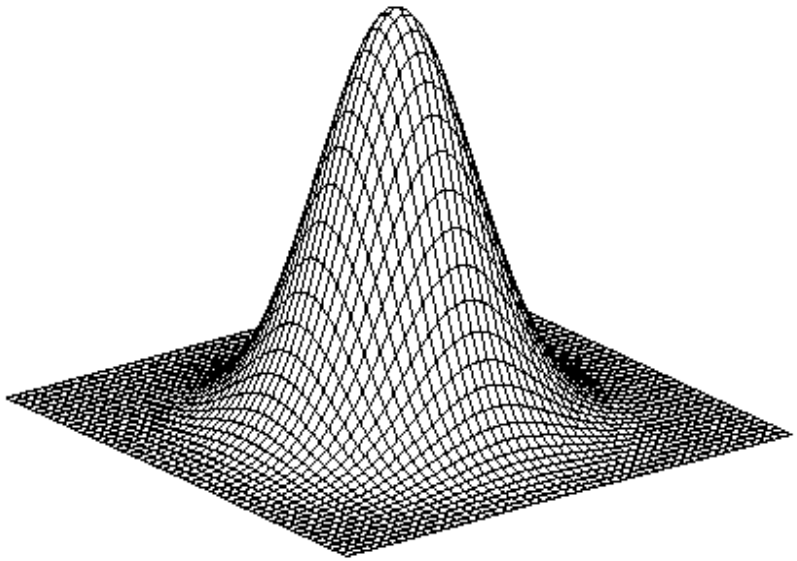
Derivative theorem for the continuous convolution

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

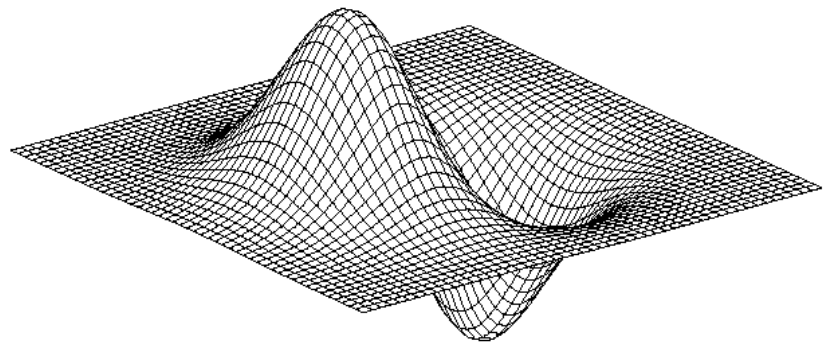
$$\partial_i(f * g) = (\partial_i f) * g = f * (\partial_i g).$$

Taking a partial derivative and computing the continuous convolution commutes.

The “Derivative of Gaussian” convolution filters



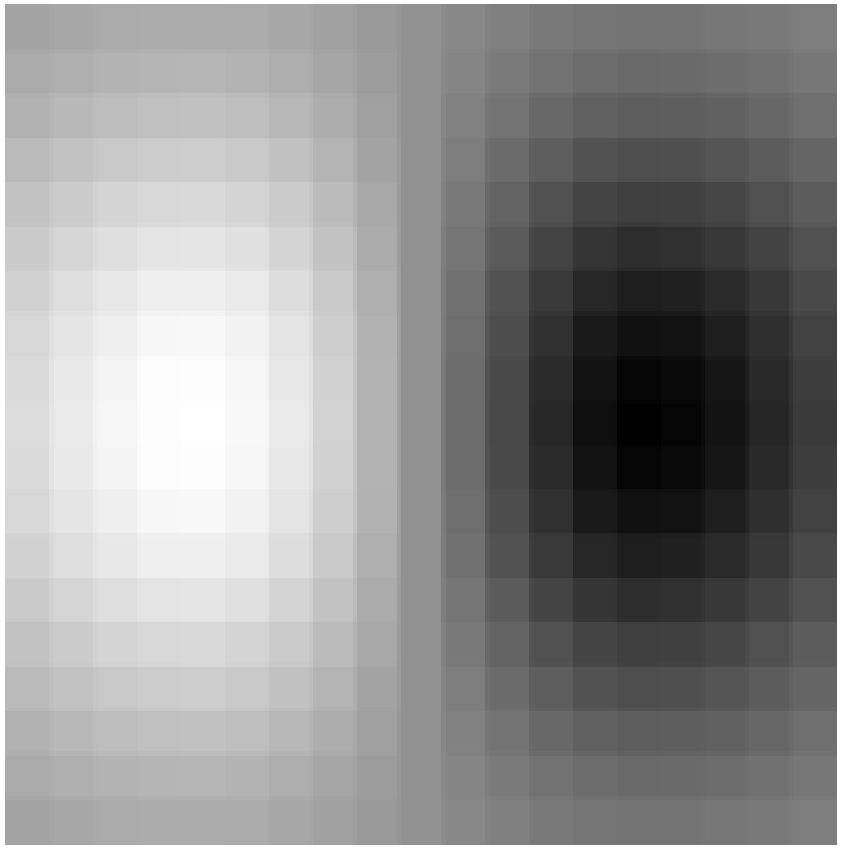
Gaussian $h_\sigma(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$



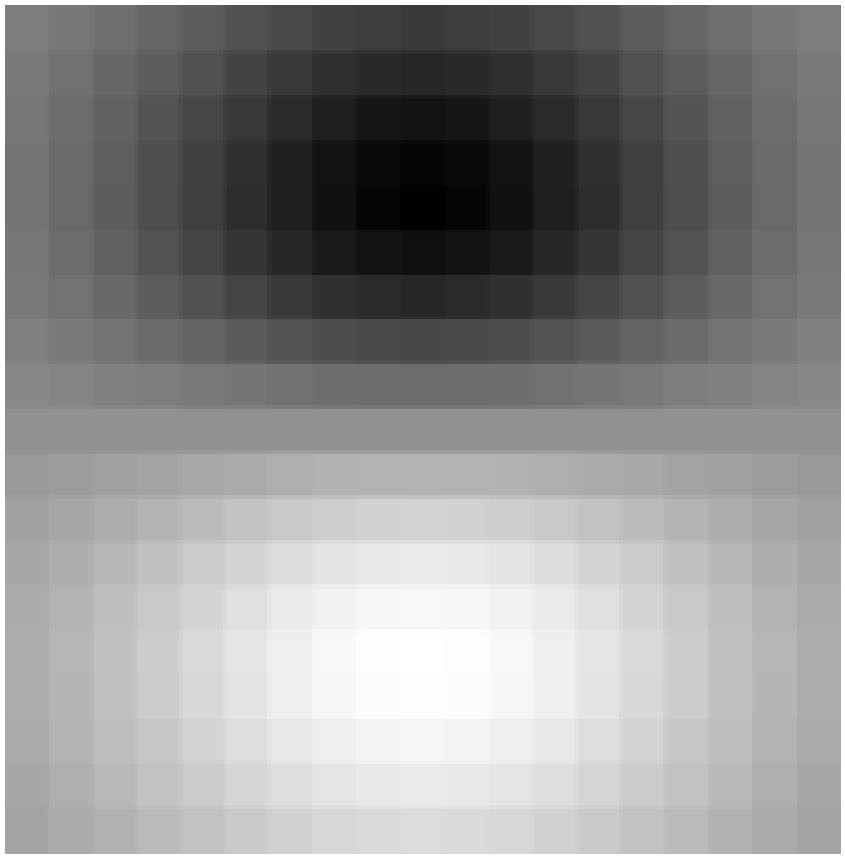
Derivative $\partial_x h_\sigma(x, y)$

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.]*

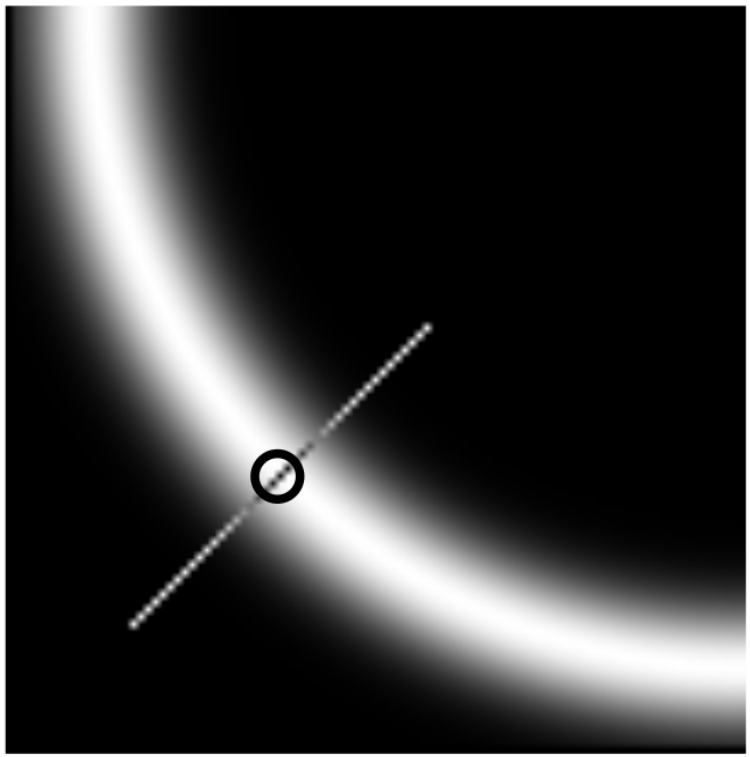
An example image

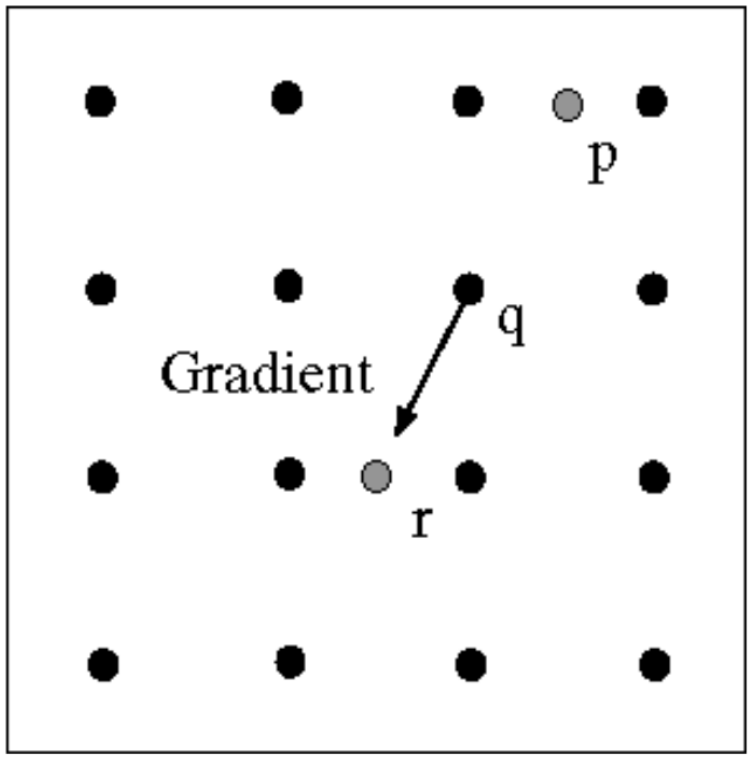
Gradient magnitude at certain scale σ

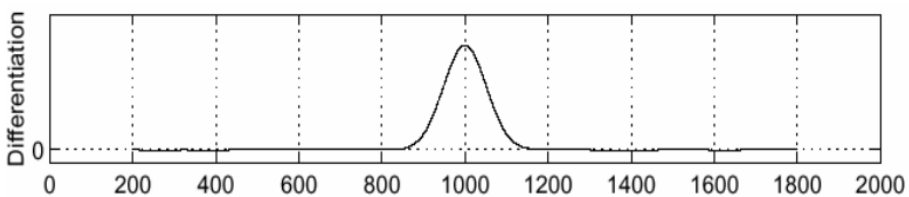
Derivatives computed with “Derivative of Gaussian” filters.

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

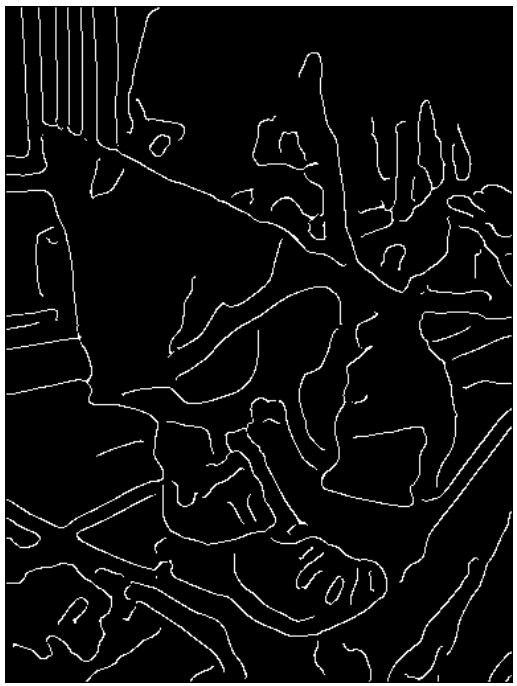
We did not discuss the last step, but it's something of a hack and not very interesting all things considered.

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 ...

$\sigma = 0$

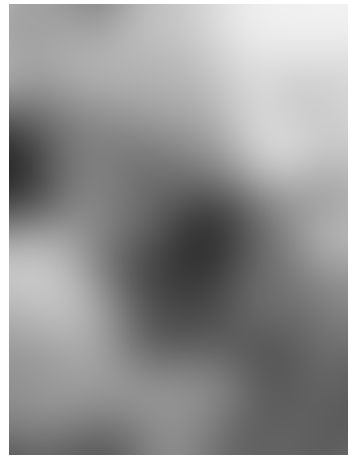
$\sigma = 1.0$

$\sigma = 2.0$

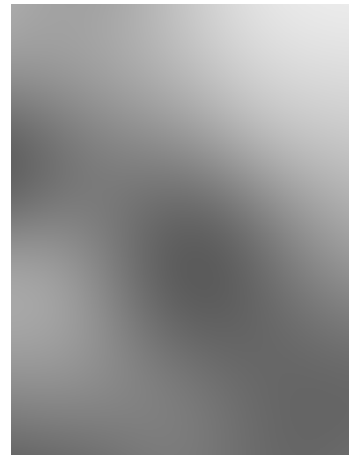
$\sigma = 4.0$

$\sigma = 8.0$

$\sigma = 16.0$



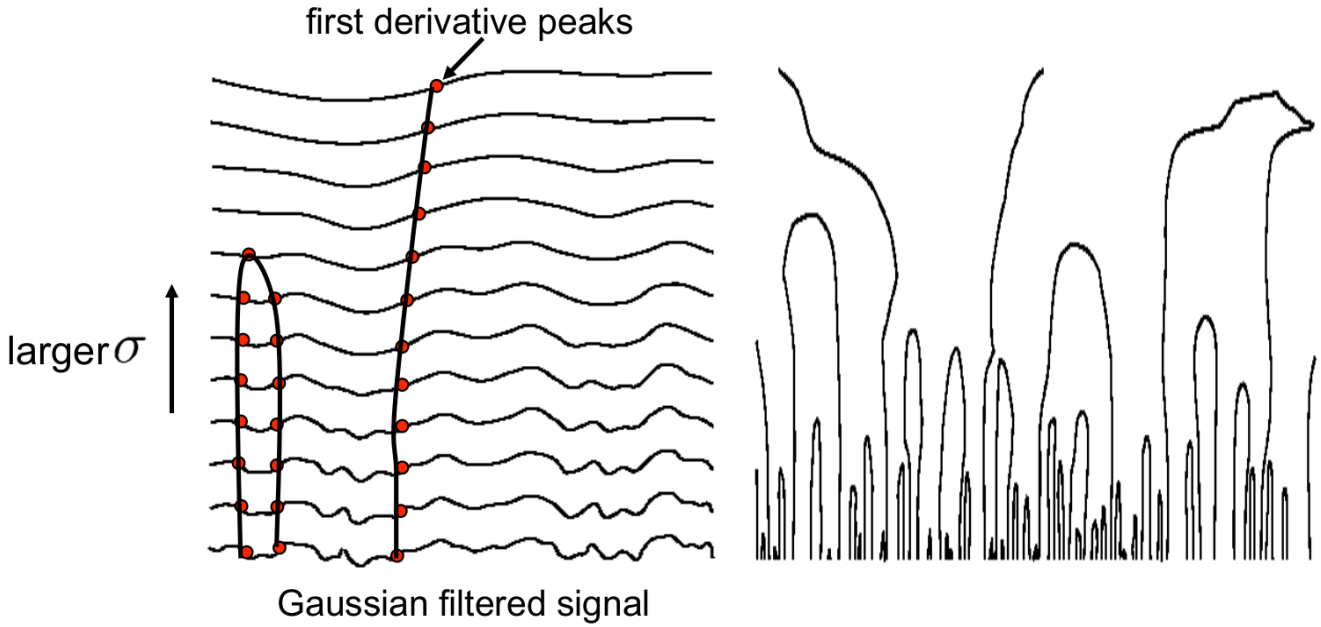
$\sigma = 32.0$



$\sigma = 64.0$

Increasing σ gradually removes detail ...

The scale space idea

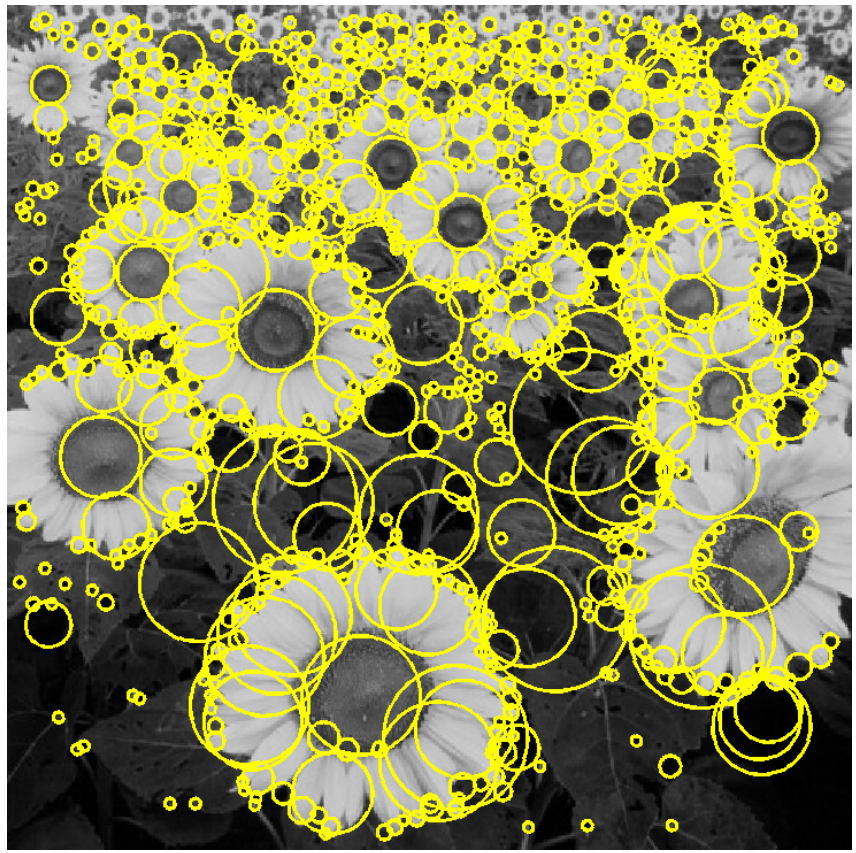


- edge position may shift with increasing scale σ
- two edges may merge with increasing scale
- an edge may not split into two with increasing scale
- note: there are other ways to construct scale spaces, but the Gaussian is the “canonical” one in a certain sense.
- the idea is very powerful, and sparked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

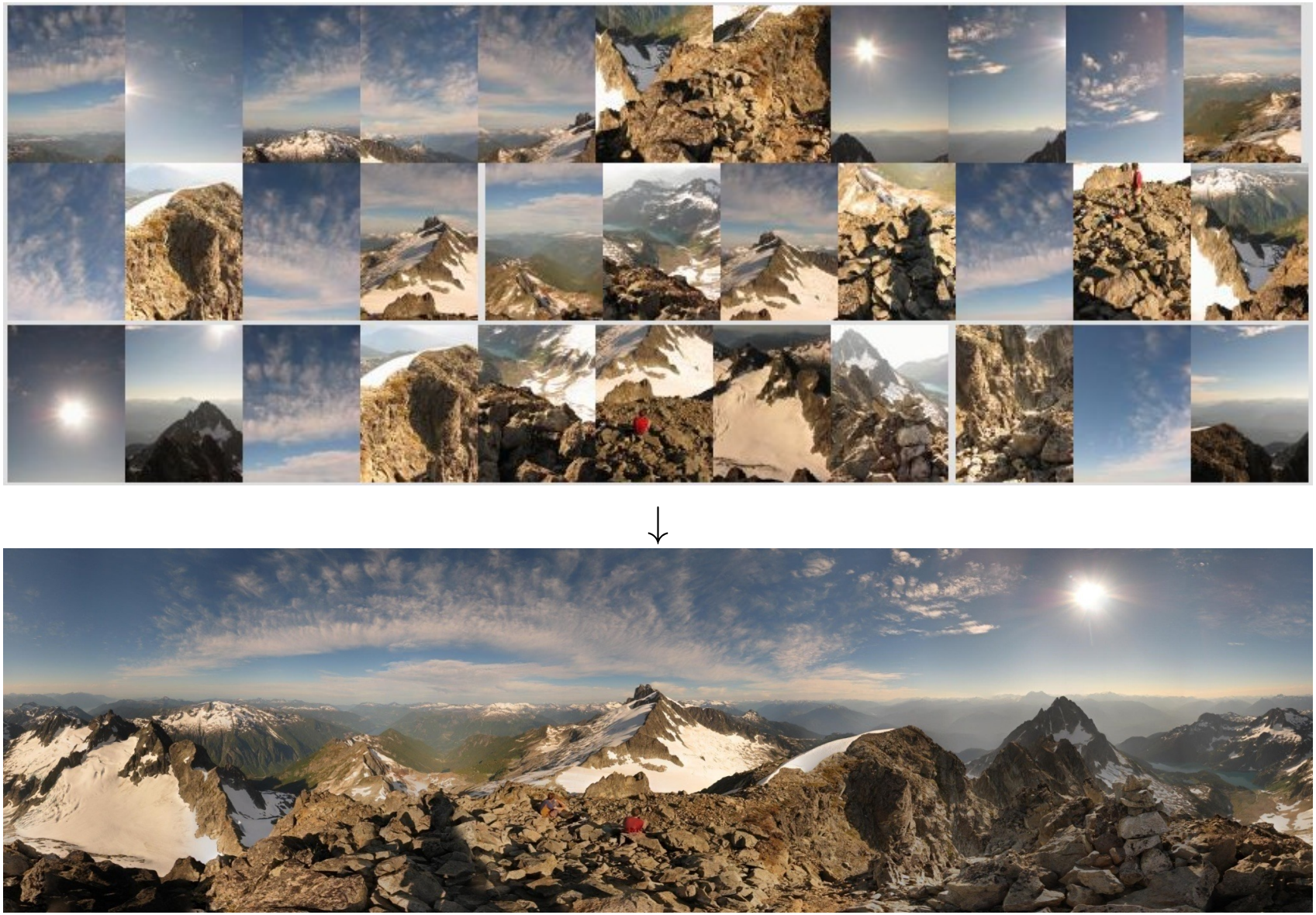
Maybe first paper on scale space: [Witkin 1983]

Overview

Feature extraction: corners and blobs



Motivation: automatic panoramas



Motivation: automatic panoramas

- HDPan (Microsoft Research)

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

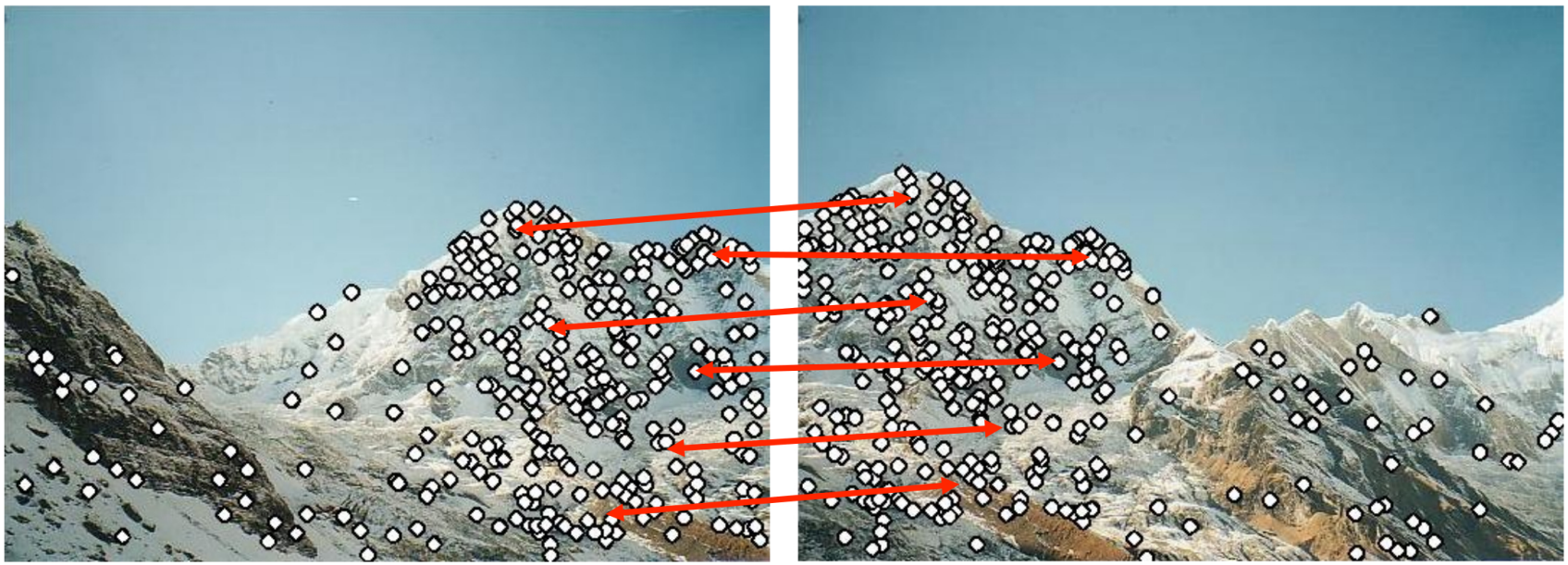
- GigaPan

  `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

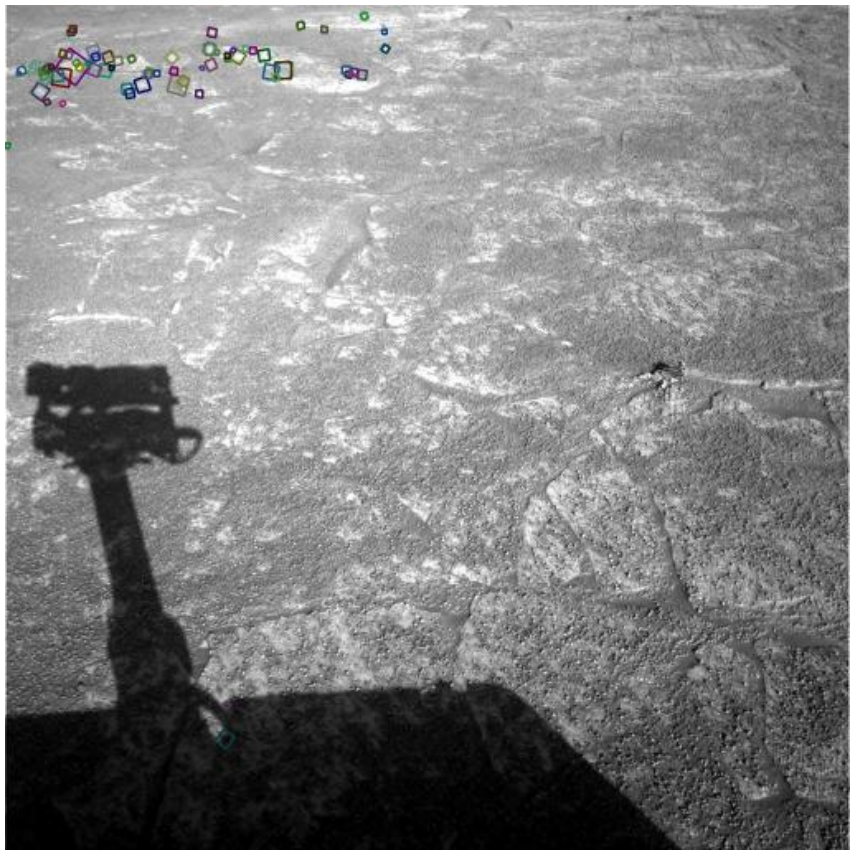
How to combine two images to form a panorama?

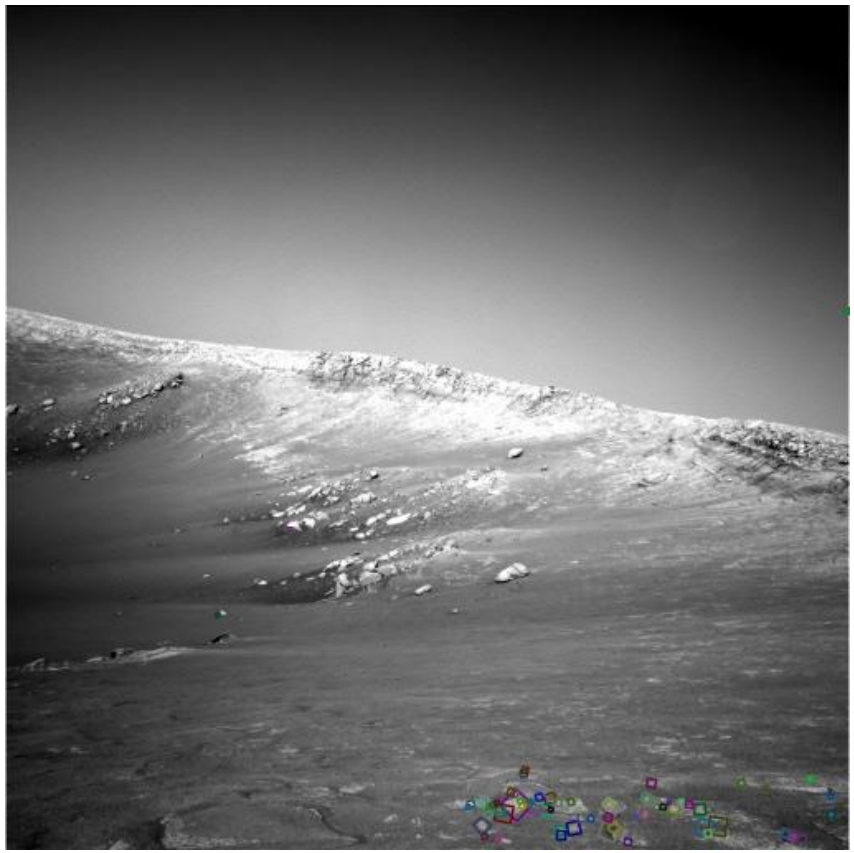
Step 2: image alignment

A difficult case ...

NASA Mars rover images - which parts match?

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?

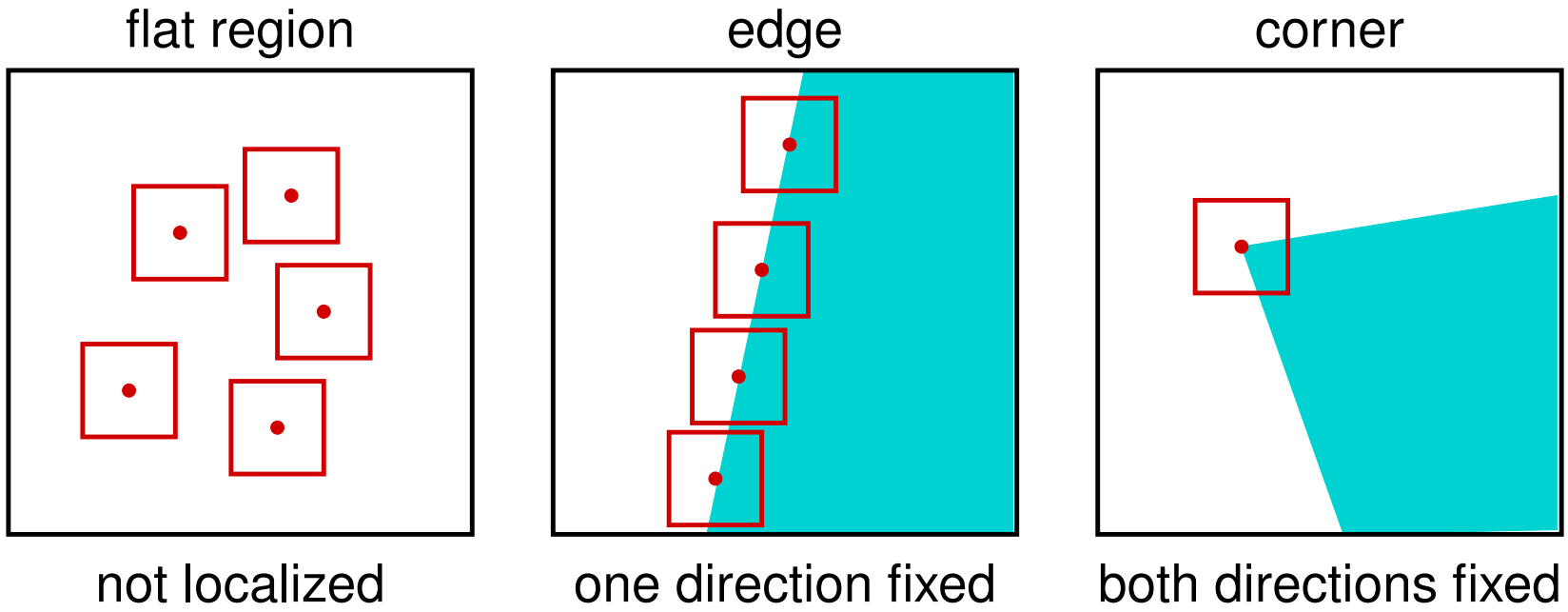
What points to choose?

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

What points to choose?

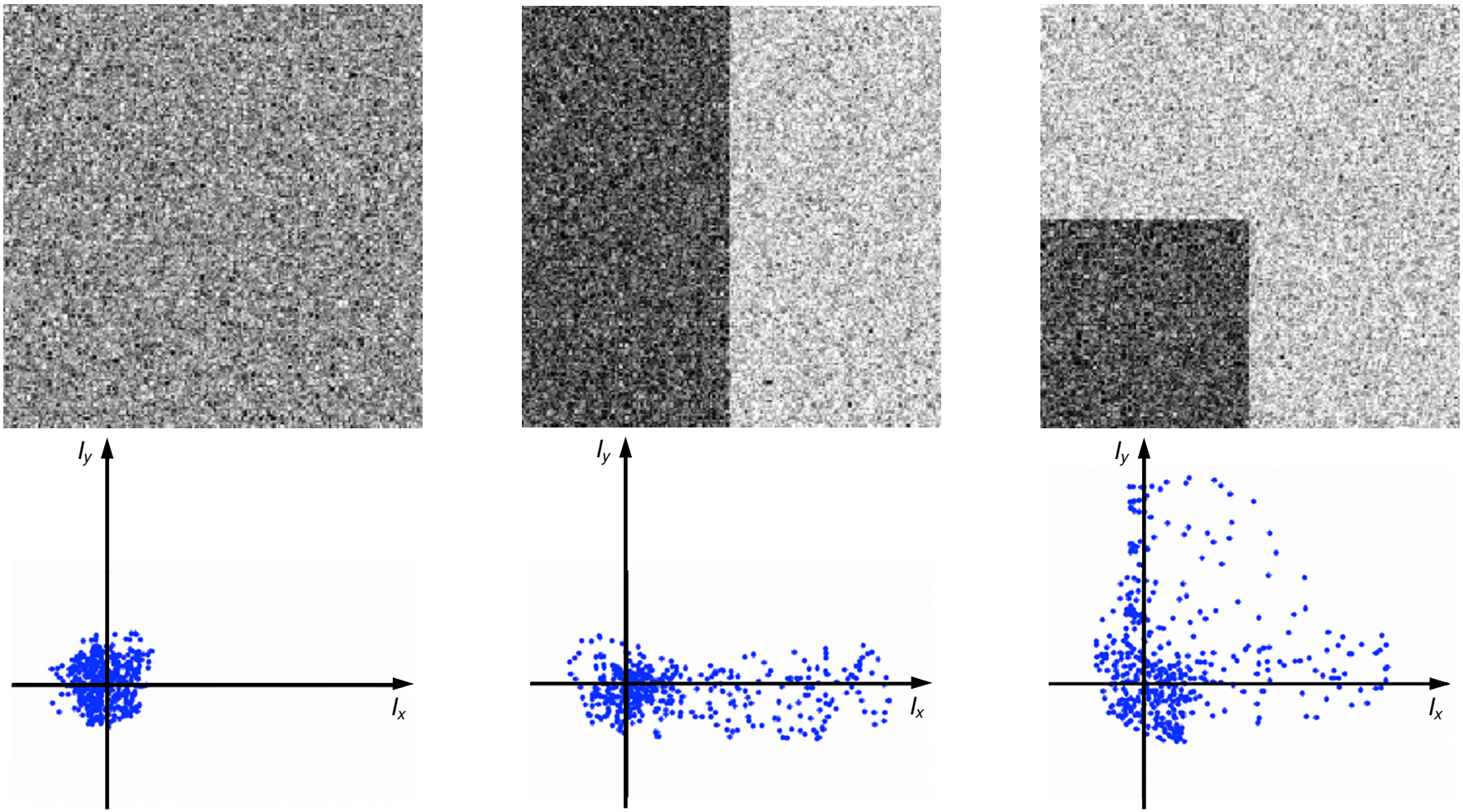
- Textureless patches are nearly impossible to localize.
- Patches with large contrast changes (gradients) are easier to localize.
- But straight line segments at a single orientation suffer from the **aperture problem**, i.e. it is only possible to align the patches along the direction normal to the 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 delve into statistics again.

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. 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 non-negative real numbers $\sigma_1 \geq \sigma_2 \geq \sigma_3 ... \geq 0$. These are called the **singular values** of A.

- *The singular values are uniquely determined by A, but the matrices U and V are not.*
- *Interpretation: Equivalent to the above is $AV = \Sigma U$. Hence, if A is viewed as a linear map from $\mathbb{R}^n$ to $\mathbb{R}^m$, then A maps the columns of V onto the columns of U scaled by the singular values.*

Some useful more or less obvious properties

- The rank of A is equal to the number of non-zero singular values.
- The range of A is spanned by the columns of U corresponding to the non-zero singular values (thus the first k columns, where $k = rank(A)$).
- The columns of V are Eigenvectors of $A^T A$.
- The columns of U are Eigenvectors of AA^T.
- The Eigenvalues of both $A^T A$ and AA^T are the squared singular values. In the case of AA^T, the remaining Eigenvalues are zero.
- In particular, both $A^T A$ and AA^T have only positive or zero Eigenvalues (they are “positive semi-definite”).

Why is the SVD useful?

Another way to write down the decomposition:

$$A = \sum_{i=1}^{\min(m,n)} \sigma_i (U_i V_i^T),$$

where U_i and V_i are the corresponding columns of the SVD matrices. The partial sum $A_k := \sum_{i=1}^{k} \sigma_i (U_i V_i^T)$ is an approximation to A of increasing quality - the singular value measures in a sense the "importance" of the corresponding component.

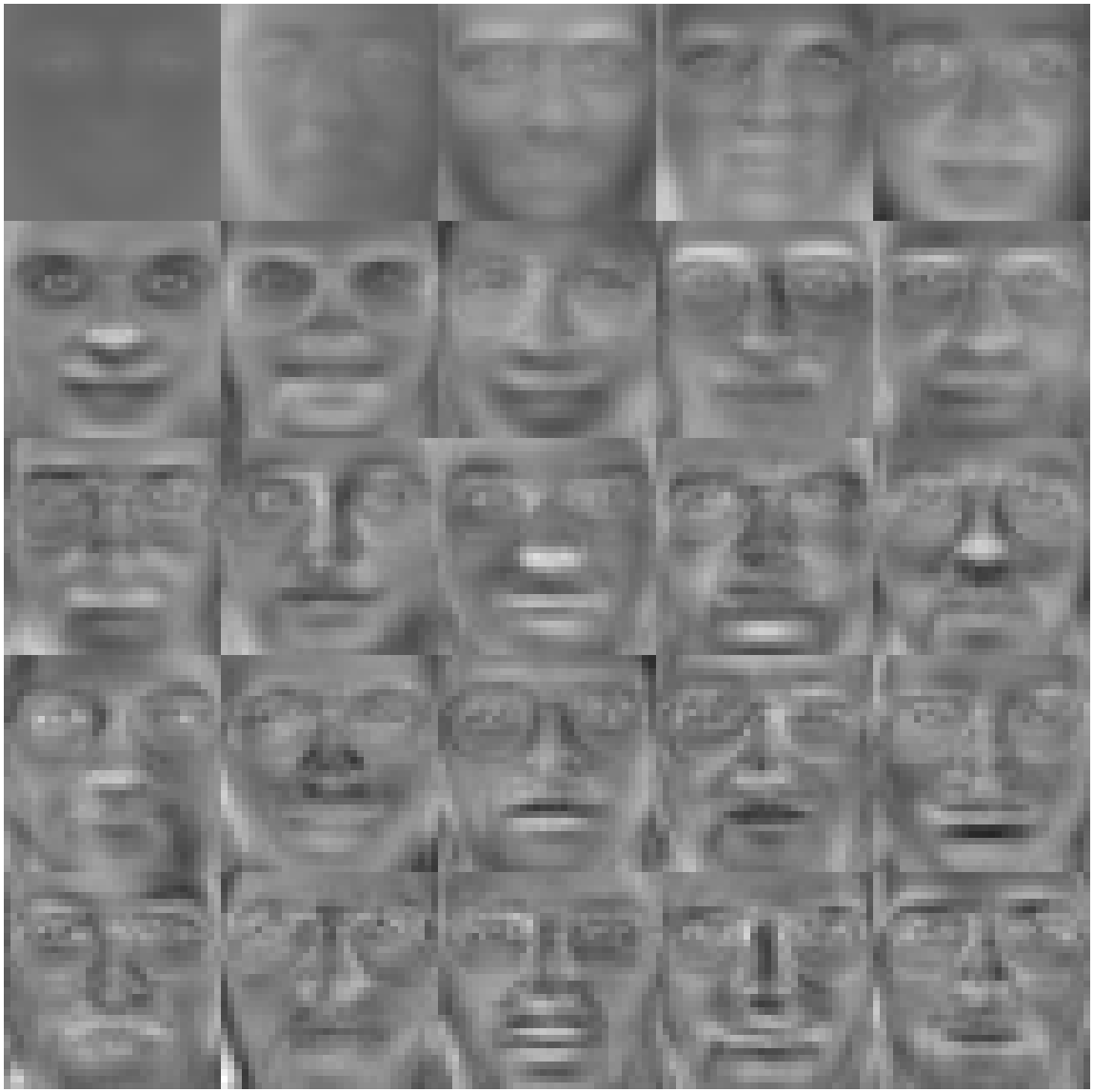
- Optimization, low-rank approximation
  $\rightarrow$ Later chapter

- Data analysis and compression
  - find the most meaningful "components" of data
  - relation to statistics via PCA, discussed soon

Example: face analysis

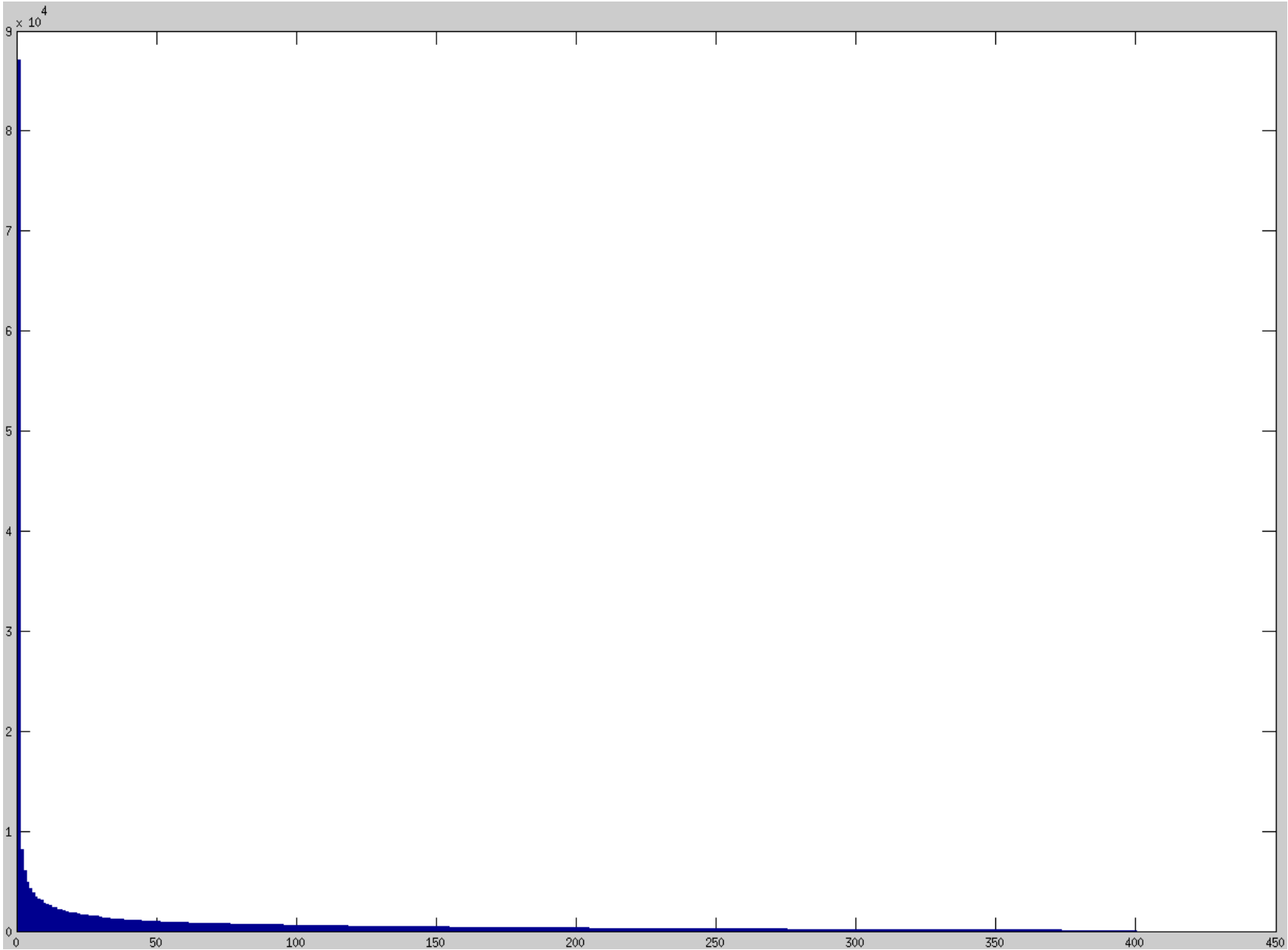
- Database of 400 faces, $32 \times 32 = 1024$ pixels each.
- Build large 400×1024 matrix with one face per row, compute SVD.
- What happens?

The “Eigenfaces”



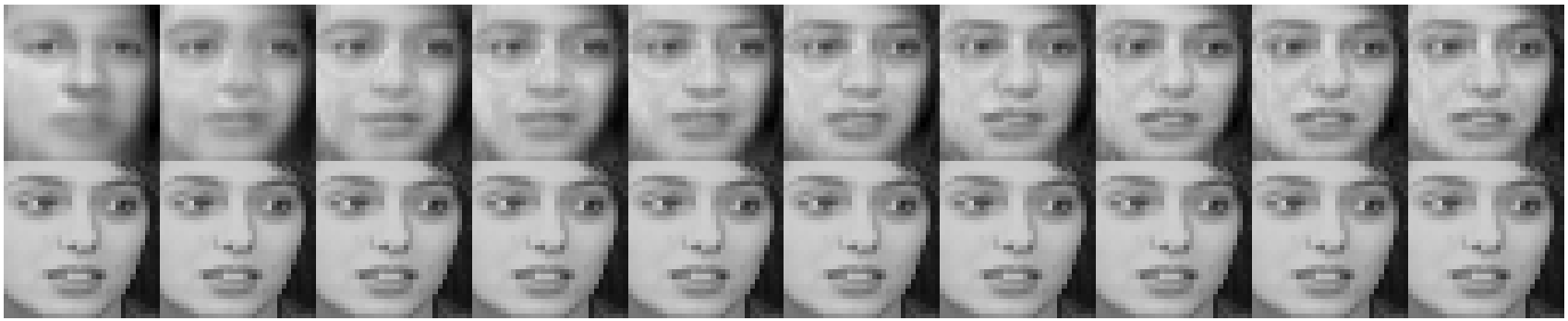
- First 25 columns of V

The singular values



Face approximation via the Eigenfaces

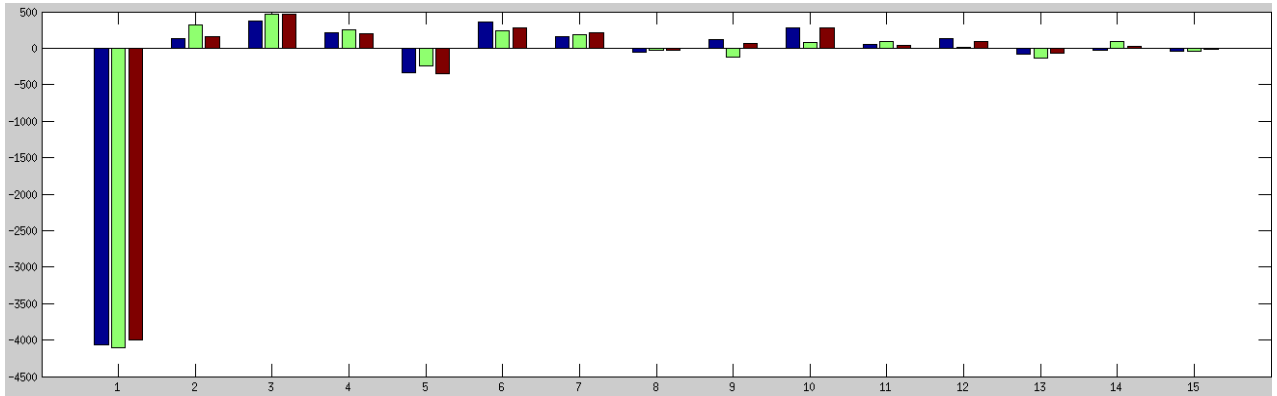
Approximated with k=10, 20, ..., 100 Eigenfaces, $f_k = \sum_{i=1}^{k} (f, V_i) V_i$
(top: approximation, bottom: original)



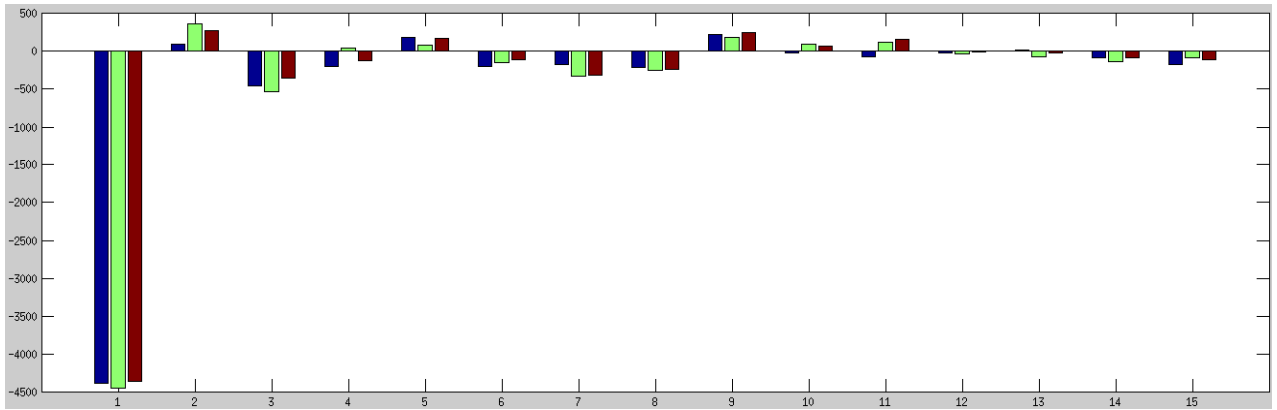
Approximated with 110, 120, ..., 200 Eigenfaces
(top: approximation, bottom: original)

Idea: face recognition

Coefficients of the approximation can be used to discriminate between two different persons:
Person 1, three different photos (blue, green, red bars)



Person 2, three different photos (blue, green, red bars)



Data and code to reproduce figures on slides is on Ilias, feel free to play around with it.

SVD: an informal interpretation

- In the singular value decomposition, the columns of the matrix V give a new basis for the vector space $\mathbb{R}^n$.
- In the face database example, this space corresponds to the space of possible images of faces.
- The new basis vectors are chosen in a way that they capture the most important aspects of the data in descending order of importance.
- Exactly how important the basis vector is corresponds to the magnitude of the singular value.

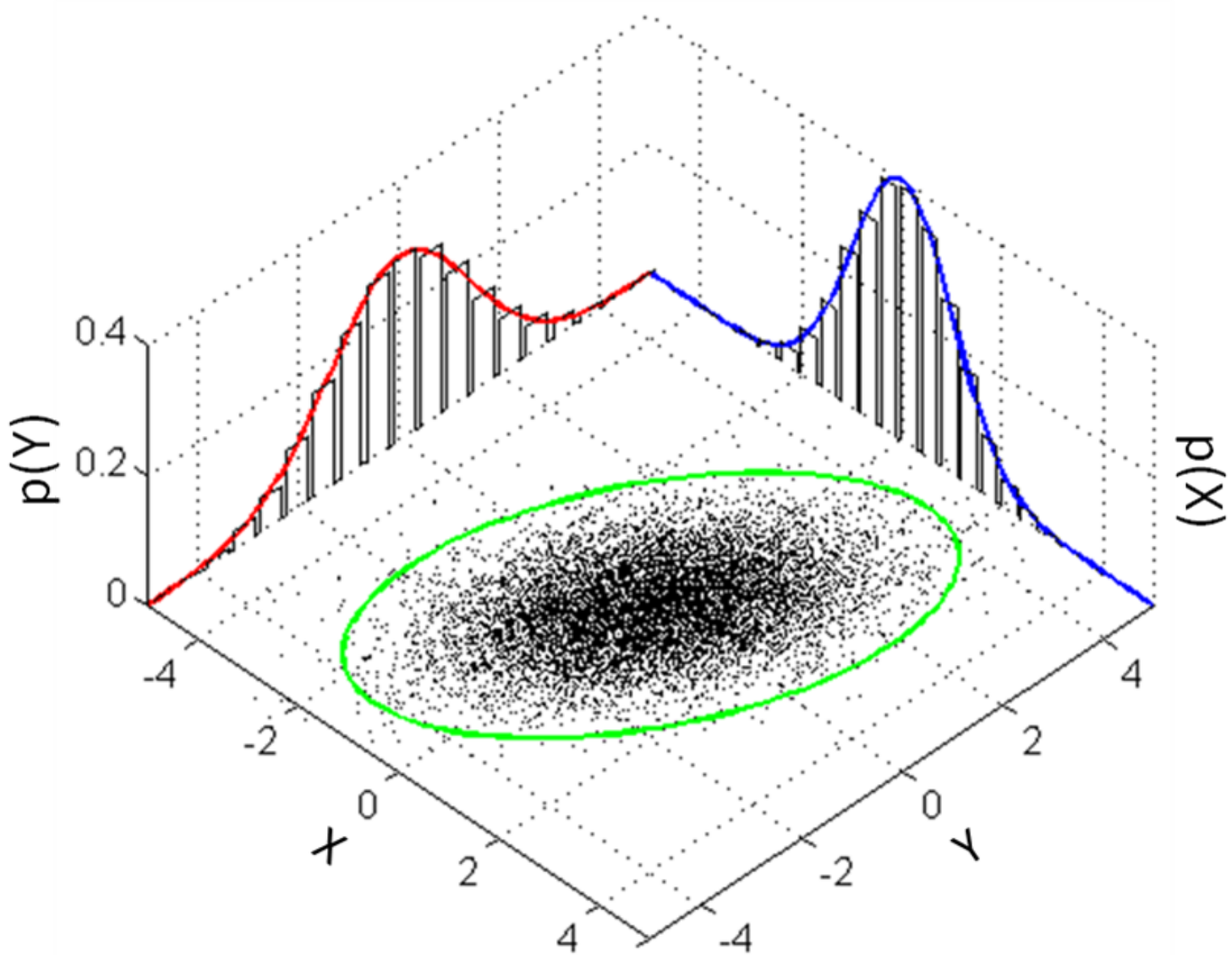
SVD: an informal interpretation

- In the singular value decomposition, the columns of the matrix V give a new basis for the vector space $\mathbb{R}^n$.
- In the face database example, this space corresponds to the space of possible images of faces.
- The new basis vectors are chosen in a way that they capture the most important aspects of the data in descending order of importance.
- Exactly how important the basis vector is corresponds to the magnitude of the singular value.

This intuition is formalized in the language of statistics
in the Principal Component Analysis (PCA) of a data set.

Principal component analysis (PCA)

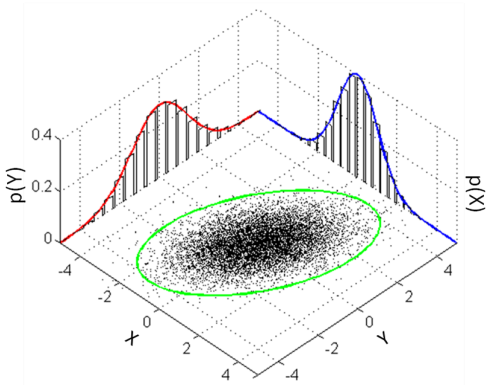
Goal: characterize joint probability distributions of several variables, for example find the direction(s) with greatest variance.



Graphics: Wikipedia (*Joint probability distribution*)

Joint probability distributions: some terminology

- **Random variables** $X_j, j = 1, \ldots, n$, if there are only two variables ("bivariate distribution"), we sometimes call them X and Y
- **Joint distribution** $P(X_j = x_j \text{ for all } j)$ (sampled inside green ellipse for bivariate case)
- **Marginal distributions** $P(X_j = x_j)$, e.g. $P(X = x)$ (blue) and $P(Y = y)$ (red) for bivariate case
- X_j and X_k are called **independent** iff $P(X_j = x_j, X_k = x_k) = P(X_j = x_j)P(X_k = x_k)$.



Graphics: Wikipedia (*Joint probability distribution*)

Sample expectation and standard deviation

- Suppose you measure a bunch of samples (say $m > n$) of the joint distribution, each described by a vector $\mathbf{x}_i \in \mathbb{R}^n$, $i = 1, \ldots, m$.
- We assume we have a probability distribution over the samples, in effect assign each sample a weight p_i such that $\sum_i p_i = 1$.
- The **sample expectation value** for a random variable Z (which can e.g. be one of the X_j) is

$$E(Z) = \sum_{i=1}^{m} p_i z_i,$$

  i.e. a weighted average over all measurements for Z.
- The **sample variance** for a random variable Z is

$$Var(Z) = \sum_{i=1}^{m} p_i (Z_i - E(Z))^2,$$

  i.e. the expected squared deviation from the expectation value.
- The **sample standard deviation** is $\sigma(Z) = \sqrt{Var(Z)}$.

The covariance

- The covariance of two random variables X and Y is defined as

$$cov(X, Y) := \sum_{i=1}^{m} p_i \, (x_i - E(X)) \, (y_i - E(Y)).$$

- From now on, we will assume that we have **"centered" the measurements** such that $E(X_j) = 0$ for all variables X_j. This can be done by just subtracting the expectation value from each measurement. In particular, we then have

$$cov(X_j, X_k) = \sum_{i=1}^{m} p_i \, x_{ji} \, x_{ki}.$$

Intuition: *If X_j usually increases when X_i increases, the covariance is positive. If X_j usually decreases when X_i increses, the covariance is negative. If X_j is not biased towards changes in X_i, then both variables are uncorrelated.*

The covariance matrix

For the random variables $\mathbf{X} = X_1, \ldots, X_n$, the **covariance matrix** is defined as the $n \times n$ matrix

$$C(\mathbf{X}) := (cov(X_j, X_k))_{j,k=1,\ldots,n}.$$

Properties:

- The covariance matrix is symmetric.
- Since $cov(X_j, X_j) = var(X_j)$, the diagonal consists of the variances.
- Let's consider the covariance of the measured samples. We write all the measurements as row vectors into a large matrix $M \in \mathbb{R}^{m \times n}$ (remember the face example), to account for weights, the ith row is multiplied by $\sqrt{p_i}$.
- The formula on the previous slide then shows that

$$C(\mathbf{X}) = M^T M.$$

  In particular, $C(\mathbf{X})$ is positive semi-definite.

Covariance matrix and SVD

Let $M = U\Sigma V^T$ be the singular value decomposition of the measurement matrix. Assume we transform all measurements by V, then we obtain the new measurement matrix $M' = MV = U\Sigma$, which has covariance matrix $C' = \Sigma^T U^T U\Sigma = \Sigma^2$.

Covariance matrix and SVD

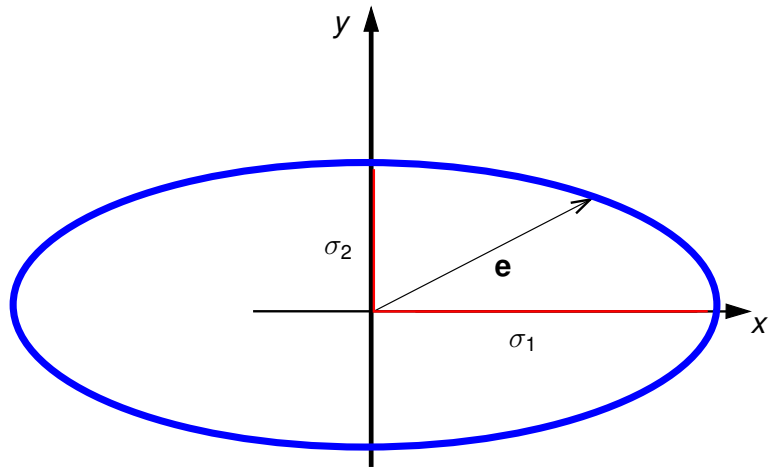
Let $M = U\Sigma V^T$ be the singular value decomposition of the measurement matrix. Assume we transform all measurements by V, then we obtain the new measurement matrix $M' = MV = U\Sigma$, which has covariance matrix $C' = \Sigma^T U^T U\Sigma = \Sigma^2$.

- The covariance matrix can always be made diagonal using an orthogonal transformation in joint measurement space.
- For this, the Eigenvectors of the old covariance matrix are mapped onto the unit vectors.
- In the new basis in measurement space, variables are sorted by descending standard deviation and pairwise uncorrelated.
- The standard deviations are equal to the singular values.

And finally .... the ellipses

Consider the case of two random variables which have been decorrelated. Also consider an ellipse given in parametric form via

$$\left\{ \begin{bmatrix} \sigma_1 \cos(\phi) \\ \sigma_2 \sin(\phi) \end{bmatrix} ; 0 \leq \phi \leq 2\pi \right\}.$$



What does it visualize? Turns out that the radius in each given direction $\mathbf{e} = (\sin(\phi), \cos(\phi))^T$ is

$$\begin{aligned} \sqrt{\sigma_1^2 e_1^2 + \sigma_2^2 e_2^2} &= \sqrt{e_1^2\, var(X) + e_2^2\, var(Y)} \\ &= \sqrt{var(e_1\, X \; + \; e_2\, Y)}. \end{aligned}$$

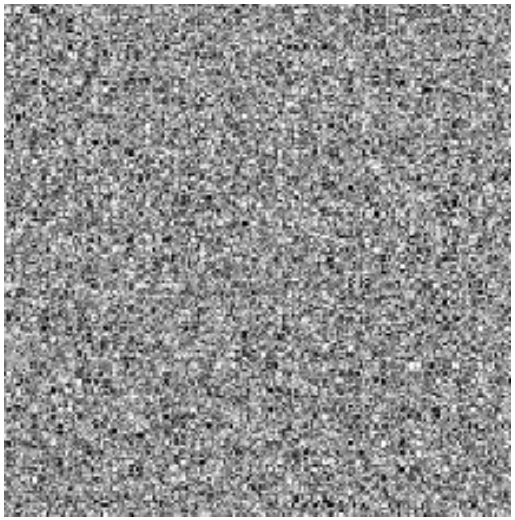
The radius of the ellipse yields the standard deviation of the measurements projected onto the corresponding direction.

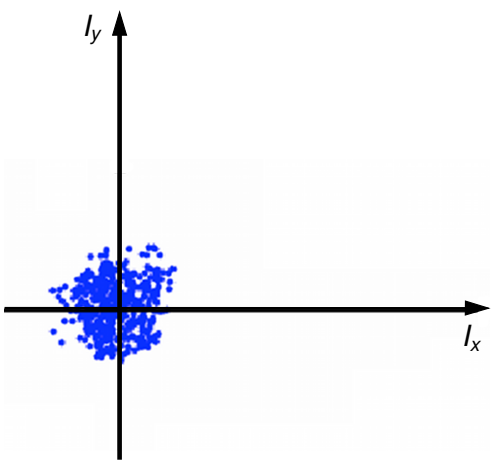
PCA - summary

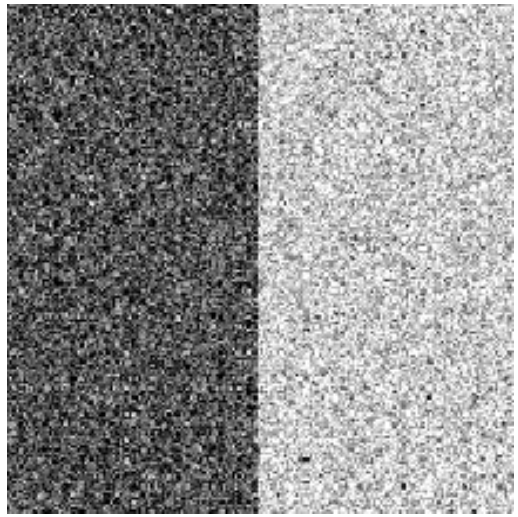
- PCA is very closely related to SVD, and can be used to decorrelate data via a basis transformation in measurement space, which makes subsequent analysis simpler.
- Dimensions which correspond to larger singular values have higher variance and are thus statistically more meaningful.
- This is extremely useful to analyse high-dimensional data and extract the most meaningful lower-dimensional representation.
- We will now apply it to the gradient measurements around a pixel to analyze local orientation in an image.

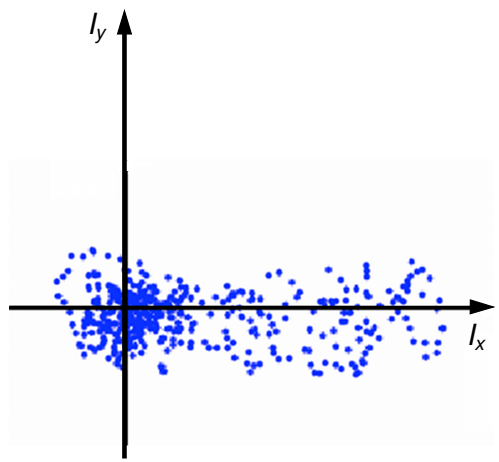
We know a lot more about statistics now ...
let's get back to corner detection !

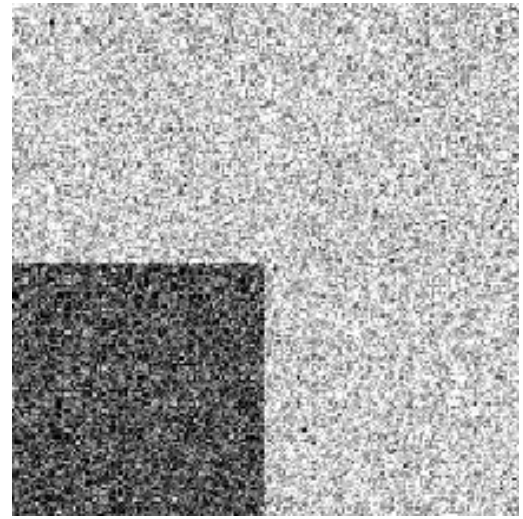
Remember what we were interested in ...

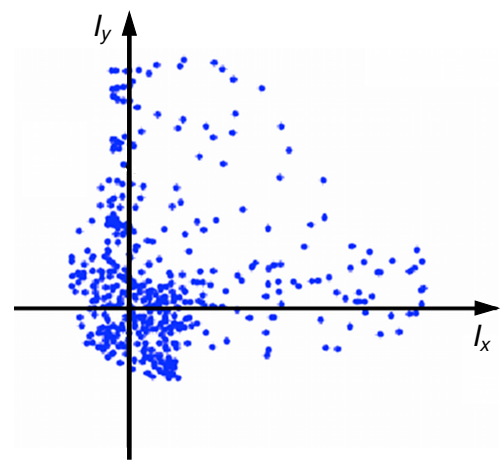












The distribution of the partial derivatives is characteristic
$\Rightarrow$ we now know how to analyze this!

Notation to define the structure tensor

- A Gaussian kernel with standard deviation σ is written as G_σ.
- the Derivative of Gaussian operators at scale σ are denoted by $\partial_x^\sigma := \partial_x * G_\sigma$ and $\partial_y^\sigma = \partial_y * G_\sigma$.
- For an image $f = (f_{i,j})$, the partial derivatives at scale σ are $f_x^\sigma := \partial_x^\sigma f$ and $f_y^\sigma := \partial_y^\sigma f$.

We'll also use this notation in the future for other things.

The structure tensor as gradient statistics

- Consider a pixel i, j and a window around the pixel. We sample the gradient at scale σ inside the window.
- Since we are interested in local structure, we assign a weight depending on the distance to i, j according to a Gaussian with standard deviation τ.
- We consider the gradient to be distributed according to a joint distribution of the partial derivatives f_x^σ and f_y^σ.

The structure tensor as gradient statistics

- Consider a pixel i, j and a window around the pixel. We sample the gradient at scale σ inside the window.
- Since we are interested in local structure, we assign a weight depending on the distance to i, j according to a Gaussian with standard deviation τ.
- We consider the gradient to be distributed according to a joint distribution of the partial derivatives f_x^σ and f_y^σ.
- As we have seen, the distribution around (i, j) is characterized by the **covariance matrix**

$$C(\nabla^\sigma f) = \begin{bmatrix} cov(f_x^\sigma, f_x^\sigma) & cov(f_x^\sigma, f_y^\sigma) \\ cov(f_x^\sigma, f_y^\sigma) & cov(f_y^\sigma f_y^\sigma) \end{bmatrix} = \begin{bmatrix} E(f_x^\sigma f_x^\sigma) & E(f_x^\sigma f_y^\sigma) \\ E(f_x^\sigma f_y^\sigma) & E(f_y^\sigma f_y^\sigma) \end{bmatrix}.$$

The structure tensor as gradient statistics

- Consider a pixel i, j and a window around the pixel. We sample the gradient at scale σ inside the window.
- Since we are interested in local structure, we assign a weight depending on the distance to i, j according to a Gaussian with standard deviation τ.
- We consider the gradient to be distributed according to a joint distribution of the partial derivatives f_x^σ and f_y^σ.
- As we have seen, the distribution around (i, j) is characterized by the **covariance matrix**

$$C(\nabla^\sigma f) = \begin{bmatrix} cov(f_x^\sigma, f_x^\sigma) & cov(f_x^\sigma, f_y^\sigma) \\ cov(f_x^\sigma, f_y^\sigma) & cov(f_y^\sigma f_y^\sigma) \end{bmatrix} = \begin{bmatrix} E(f_x^\sigma f_x^\sigma) & E(f_x^\sigma f_y^\sigma) \\ E(f_x^\sigma f_y^\sigma) & E(f_y^\sigma f_y^\sigma) \end{bmatrix}.$$

- Expand the formulas, and it is exactly the the usual definition of the structure tensor ...

Standard definition of the structure tensor

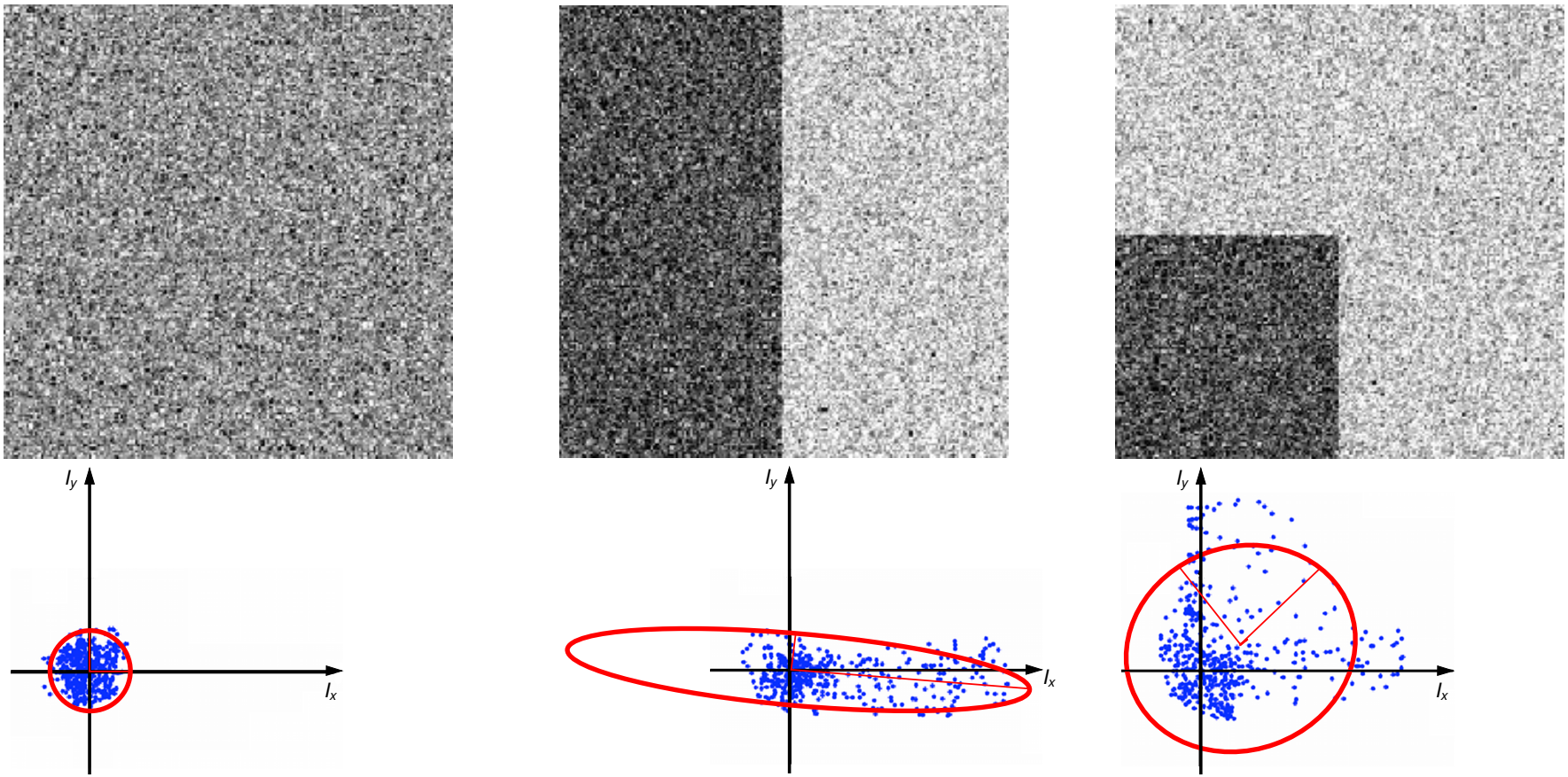
The **structure tensor** $\mathcal{T}$ of a grayscale image f is an image whose values are 2×2 matrices, $\mathcal{T} : \mathbb{Z}^2 \to \mathbb{R}^{2\times 2}$:

$$\mathcal{T} := \begin{bmatrix} G_\tau * (f_x^\sigma)^2 & G_\tau * (f_x^\sigma f_y^\sigma) \\ G_\tau * (f_x^\sigma f_y^\sigma) & G_\tau * (f_y^\sigma)^2 \end{bmatrix} = G_\tau * \begin{bmatrix} (f_x^\sigma)^2 & f_x^\sigma f_y^\sigma \\ f_x^\sigma f_y^\sigma & (f_y^\sigma)^2 \end{bmatrix}.$$

The structure tensor depends on two parameters:

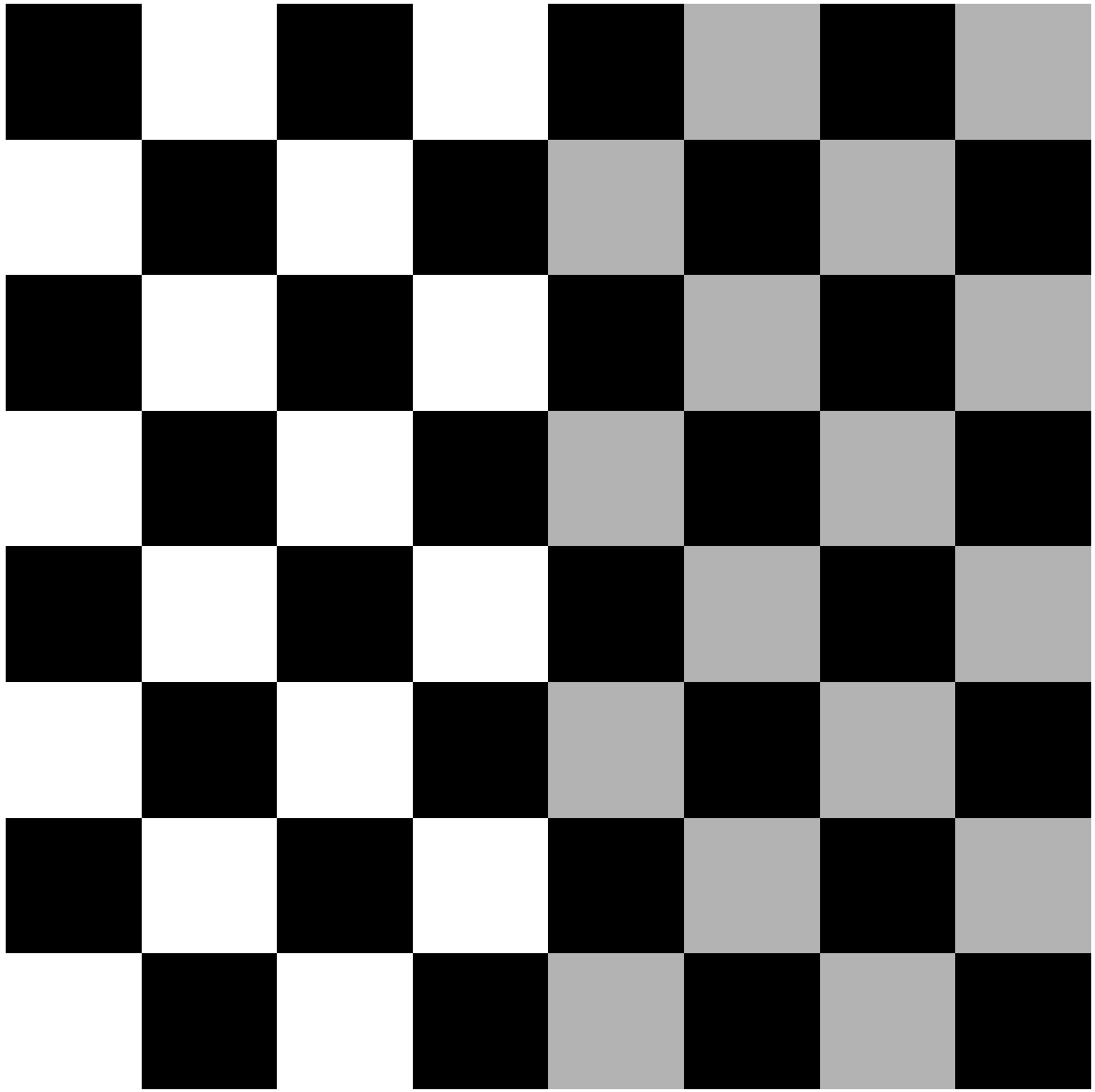
- an *inner scale parameter* $\sigma > 0$, which determines the image scale at which derivatives are taken.
- an *outer scale parameter* $\tau > 0$, which determines the size of the window on which derivative information is sampled from.

Eigenvectors and Eigenvalues characterize the distribution



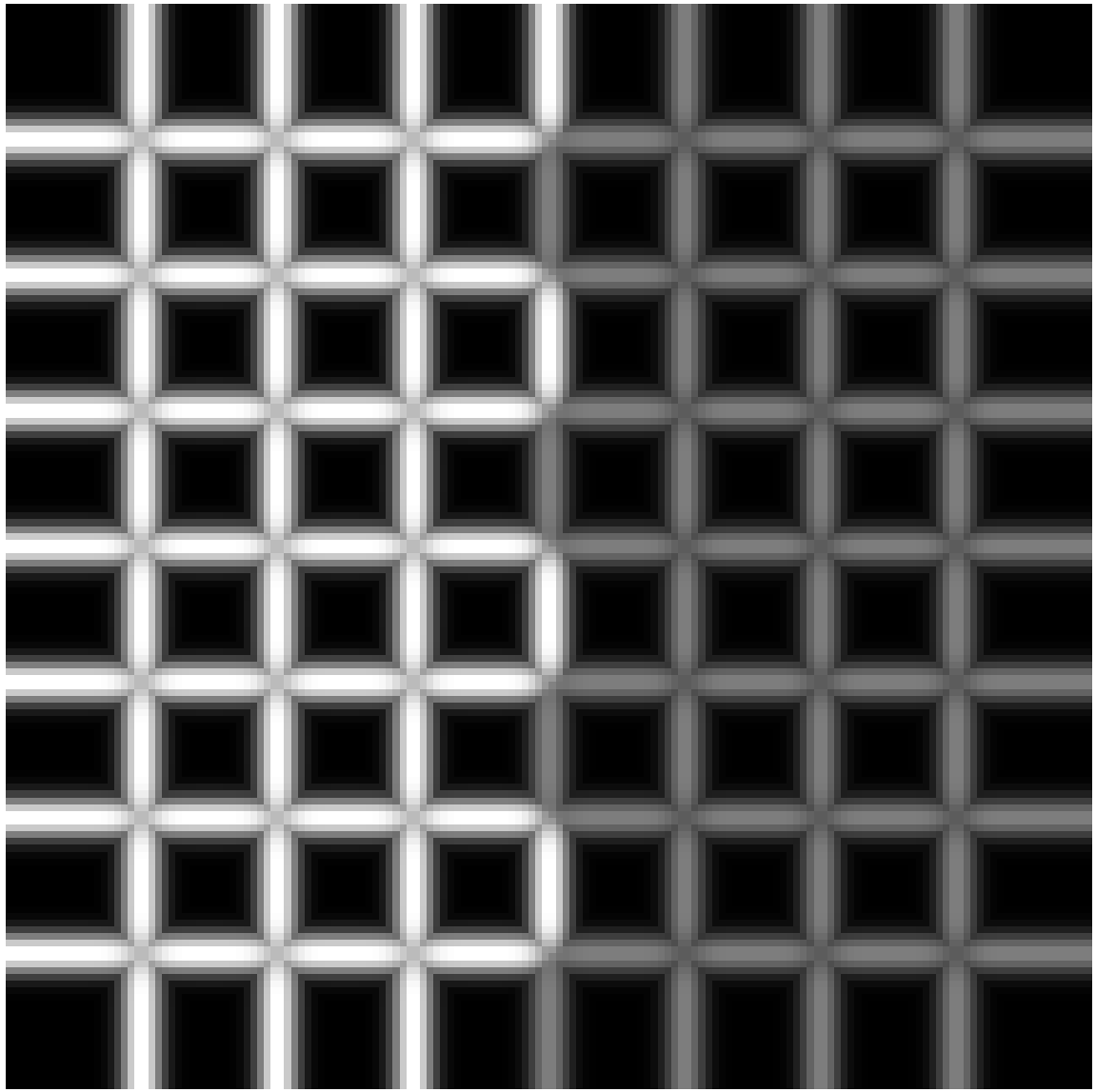
Reminder: let $\lambda_1 \geq \lambda_2$ be the Eigenvalues of the structure tensor (they differ from point to point), then $\sigma_1 = \sqrt{\lambda_1}$ and $\sigma_2 = \sqrt{\lambda_2}$.

Example



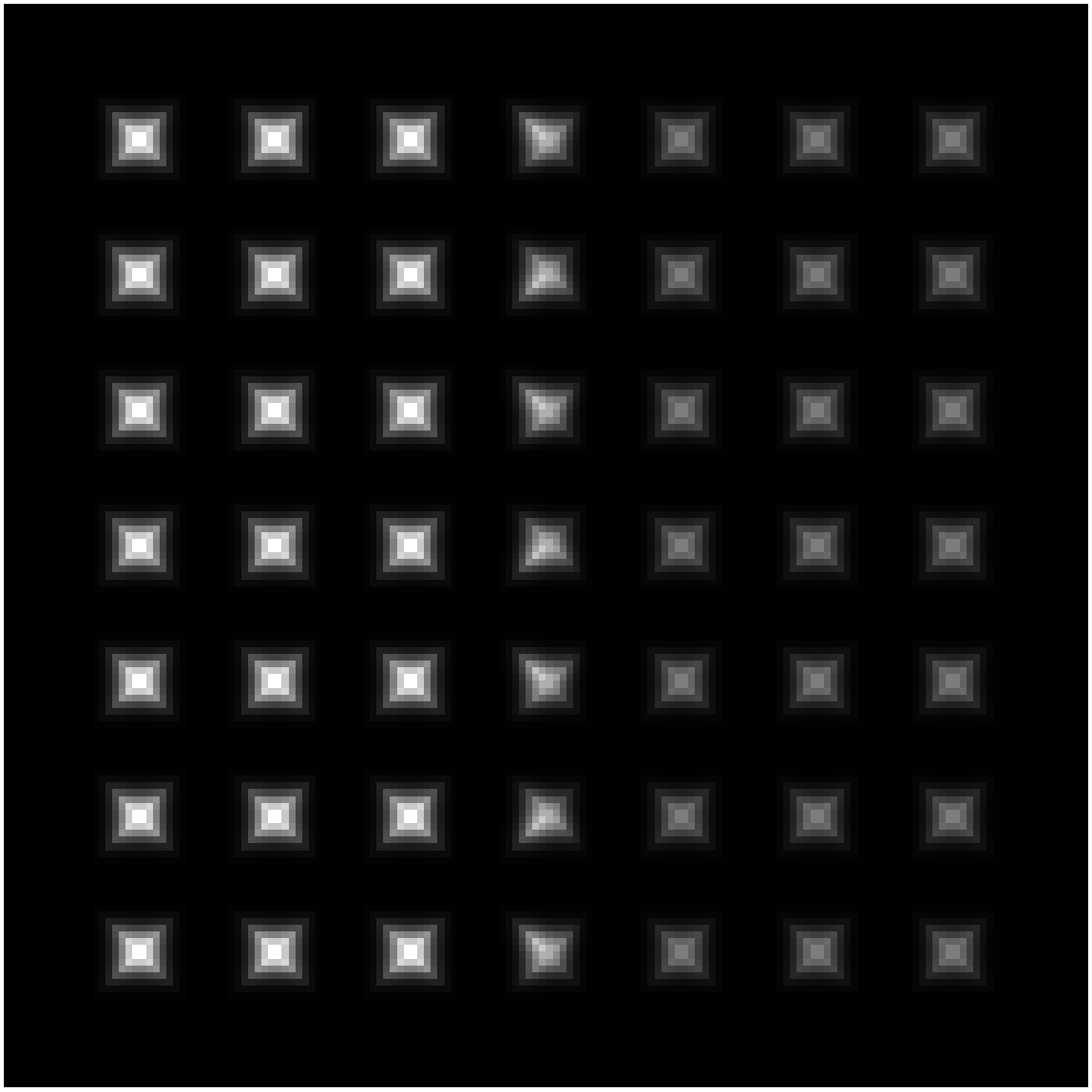
Input image f

Example



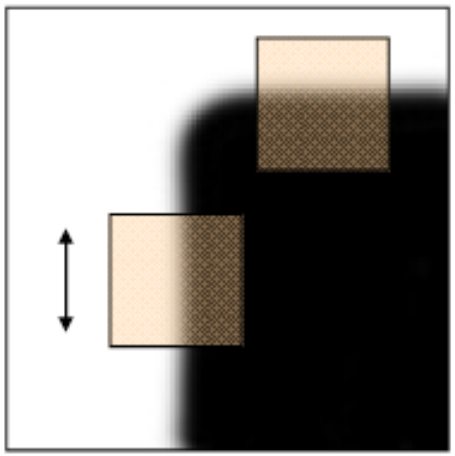
$\lambda_1, \sigma = 1.0, \tau = 2.0$

Example



$\lambda_2, \sigma = 1.0, \tau = 2.0$

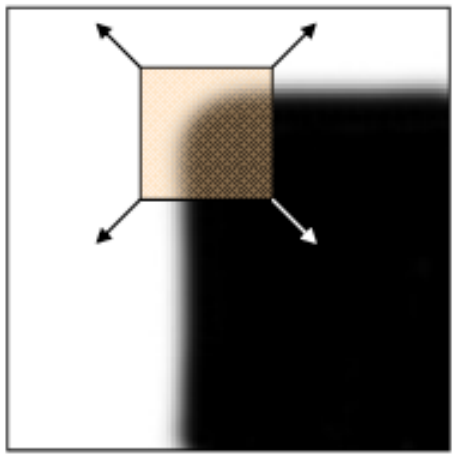
Pixel characterization based on Eigenvalues λ_1, λ_2 of $\mathcal{T}$



"edge":

$\lambda_1 >> \lambda_2$

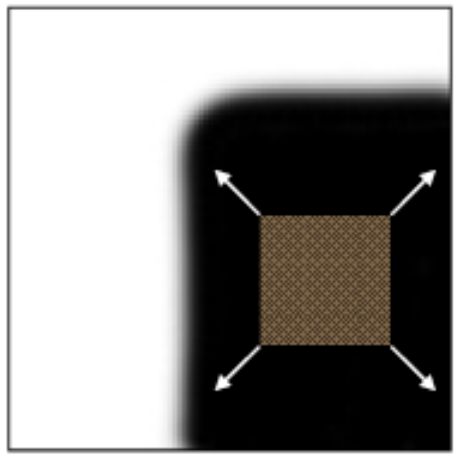
$\lambda_2 >> \lambda_1$



"corner":

λ_1 and λ_2 are large,

$\lambda_1 \sim \lambda_2$;

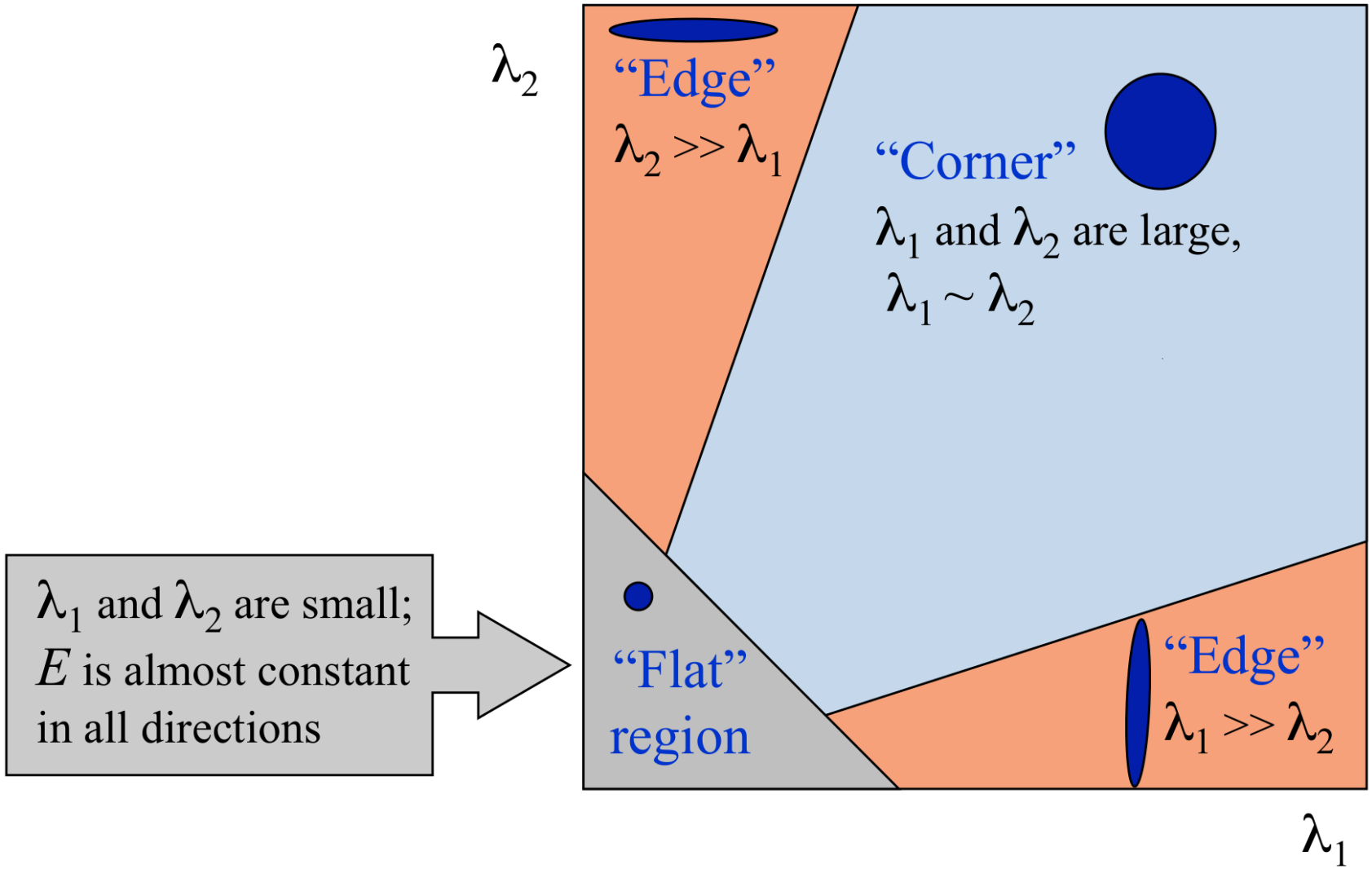


"flat" region

λ_1 and λ_2 are small;

In this illustration, the Eigenvalues are not sorted!

Pixel characterization based on Eigenvalues λ_1, λ_2 of $\mathcal{T}$



In this illustration, the Eigenvalues are not sorted!

“Cornerness” response functions

Let $\lambda_1 \geq \lambda_2$. Different functions have been proposed to quantify how much the point looks like a corner.

“Cornerness” response functions

Let $\lambda_1 \geq \lambda_2$. Different functions have been proposed to quantify how much the point looks like a corner.

- **Shi and Tomasi, 1994**: use smaller Eigenvalue λ_2.

“Cornerness” response functions

Let $\lambda_1 \geq \lambda_2$. Different functions have been proposed to quantify how much the point looks like a corner.

- **Shi and Tomasi, 1994**: use smaller Eigenvalue λ_2.
- **Harris 1988, Foerstner 1986**: use

$$\det(\mathcal{T}) - \kappa \operatorname{trace}(\mathcal{T})^2 = \lambda_1 \lambda_2 - \kappa(\lambda_1 + \lambda_2)^2$$

for parameter $\kappa > 0$ (needs to be adjusted, $0.04 - 0.15$ are reported to work well).

“Cornerness” response functions

Let $\lambda_1 \geq \lambda_2$. Different functions have been proposed to quantify how much the point looks like a corner.

- **Shi and Tomasi, 1994**: use smaller Eigenvalue λ_2.
- **Harris 1988, Foerstner 1986**: use

$$\det(\mathcal{T}) - \kappa \operatorname{trace}(\mathcal{T})^2 = \lambda_1 \lambda_2 - \kappa(\lambda_1 + \lambda_2)^2$$

  for parameter $\kappa > 0$ (needs to be adjusted, $0.04 - 0.15$ are reported to work well).

A corner is reported when the detector value

- is above a certain threshold $\theta > 0$
- and attains a local maximum

Example

Input image f

Example

$\lambda_1, \sigma = 1.0, \tau = 2.0$

Example

$\lambda_2, \sigma = 1.0, \tau = 2.0$

Example

Harris response, $\kappa = 0.04$

Example

Detected corners, $\theta = 0.0002$

Example

Detected corners, $\theta = 0.00002$

Example

Orientation (Eigenvector corresponding to λ_1)

Example

Orientation (Eigenvector corresponding to λ_1) scaled by λ_1

Overview

We now have detected useful feature points ...
but how to compare features in different images?

Patch comparison: sum of squared differences

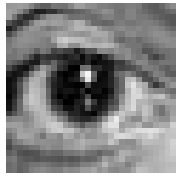
Given two rectangular image patches f, g on $\{1, \ldots, W\} \times \{1, \ldots H\}$, then the **sum of squared differences** patch distance is given as

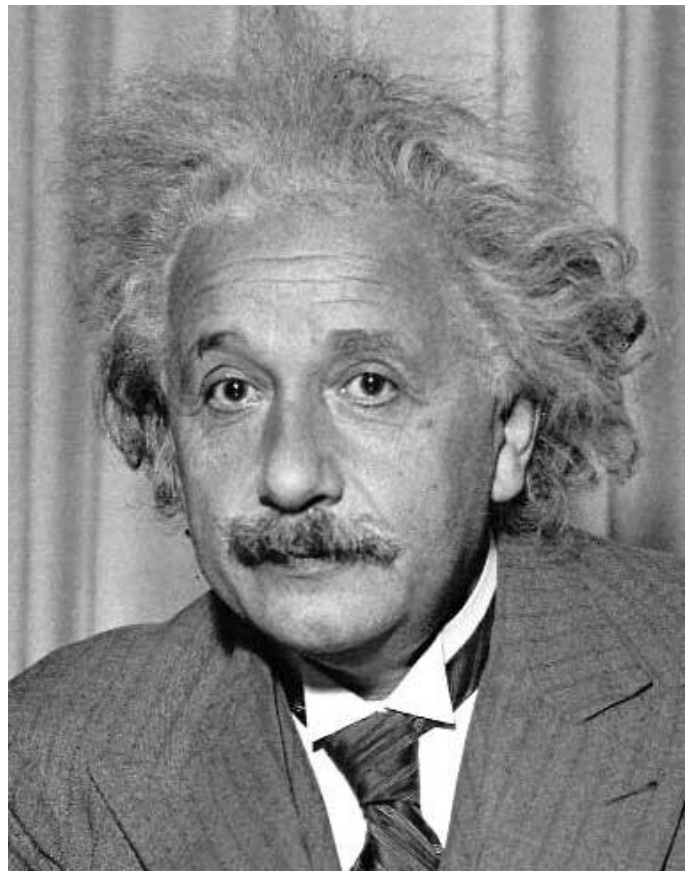
$$E_{\text{SSD}}(f, g) = \sum_{i=1}^{W} \sum_{j=1}^{H} \big(f(i,j) - g(i,j)\big)^2.$$

The smaller, the better the fit.

- *Often, SSD is divided by $W \cdot H$ to normalize it.*
- *Compare to mean square error, or the patch distance which is used for non-local means.*

Using SSD to find a patch in an image

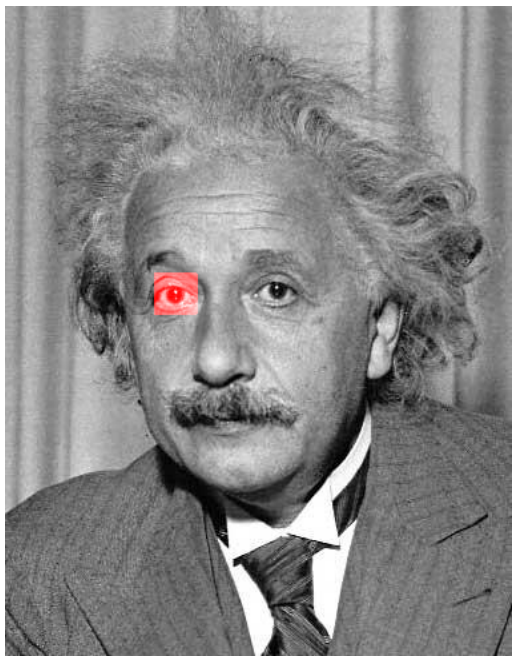
Goal: Find  in the image.



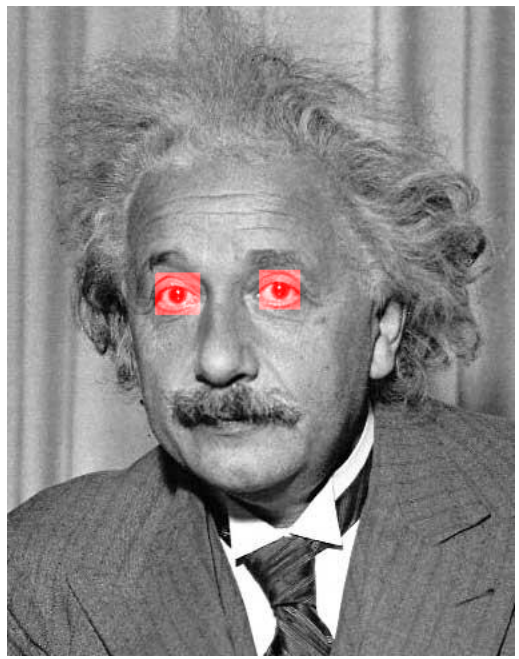
Method: Compare patch to be found to patch of same size at every image location, threshold result.

Results: SSD

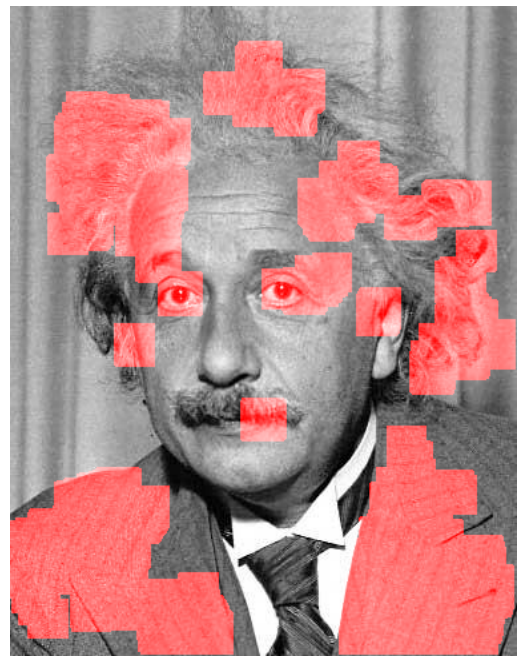
Threshold becoming less strict from left to right: within θ% of maximum error, value of θ shown below each image.



$\theta = 90$



$\theta = 88$



$\theta = 85$

How good is SSD as a patch comparison metric?

Fast

How good is SSD as a patch comparison metric?

Fast	yes
Invariant to translation	

How good is SSD as a patch comparison metric?

Fast	yes
Invariant to translation	yes
Invariant to rotation	

How good is SSD as a patch comparison metric?

Fast	yes
Invariant to translation	yes
Invariant to rotation	no
Invariant to scaling	

How good is SSD as a patch comparison metric?

Fast	yes
Invariant to translation	yes
Invariant to rotation	no
Invariant to scaling	no
Invariant to contrast/brightness changes	?

Let's test it ...

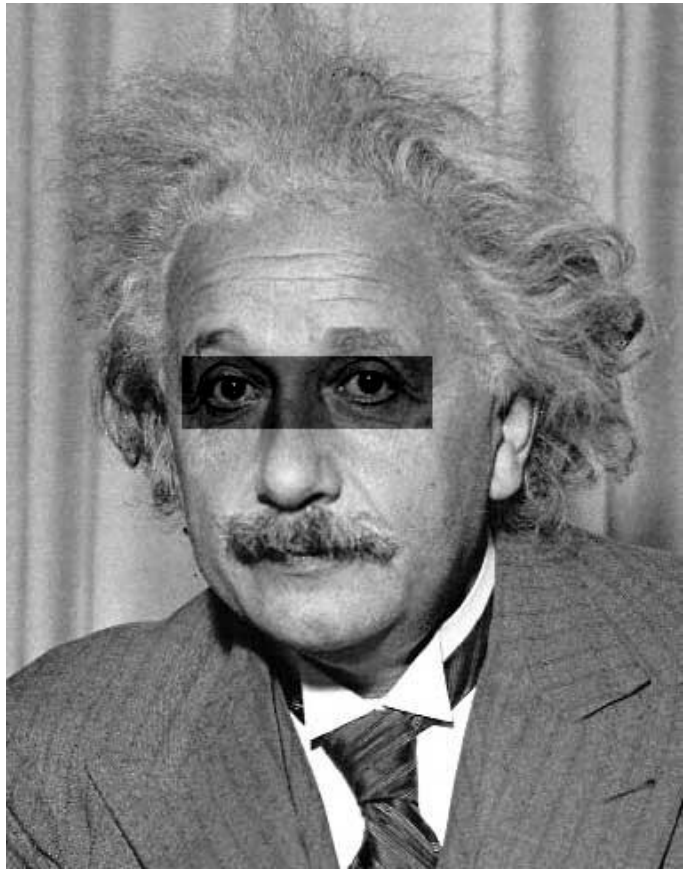
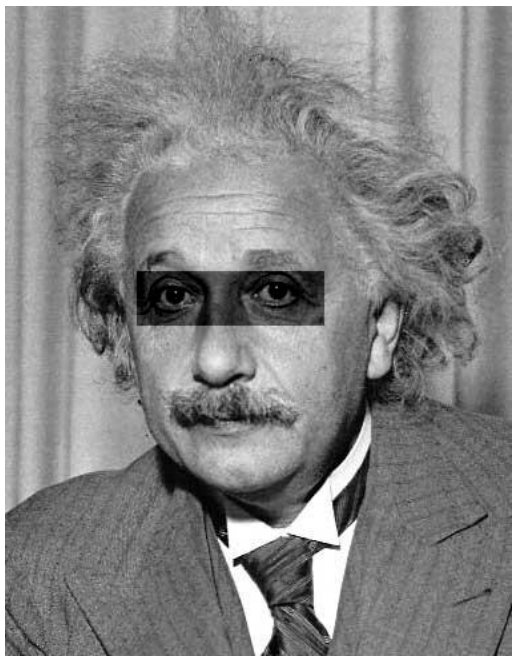


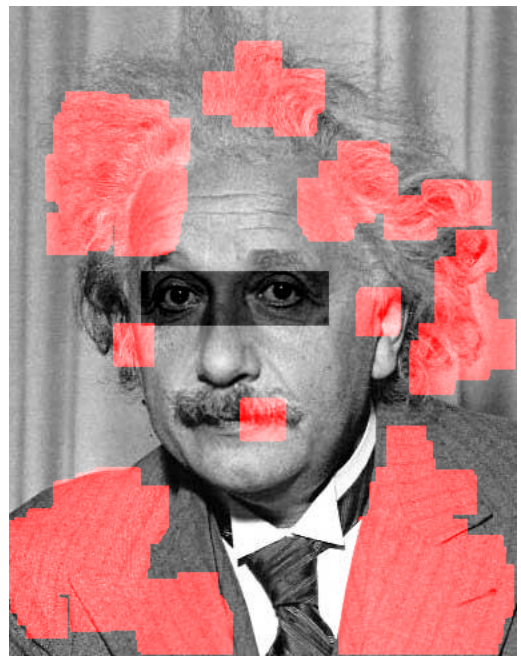
Image intensity darkened in regions where the patch can be found.

Results: SSD, brightness invariance

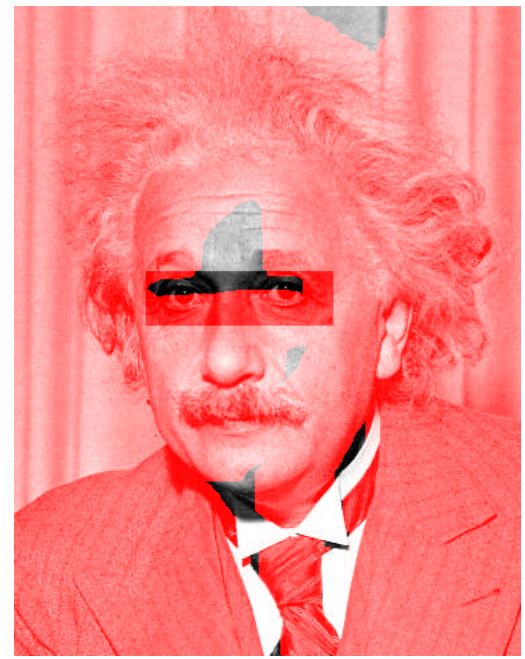
Threshold becoming less strict from left to right: within θ% of maximum error, value of θ shown below each image.



$\theta = 90$



$\theta = 85$



$\theta = 70$

How good is SSD as a patch comparison metric?

Fast	yes
Invariant to translation	yes
Invariant to rotation	no
Invariant to scaling	no
Invariant to contrast/brightness changes	no

Let's look for something better ...

Given two rectangular image patches f, g on $\{1, \ldots, W\} \times \{1, \ldots H\}$,

Let's look for something better ...

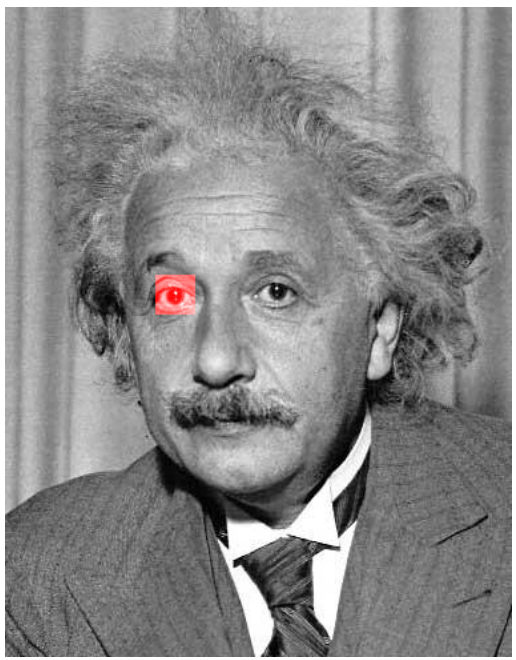
Given two rectangular image patches f, g on $\{1, \ldots, W\} \times \{1, \ldots H\}$, then the **normalized cross-correlation** patch distance is given as

$$E_{\text{NCC}}(f, g) = \frac{cov(f, g)}{\sigma(f)\sigma(g)}.$$

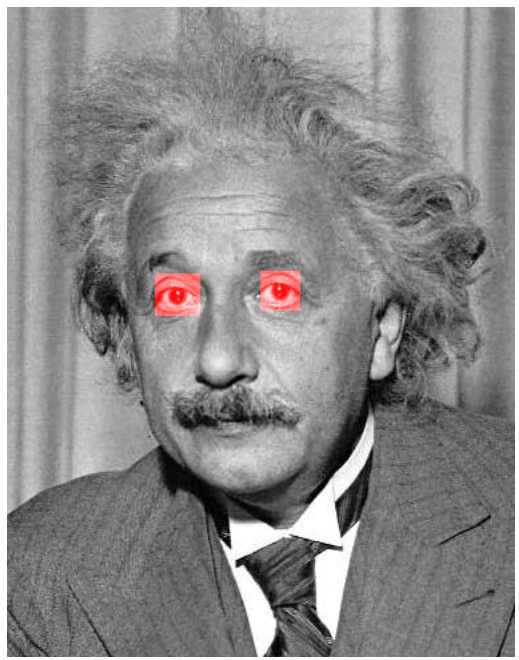
The range is within $[-1, 1]$, larger means a better fit (below zero it is actually the opposite of a fit).

Results: NCC

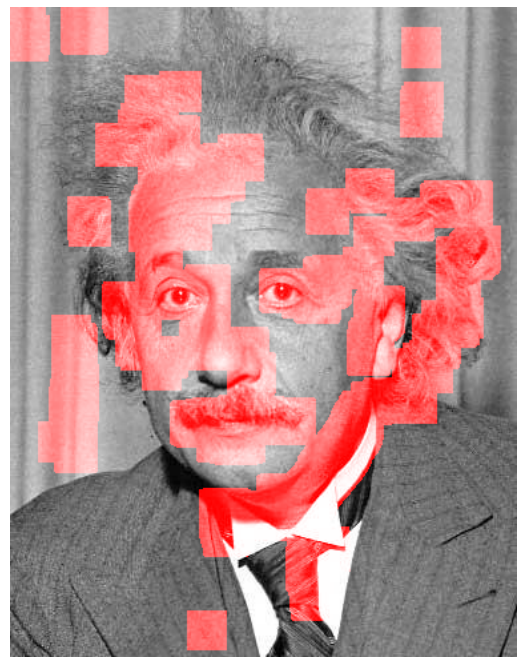
Threshold becoming less strict from left to right: patches show with NCC of at least θ of maximum error, value of θ shown below each image.



$\theta = 0.9$



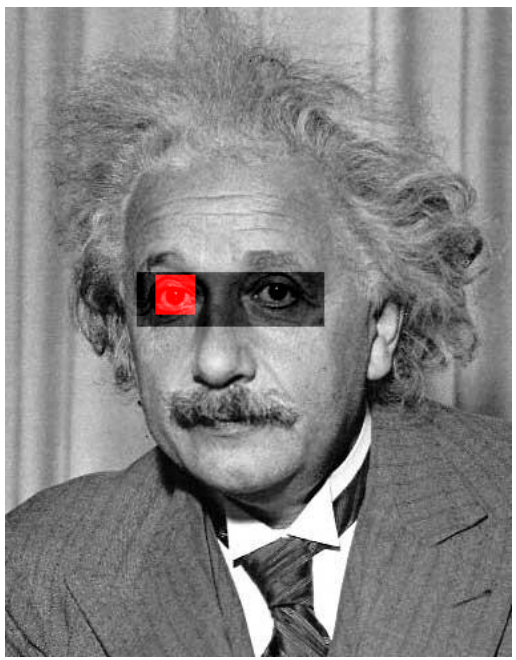
$\theta = 0.6$



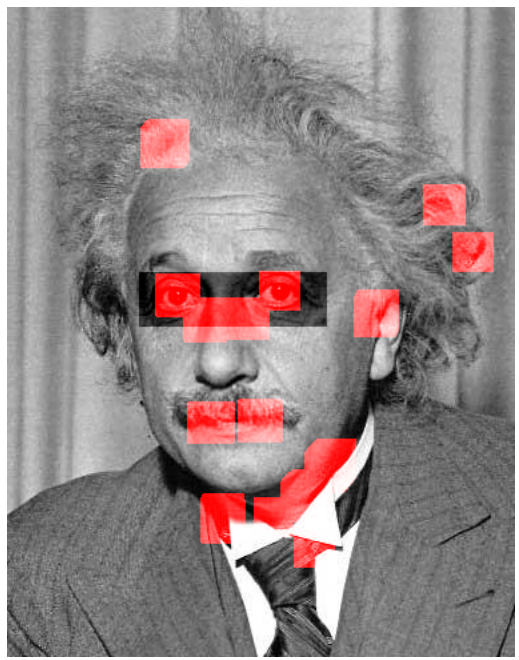
$\theta = 0.3$

Results: NCC, brightness invariance

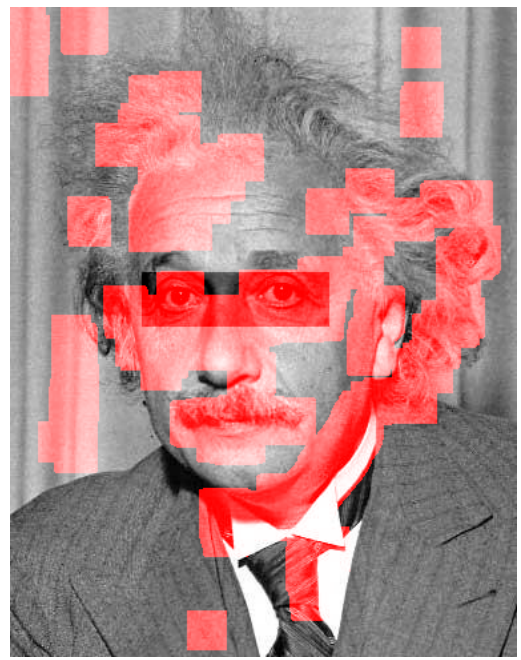
Threshold becoming less strict from left to right: patches show with NCC of at least θ of maximum error, value of θ shown below each image.



$\theta = 0.8$



$\theta = 0.44$



$\theta = 0.3$

How good is NCC as a patch comparison metric?

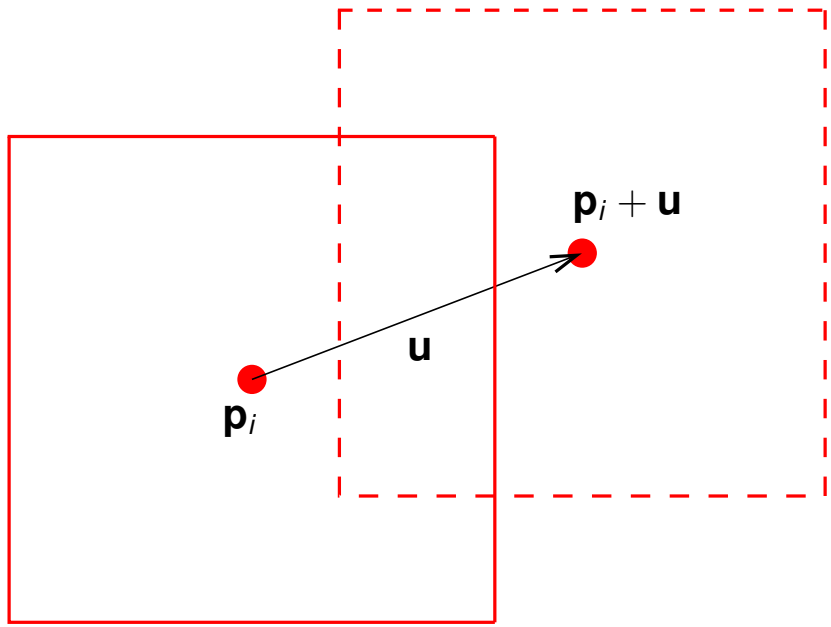
Fast	no
Invariant to translation	yes
Invariant to rotation	no
Invariant to scaling	no
Invariant to contrast/brightness changes	yes

Scale will be considered in chapter 3, rotation invariance in chapter 4, when we finally discuss the scale-invariant feature transform.

We will now check what happens to the error metrics when we slightly vary the patch location.

Autocorrelation idea

- **Idea:** we displace a patch location slightly by a shift vector **u**, and check what happens when we compare the previous patch to the new one using (weighted) SSD.
- Should give an idea about the *stability* of matching this patch.



Notation: *pixels in patch indexed by i, pixel locations $\mathbf{p}_i$.*
Side note: *the naming is somewhat confusing, it would be better to call it "Auto-SSD" instead - in signal processing, one usually uses cross-correlation instead of SSD to define autocorrelation, then the name makes sense.*

Autocorrelation definition

- As usual, we allow a weight $w(\mathbf{p}_i)$ to be defined in each pixel, e.g. a Gaussian distribution of standard deviation τ around the patch center.

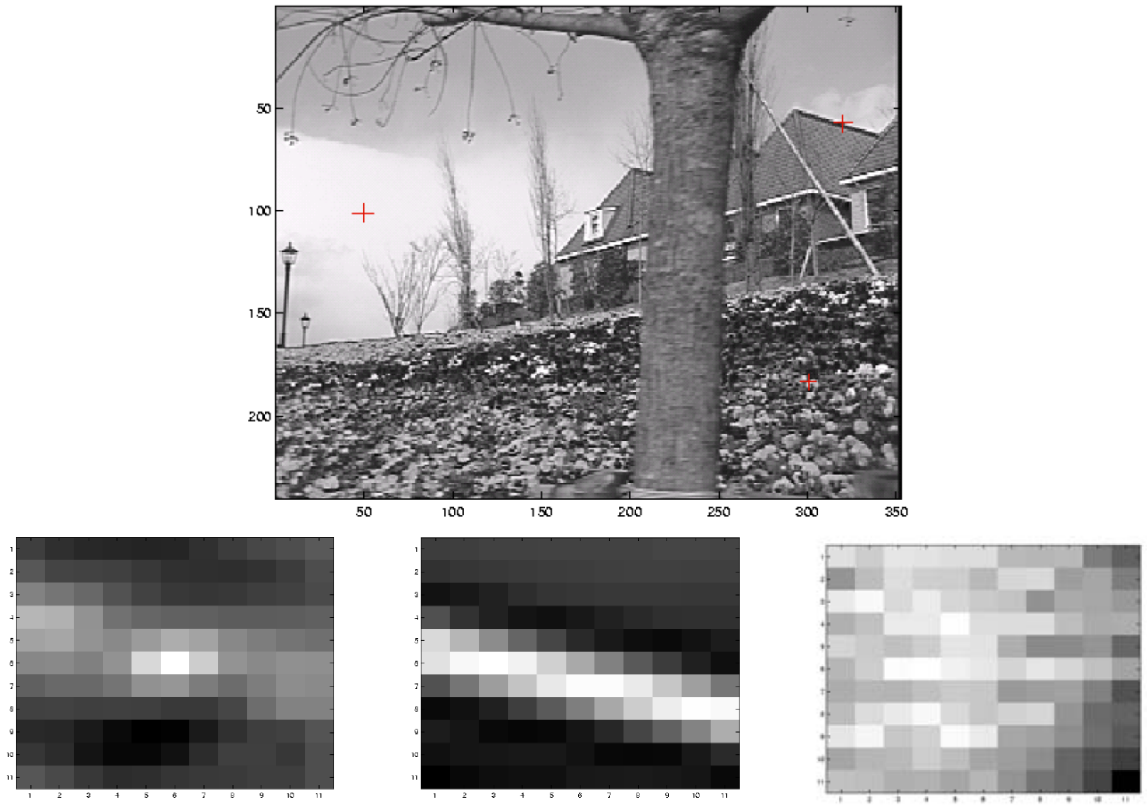
- The **autocorrelation** for patch f and shift $\mathbf{u}$ and weight w is defined as the weighted SSD between f and f shifted by $\mathbf{u}$,

$$E_{\mathrm{AC}}(f, \mathbf{u}) := \sum_i w(\mathbf{p}_i) \left(f(\mathbf{p}_i + \mathbf{u}) - f(\mathbf{p}_i)\right)^2,$$

  where the sum runs over all pixels.

Autocorrelation and feature quality

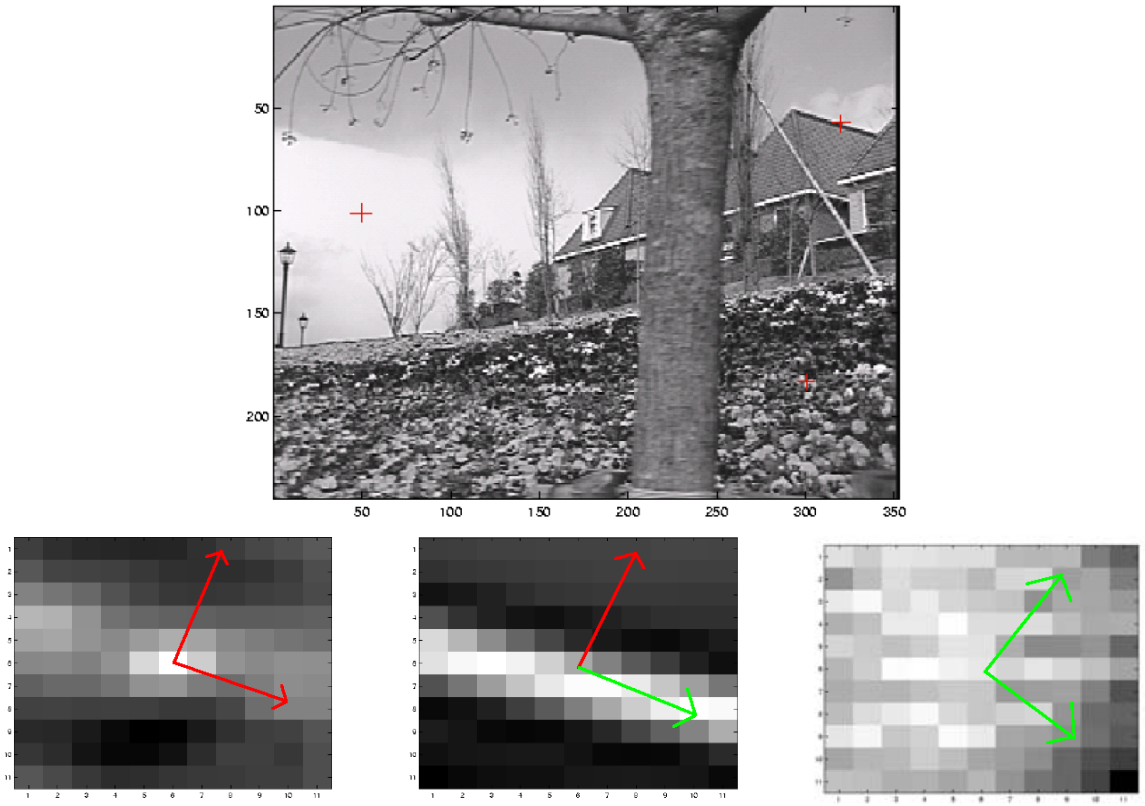
Intuition: autocorrelation will be small (there is small patch difference) if a patch is shifted such that it still looks similar.



Images: R. Szeliski

Autocorrelation and feature quality

Intuition: autocorrelation will be small (there is small patch difference) if a patch is shifted such that it still looks similar.



- green arrows: $E_{\mathrm{AC}}(\mathbf{u})$ small for points in this direction,
- red arrows: $E_{\mathrm{AC}}(\mathbf{u})$ large for points in this direction,
- features where all directions are red can be optimally localized.

Images: R. Szeliski

Autocorrelation approximation for small shifts u

- Assume small shift **u**
- **Step 1:** Taylor series approximation of the image f around each pixel,

$$f(\mathbf{p}_i + \mathbf{u}) \approx f(\mathbf{p}_i) +$$

Autocorrelation approximation for small shifts u

- Assume small shift **u**
- **Step 1:** Taylor series approximation of the image f around each pixel,

$$f(\mathbf{p}_i + \mathbf{u}) \approx f(\mathbf{p}_i) + \nabla f(\mathbf{p}_i)^T \cdot \mathbf{u}.$$

Autocorrelation approximation for small shifts u

- Assume small shift **u**
- **Step 1:** Taylor series approximation of the image f around each pixel,

$$f(\mathbf{p}_i + \mathbf{u}) \approx f(\mathbf{p}_i) + \nabla f(\mathbf{p}_i)^T \cdot \mathbf{u}.$$

- **Step 2:** Compute E_{AC} using this approximation:

$$E_{\mathrm{AC}}(f, \mathbf{u}) := \sum_i w(\mathbf{p}_i) \left(f(\mathbf{p}_i + \mathbf{u}) - f(\mathbf{p}_i)\right)^2$$

Autocorrelation approximation for small shifts u

- Assume small shift **u**
- **Step 1:** Taylor series approximation of the image f around each pixel,

$$f(\mathbf{p}_i + \mathbf{u}) \approx f(\mathbf{p}_i) + \nabla f(\mathbf{p}_i)^T \cdot \mathbf{u}.$$

- **Step 2:** Compute E_{AC} using this approximation:

$$\begin{aligned} E_{\mathrm{AC}}(f, \mathbf{u}) &:= \sum_i w(\mathbf{p}_i) \left(f(\mathbf{p}_i + \mathbf{u}) - f(\mathbf{p}_i) \right)^2 \\ &\approx \sum_i w(\mathbf{p}_i) \left(f(\mathbf{p}_i) + \nabla f(\mathbf{p}_i)^T \cdot \mathbf{u} - f(\mathbf{p}_i) \right)^2 \end{aligned}$$

Autocorrelation approximation for small shifts u

- Assume small shift **u**
- **Step 1:** Taylor series approximation of the image f around each pixel,

$$f(\mathbf{p}_i + \mathbf{u}) \approx f(\mathbf{p}_i) + \nabla f(\mathbf{p}_i)^T \cdot \mathbf{u}.$$

- **Step 2:** Compute E_{AC} using this approximation:

$$\begin{aligned} E_{\mathrm{AC}}(f, \mathbf{u}) &:= \sum_i w(\mathbf{p}_i) \left(f(\mathbf{p}_i + \mathbf{u}) - f(\mathbf{p}_i)\right)^2 \\ &\approx \sum_i w(\mathbf{p}_i) \left(f(\mathbf{p}_i) + \nabla f(\mathbf{p}_i)^T \cdot \mathbf{u} - f(\mathbf{p}_i)\right)^2 \\ &= \sum_i w(\mathbf{p}_i) \left(\nabla f(\mathbf{p}_i)^T \cdot \mathbf{u}\right)^2 \end{aligned}$$

Autocorrelation approximation for small shifts u

- Assume small shift **u**
- **Step 1:** Taylor series approximation of the image f around each pixel,

$$f(\mathbf{p}_i + \mathbf{u}) \approx f(\mathbf{p}_i) + \nabla f(\mathbf{p}_i)^T \cdot \mathbf{u}.$$

- **Step 2:** Compute E_{AC} using this approximation:

$$\begin{aligned} E_{\mathrm{AC}}(f, \mathbf{u}) &:= \sum_i w(\mathbf{p}_i) \left(f(\mathbf{p}_i + \mathbf{u}) - f(\mathbf{p}_i)\right)^2 \\ &\approx \sum_i w(\mathbf{p}_i) \left(f(\mathbf{p}_i) + \nabla f(\mathbf{p}_i)^T \cdot \mathbf{u} - f(\mathbf{p}_i)\right)^2 \\ &= \sum_i w(\mathbf{p}_i) \left(\nabla f(\mathbf{p}_i)^T \cdot \mathbf{u}\right)^2 \\ &= \sum_i w(\mathbf{p}_i)\, \mathbf{u}^T \left(\nabla f(\mathbf{p}_i) \nabla f(\mathbf{p}_i)^T\right) \mathbf{u} \end{aligned}$$

Autocorrelation approximation for small shifts u

- Assume small shift **u**
- **Step 1:** Taylor series approximation of the image f around each pixel,

$$f(\mathbf{p}_i + \mathbf{u}) \approx f(\mathbf{p}_i) + \nabla f(\mathbf{p}_i)^T \cdot \mathbf{u}.$$

- **Step 2:** Compute E_{AC} using this approximation:

$$\begin{aligned}
E_{\mathrm{AC}}(f, \mathbf{u}) :&= \sum_i w(\mathbf{p}_i) \left(f(\mathbf{p}_i + \mathbf{u}) - f(\mathbf{p}_i)\right)^2 \\
&\approx \sum_i w(\mathbf{p}_i) \left(f(\mathbf{p}_i) + \nabla f(\mathbf{p}_i)^T \cdot \mathbf{u} - f(\mathbf{p}_i)\right)^2 \\
&= \sum_i w(\mathbf{p}_i) \left(\nabla f(\mathbf{p}_i)^T \cdot \mathbf{u}\right)^2 \\
&= \sum_i w(\mathbf{p}_i)\, \mathbf{u}^T \left(\nabla f(\mathbf{p}_i) \nabla f(\mathbf{p}_i)^T\right) \mathbf{u} \\
&= \mathbf{u}^T \left(\sum_i w(\mathbf{p}_i) \nabla f(\mathbf{p}_i) \nabla f(\mathbf{p}_i)^T\right) \mathbf{u}
\end{aligned}$$

So what is this object in the middle ...

It's a 2×2 matrix, let's call it $\mathcal{T}$:

$$\mathcal{T} = \sum_i w(\mathbf{p}_i) \nabla f(\mathbf{p}_i) \nabla f(\mathbf{p}_i)^T$$

So what is this object in the middle ...

It's a 2×2 matrix, let's call it $\mathcal{T}$:

$$\mathcal{T} = \sum_i w(\mathbf{p}_i) \nabla f(\mathbf{p}_i) \nabla f(\mathbf{p}_i)^T$$

$$= \sum_i w(\mathbf{p}_i) \begin{bmatrix} f_x^2(\mathbf{p}_i) & f_x(\mathbf{p}_i) f_y(\mathbf{p}_i) \\ f_x(\mathbf{p}_i) f_y(\mathbf{p}_i) & f_y^2(\mathbf{p}_i) \end{bmatrix}$$

So what is this object in the middle ...

It's a 2×2 matrix, let's call it $\mathcal{T}$:

$$\mathcal{T} = \sum_i w(\mathbf{p}_i) \nabla f(\mathbf{p}_i) \nabla f(\mathbf{p}_i)^T$$

$$= \sum_i w(\mathbf{p}_i) \begin{bmatrix} f_x^2(\mathbf{p}_i) & f_x(\mathbf{p}_i) f_y(\mathbf{p}_i) \\ f_x(\mathbf{p}_i) f_y(\mathbf{p}_i) & f_y^2(\mathbf{p}_i) \end{bmatrix}$$

$$= \begin{bmatrix} cov(f_x, f_x) & cov(f_x, f_y) \\ cov(f_y, f_x) & cov(f_y, f_y) \end{bmatrix}$$

So what is this object in the middle ...

It's a 2×2 matrix, let's call it $\mathcal{T}$:

$$\mathcal{T} = \sum_i w(\mathbf{p}_i) \nabla f(\mathbf{p}_i) \nabla f(\mathbf{p}_i)^T$$

$$= \sum_i w(\mathbf{p}_i) \begin{bmatrix} f_x^2(\mathbf{p}_i) & f_x(\mathbf{p}_i) f_y(\mathbf{p}_i) \\ f_x(\mathbf{p}_i) f_y(\mathbf{p}_i) & f_y^2(\mathbf{p}_i) \end{bmatrix}$$

$$= \begin{bmatrix} cov(f_x, f_x) & cov(f_x, f_y) \\ cov(f_y, f_x) & cov(f_y, f_y) \end{bmatrix}$$

It's the structure tensor again !

The structure tensor revisited

We now have two interpretations of the structure tensor $\mathcal{T}$, derived from two completely different contexts:

1. $\mathcal{T}$ is **the covariance matrix of the gradient distribution**, giving information about the dominant image gradients in the patch,

2. The function

$$\mathbf{u} \mapsto \mathbf{u}^T \mathcal{T} \mathbf{u}$$

is an approximation to the **autocorrelation for small shifts**, giving information about how stable the patch is with regard to small shifts.

The structure tensor revisited

We now have two interpretations of the structure tensor $\mathcal{T}$, derived from two completely different contexts:

1. $\mathcal{T}$ is **the covariance matrix of the gradient distribution**, giving information about the dominant image gradients in the patch,

2. The function

$$\mathbf{u} \mapsto \mathbf{u}^T \mathcal{T} \mathbf{u}$$

   is an approximation to the **autocorrelation for small shifts**, giving information about how stable the patch is with regard to small shifts.

Question: what is the interpretation of the Eigenvalues and Eigenvectors in the second case?

Autocorrelation: maxima / minima

- Let's look at the direction of fastest increase in error, i.e. find $\mathbf{u}$ with $\|\mathbf{u}\| = 1$ such that $\mathbf{u}^T \mathcal{T} \mathbf{u}$ is maximal.

Autocorrelation: maxima / minima

- Let's look at the direction of fastest increase in error, i.e. find $\mathbf{u}$ with $\|\mathbf{u}\| = 1$ such that $\mathbf{u}^T \mathcal{T} \mathbf{u}$ is maximal.

- Eigenvalue decomposition of $\mathcal{T}$, $\lambda_1 \geq \lambda_2$:

$$\mathbf{u}^T \mathcal{T} \mathbf{u} = (\mathbf{u}^T V) \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} (V^T \mathbf{u}).$$

  This expression is maximal if $V^T \mathbf{u} = \mathbf{e}_1 \Leftrightarrow \mathbf{u} = V\mathbf{e}_1$, i.e. in the direction of the first Eigenvector of $\mathcal{T}$.

Autocorrelation: maxima / minima

- Let's look at the direction of fastest increase in error, i.e. find $\mathbf{u}$ with $\|\mathbf{u}\| = 1$ such that $\mathbf{u}^T \mathcal{T} \mathbf{u}$ is maximal.

- Eigenvalue decomposition of $\mathcal{T}$, $\lambda_1 \geq \lambda_2$:

$$\mathbf{u}^T \mathcal{T} \mathbf{u} = (\mathbf{u}^T V) \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} (V^T \mathbf{u}).$$

  This expression is maximal if $V^T \mathbf{u} = \mathbf{e}_1 \Leftrightarrow \mathbf{u} = V\mathbf{e}_1$, i.e. in the direction of the first Eigenvector of $\mathcal{T}$.

- If you think about it, that is as is should be, as in this direction there are statistically the strongest image gradients.

Autocorrelation: maxima / minima

- Let's look at the direction of fastest increase in error, i.e. find $\mathbf{u}$ with $\|\mathbf{u}\| = 1$ such that $\mathbf{u}^T \mathcal{T} \mathbf{u}$ is maximal.

- Eigenvalue decomposition of $\mathcal{T}$, $\lambda_1 \geq \lambda_2$:

$$\mathbf{u}^T \mathcal{T} \mathbf{u} = (\mathbf{u}^T V) \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} (V^T \mathbf{u}).$$

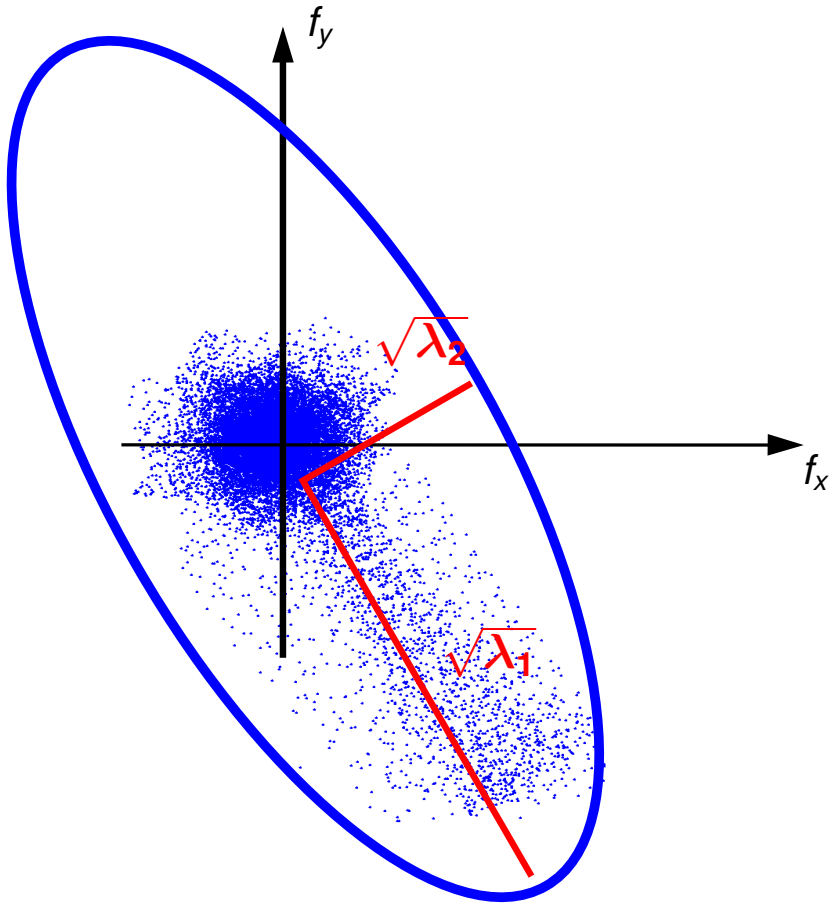
  This expression is maximal if $V^T \mathbf{u} = \mathbf{e}_1 \Leftrightarrow \mathbf{u} = V\mathbf{e}_1$, i.e. in the direction of the first Eigenvector of $\mathcal{T}$.

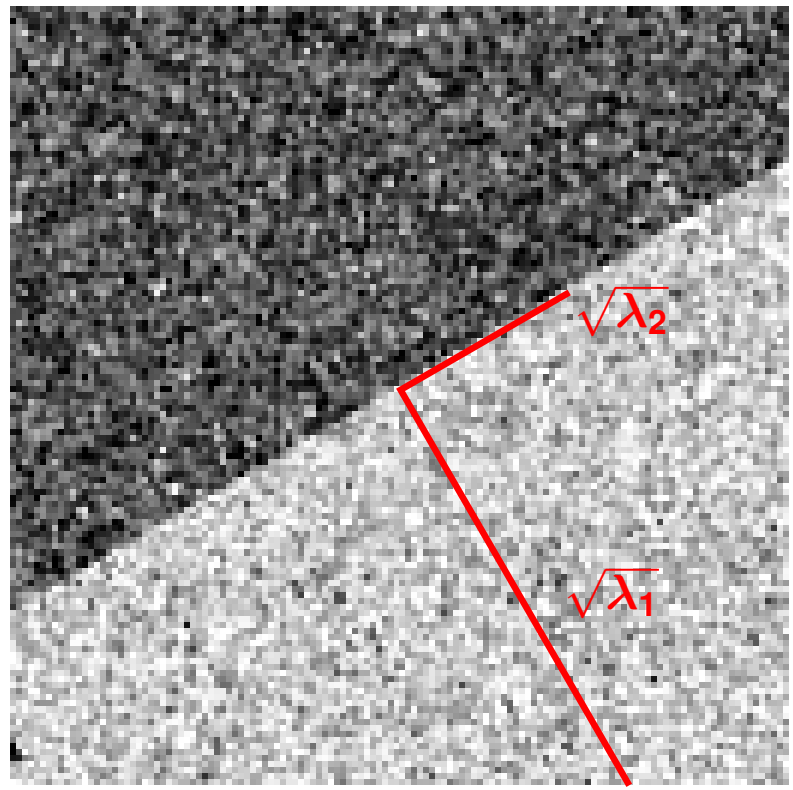
- If you think about it, that is as is should be, as in this direction there are statistically the strongest image gradients.

- Similarly, the direction of minimal increase in error corresponds to the second Eigenvector of $\mathcal{T}$.

The structure tensor: a slide to remember

Visualization of Eigenvalues $\lambda_1 \geq \lambda_2$ of the structure tensor:





Statistics view:
λ_1, λ_2 are Eigenvalues of covariance matrix for f_x, f_y, given variance in Eigenvector direction.

Calculus view:
λ_1, λ_2 measure increase of SSD patch comparison metric when shifting in Eigenvector direction.

Overview

Summary I: edges

- This chapter was mostly about detecting things in images.
- The first thing of interest are edges. Humans usually associate them with object boundaries, but finding object boundaries is impossible at a low level of image understanding.
- We therefore resort to defining edges as locations of strong image gradients.
- The gradient can be computed at different image scales by pre-filtering with a Gaussian - besides noise removal, this corresponds to selecting more dominant edges when moving towards larger scales.
- A popular edge detector which combines this with non-maxima suppression and a thresholding scheme to follow contours is the Canny edge detector.

Summary II: corners

- Edges are only “localized” in one direction - if a patch containing the edge is moved into the edge direction, it does not change much.
- For image features for matching, we therefore look for corners.
- Corners are characterized by the distribution of the gradient - in particular, strong gradients in two orthogonal directions.
- The structure tensor as the covariance matrix of the gradient distribution is used to detect corners, as its Eigenvalues give the edge strength in Eigenvector direction.
- An alternative interpretation of the structure tensor is as an approximation of autocorrelation, where the Eigenvectors and Eigenvalues describe the rate of change when the patch is shifted.

Summary III: pattern matching

- The sum of squared differences (SSD) can be used for comparing patches in images, in particular to detect occurences of a patch in a target image.
- The SSD is not robust to brightness and contrast changes (changes of the image intensity according to a linear function).
- A matching score which is robust to these changes is the normalized cross correlation (NCC).
- SSD is however much faster to compute than NCC, it depends on the application which one is preferrable.

References

Lecture companion books:

- *Filtering and edges:*
  Szeliski, chapters 3.1, 3.2 and 4.2.
  Forsyth and Ponce, chapters 7.1, 7.2, 7.5, 7.6, 8.1-8.3.
- *Structure tensor, autocorrelation:*
  Szeliski, chapters 4.1.
  Forsyth and Ponce, chapters 8.4 (very superficial treatment).
- *PCA, face classification:*
  Forsyth and Ponce, 22.3.1, 22.3.2.

Additional material:

- *Eigenvectors, PCA:* "A tutorial on Principal Components Analysis" by Lindsay I. Smith, on Ilias (`PCA_Tutorial.pdf`). Note that it has slightly different definitions for some of the statistical measures since it does not use weights.