Robot Vision

TTK4255

Lecture 03 – Feature Detection and Matching

Annette Stahl

(Annette.Stahl@ntnu.no)

Department of Engineering Cybernetics – ITK

NTNU, Trondheim

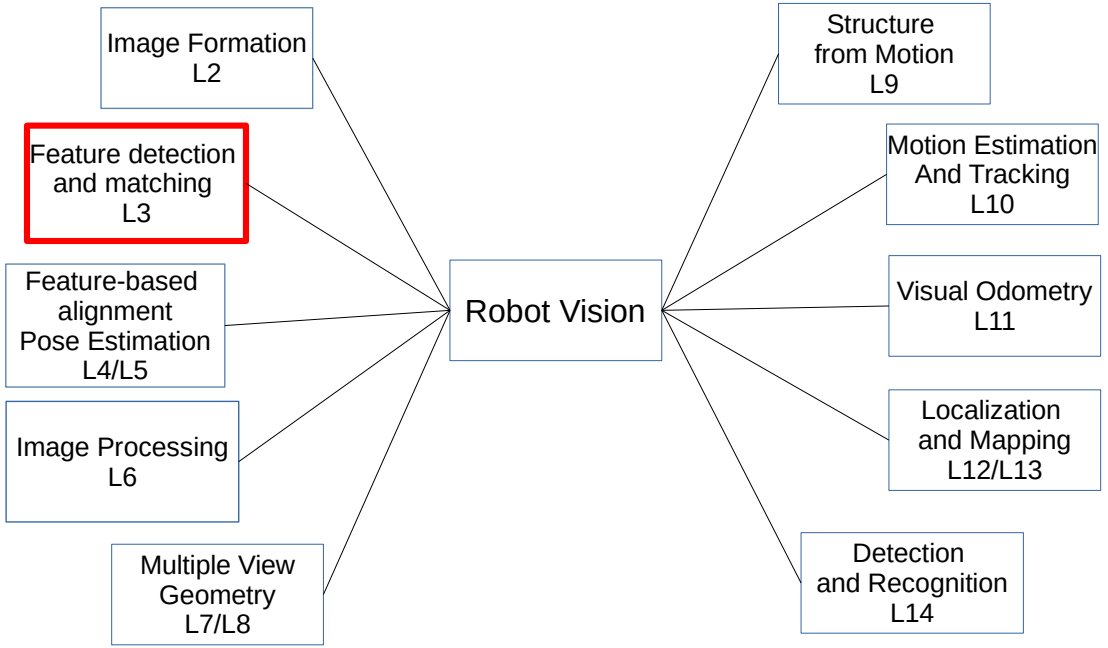
Spring Semester

20. January 2020

Lecture 03 – Feature Detection and Matching

Annette Stahl (Annette.Stahl@ntnu.no)

Simen Haugo (Simen.Haugo@ntnu.no)



Outline of the second lecture:

- Keypoint - correspondences
- Image Gradients
- Features
- Feature Descriptors
- Feature Matching

Recap L02

The perspective camera model is decomposed into 3 transformations:

- Rigid-body motion:

  coordinate transformation between the camera frame and the world frame
- Perspective projection/pinhole camera model:

  projection of 3D camera onto 2D image coordinates
- Image transformation:

  Convert (x,y) to (discretized) pixel coordinates (x',y')

Perspective projection matrix Π

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} fs_x & fs_\theta & o_x \\ 0 & fs_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \\ 1 \end{bmatrix}$$

Intrinsic parameter matrix
Camera matrix K

Extrinsic parameter matrix
3D rotation and translation

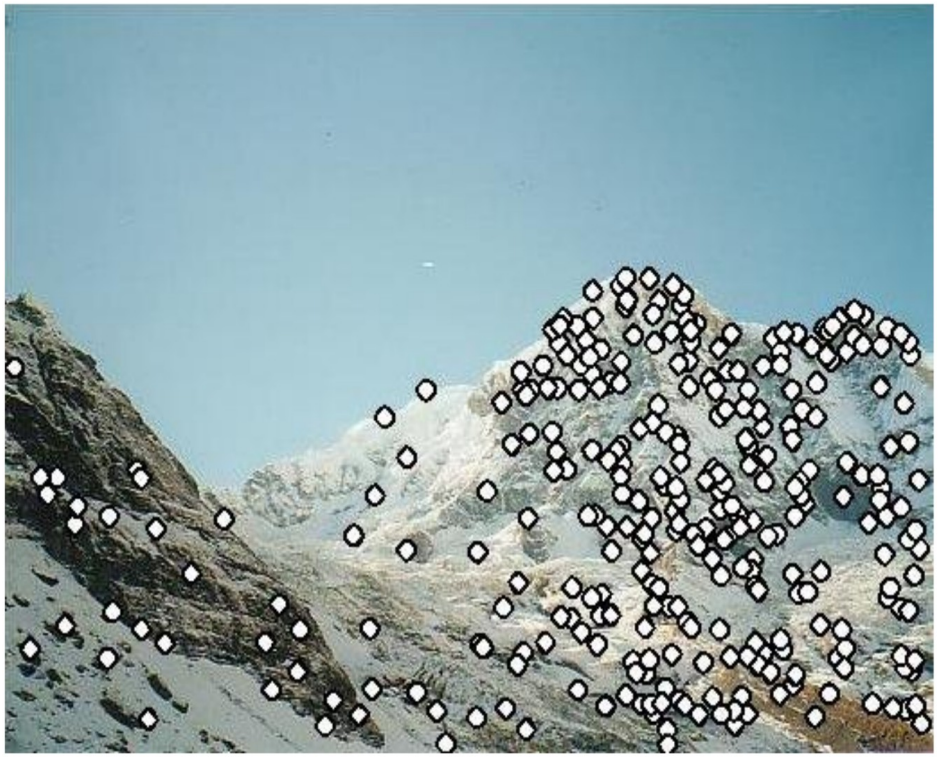
How would you align these images?

Szeliski Book

What kind of features should you detect and then match?

Look for specific locations in the images
→ localized features/keypoint features/interest points, corners
→ edges

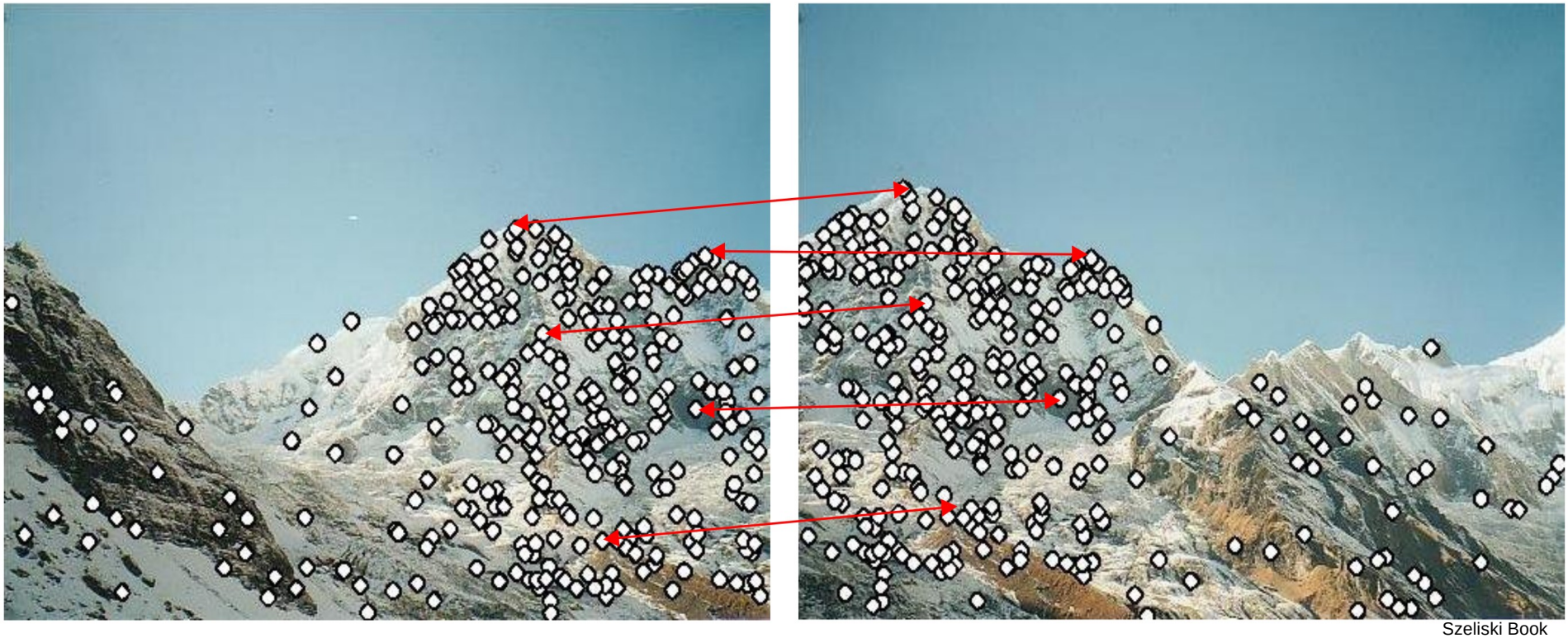
Detect Features



Szeliski Book

Point features / keypoints

Point Correspondences



Szeliski Book

Find corresponding points

Matching with Features

Szeliski Book

- Detect point features in both images
- Find corresponding points
- Use this pairs to align images (matching)

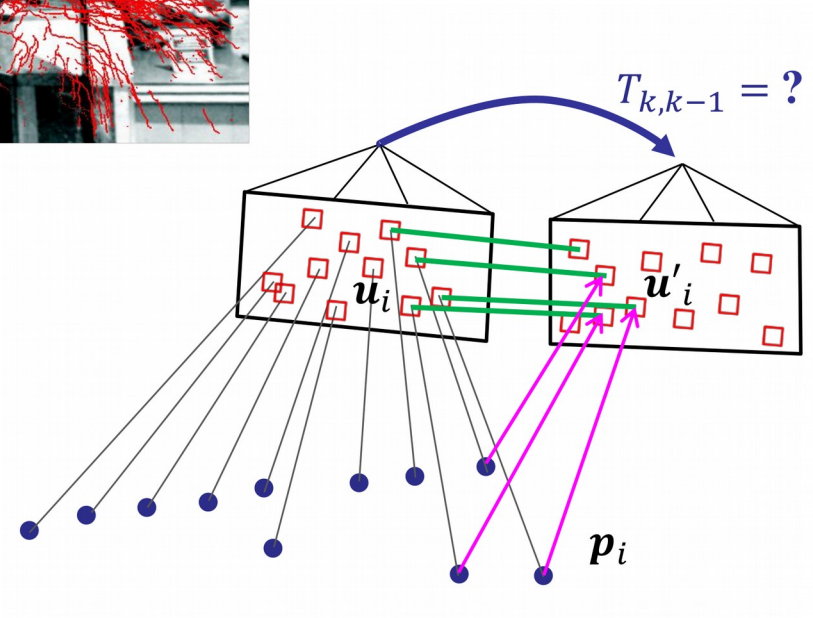
Application – Image Stitching

In the shown example in each image features are computed, matched and the images geometrically transformed. (Example created with Autostitch)
http://matthewalunbrown.com/autostitch/autostitch.html

Feature Matching plays a role in

- Pose estimation
- Motion estimation
- Object tracking

- 3D reconstruction
- Object recognition
- Place recognition
- Robot navigation
- Index and context based retrieval



→ **Image matching problem:** find similar keypoints/point features between two images of the same scene taken under different conditions.

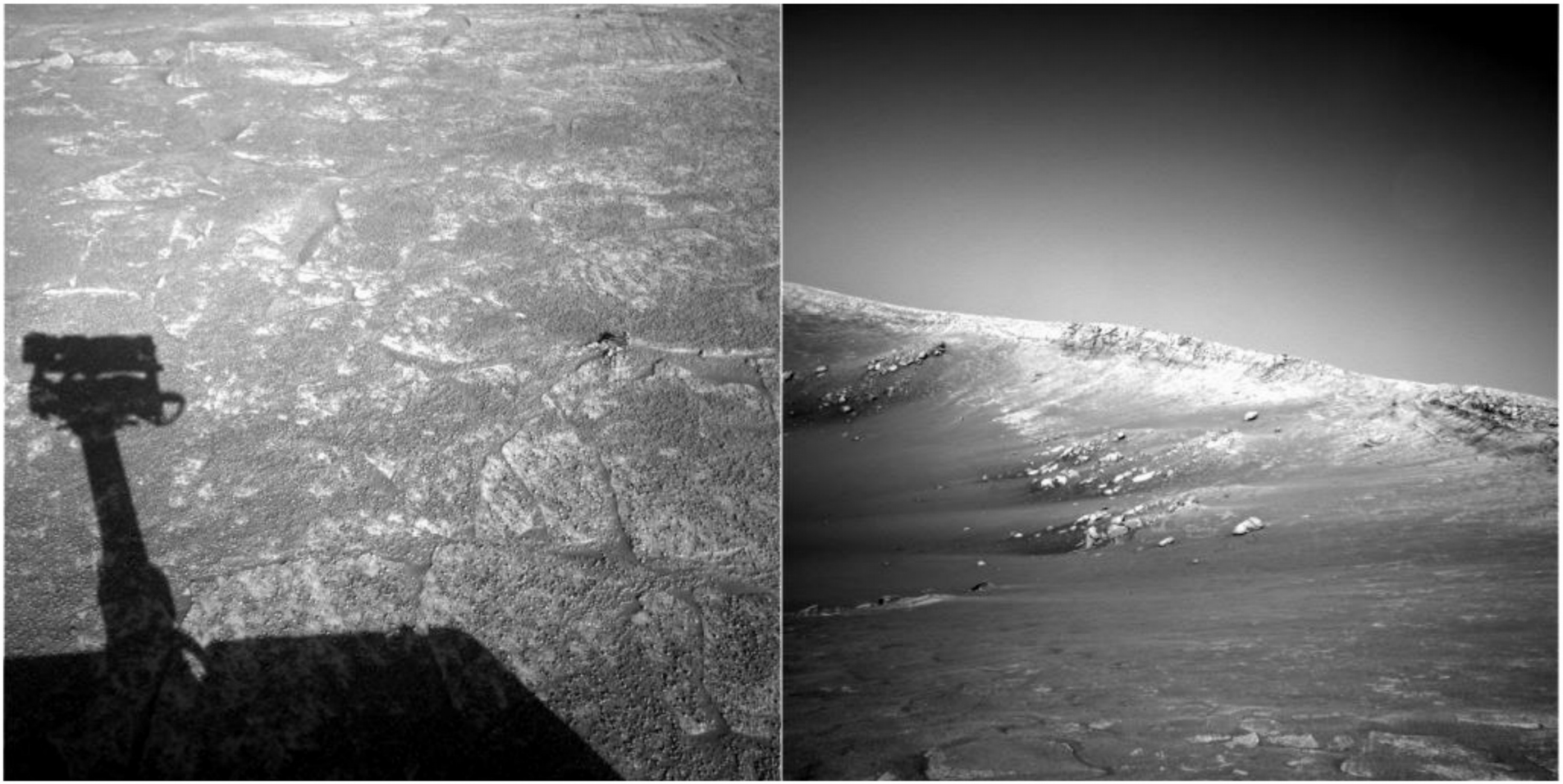
Challenges – Matching with Features

Courtesy: Diva Sian

Courtesy: Matt Northam

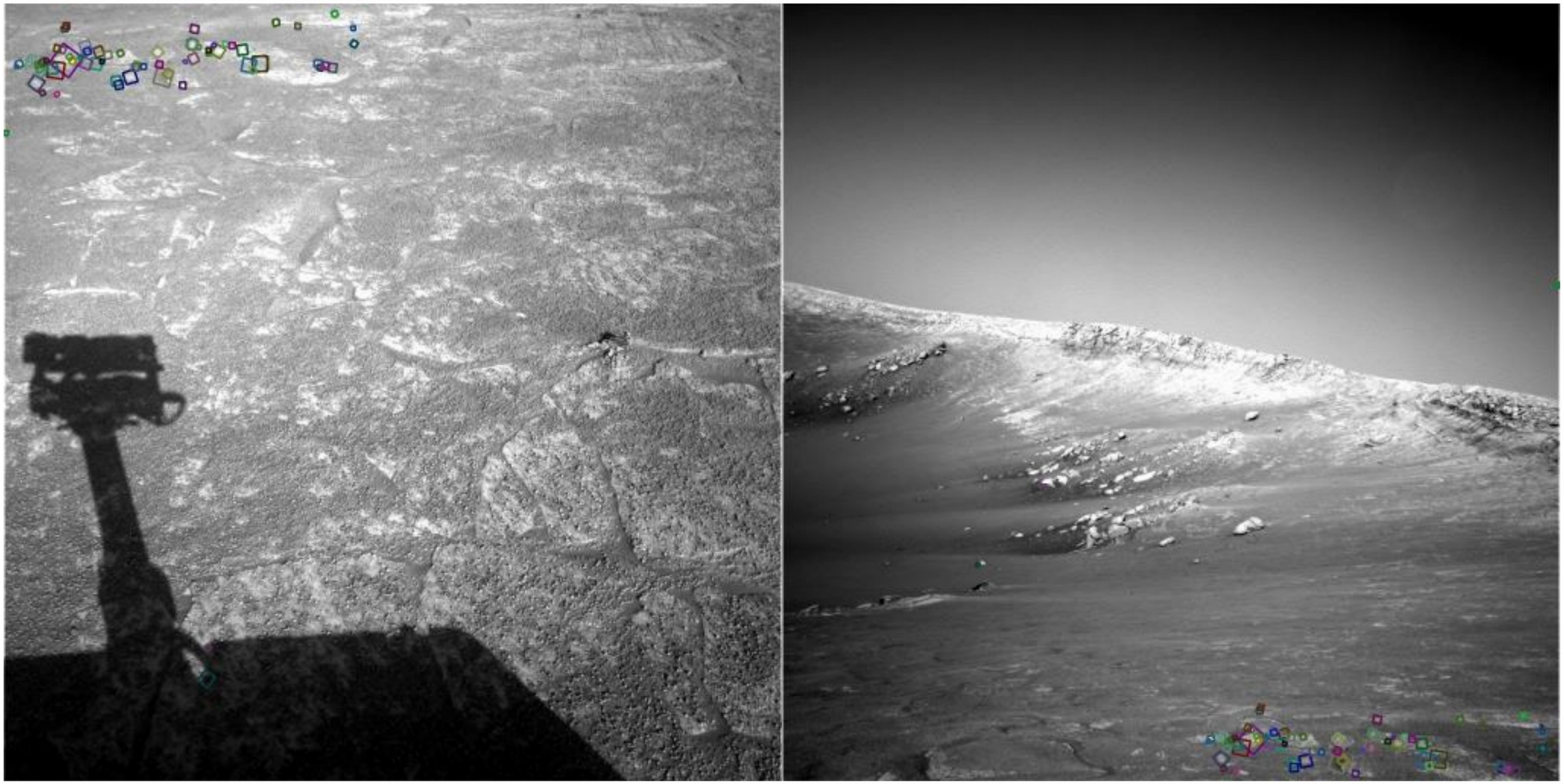
- Illumination changes (small changes are typically modeled with a linear transformations)
- Geometric changes (rotation, scale (i.e. zoom))
- View point (i.e. perspective changes)

Challenges – Feature Detection



Courtesy: NASA (Mars Rover Images)

Challenges – Feature Detection



Courtesy: NASA (Mars Rover Images)
With SIFT features
Figure by Noah Snavely

Feature detection and matching stages

1. Feature detection (extraction)

   Search for well identifiable locations that likely match well in other images

2. Feature description

   Keypoint locations transformed into compact stable/invariant descriptor (vector representation)

3. Feature matching

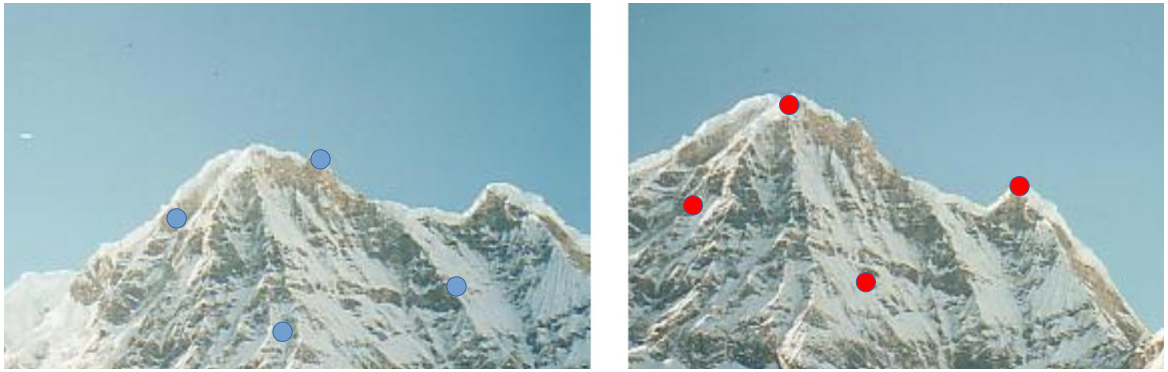
   Search efficiently for the best matching candidates in other images

4. Feature tracking (only for video processing)

   Feature matching that searches only a small neighborhood around detected features in the next image frame

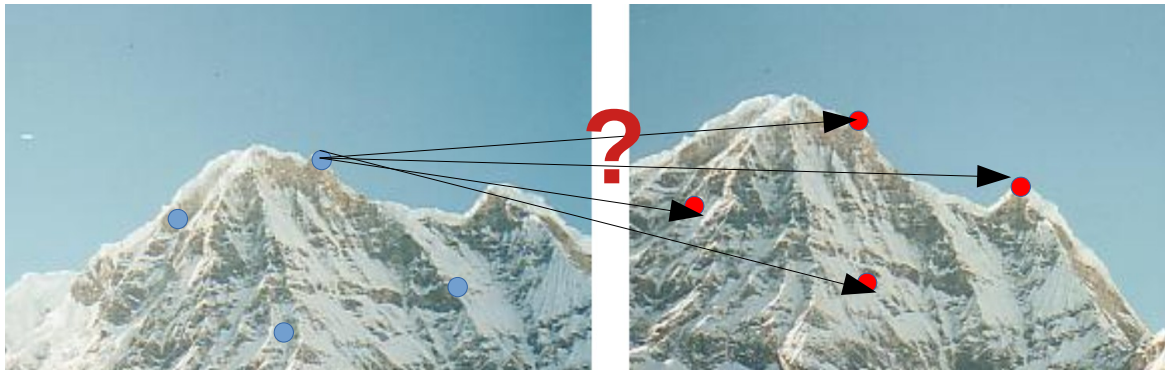
Challenges:

1: Detect the same points independently in both images



No Match possible!!!

2: For each point, identify its correct **correspondence** in the other image(s)



Feature Detector - Properties

What is needed?

- Repeatability

  the detector should be able to re-detect the same feature in different images of the same scene.

- Distinctiveness

  the descriptor uniquely identifies a feature from its surrounding without ambiguity. In general, a descriptor is a “description” of the pixel information around a feature (e.g., patch intensity values, gradient values, etc.).

- Robustness

  the descriptor must also be robust to noise, geometric and illumination changes.

- Compactness and Efficiency

  much less features than image pixels

- Locality

  a feature is assigned to a relatively small patch of the image (robustness to occlusion)

Ideal Feature Detector

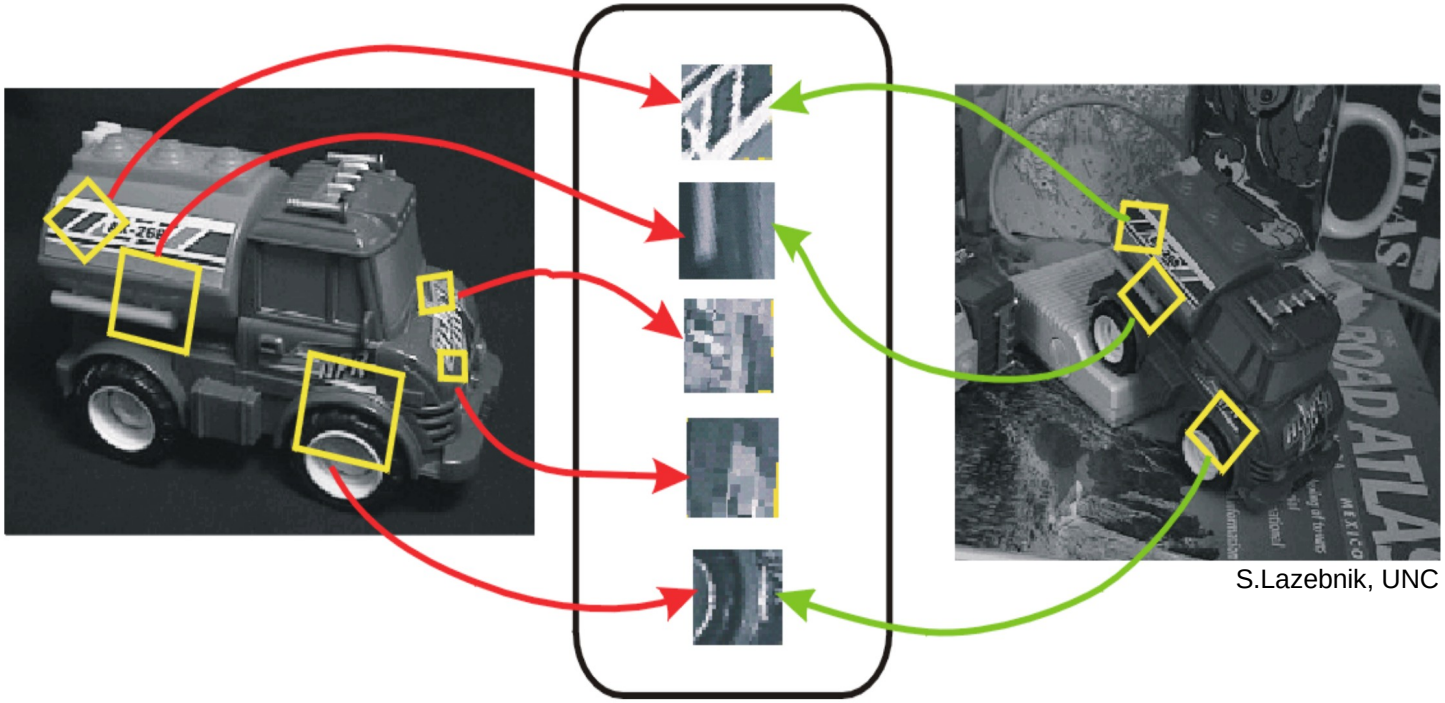
Always finds the same point on an object/scene, regardless of changes to the image (= Subset of local feature types designed to be invariant to common geometric(translation, rotation, scale) and photometric (brightness, exposure, etc.) transformations.)

→ Insensitive (invariant) against changes in scale, lightning, perspective imaging, partial occlusion)

Basic steps:

1) Detect repeatable and distinctive interest points

2) Extract invariant descriptors

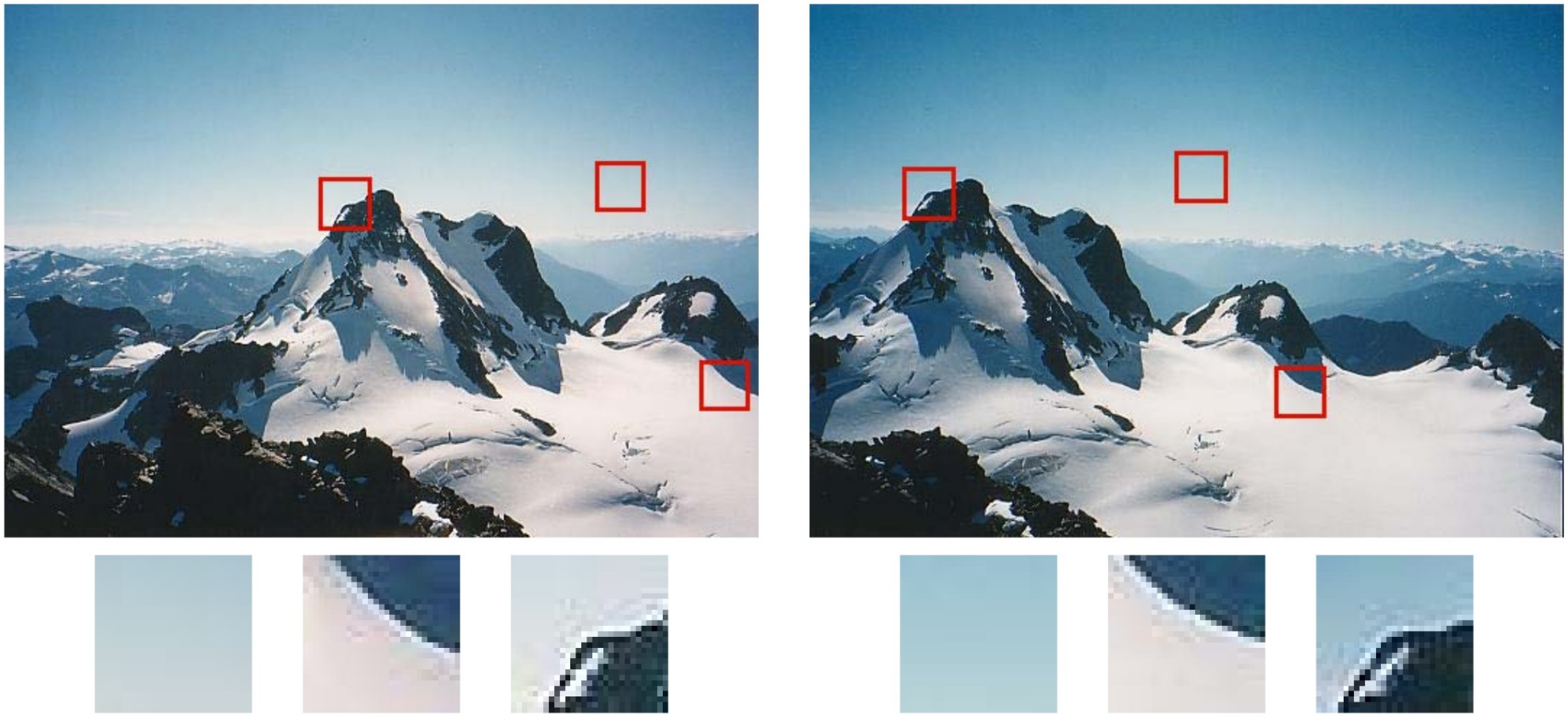


S.Lazebnik, UNC

Feature Detection

- How to find locations that match well with other images, which are repeatable and distinctive?

How to find image locations for reliable correspondences?

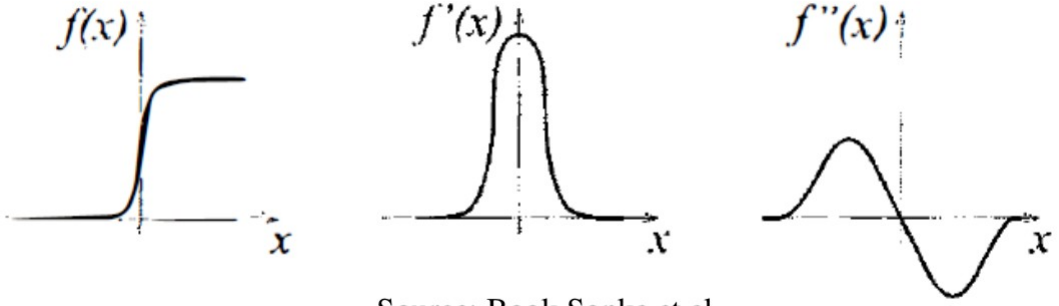


Szeliski Book

Some patches can be localized or matched with a higher accuracy.

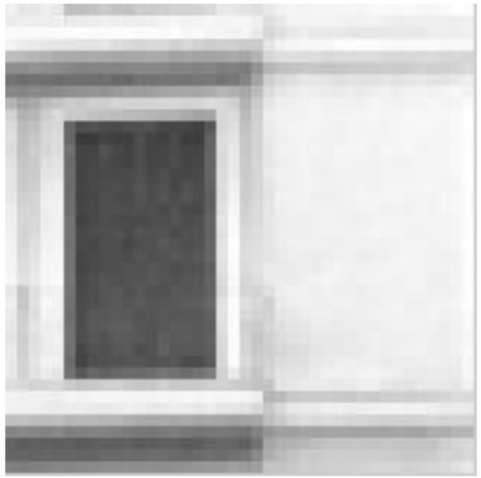
Finding interest points in an image

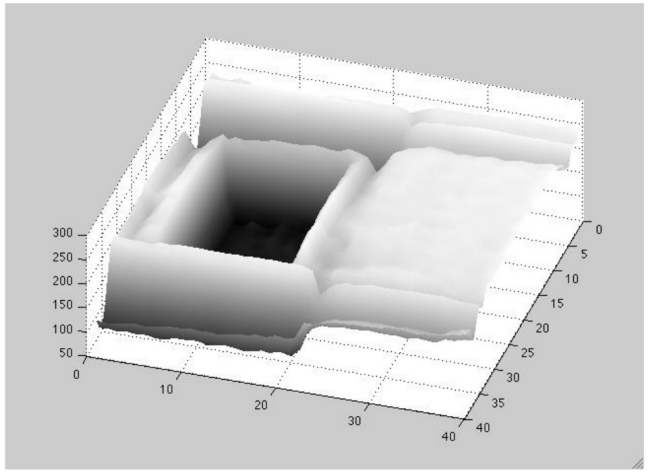
Recap first derivative:



Source: Book Sonka et.al.

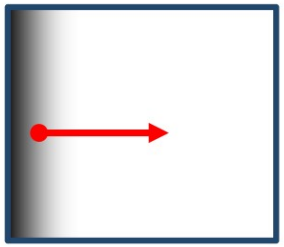
The first derivative is expected to have an extrema at the edge-location





The image gradient $\nabla I = \begin{bmatrix} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \end{bmatrix} = \begin{bmatrix} I_x \\ I_y \end{bmatrix}$ points in the direction of increasing intensity (steepest ascend)

Finding interest points in an image



$$\nabla I = [\frac{\partial I}{\partial x} \quad 0]^\top$$

$$\nabla I = [0 \quad \frac{\partial I}{\partial y}]^\top$$

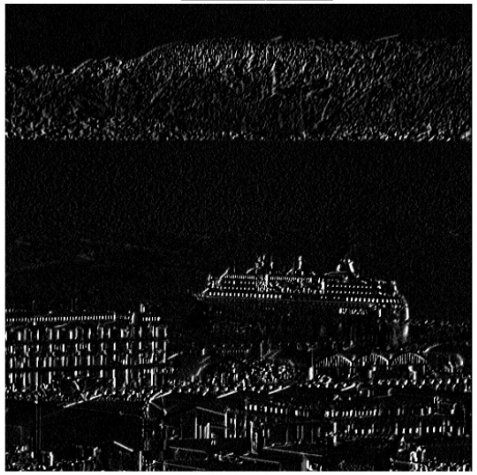
$$\nabla I = [\frac{\partial I}{\partial x} \quad \frac{\partial I}{\partial y}]^\top$$

One possibility is to compute the derivative of an image is to take the discrete derivative → finite difference filter

Backward finite difference: $\frac{\partial I}{\partial x}(x,y) \approx I_{i,j} - I_{i-1,j}$ $\frac{\partial I}{\partial y}(x,y) \approx I_{i,j} - I_{i,j-1}$

=>

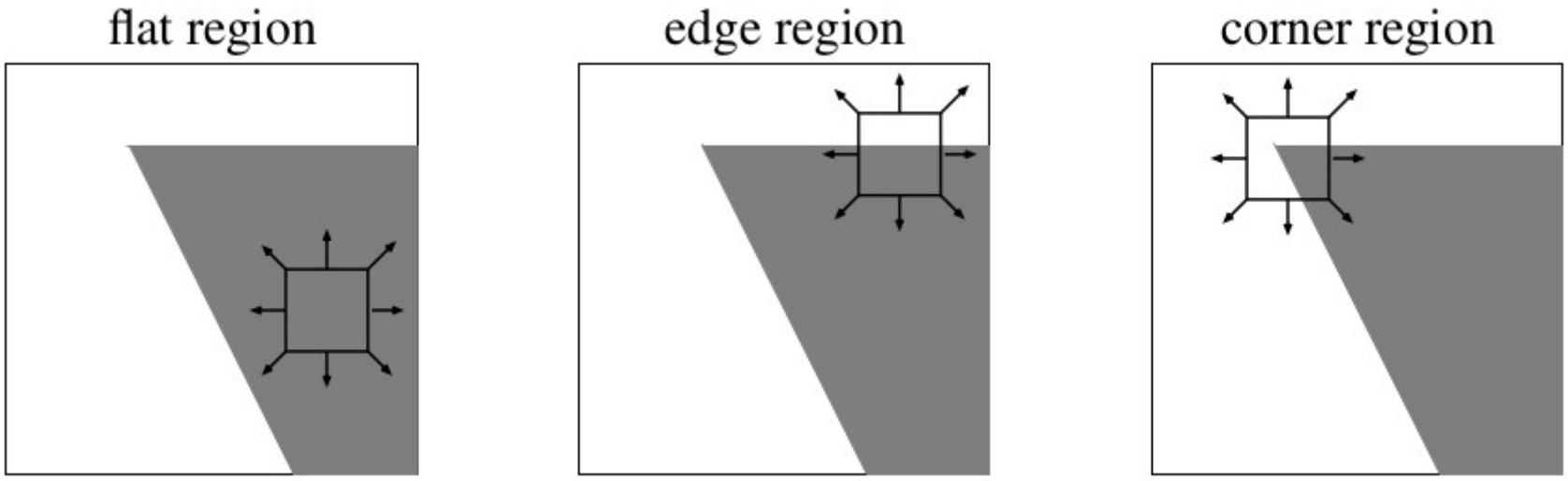
-1 1



-1
1

Finding interest points in an image

Consider the gray values within a small window of an image:



- Ideally in a 'flat' region we expect no change of intensity if we shift the window arbitrary around.

  → textureless regions are impossible to localize

- For edge regions we expect no change if the window is shifted along the edge.

  → patches with gradients are easier to localize → only possible to align the patches along the normal direction (aperture problem)

- For corners we would expect a change of intensity for all directions the window is shifted.

  → patches with at least two significantly different orientations are easiest to localize

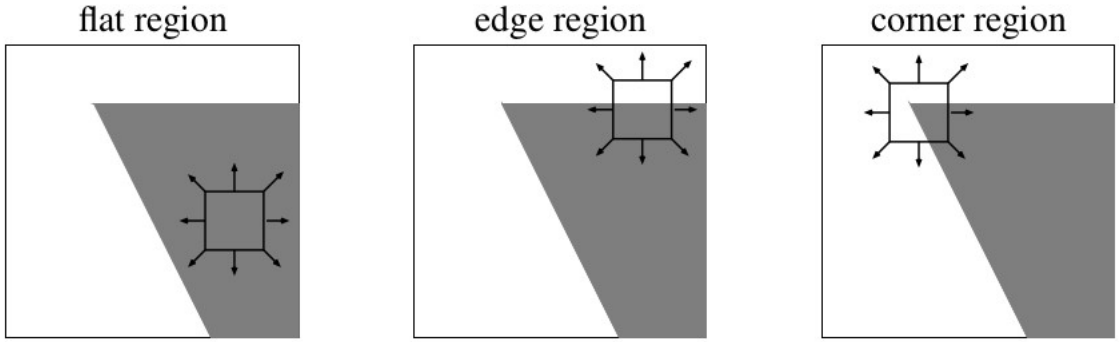
→ this idea was exploited by C.Harris and M.Stephens: "A Combined Corner and Edge Detector„ (1988)

Finding interest points in an image

Find locations such that shifting the window in any direction results in a large intensity change (difference between neighbor pixels at least in 2 directions is large → **Sum of Squared Differences (SSD)** is large).

- Consider shifting the window W by the displacement vector $[u\ v]^\top = [\Delta x, \Delta y]^\top$
- How do the pixels in W change?
- Compare each pixel before and after using the SSD
- This defines a SSD energy E(u,v):

$$E_{SSD}(u,v) = \sum_{u,v \in W} (I(x+u, y+v) - I(x,y))^2$$



H. Moravec, Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover, PhD thesis, Chapter 5, 39, Stanford University, Computer Science Department, 1980.

Finding interest points - Motion Assumption

Taylor Series expansion of I:

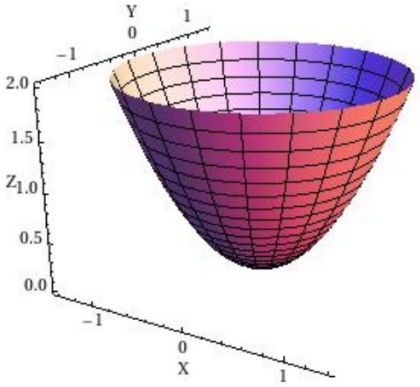
$$I(x+u, y+v) = I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \text{ higher order terms}$$

If we assume a small change in the displacement (u,v), then the fist order approximation is accurate enough

$$\begin{aligned} I(x+u, y+v) &\approx I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v \\ &= I(x,y) + I_x u + I_y v \end{aligned}$$

We substitute and obtain the approximation for the SSD Energy

$$\begin{aligned} E_{SSD}(u,v) &= \sum_{u,v \in W} (I(x+u, y+v) - I(x,y))^2 \\ &\approx \sum_{u,v \in W} (I(x,y) + I_x u + I_y v - I(x,y))^2 \\ &= \sum_{u,v \in W} (I_x u + I_y v)^2 \end{aligned}$$



Finding interest points – Structure Tensor

This can be written in matrix form as

$$
\begin{aligned}
E_{SSD}(u,v) &\approx \sum_{u,v\in W} (I_x u + I_y v)^2 \\
&= \sum_{u,v\in W} [u\ v] \begin{bmatrix} I_x \\ I_y \end{bmatrix} [I_x\ I_y] \begin{bmatrix} u \\ v \end{bmatrix} \\
&= \sum_{u,v\in W} [u\ v] \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \\
&= \sum_{u,v\in W} [u\ v] A \begin{bmatrix} u \\ v \end{bmatrix}
\end{aligned}
$$

The Matrix A can be written in a form of a structure tensor

$$
A = \sum_{u,v\in W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}
$$

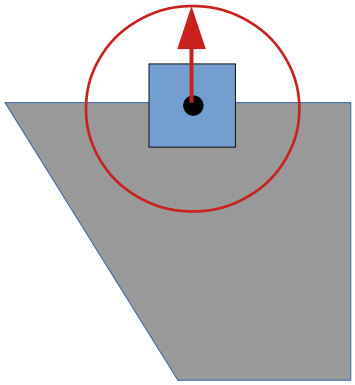
Finding interest points – Structure Tensor

This can be rewritten as:

$$E_{SSD}(u,v) \approx [u\ v] \left(\sum_{u,v \in W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix} = [u\ v] \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

A A

Example:



- You can move the center of the blue window to anywhere on the red unit circle
- How do we find directions that will result in the larges and smallest energy values?
- We can find these directions by utilizing the eigenvectors of A

Recal: Eigenvalue / Eigenvector

The eigenvectors of a matrix A are the vectors that satisfy:

$$Ax = \lambda x$$

The scalar λ is the eigenvalue corresponding to x

The eigenvalues are found by solving

$$\det(A - \lambda I) = 0$$

In our case, A is a 2x2 matrix:

$$\det \begin{bmatrix} a_{11} - \lambda & a_{12} \\ a_{21} & a_{22} - \lambda \end{bmatrix} = 0$$

The solution

$$\lambda_{1,2} = \frac{1}{2}[(a_{11} + a_{22} \pm \sqrt{4a_{12}a_{21} + (a_{11} - a_{22})^2}]$$

Once you know λ you find the two eigenvectors x by solving

$$\begin{bmatrix} a_{11} - \lambda & a_{12} \\ a_{21} & a_{22} - \lambda \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 0$$

Finding interest points – Structure Tensor

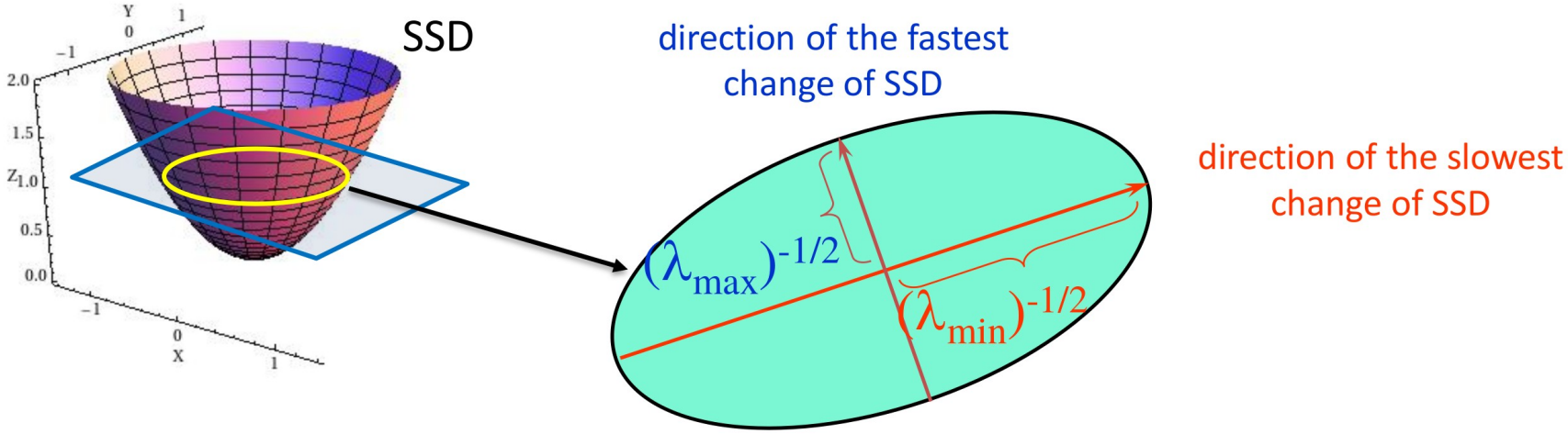
Since A is a symmetric matrix, it can always be decomposed into

$$A = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$$

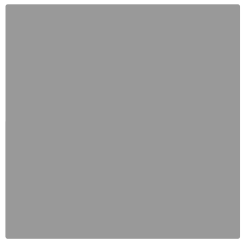
We can visualize $[u\ v]A \begin{bmatrix} u \\ v \end{bmatrix} = const$ as an ellipse with axis lengths determined

by the eigenvalues and the two axes' orientations determined by R (i.e., the eigenvectors of A)

→ The two eigenvectors identify the directions of largest and smallest changes of SSD



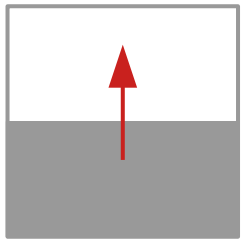
Finding interest points – Structure Tensor



Flat region

$$A = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

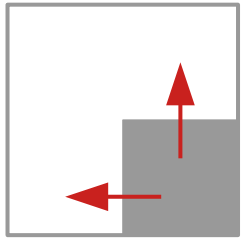
Flat region:
→ eigenvalues are close to 0!



Edge

$$A = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

Edge:
→ least one of the eigenvalues is close to 0!



Corner

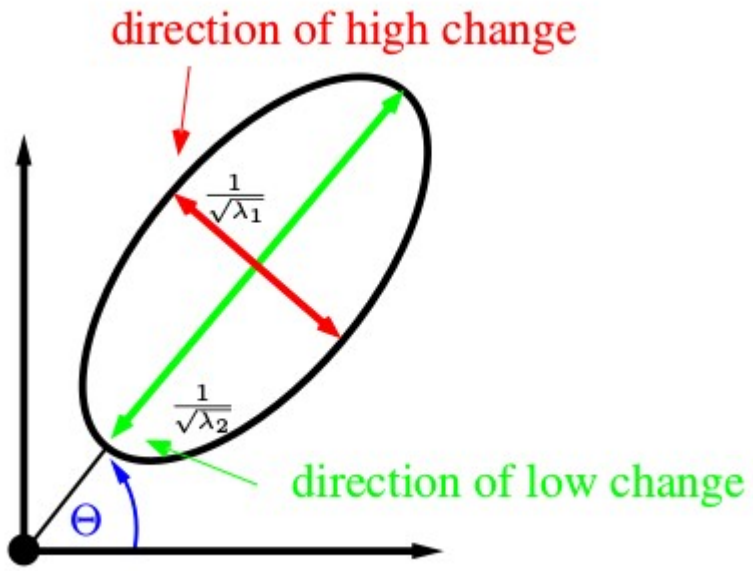
$$A = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} = \begin{bmatrix} \cos\frac{\pi}{4} & -\sin\frac{\pi}{4} \\ \sin\frac{\pi}{4} & \cos\frac{\pi}{4} \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} \cos\frac{\pi}{4} & \sin\frac{\pi}{4} \\ -\sin\frac{\pi}{4} & \cos\frac{\pi}{4} \end{bmatrix}$$

Corner:
→ both eigenvalues >> 0

Note: The dominant gradient directions in this axis-aligned corner example are at 45 degrees with x and y axes

Visualization of the Structure Tensor

The 2x2 structure tensor A is also known as **second moment matrix** involving the gradient of the image function I. Its eigenvalues and eigenvectors can be represented as a oriented ellipse.



The structure tensor allows to identify the dominant directions of the image gradient.

Note: Here the longer side corresponds to the smaller change $\left(\frac{1}{\sqrt{\lambda_2}}, \lambda_1 \geq \lambda_2 \geq 0\right)$.

Visualization of Structure Tensor

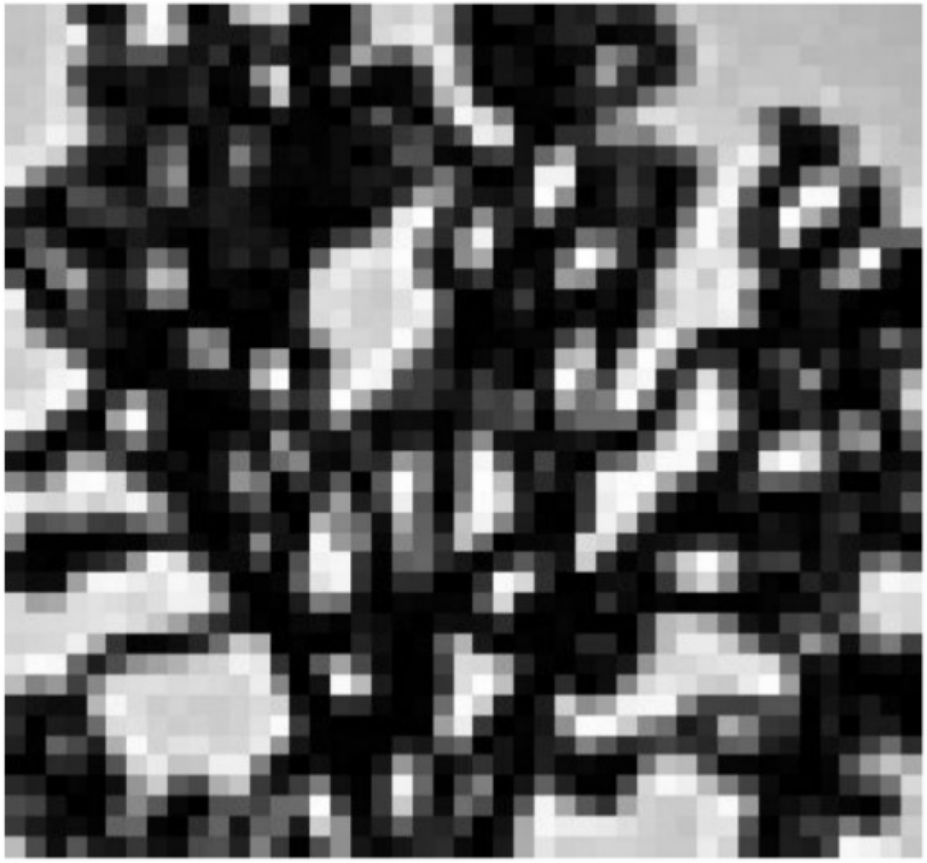


Image Source: Brown University

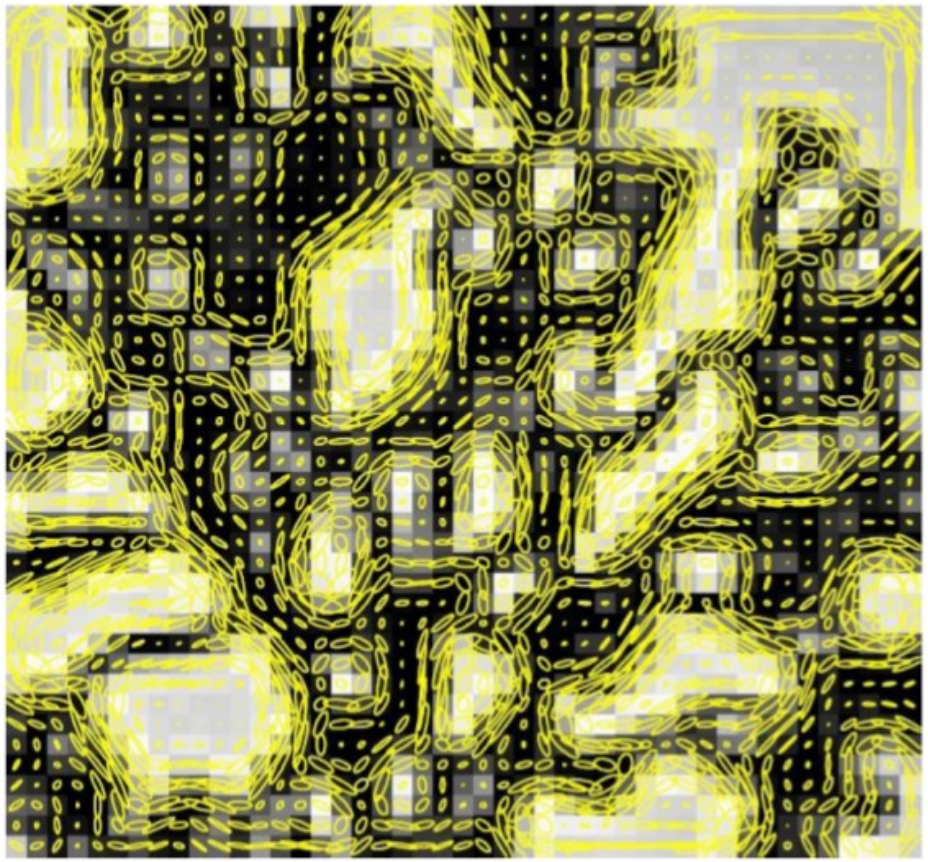


Image Source: Brown University

Note: The ellipses are plotted proportionally to the eigenvalues. Small ellipses denote a flat region, and big ones, a corner.

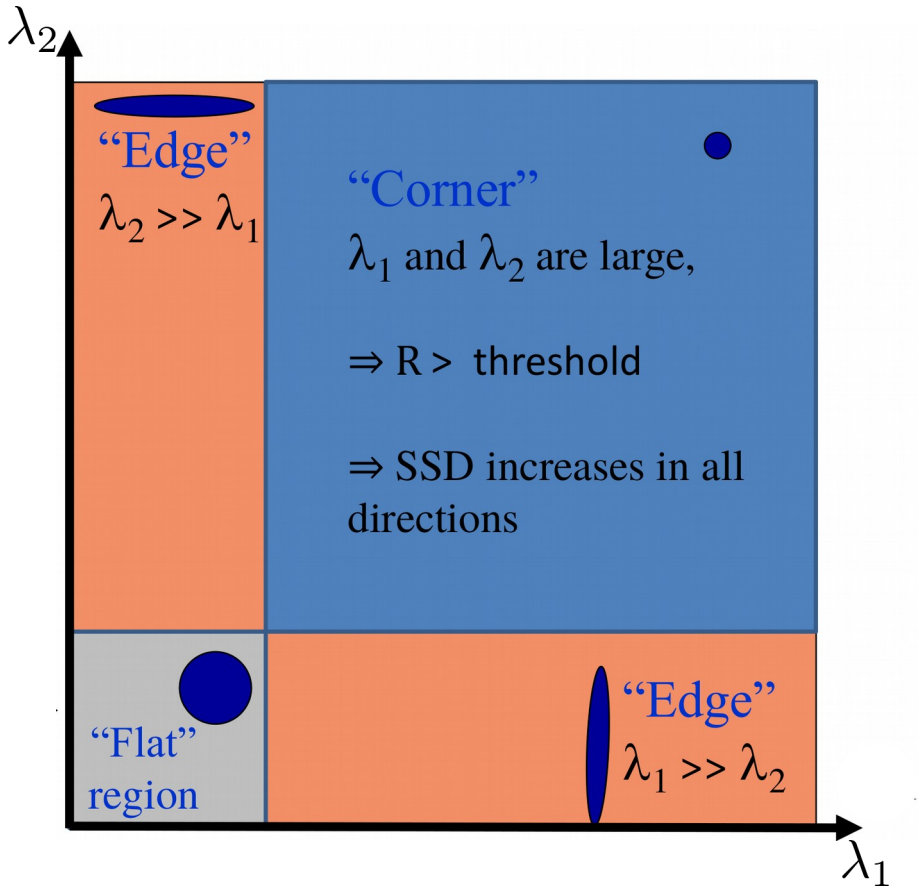
Interpretation of the Structure Tensor

Classification of image points using eigenvalues of A

- Corner: minimum of the two eigenvalues of A is larger than a certain user-defined threshold, here $R = \min(\lambda_1, \lambda_2)$

- R is called the **corner response function**
- The corner detector using this criterion is called **«Shi-Tomasi» detector**

J. Shi and C. Tomasi (June 1994). "Good Features to Track,". 9th IEEE Conference on Computer Vision and Pattern Recognition



Structure Tensor – Harris

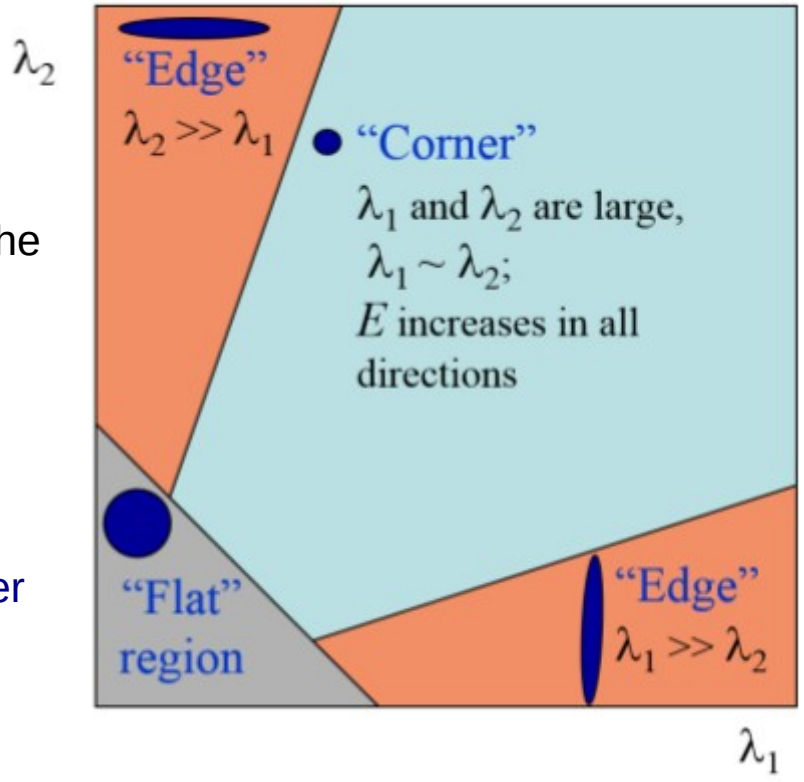
Harris and Stephens suggested the following corner response function for the corner/edge evaluation:

$$R = \lambda_1\lambda_2 - \kappa(\lambda_1 + \lambda_2)^2 = det(A) - \kappa\ \mathrm{trace}^2(A)$$

This avoids the direct computation of the eigenvalues of A

The parameter κ was found empirically and

good values are in the range $\kappa \in [0.04, 0.15]$.



The eigenvalues of A (positive semidefinite) correspond to the following cases:

$\lambda_1 \approx 0$ and $\lambda_2 \approx 0$: pixel belongs to a flat region

$\lambda_1 \approx 0$ and $\lambda_2 > 0$ (large positive value): edge region

Both $\lambda_1 > 0$ and $\lambda_2 > 0$ have large positive values: corner

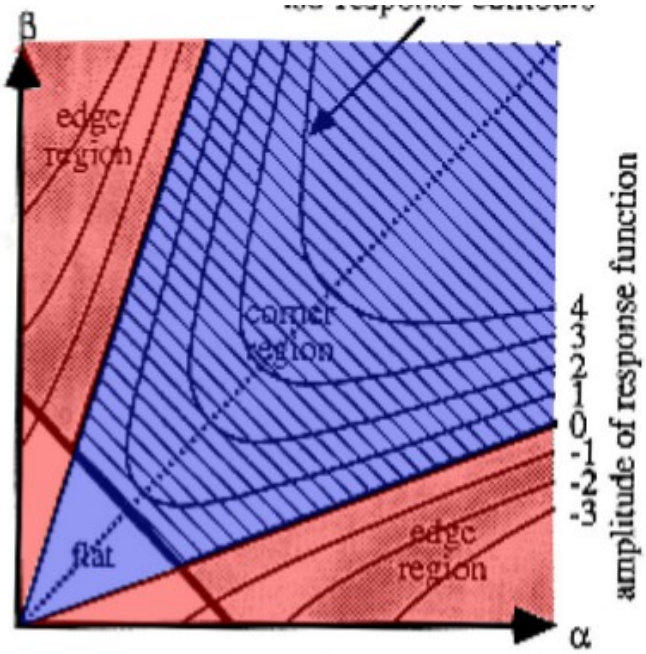
C.Harris and M.Stephens. "A Combined Corner and Edge Detector."
Proceedings of the 4th Alvey Vision Conference: pages 147-151, 1988.

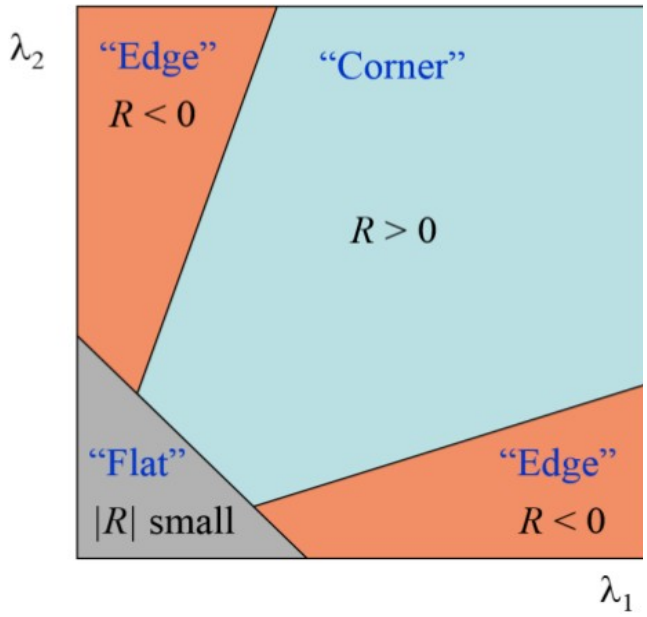
Harris Corner Detector

The corner response function R is

- positive in the corner region
- negative in the edge regions
- and small in the flat region

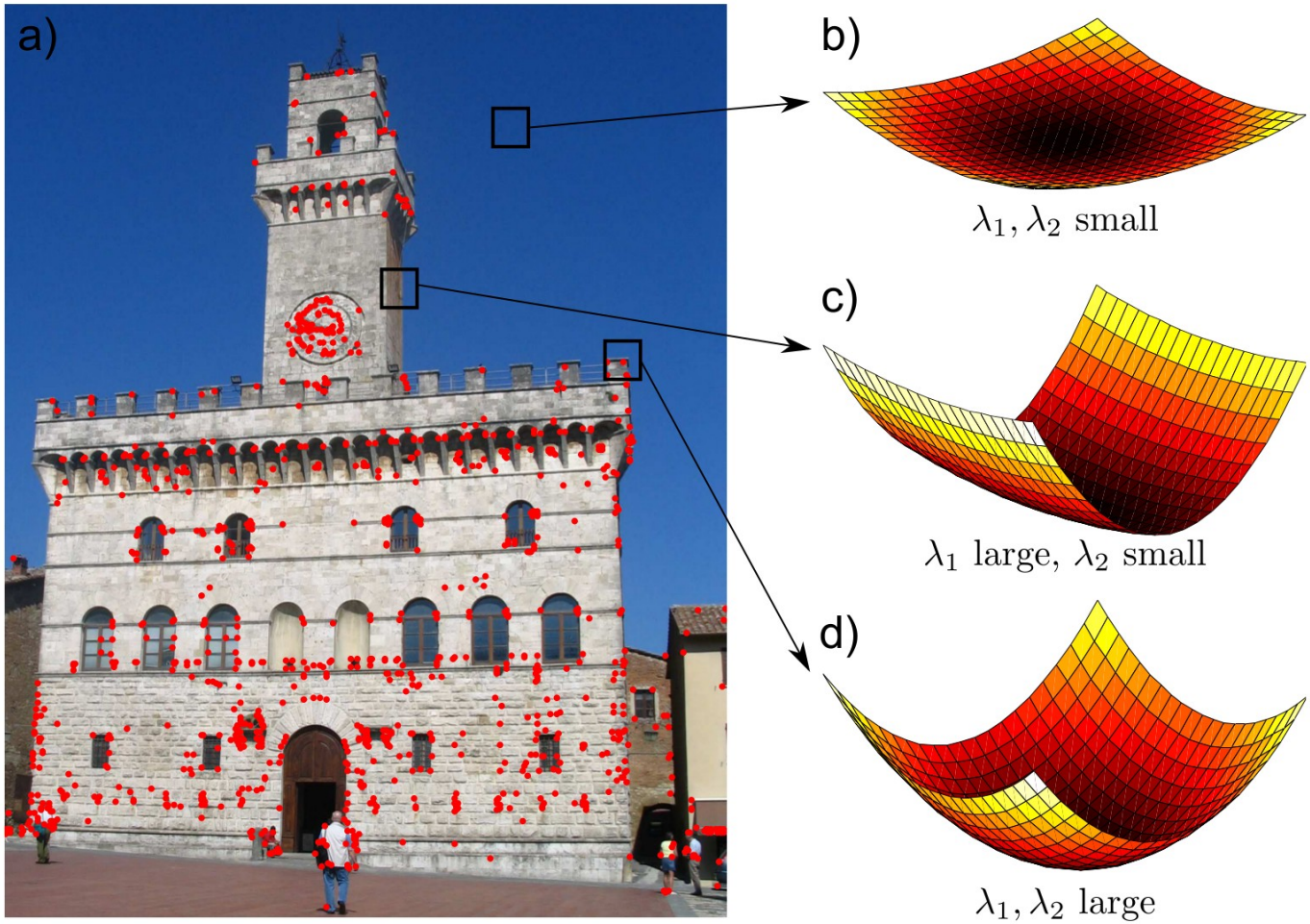
The flat region can be specified by trace (A) falling below a threshold.





Harris Corner Detector

Example

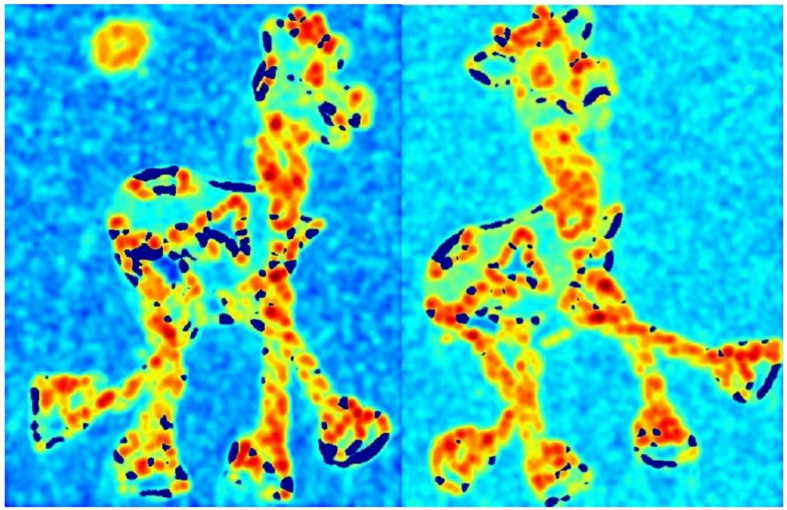


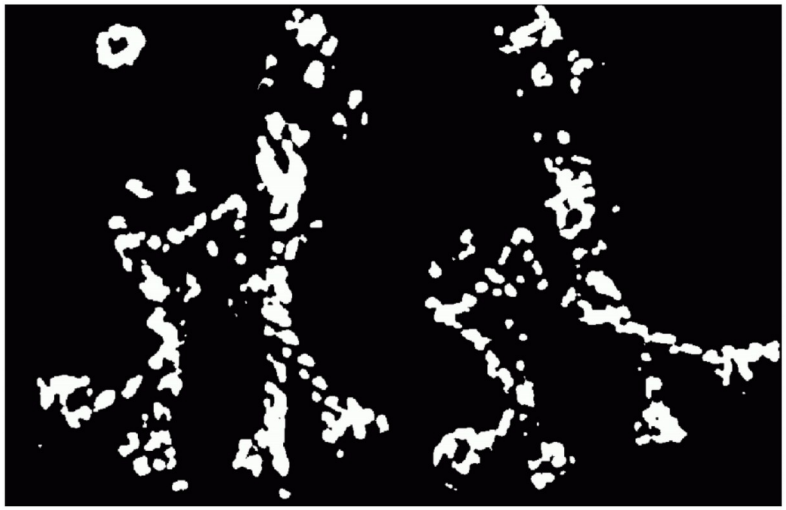
Computer vision: models, learning and inference. ©2011 Simon J.D. Prince

Harris Corner Detector

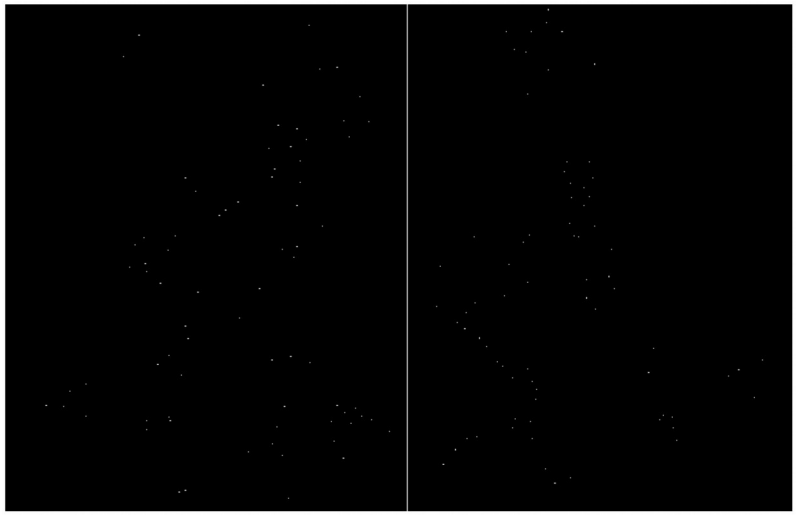
How are the feature points determined?

- Compute R and perform Thresholding (pixels with high corner response function (R > threshold)



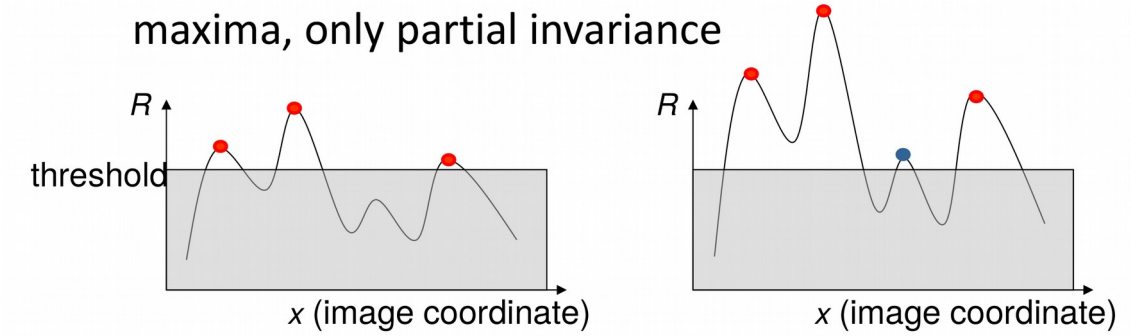


- Non-maxima suppression: Select a local maxima of R in a certain neighborhood as feature point

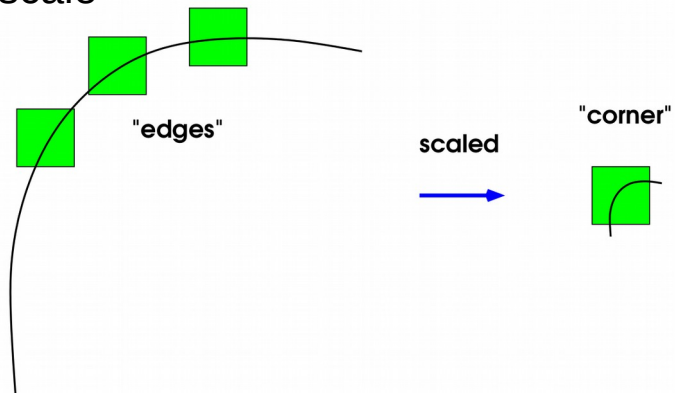


Harris Corner Detector – Invariance Prop.

- **Translation-invariant** (derivatives are shift-invariant)
- Corner response function is **invariant to image rotation** (ellipse rotates but its shape remains the same
- **Invariant to intensity shift** $I \rightarrow I + b$ (derivatives do not change);
- Partial invariant to intensity scale $I \rightarrow aI$: because of fixed intensity threshold on local maxima



- **NOT** invariant to image scale

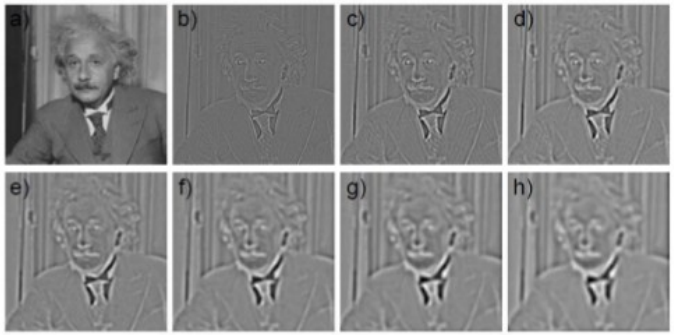


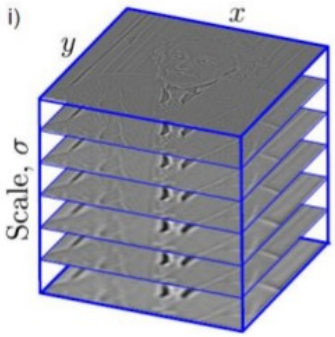
Note: Harries corner detector was one of the first feature detectors.
Ideas used occur also in other more advanced detectors.

Scale Invariance

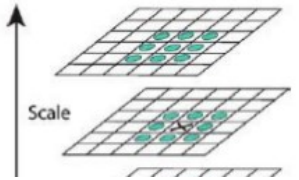
Idea:

- Generate copies of image at multiple scales by convolving with Gaussians of different σ and using a pyramid



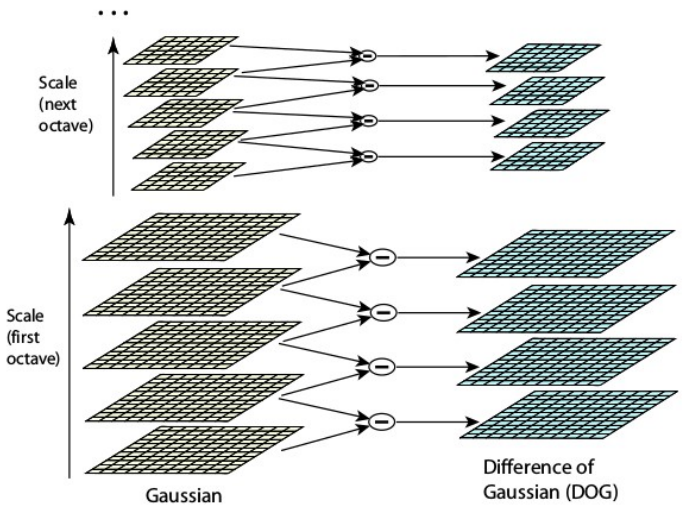


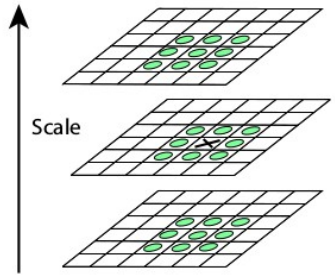
- The image is filtered with difference of Gaussian kernels at a range of increasing scales. Candidate points that are the local extrema in the stacked image volume



- Find local maxima points by comparing to neighboring points at current and adjacent scales

- Each interest point is a local maximum in both position and scale





Next steps:

Recall: Steps for Feature Detection and Matching

- 1. Feature detection (extraction)

  Search for locations that match well in other images

- 2. Feature description

  Keypoint locations transformed into compact stable/invariant descriptor

- 3. Feature matching

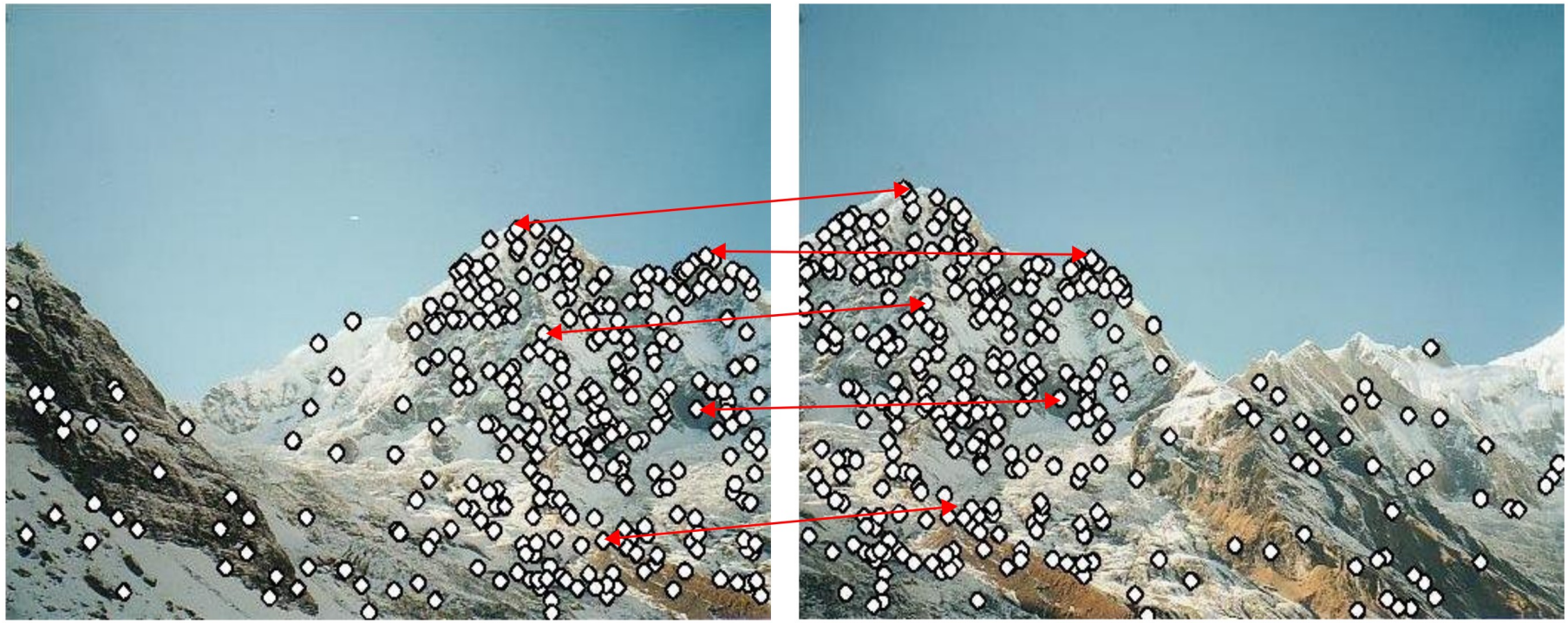
  searches efficiently for likely matching candidates in other images

Related Questions:

- How to find locations that match well with other images, which are repeatable and distinctive?
- How to describe a feature / design a feature descriptor?
- How to establish correspondences, i.e., compute matches?

Feature Descriptors

- We know how to detect good interest points
- Next question: How to match image regions around interest points?
- Answer: Need feature descriptors



Feature Descriptor

Two steps:

1) Make sure your feature detector is invariant

- Harris is invariant to translation and rotation
- Scale invariance is more difficult to achieve
  - common approach is to detect features at many scales using a Gaussian pyramid (e.g., MOPS)
  - More sophisticated methods find “the best scale” to represent each feature (e.g., SIFT)

2) Design an invariant feature descriptor

- A descriptor captures the intensity information in a region around the detected feature point
- The simplest descriptor: a square window of pixels

Multiscale Oriented PatchS descriptor - MOPS

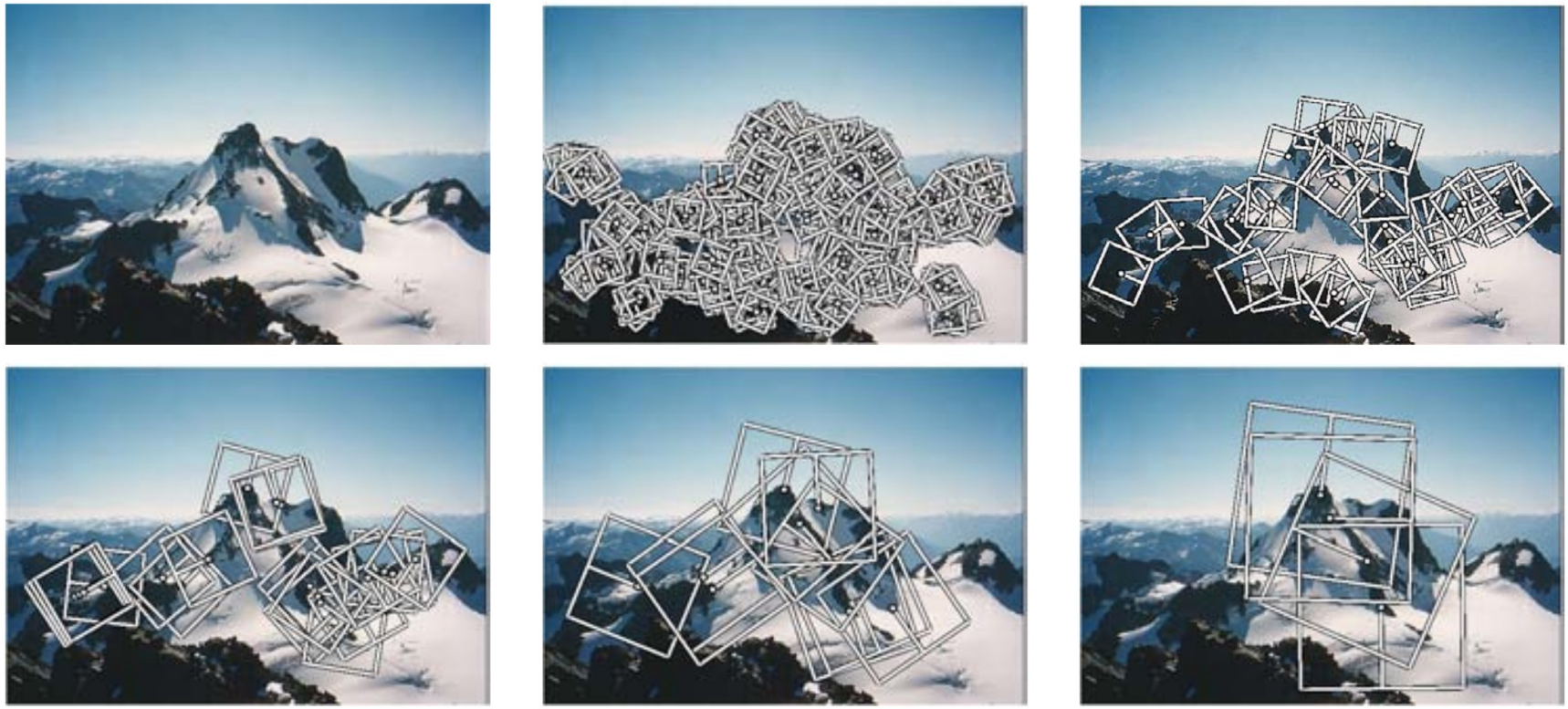


Figure 4.10 Multi-scale oriented patches (MOPS) extracted at five pyramid levels (Brown, Szeliski, and Winder 2005) © 2005 IEEE. The boxes show the feature orientation and the region from which the descriptor vectors are sampled.

Scale Invariant Feature Transform (SIFT)

Lit.: David G. Lowe, “Distinctive image features from scale-invariant keypoints,“
International Journal of Computer Vision, 60, 2 (2004), pp. 91-110

SIFT - Scale Invariant Feature Transform

SIFT image features provide a set of robust features that are independent of the scaling, rotation and translation of the image.

The SIFT algorithm consists of 4 stages

1. Scale-Space Extrema Detection (determine candidate locations)

2. Key-point Localization (determine scale and location)

3. Orientation Assignment (determine main orientation)

4. Key-point Descriptor (create high dimensional description vector)

→ results in a large collection of local feature vectors

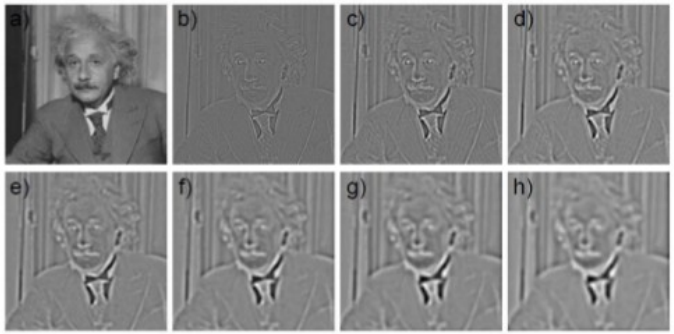
Extrema in the Scale-Space

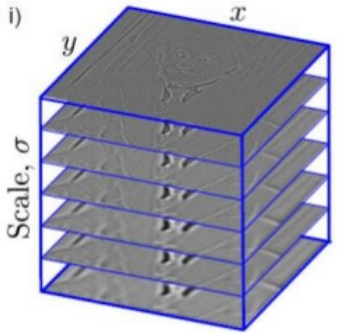
The scale-space representation of an image $I(x, y)$ is given by the image function $L(x, y, \sigma)$ that is defined as the convolution of the Gaussian $G(x, y, \sigma)$ with the image:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

with $G(x, y, \sigma) = \frac{1}{1\pi\sigma} e^{-(x^2+y^2)/2\sigma}$

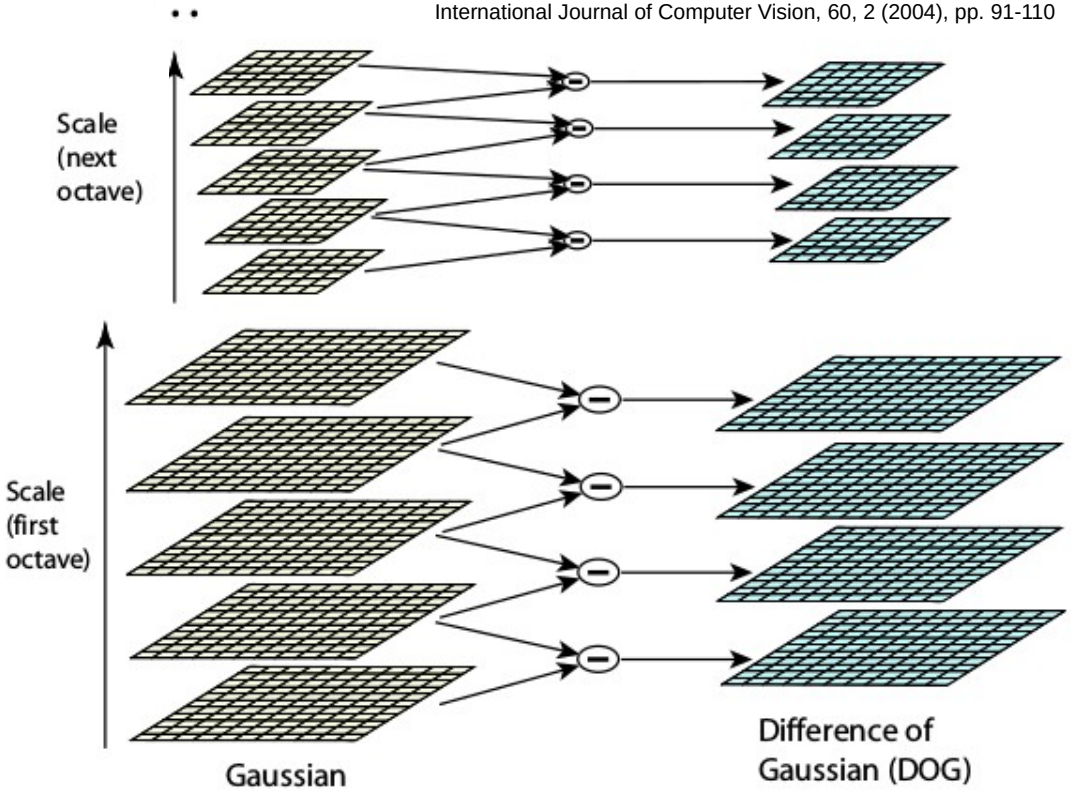
Extrema in the Scale-Space





- The image is filtered with difference of Gaussian kernels at a range of increasing scales. Candidate points that are the local extrema in the stacked image volume

David G. Lowe, "Distinctive image features from scale-invariant keypoints," International Journal of Computer Vision, 60, 2 (2004), pp. 91-110



Difference of Gaussian (DoG):

difference of two nearby scales separated by a constant multiplicative factor k:

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma)$$

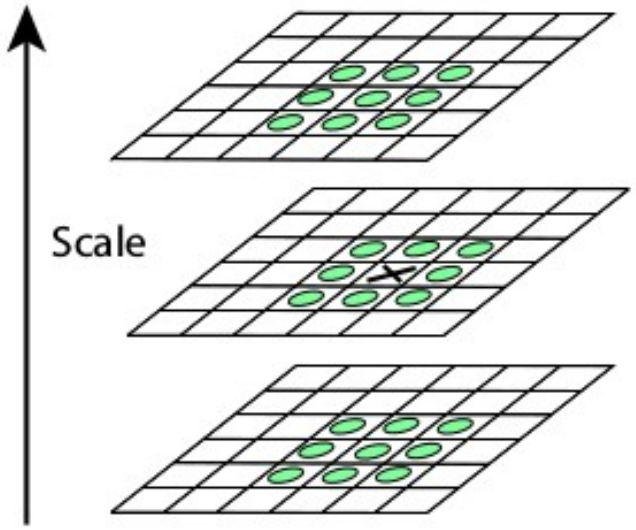
→ approximation of the Laplacian

Candidate Key-point Detection

Find key-point candidates as local extrema:

- detect and localize the (interpolated) maxima and minima of the Difference-of-Gaussian (DoG) in the scale space levels
- scale: the scale of a key-point refers to the level it is found

→ many candidates will be detected



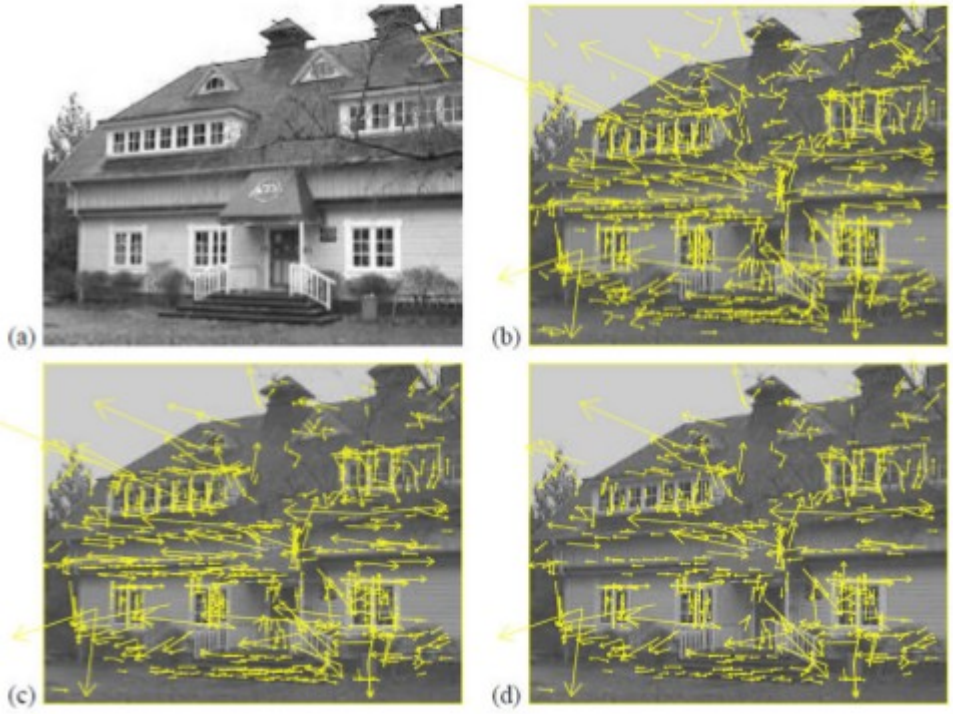
David G. Lowe, “Distinctive image features from scale-invariant keypoints,“ International Journal of Computer Vision, 60, 2 (2004), pp. 91-110

Reject Certain Key-points

Heuristics to reject key-point candidates:

- Reject low contrast candidate points. (threshold the peak value of DoG)
- Reject candidates which are along edges (curvature measure based on the Hessian matrix of DoG [=second-order partial derivatives]).

Example of key-point rejection:



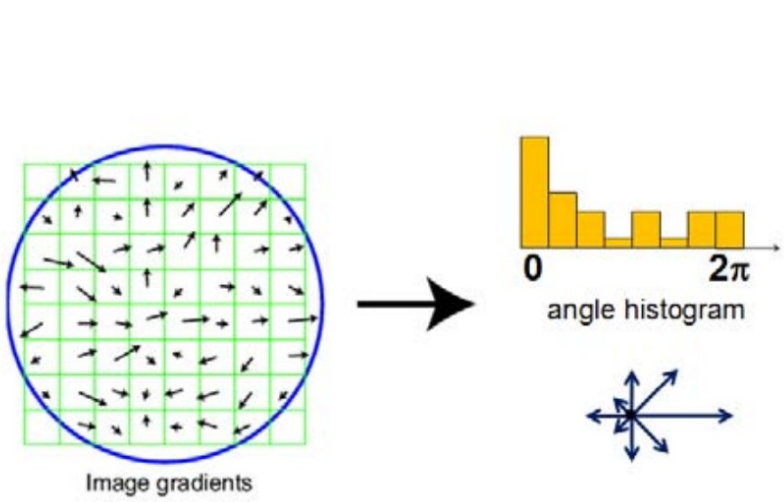
- 832 key-points locations at maxima and minima
- 729 key-points remain after applying a threshold on minimum contrast
- 536 key-points remain after thresholding low curvature points

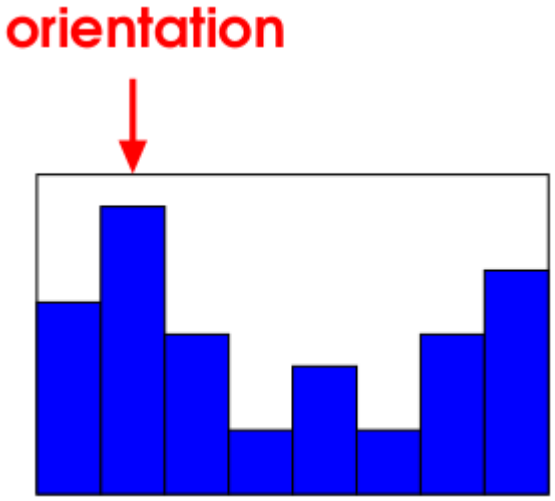
David G. Lowe, “Distinctive image features from scale-invariant keypoints,“ International Journal of Computer Vision, 60, 2 (2004), pp. 91-110

Orientation Assignment

- create a histogram of local gradient directions that are computed at the determined scale.
- determine peak of the (smoothed) histogram and assign it as canonical orientation. (create new key-point when multiple peaks of similar magnitude)
- store for each key-point its “robust“ coordinates: (x, y, scale, orientation)

→ features are compared relative to the canonical orientation (this makes the approach rotation invariant)



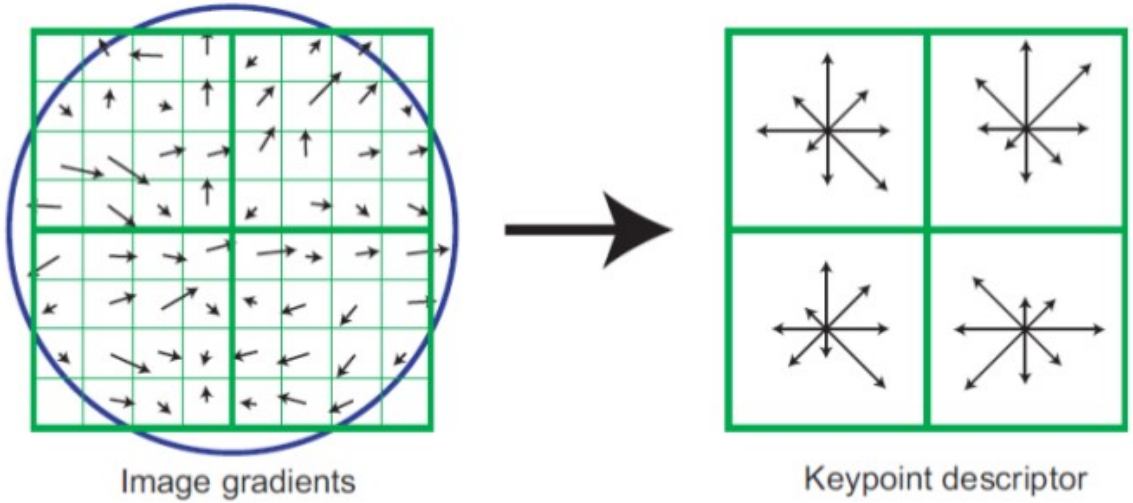


David G. Lowe, “Distinctive image features from scale-invariant keypoints,“ International Journal of Computer Vision, 60, 2 (2004), pp. 91-110

Key-point Descriptor Creation

- compute local gradient directions in a neighborhood (usually 16x16)
- around the key-point compute orientation histograms (with 8 bins) of 4x4 subregions
- rotate and store the orientations relative to the key-point orientation

The image below shows a 2x2 descriptor array computed from a 8x8 neighborhood (2x2x8=32 dimensions):



→ The typical SIFT descriptor has 4x4x8=128 dimensions !

David G. Lowe, "Distinctive image features from scale-invariant keypoints," International Journal of Computer Vision, 60, 2 (2004), pp. 91-110

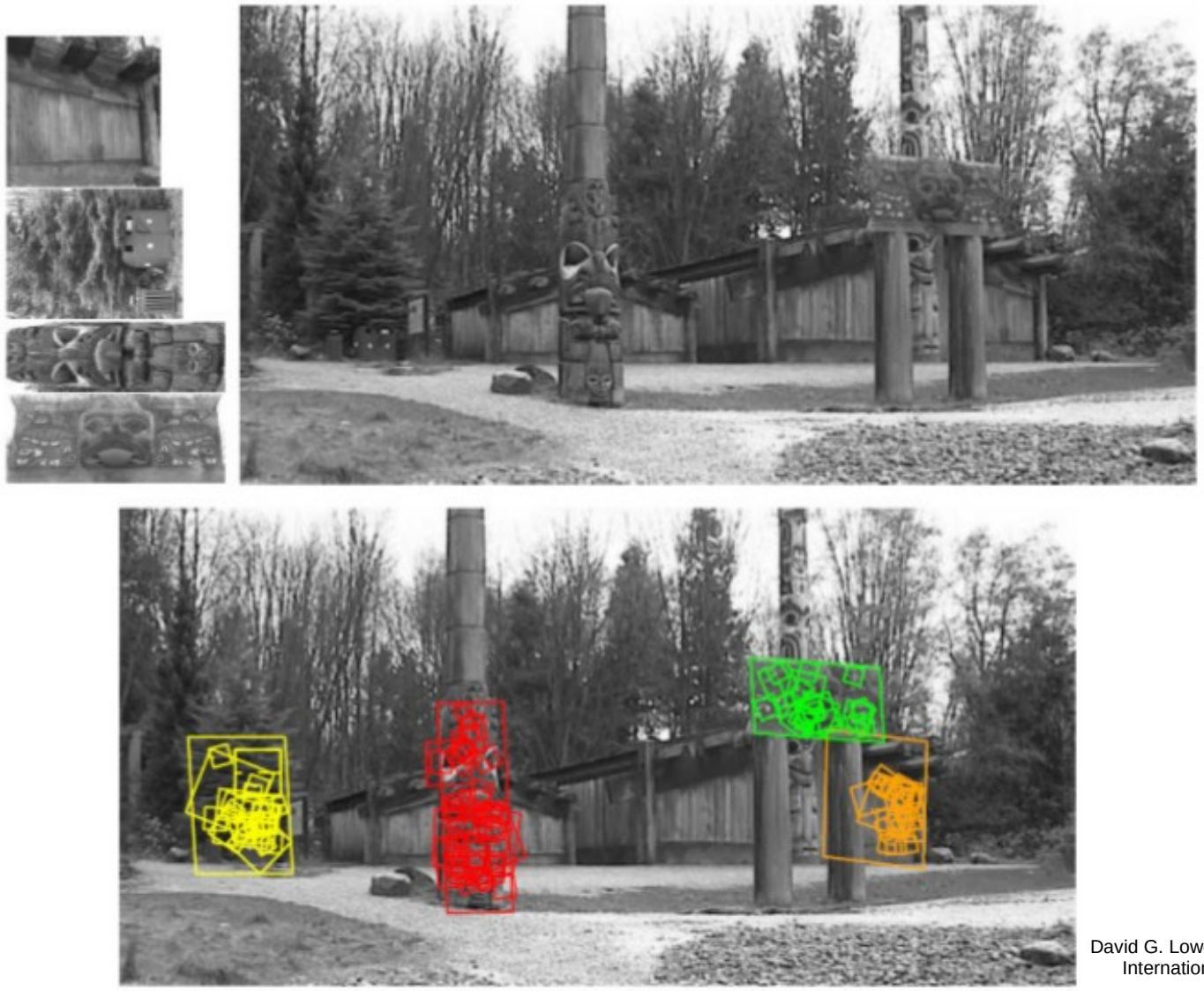
SIFT Example

Shown: Features from the small image that fit within the larger scene image. The larger rectangles correspond to the boundary of the small images. The size of the inner rectangles indicates the scale on which the feature fits.

→ Good match even with large occlusions.

David G. Lowe, “Distinctive image features from scale-invariant keypoints,“ International Journal of Computer Vision, 60, 2 (2004), pp. 91-110

SIFT Example



David G. Lowe, “Distinctive image features from scale-invariant keypoints,“ International Journal of Computer Vision, 60, 2 (2004), pp. 91-110

More Feature Detectors

Focus within the last years:

Perform as good as SIFT but with lower computational complexity:

- Speeded up Robust Feature (SURF) H. Bay, T. Tuytelaars, L. Van Gool, "Surf: Speeded up robust features", Computer vision-ECCV 2006. Springer, pp. 404-417, 2006.
- The Binary Robust Independent Elementary Feature (BRIEF)
- Oriented Fast and Rotated BRIEF (ORB)
- Binary Robust Invariant Scalable Keypoints (BRISK)

- ... and more ...

Next steps:

Recall: Steps for Feature Detection and Matching

- 1. Feature detection (extraction)

  Search for locations that match well in other images

- 2. Feature description

  Keypoint locations transformed into compact stable/invariant descriptor

- 3. Feature matching

  searches efficiently for likely matching candidates in other images

Related Questions:

- How to find locations that match well with other images, which are repeatable and distinctive?
- How to describe a feature / design a feature descriptor?
- How to establish correspondences, i.e., compute matches?

Feature Matching

Extracted features and their descriptors from two or more images

→ Establish feature matches between these images

- Select matching strategy and error rates
  - Feature descriptors have been designed so that the Euclidean (vector magnitude) distances in feature space can be directly used for ranking potential matches
  - Given the Euclidean distance the simplest matching strategy is to set a threshold (max distance) and to return all matches from other images within this threshold.
  - Quantify performance of a matching algorithm at a particular threshold
- Devise efficient data structures and algorithms

Efficient Matching

For feature matching, we need to answer a large number of nearest neighbor queries

- Exhaustive search $O(n^2)$

KD-Tree

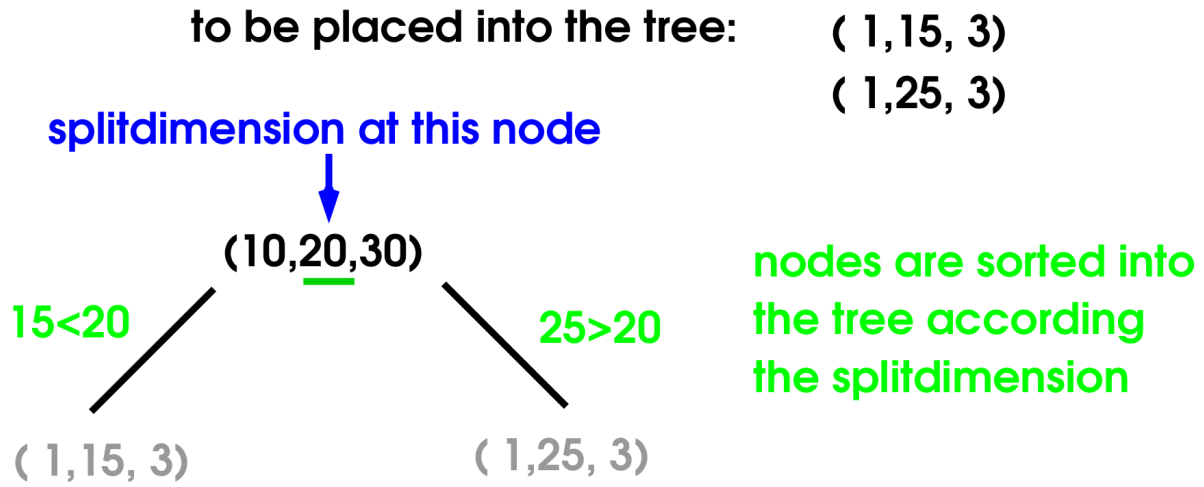
A KD-tree stores K-dimensional data points in an extended binary tree structure.

Main Idea:

- split according to one of the K dimensions at a time (introduction of a split-dimension)

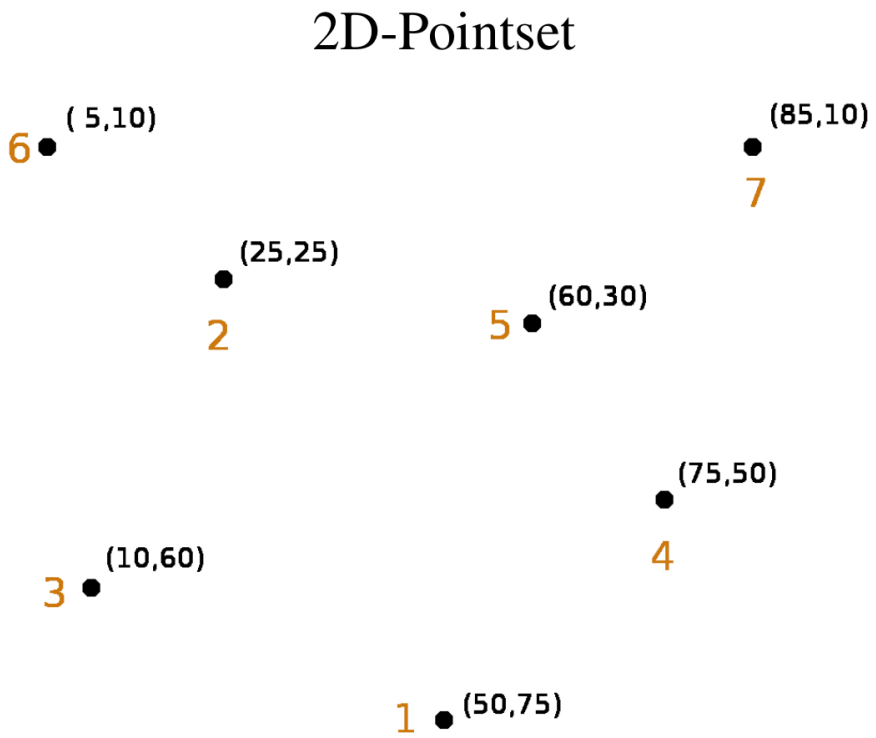
→ provides an efficient search structure

Example: Insert 3D-Points into the tree structure



KD-Tree

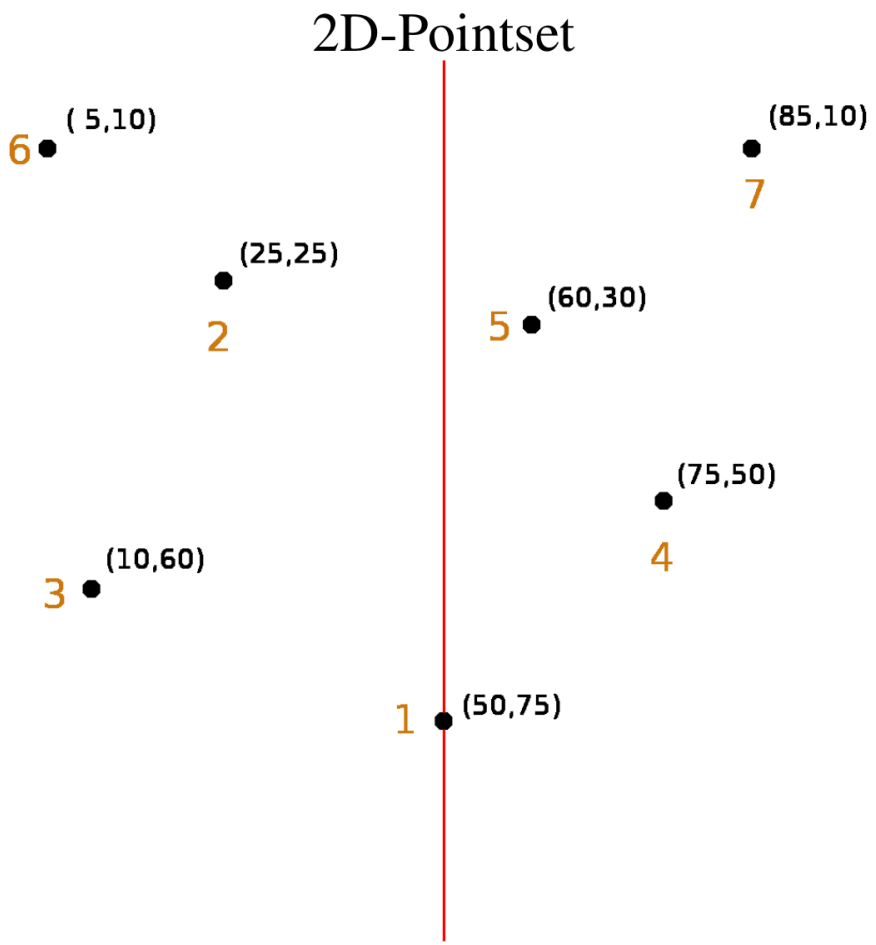
The shown 2D data-points will be used to create a KD-tree (2D-tree) by sorting them into the tree in a random (here given) order.



- Each inserted node splits the data-space according to the split-dimension (indicated by an underscore) into two parts.
- Here, the split-dimension changes with the depth of the tree (x ⇐⇒ y).

KD-Tree

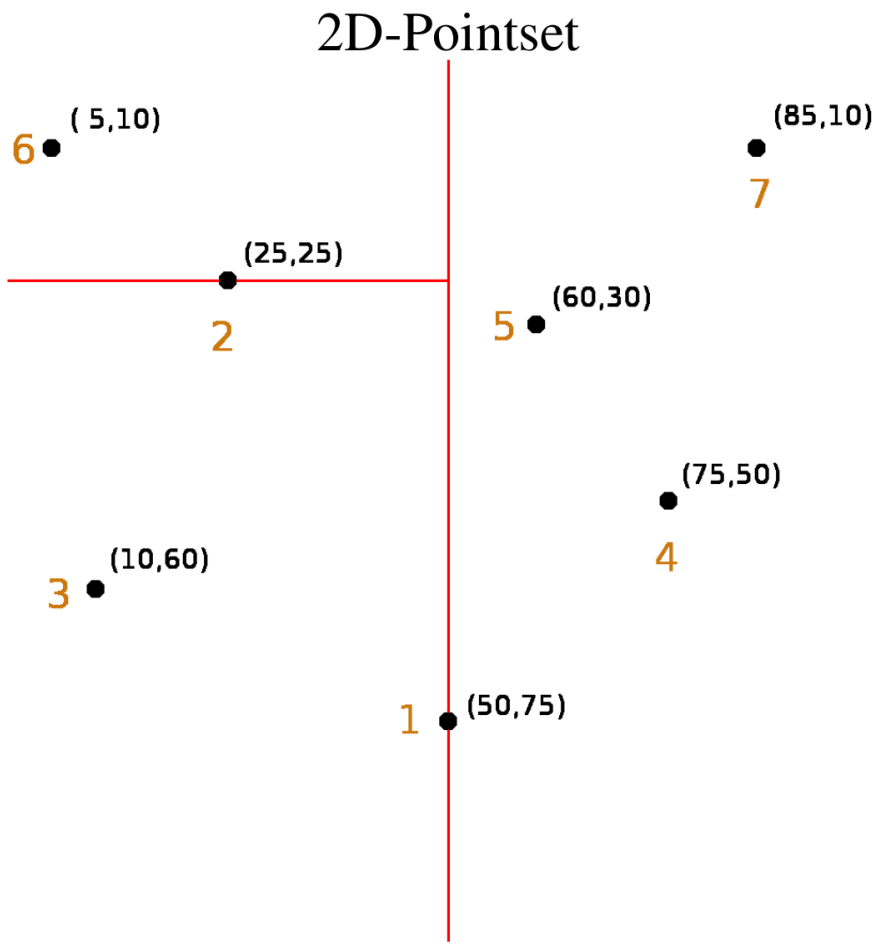
The shown 2D data-points will be used to create a KD-tree (2D-tree) by sorting them into the tree in a random (here given) order.

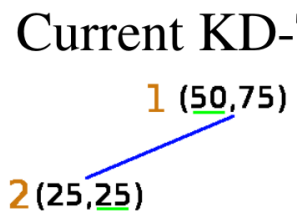


- Each inserted node splits the data-space according to the split-dimension (indicated by an underscore) into two parts.
- Here, the split-dimension changes with the depth of the tree (x ⇐⇒ y).

KD-Tree

The shown 2D data-points will be used to create a KD-tree (2D-tree) by sorting them into the tree in a random (here given) order.

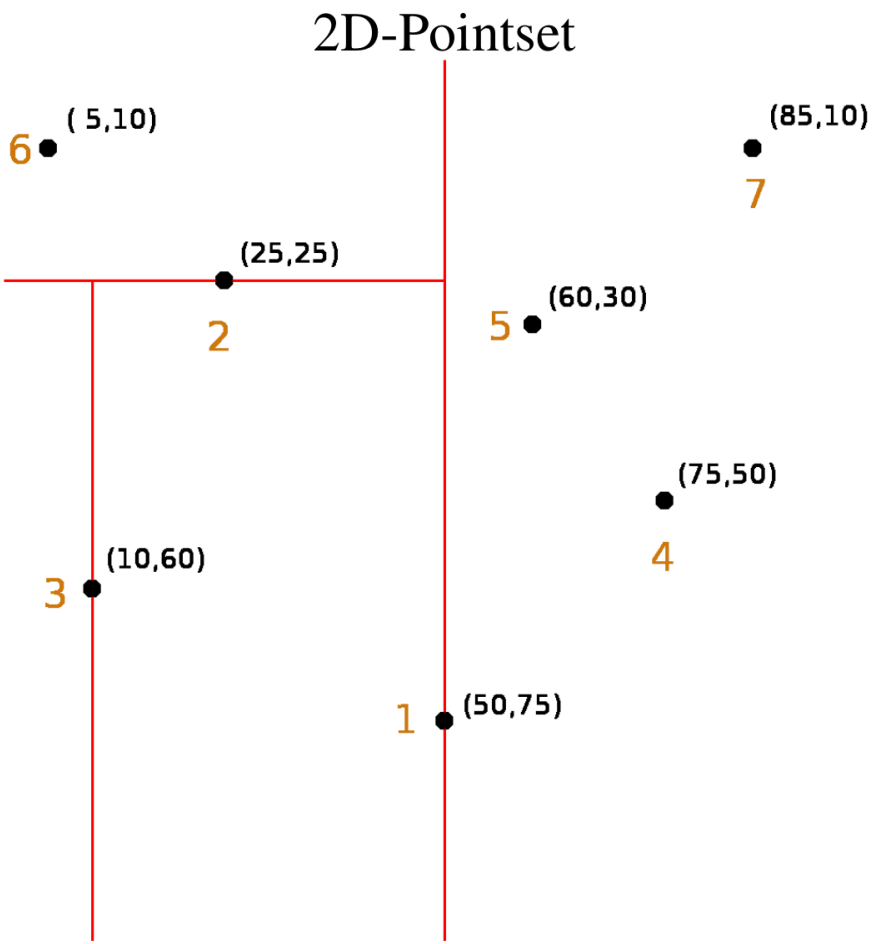


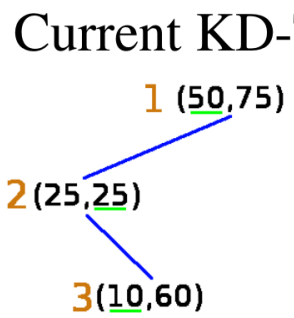


- Each inserted node splits the data-space according to the split-dimension (indicated by an underscore) into two parts.
- Here, the split-dimension changes with the depth of the tree (x ⇐⇒ y).

KD-Tree

The shown 2D data-points will be used to create a KD-tree (2D-tree) by sorting them into the tree in a random (here given) order.

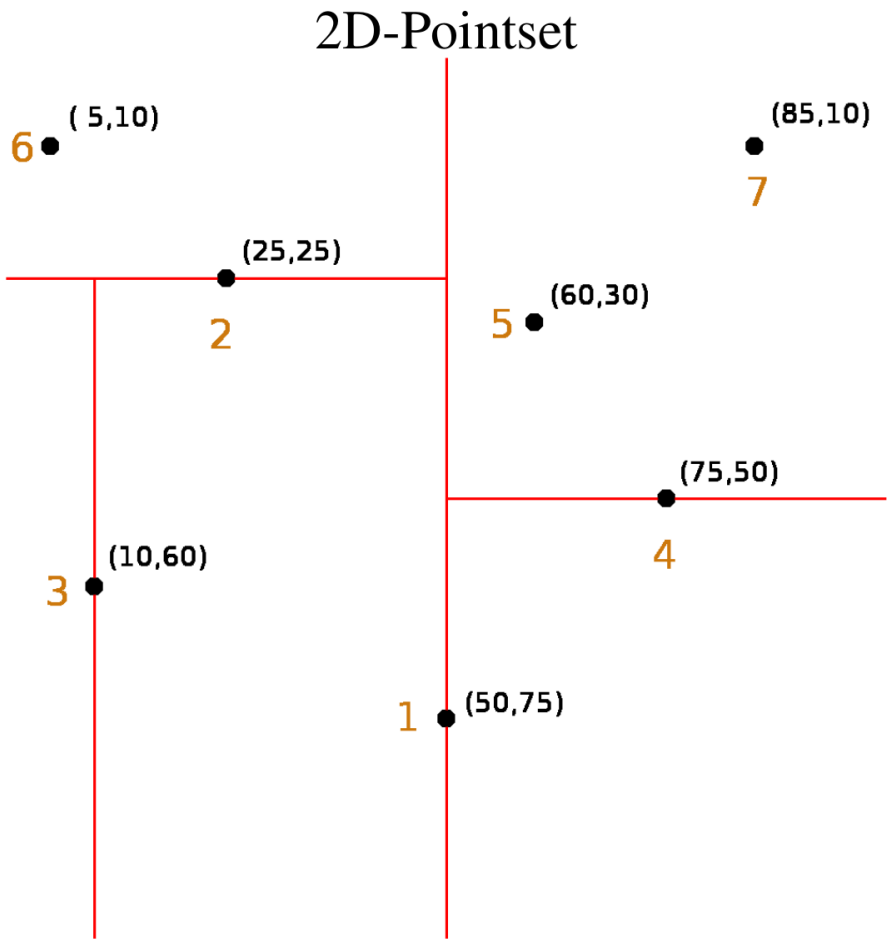


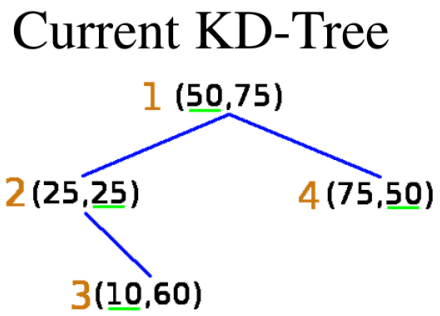


- Each inserted node splits the data-space according to the split-dimension (indicated by an underscore) into two parts.
- Here, the split-dimension changes with the depth of the tree (x ⇐⇒ y).

KD-Tree

The shown 2D data-points will be used to create a KD-tree (2D-tree) by sorting them into the tree in a random (here given) order.

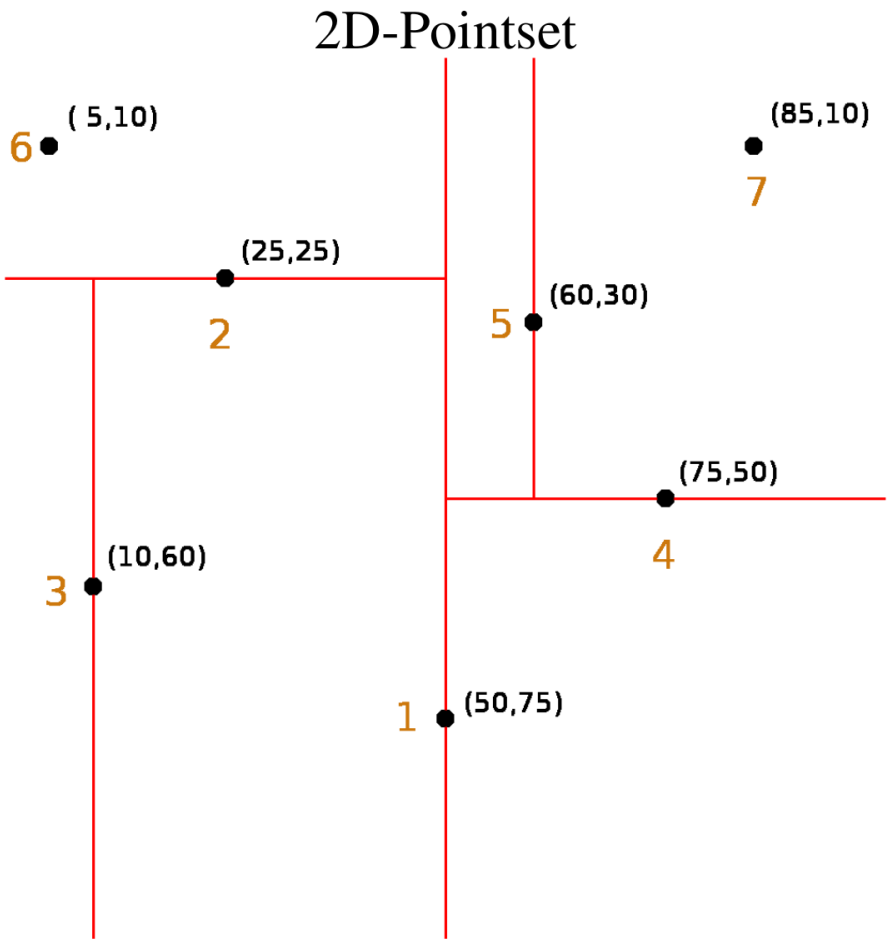


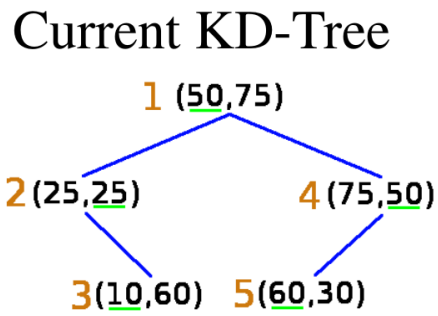


- Each inserted node splits the data-space according to the split-dimension (indicated by an underscore) into two parts.
- Here, the split-dimension changes with the depth of the tree (x ⇐⇒ y).

KD-Tree

The shown 2D data-points will be used to create a KD-tree (2D-tree) by sorting them into the tree in a random (here given) order.

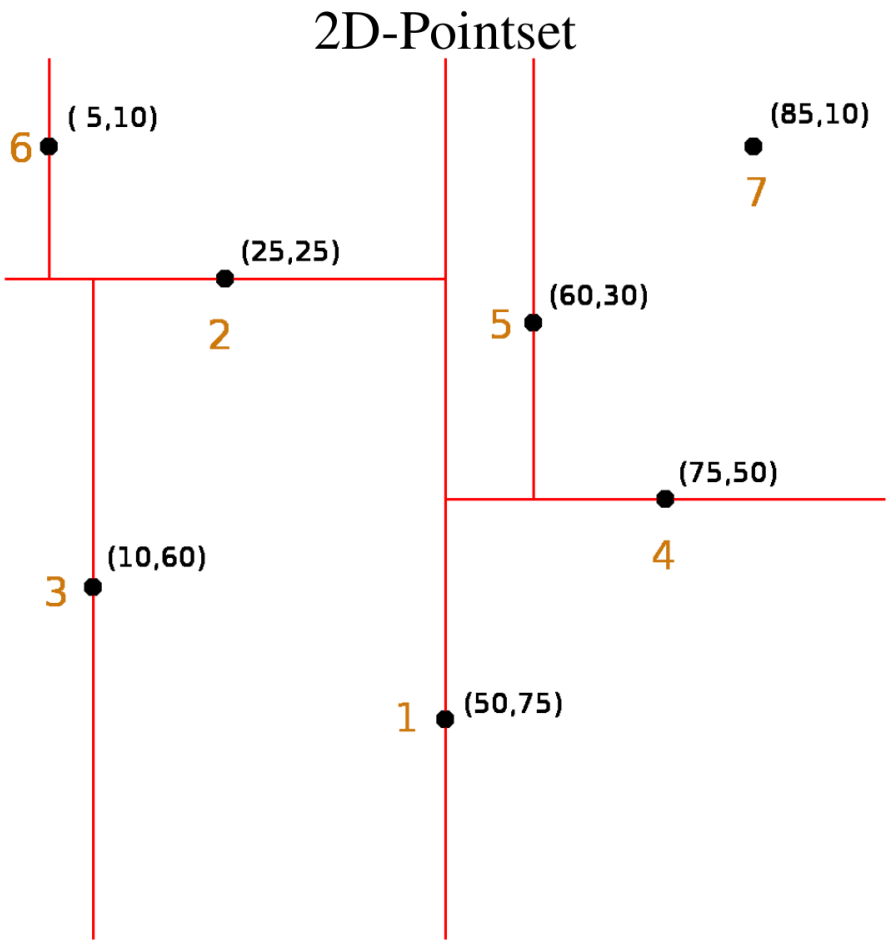


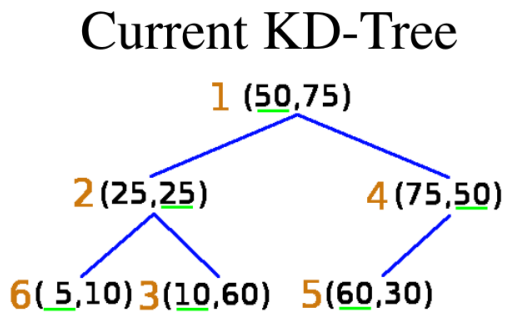


- Each inserted node splits the data-space according to the split-dimension (indicated by an underscore) into two parts.
- Here, the split-dimension changes with the depth of the tree (x ⇐⇒ y).

KD-Tree

The shown 2D data-points will be used to create a KD-tree (2D-tree) by sorting them into the tree in a random (here given) order.

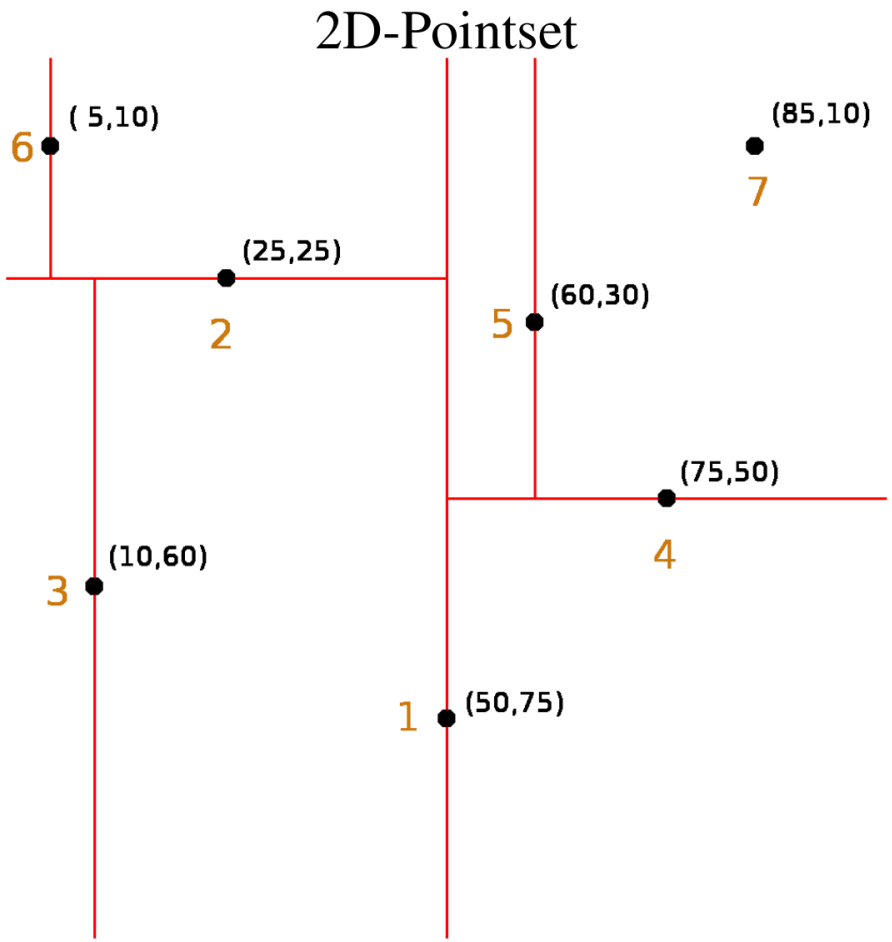


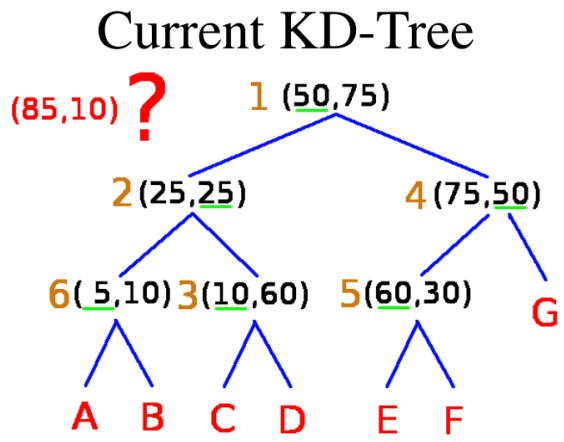


- Each inserted node splits the data-space according to the split-dimension (indicated by an underscore) into two parts.
- Here, the split-dimension changes with the depth of the tree (x ⇐⇒ y).

KD-Tree

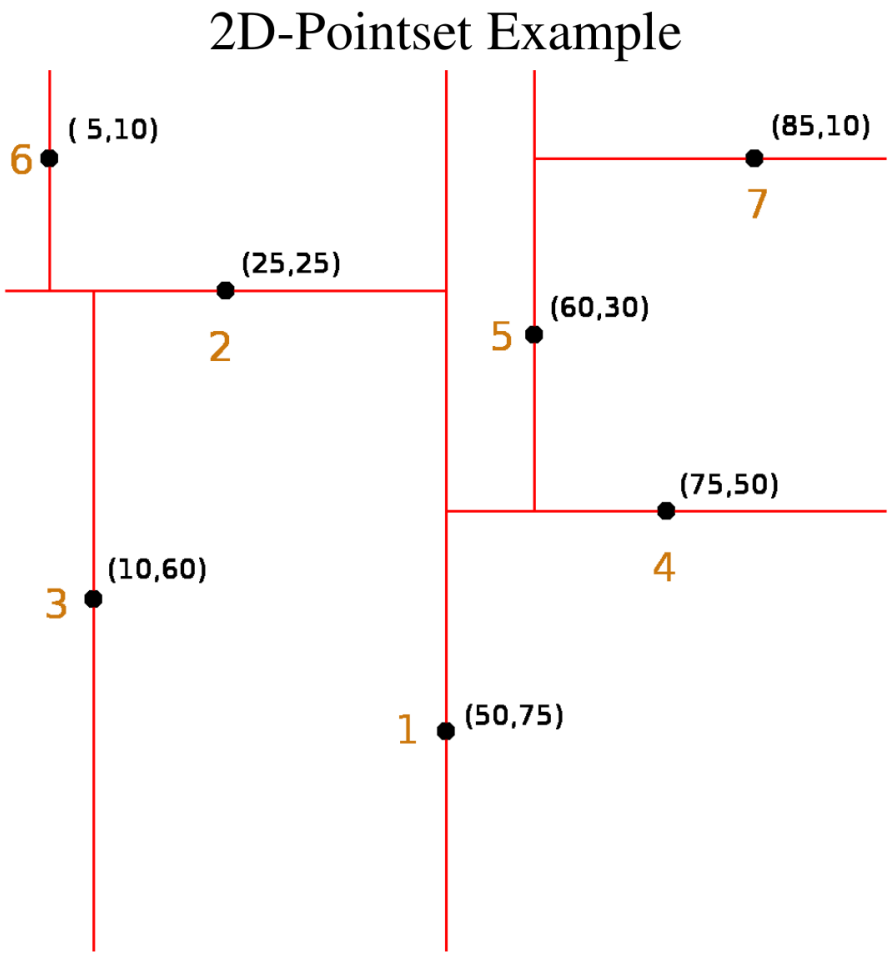
The shown 2D data-points will be used to create a KD-tree (2D-tree) by sorting them into the tree in a random (here given) order.

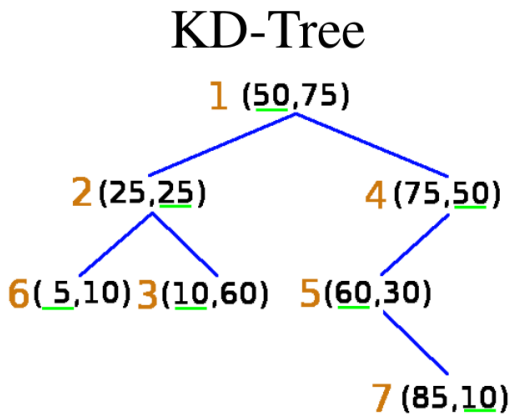




- Each inserted node splits the data-space according to the split-dimension (indicated by an underscore) into two parts.
- Here, the split-dimension changes with the depth of the tree (x ⇐⇒ y).

KD-Tree





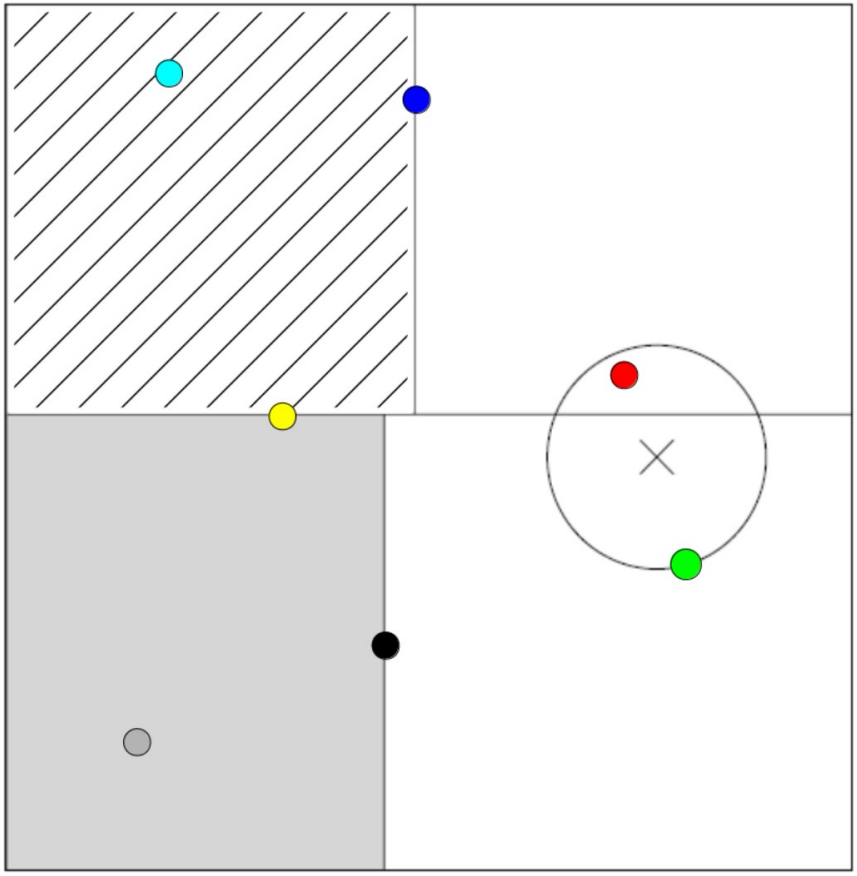
- The split-dimension changes at each level and is indicated by an underscore.
- In addition the order of insertion is indicated too.

Note: For large dimensionality, e.g., > 20 the KD-tree is not more efficient than exhaustive search

Nearest Neighbor Search

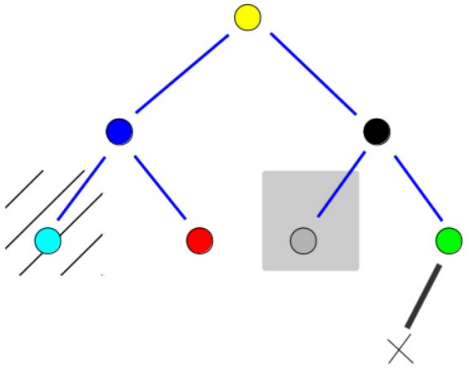
Scheme for finding the nearest neighbors:

Pointset



(Details: KD-Tree Tutorial, A.W. Moore)

Tree structure

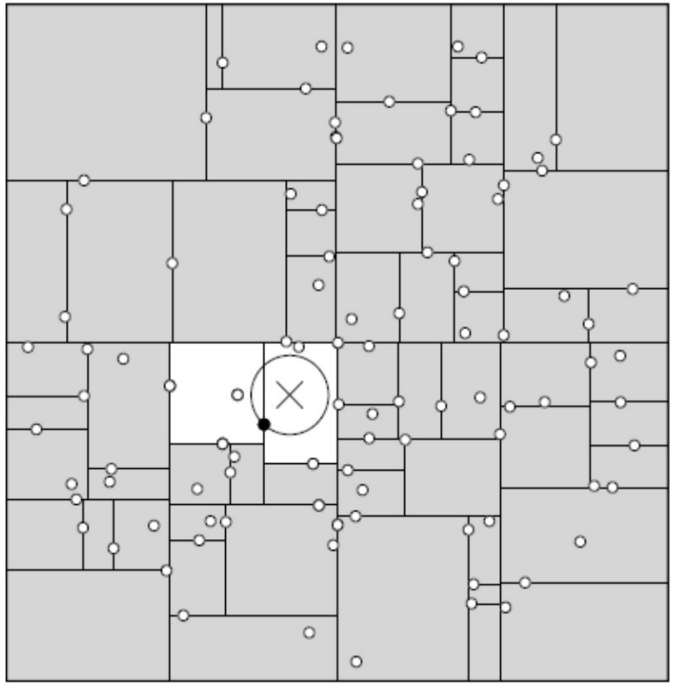


1. Sort node x into the tree.
2. Compute distance to its parent node
3. Recursively exclude areas where no points can be with a distance smaller than the current nearest
4. Update the current nearest distance/point when a nearer point is found

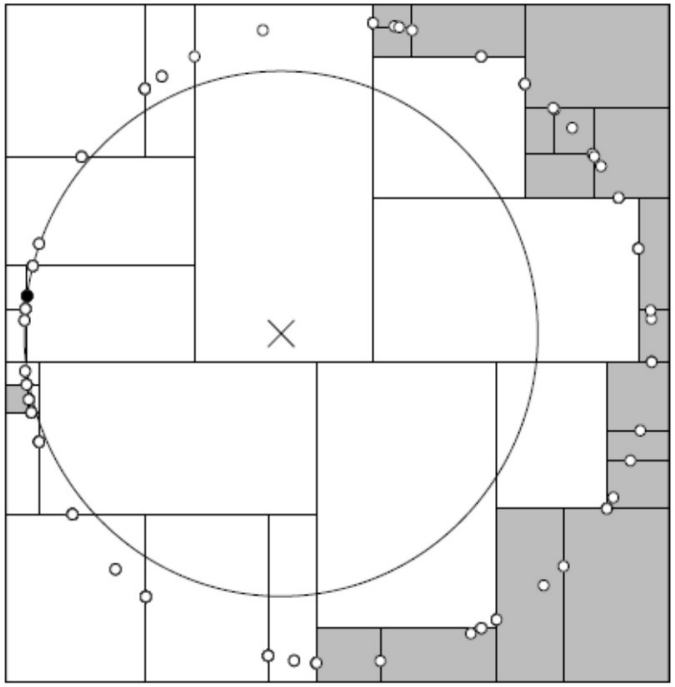
Note: The split-line is not shown for leaf nodes.

Nearest Neighbor Search

Usually only a few nodes must be visited…



… but bad configurations exist



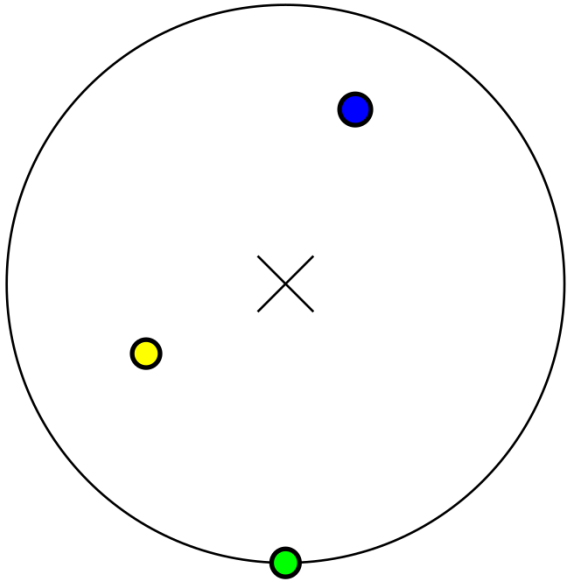
(Source: KD-Tree Tutorial, A.W. Moore)

→ Worst case is a linear search

K Nearest Neighbors

Find the K nearest neighbors needs a small modification of the nearest neighbor search.

3 nearest neighbors



(Details: KD-Tree Tutorial, A.W. Moore)

- Search within a sphere which radius is determined by the currently found K nearest neighbors.
- As long as not K neighbors are found, let the search radius be infinity.

Efficient Matching

Search for potential candidates:

- Brute force (very inefficient)
- Indexing structures
  - multi-dim search tree (KD-tree)
  - hash table
- Feature tracking over image sequences using
  - normalized cross-correlation,
  - hierarchical strategies
  - affine motion models (Knade-Lucas-Tomasi (KLT) tracker)
- Machine learning algorithms
  - reinforcement learning (Recurrent attention models)
- Approximate search
- Vocabulary trees

Literature

How to detect and match feature points

- Szeliski book chapter 4.1