# Learning from Images

## Image Features, Descriptors and Image Matching

Master DataScience

Prof. Dr. Kristian Hildebrand
khildebrand@bht-berlin.de

# Course outline

- **Image descriptors**
  - Global vs. local
  - Harris Corners
  - SIFT
- **Image matching**
  - Distance Metrics
  - FLANN
    - Randomized Kd-Tree
    - K-Means Trees

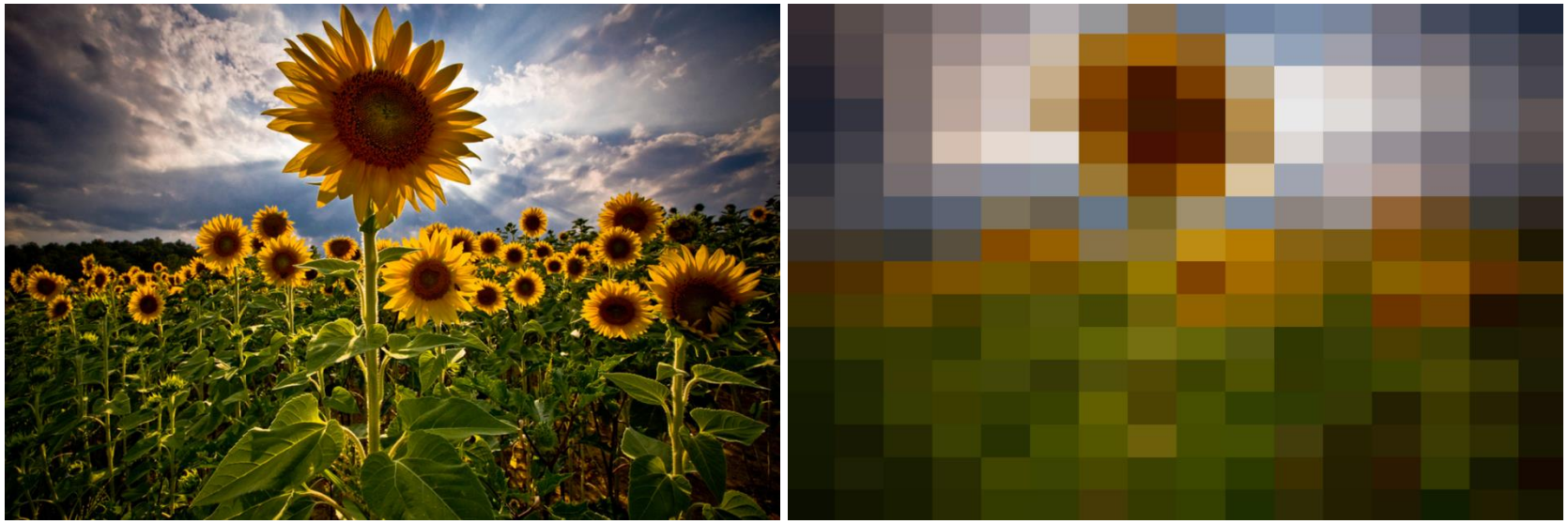

What do you see?

# One step back:
## How do we describe an image?
## How do we compare images?

# Image descriptors

- Descriptor - a piece of stored information that is used to identify an item in an information storage and retrieval system.
  - Descriptors save pertinent information, saving the processing time in future queries
  - Descriptors can save image features that are essential for search and comparison

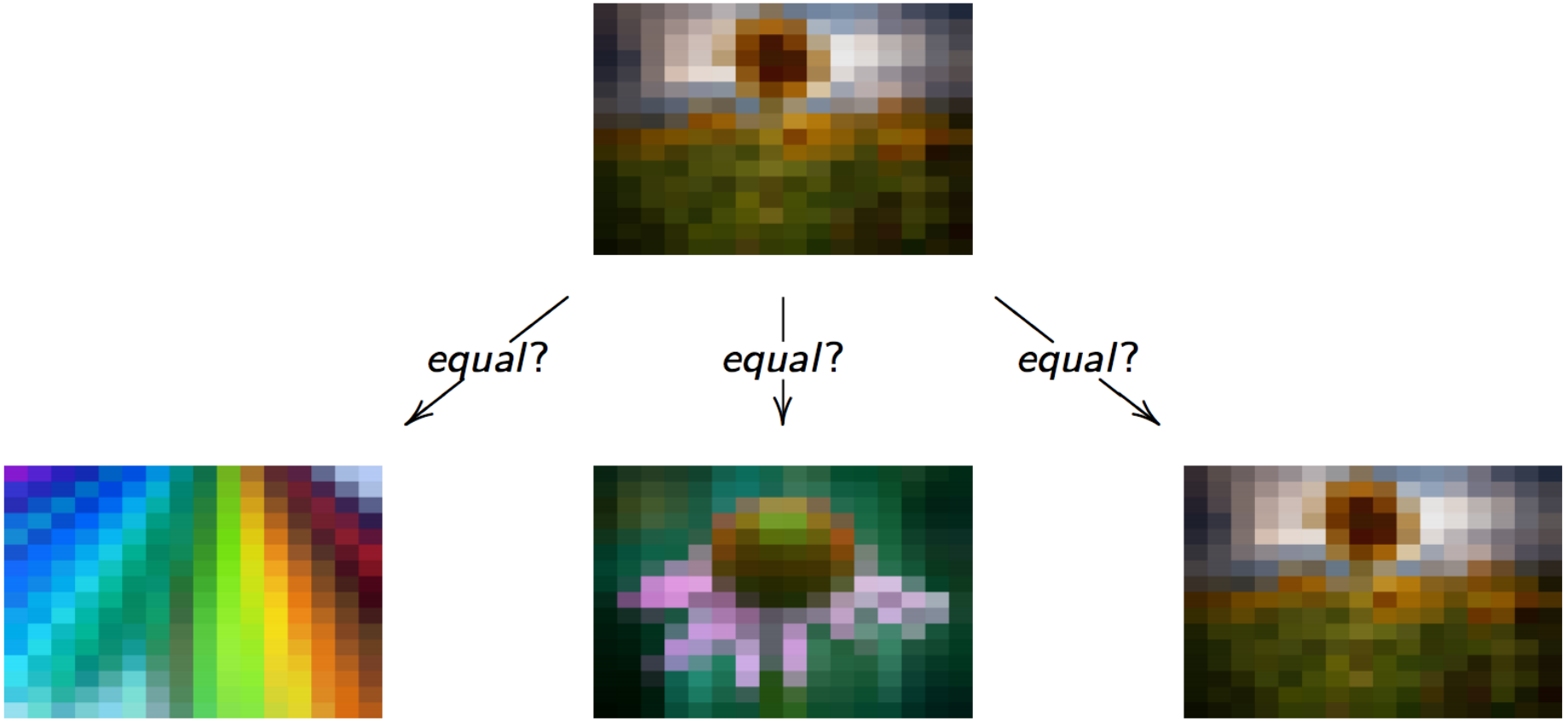
# Global image descriptors



https://groups.csail.mit.edu/vision/TinyImages/

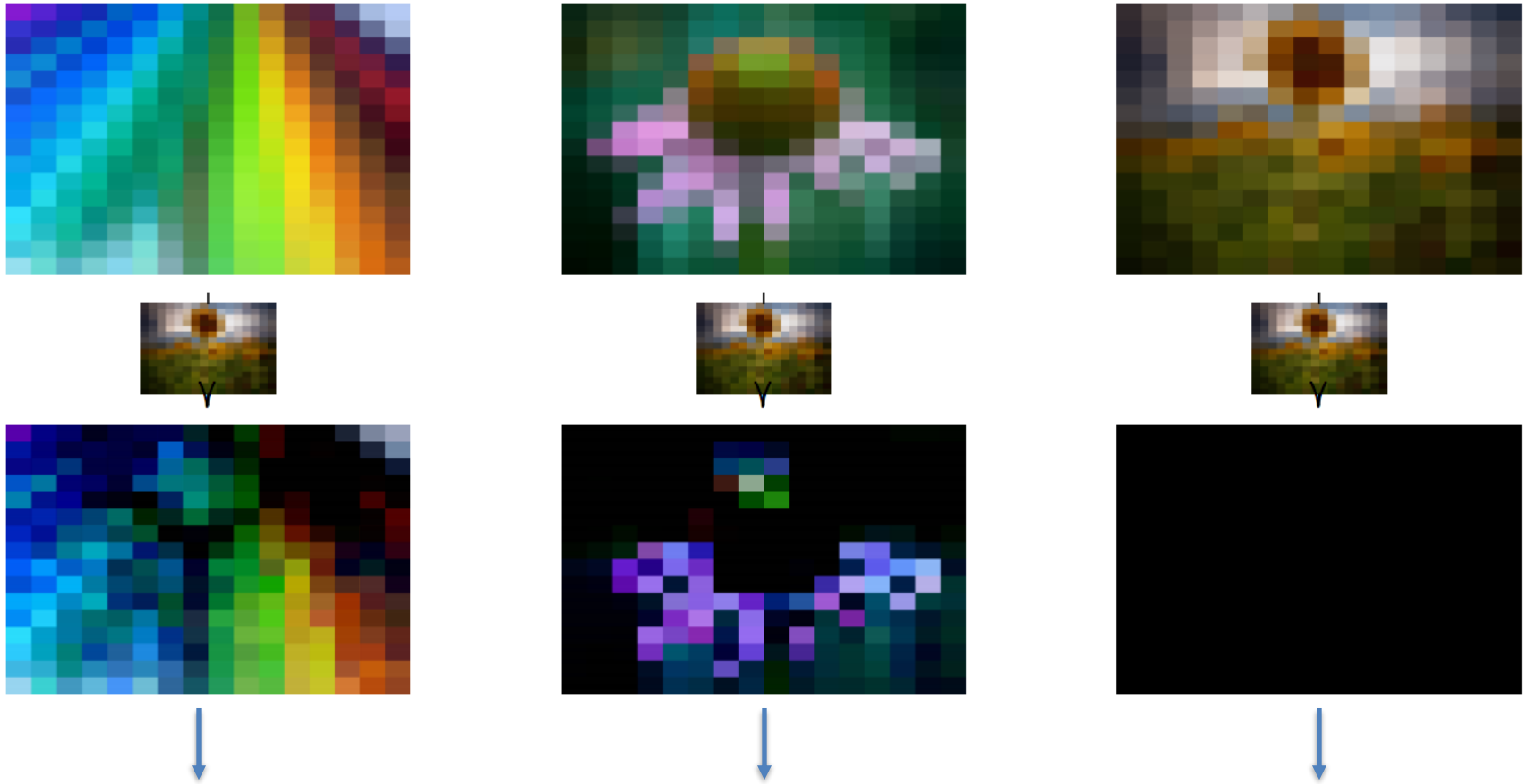
http://www.flickr.com/photos/alphageek/2759567956/

# Global image descriptors




http://www.flickr.com/photos/40513596@N00/71762740
http://www.flickr.com/photos/alphageek/2993813155/

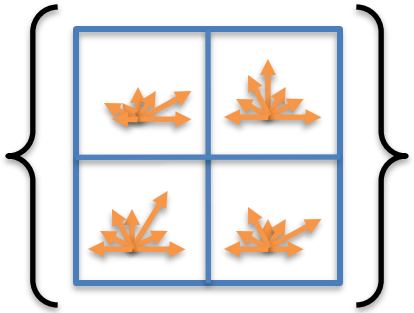
# Comparison example



# Global vs Local Features

# Image as a vector

**Feature description**

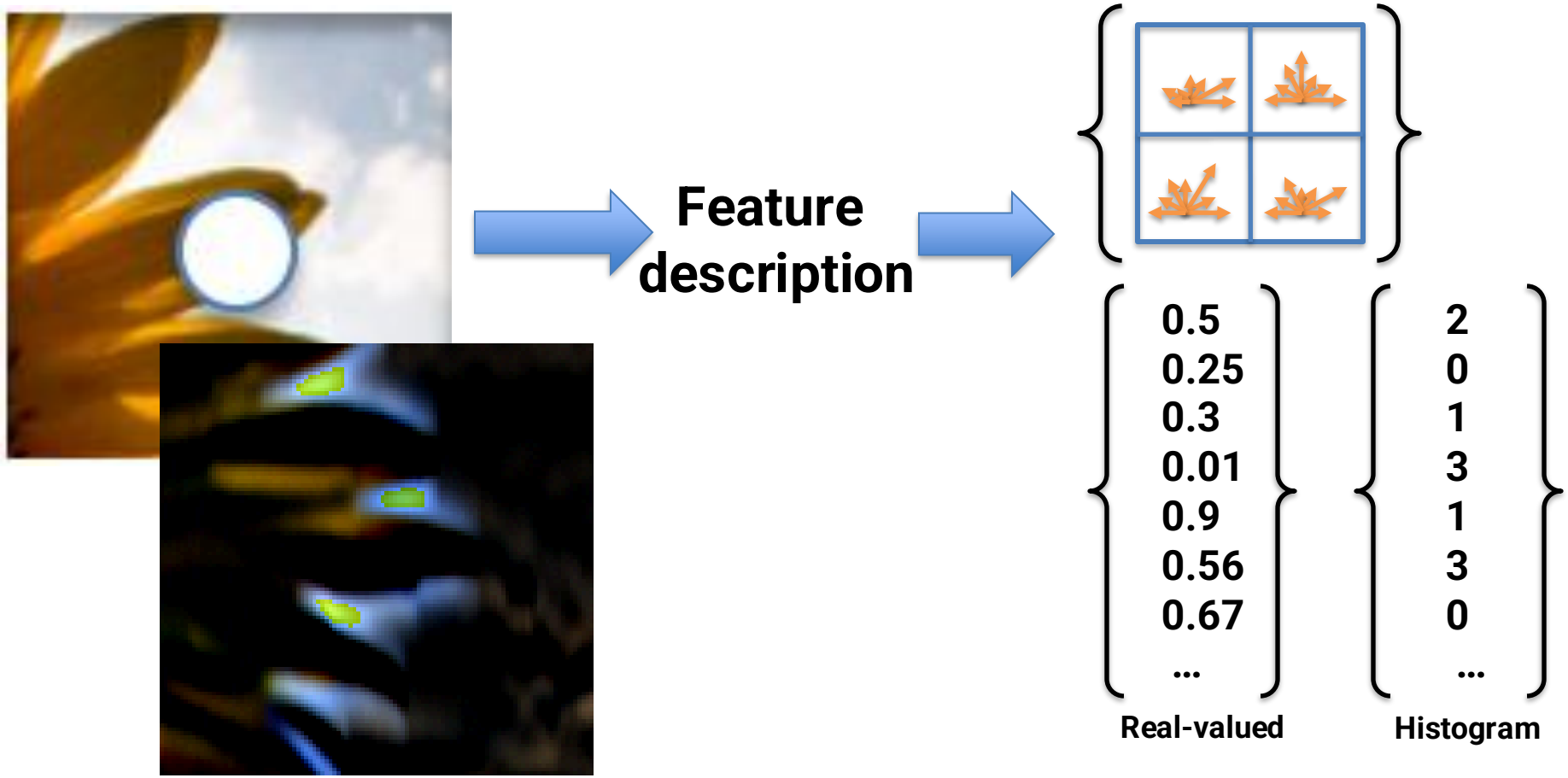


Histogram:
http://tinlizzie.org/histograms/

0.5
0.25
0.3
0.01
0.9
0.56
0.67
...

# Image as a vector

**What is the  advantage of local feature information?**



# Why local features?

- Locality
  - features are local, so robust to occlusion and clutter
- Distinctiveness:
  - can differentiate a large database of objects
- Quantity
  - hundreds or thousands in a single image
- Efficiency
  - real-time performance achievable
- Generality
  - exploit different types of features in different situations

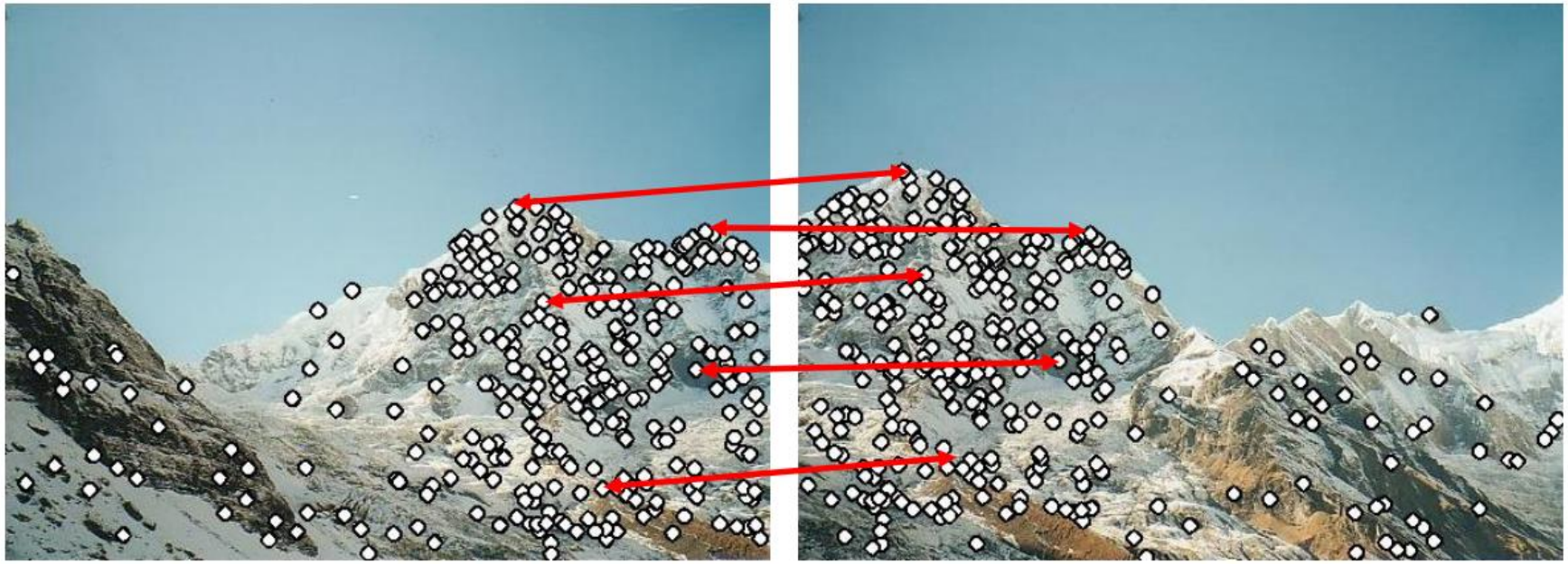
# Applications

- Features are used for:
  - Image alignment (e.g., panoramic mosaics)
  - Object recognition
  - 3D reconstruction (e.g., stereo)
  - Motion tracking
  - Indexing and content-based retrieval
  - Robot navigation
  - …

**What is a keypoint?**
**Or**
**Where  to extract features?**

# What is a good feature?

Need to detect that feature in the next frame again



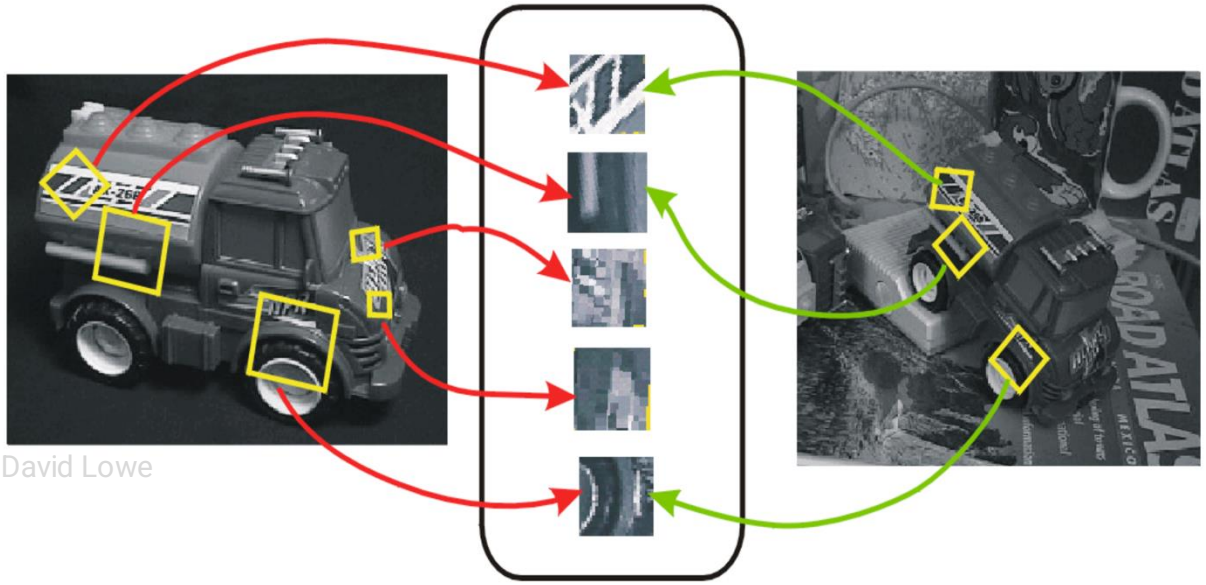
David Lowe

# What makes a good feature?

- Want uniqueness
  - Leads to unambiguous matches in other images
- Look for “interest points”
- Stable under lighting and viewpoint changes
- Keypoints that are similarity- and affine-invariant, or at least stable
- Enable accurate matching of keypoints between images
  - Object recognition / Marker detection / image registration

# Transformation invariance

- Want features that are invariant to transformations (affine-invariant)
  - geometric invariance: translation, rotation, scale
  - photometric invariance: brightness, exposure, …



David Lowe

**Features descriptors**

# Feature Detection / Location

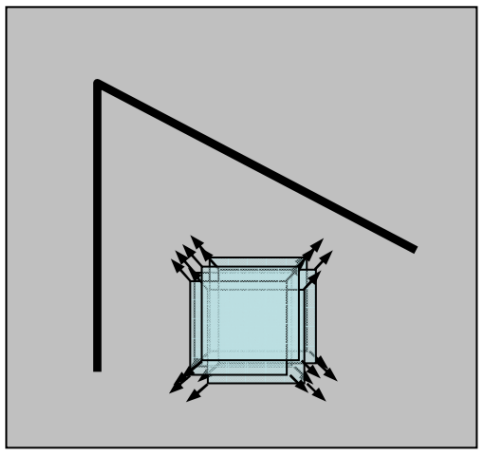
- What could be a possible detector?
  - random?
  - uniform?
  - edges?
  - **corners?**
- corners – intersection of edges
  - point where direction of intersections change
  - gradient of the image (in both directions) has high variation
    - this can be detected = we look for variation

# Harris Corner Detector

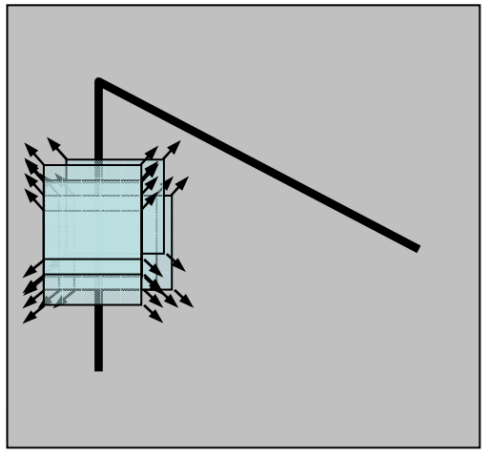
# Harris corner detector – The basic idea

- C.Harris, M.Stephens. “A Combined Corner and Edge Detector”. 1988
- We should easily recognize the point by looking through a small window
- Shifting a window in any direction should give a large change in intensity

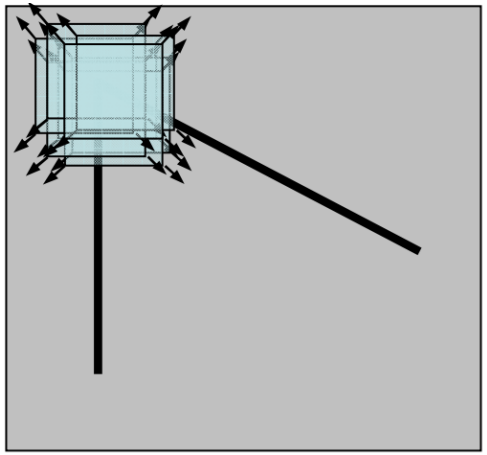
https://courses.cs.washington.edu/



“flat” region:
no change in all directions



“edge”: no change
along the edge direction

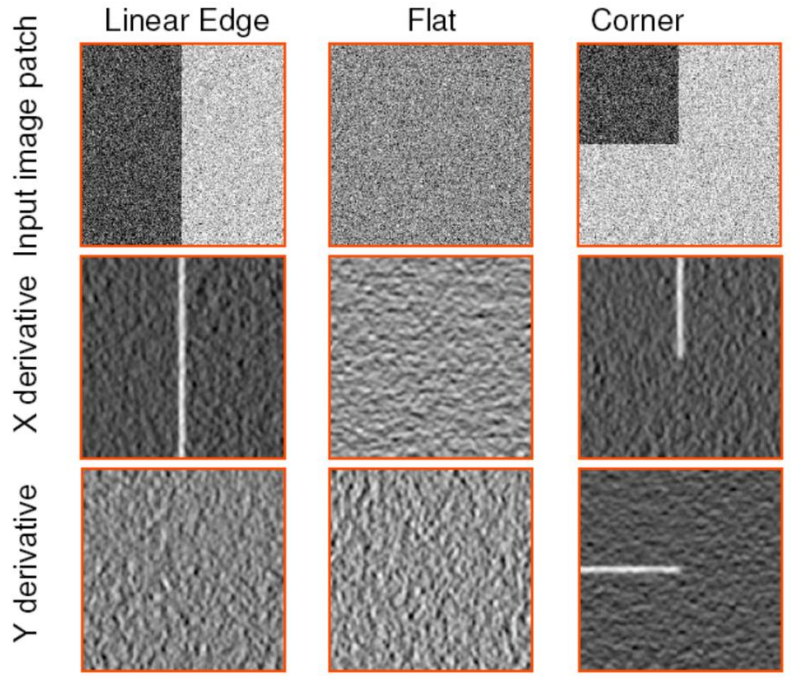


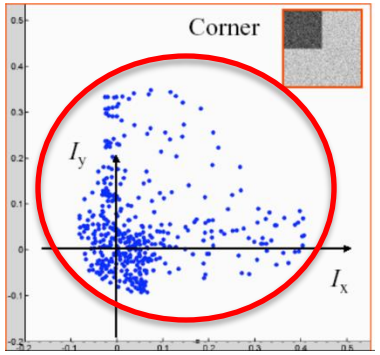
“corner”: significant
change in all directions

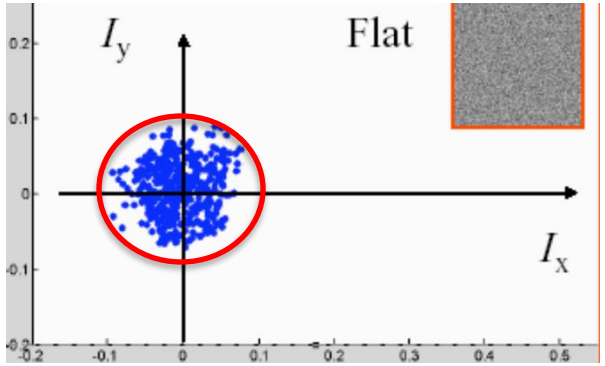
# Harris corner detector
# Demo

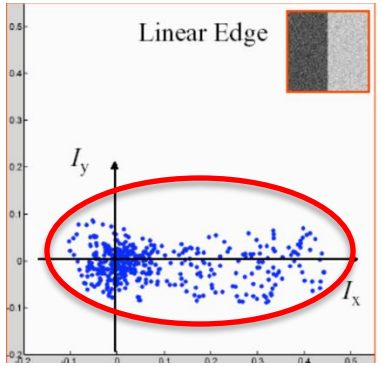
# Harris corner detector – The intuitive way

- Treat gradient vectors as a set of *(dx,dy)* points with a center of mass defined as being at (0,0)
- Fit an ellipse to that set of points via scatter matrix
- Analyze ellipse parameters for varying cases



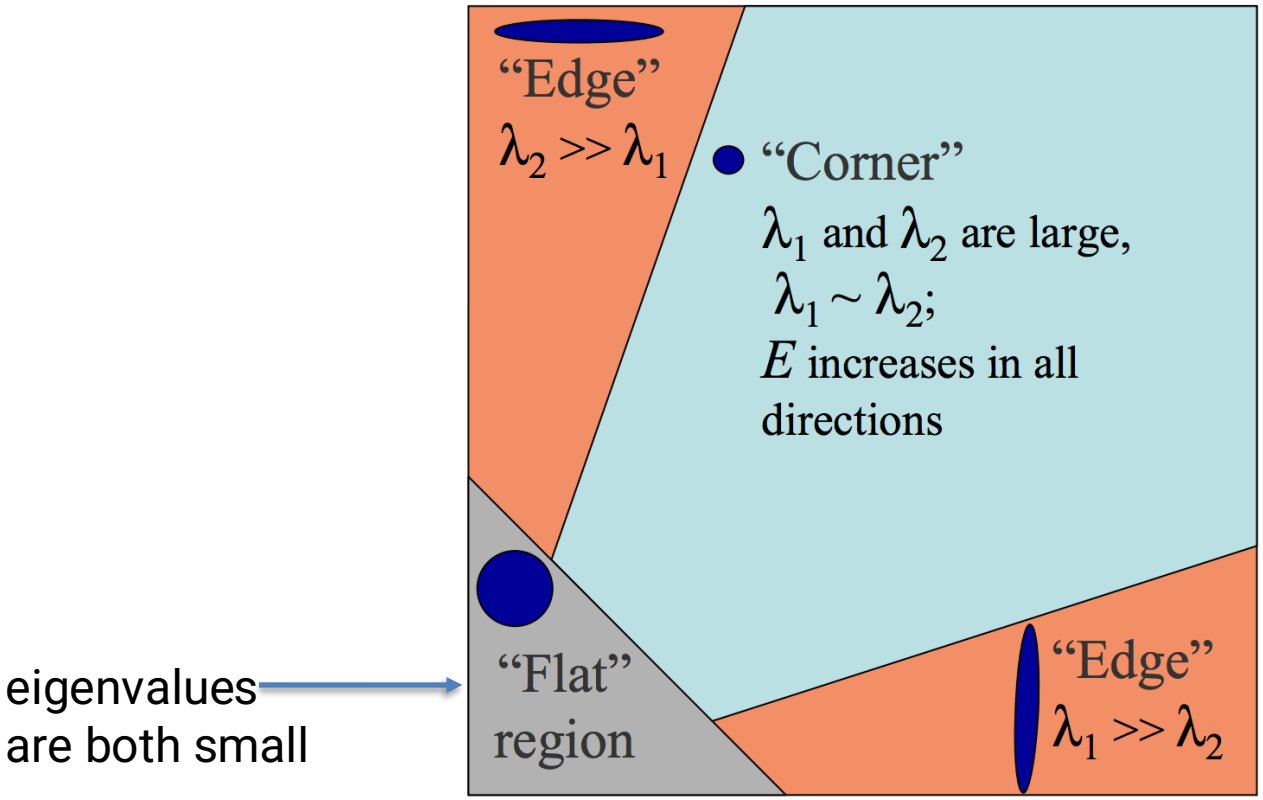



# Harris corner detector

- Classification of image points using **eigenvalues of M** ???

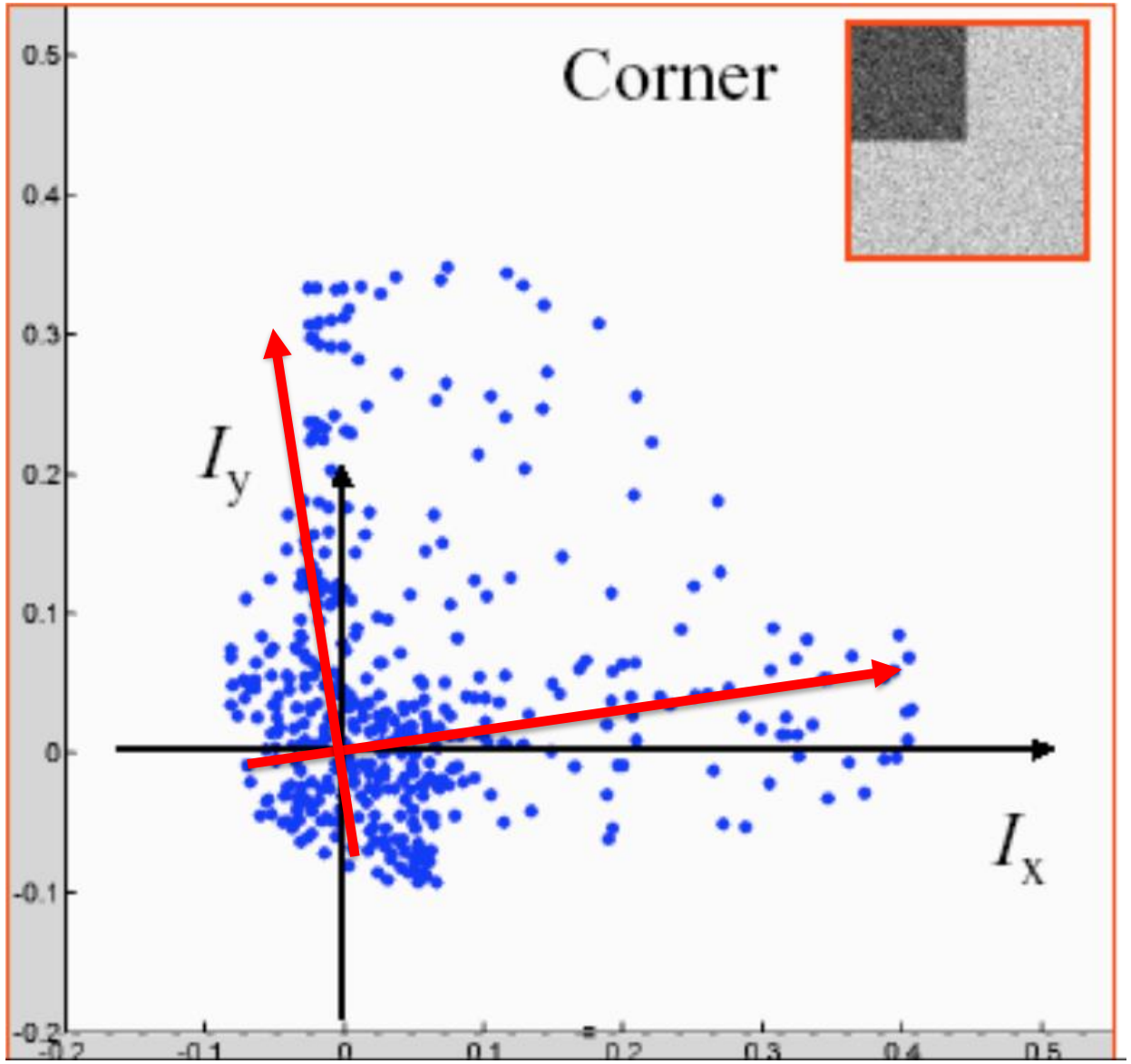



https://courses.cs.washington.edu/

# EigenWhat?

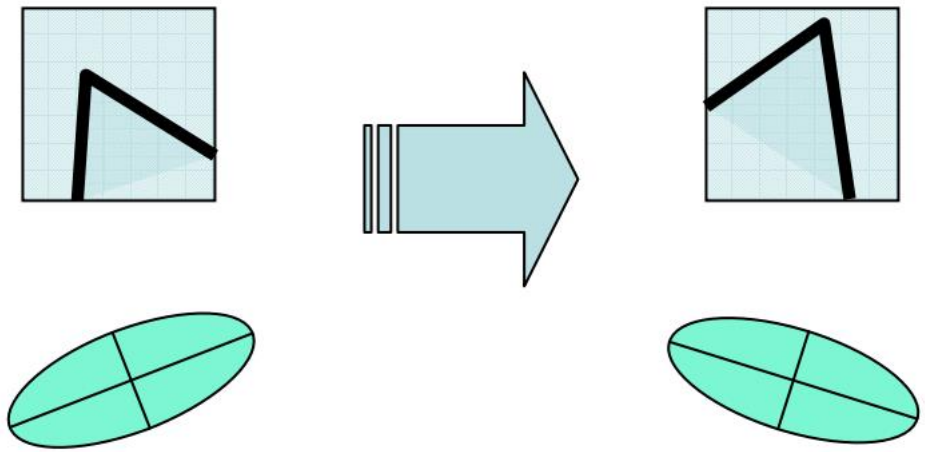
# Eigenvectors / Eigenvalues

# Eigenvectors - Definition



# Harris corner detector – Rotation invariance

- Ellipse rotates but its shape (i.e. eigenvalues) remains the same



https://courses.cs.washington.edu/

# Harris corner detector – The math

- detect change of intensity for a window shift

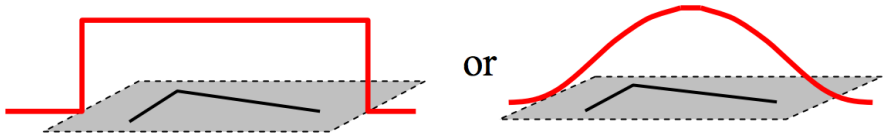
$$E(u,v) = \sum_{x,y} \; w \; [I(x+u, y+v) - I(x,y)]^2$$

w: Window function

$I(x+u, y+v)$: Shifted intensity

$I(x,y)$: Intensity

- Window function w(x,y):




1 in window, 0 outside  Gaussian

https://courses.cs.washington.edu/

- Maximize variation within window

# Harris Corner Detector – the Math

First order approximation from Taylor Series

$$f(x+u, y+v) \approx f(x,y) + uf_x(x,y) + vf_y(x,y)$$

$$\begin{aligned} E(u,v) &= \sum_{x,y} w(x,y)[I(x+u,y+v) - I(x,y)]^2 \\ &\approx \sum_{x,y} w(x,y)[I(x,y) + uI_x(x,y) + vI_y(x,y) - I(x,y)]^2 \\ &= \sum_{x,y} w(x,y)u^2 I_x^2 + 2uvI_xI_y + v^2 I_y^2 \\ &= \sum_{x,y} \begin{pmatrix} u & v \end{pmatrix} \begin{pmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} \end{aligned}$$

**rewrite as matrix equation**

$$= \begin{pmatrix} \sum I_x^2 & \sum I_xI_y \\ \sum I_xI_y & \sum I_y^2 \end{pmatrix}$$

**just the product of components of the gradient $I_x$, $I_y$**
**also known as structure tensor -> Matrix M**

# Harris corner detector – Efficient implementation

- One way to compute the eigenvalues efficiently

$$\lambda_{min} \approx \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2} = \frac{det(M)}{trace(M)}$$

- To avoid computing the eigenvalues which is computationally expensive one uses an approximation to determine if point is corner:

$$R = det(M) - k(trace(M))^2$$

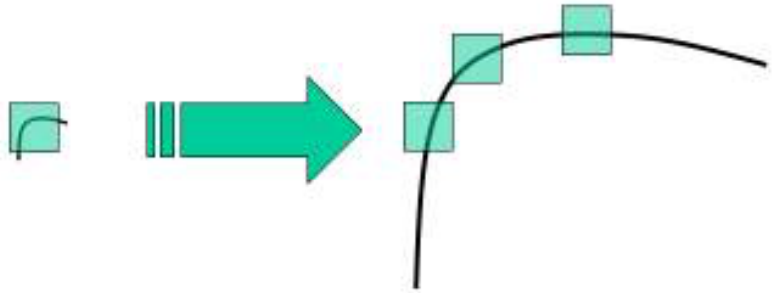
**Implementation is done as homework**

# Scale invariant feature transform (SIFT)

# Scale invariant feature transform (SIFT)

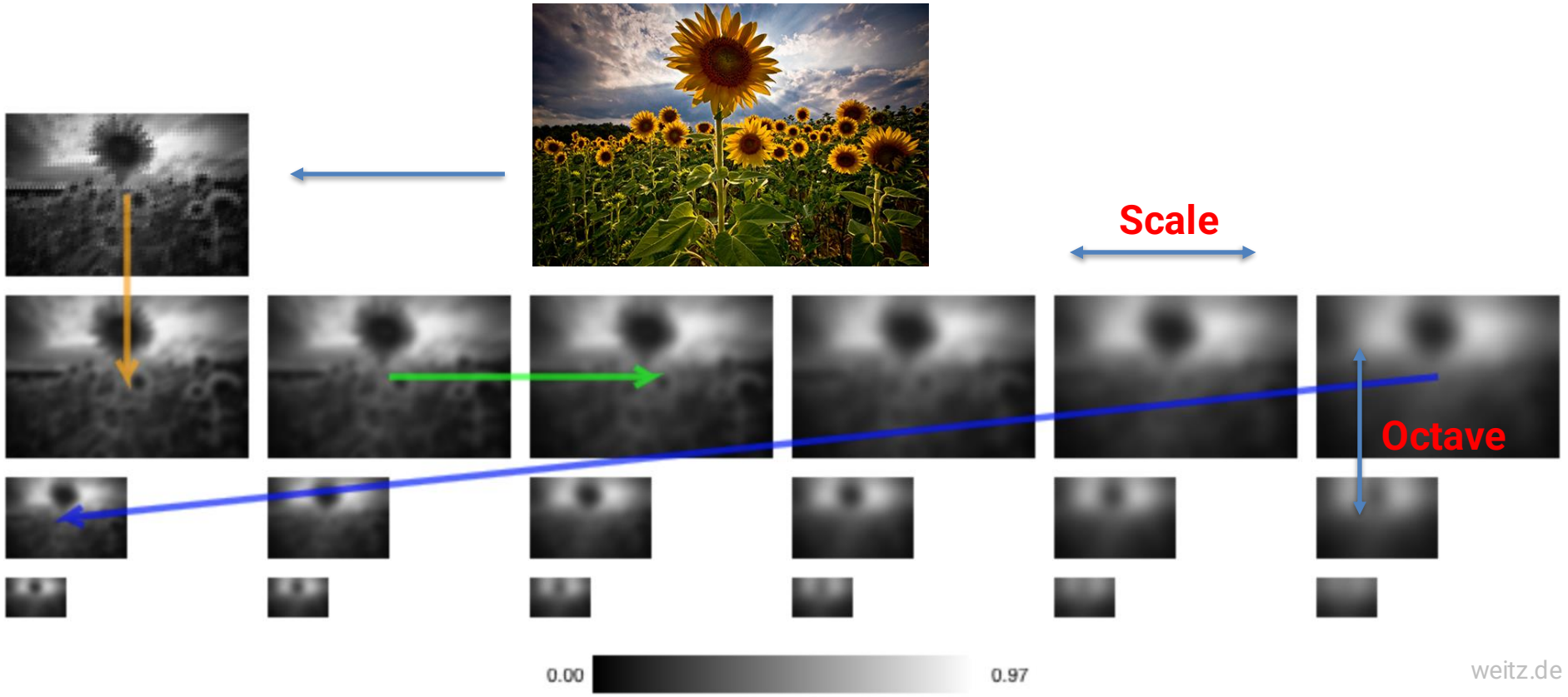
- Distinctive Image Features from Scale-Invariant Keypoints, David Lowe, 2004
- corner detectors are rotation-invariant
  - even if the image is rotated, one can find the same corners
  - corners remain corners in rotated image
- this is not true for scaling

- Harris detector is not scale-invariant



opencv.org

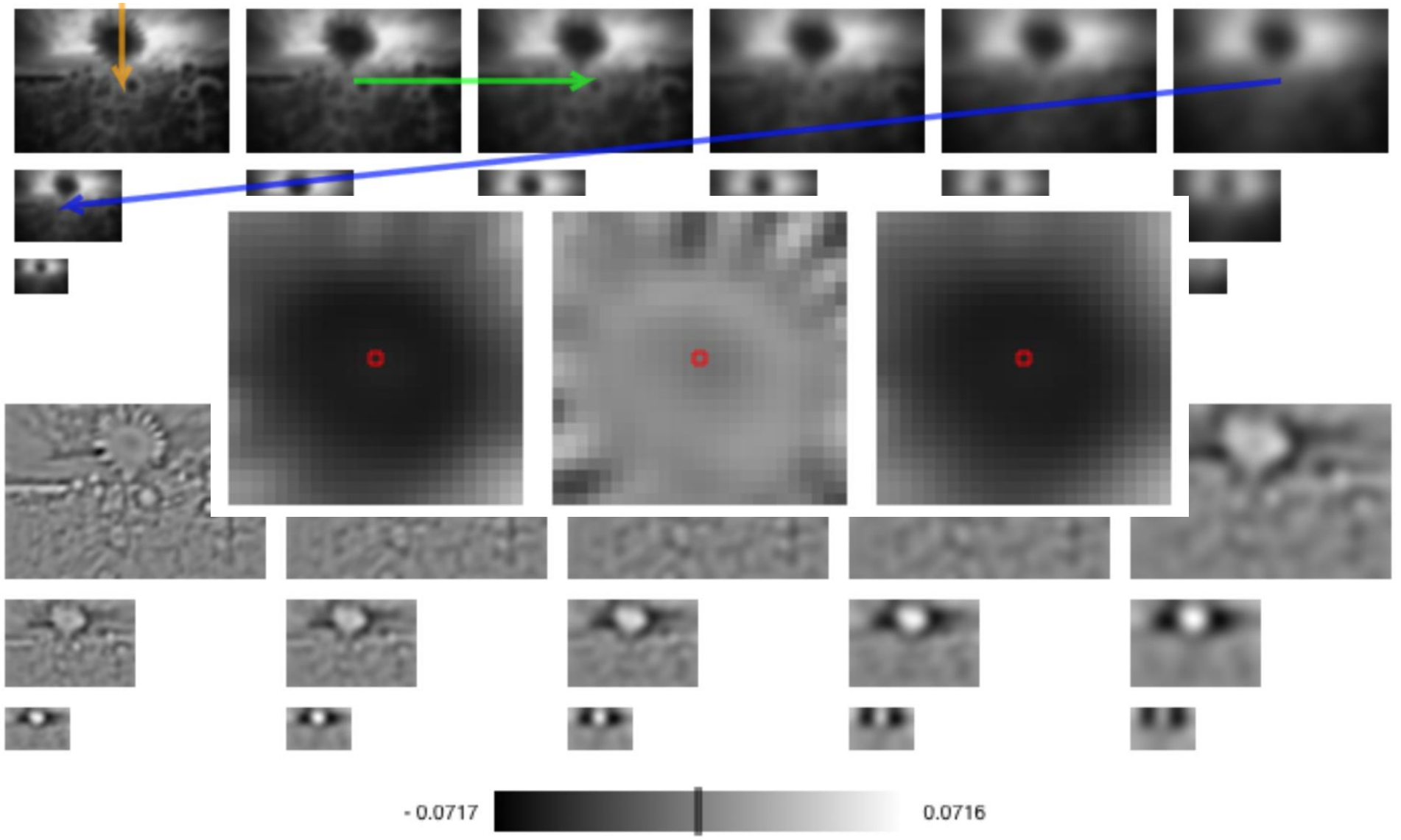
# SIFT - Scale-space Extrema Detection



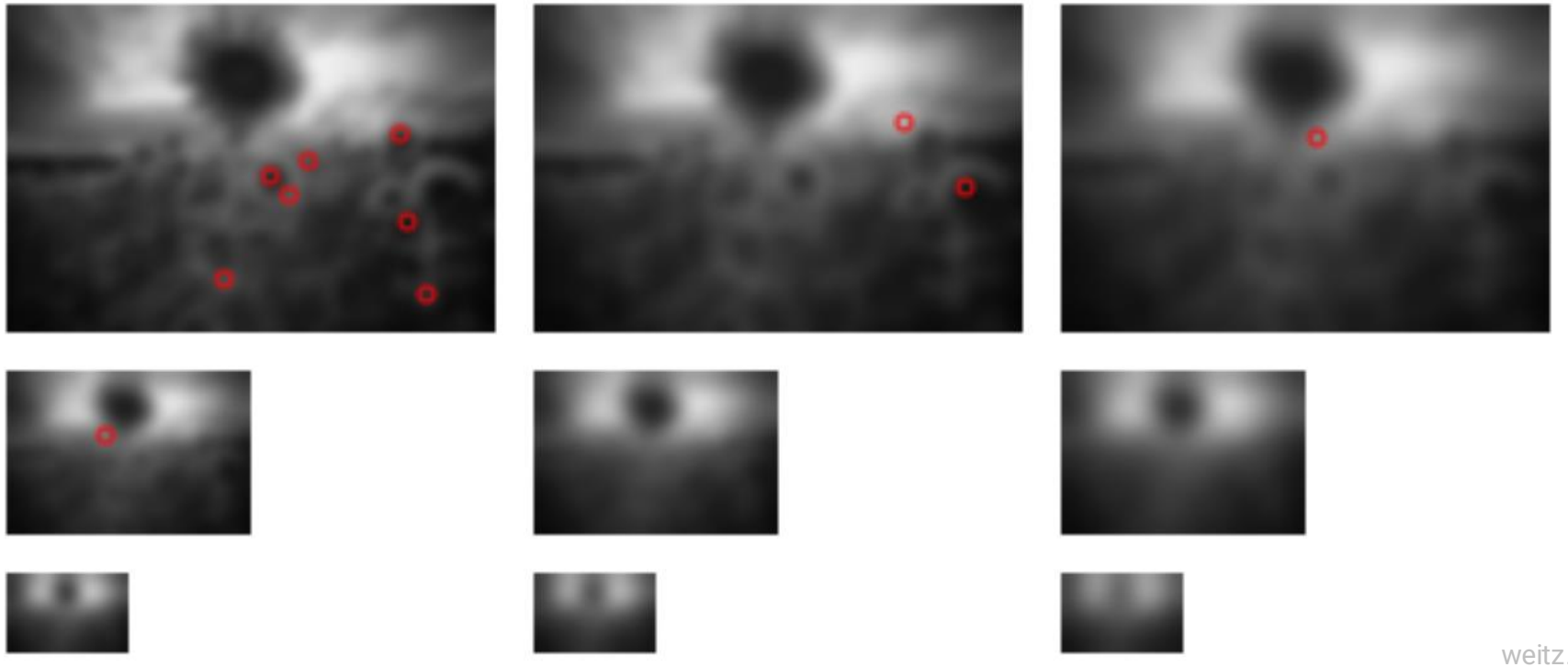

## Scale-space using bilinear interpolation and gaussian convolution

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

# SIFT - Difference of Gaussians



# SIFT - Keypoints localization




- one pixel in an image is compared with its 8 neighbours as well as 9 pixels in next scale and 9 pixels in previous scales
- If it is a local extrema, it is a potential keypoint
- It basically means that keypoint is best represented in that scale

# Feature

## Description of the image window around the keypoint

# SIFT - Keypoint Descriptor

- 16x16 neighbourhood around the keypoint
- devided into 16 sub-blocks of 4x4 size
- each sub-block, 8 bin orientation histogram is created
  - total of 128 bin values are available
- represented as a vector to form keypoint descriptor

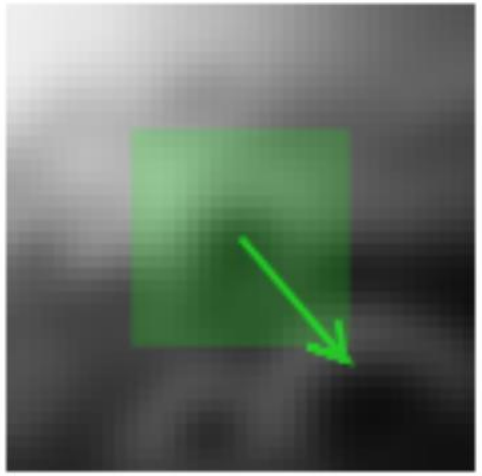
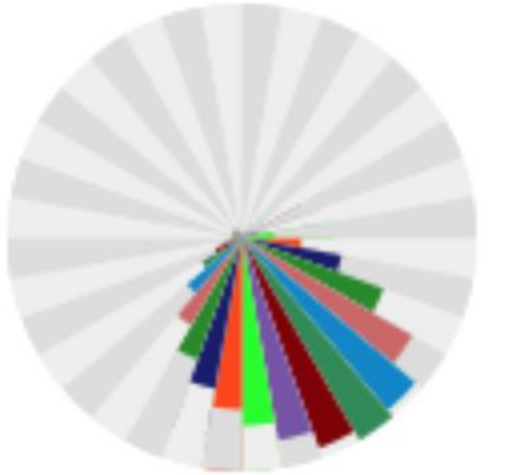


Image Gradients



weitz.de

Keypoint descriptor

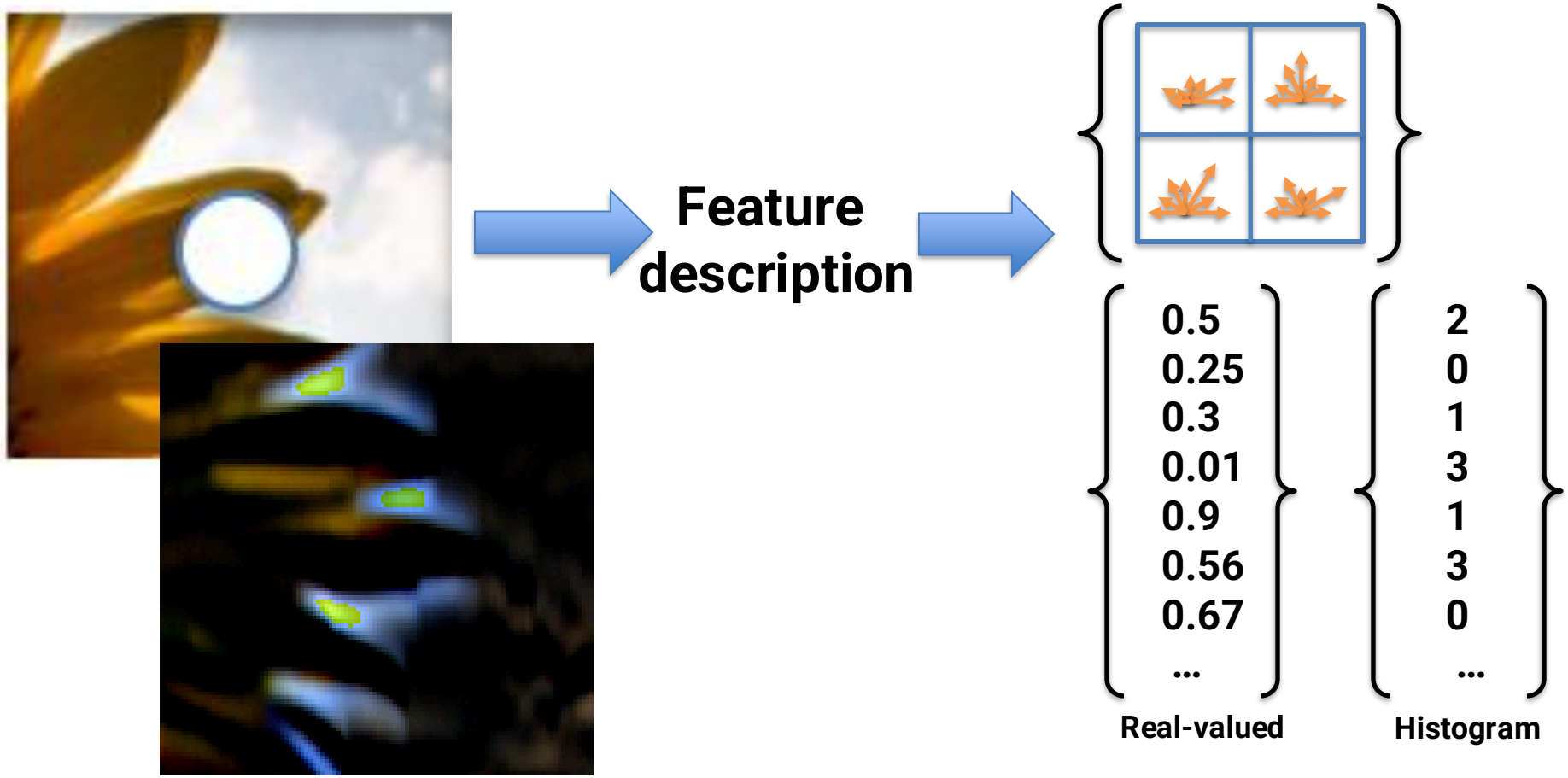
# More feature descriptors and keypoint detectors

- FAST - Features from Accelerated Segment Test
  - Detects corners using the FAST algorithm
- MSER - Maximally stable extremal region extractor.
- ORB - *oriented BRIEF* Keypoint detector and descriptor extractor
- BRISK - Binary Robust Invariant Scalable Keypoints
- SURF -Speeded-Up Robust Features
- FREAK -F*ast Retina Keypoint*

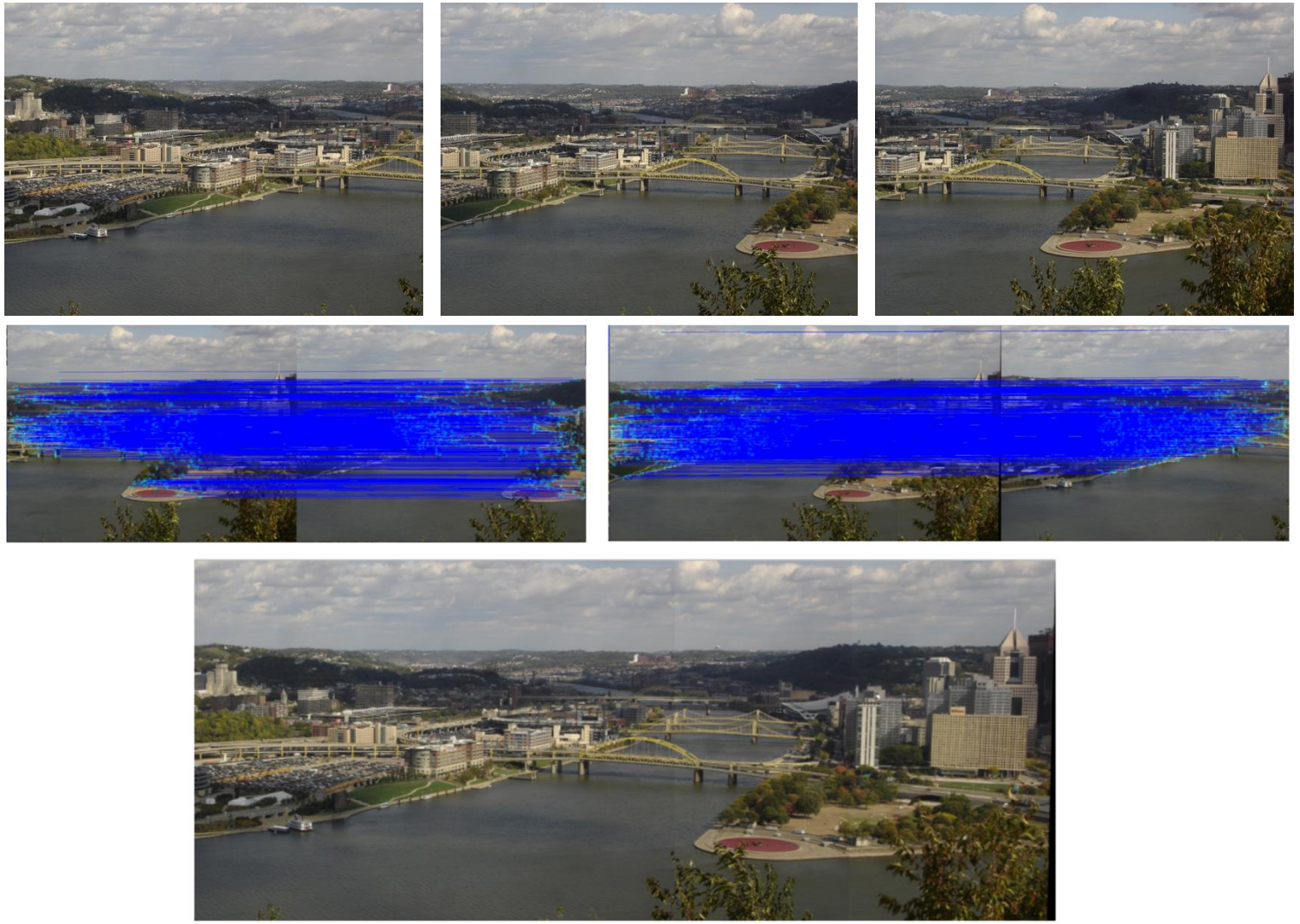
# Intermediate take away

# Image as a vector

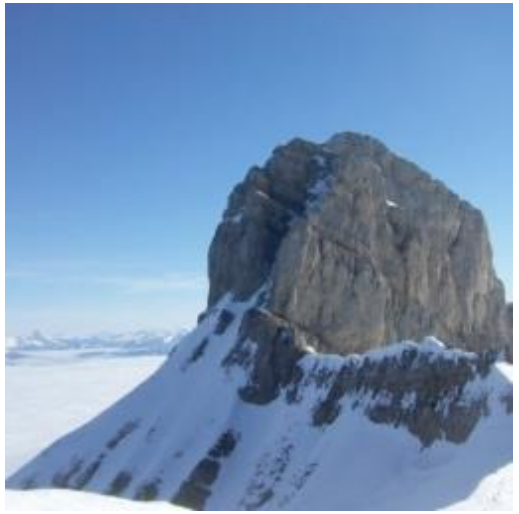
**What is the  advantage of local feature information?**



# Image feature example applications

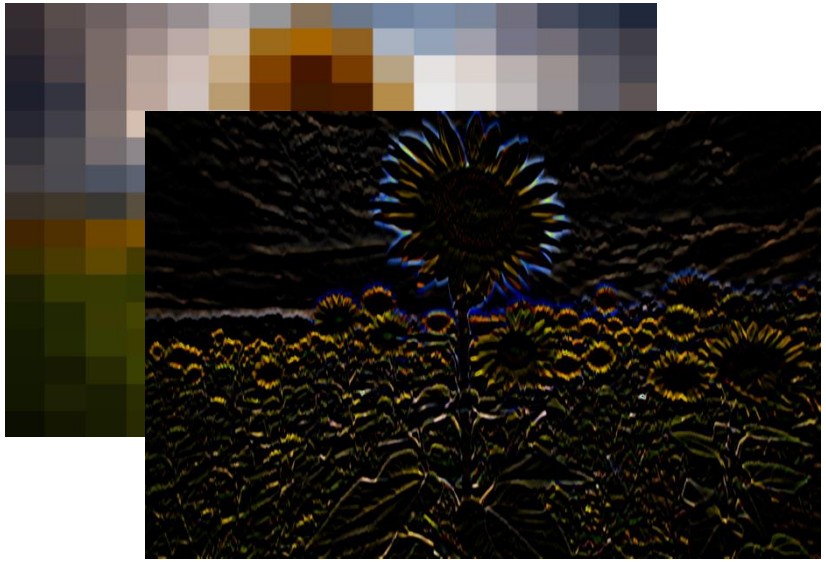


# Image feature example applications




# Image features

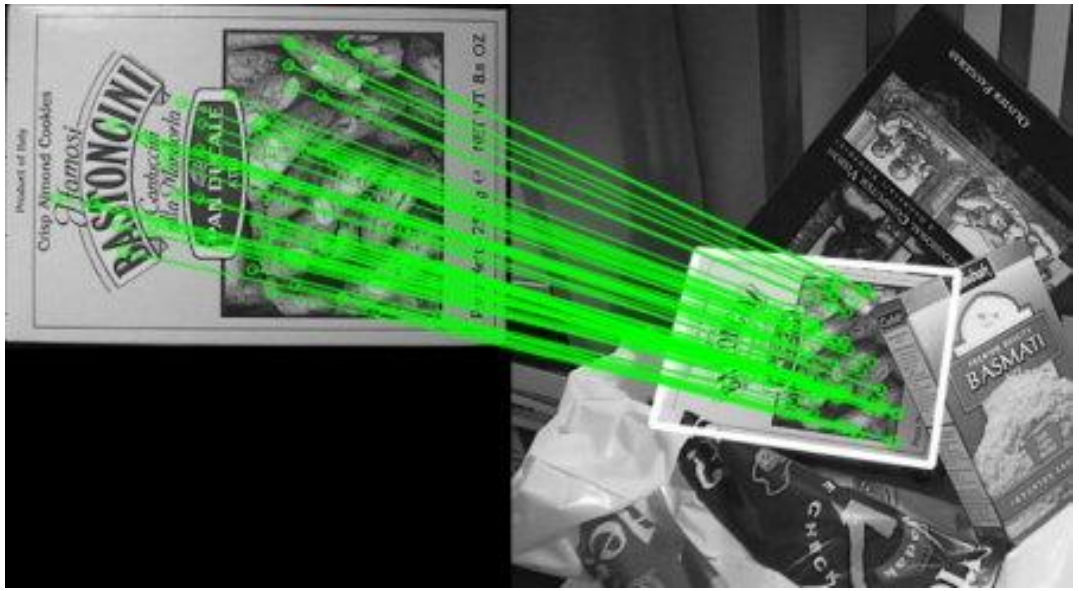
## Global Descriptor



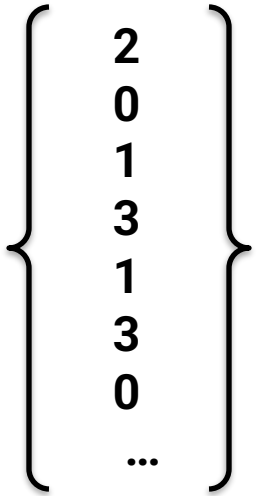
## Local Descriptor

# Feature Matching

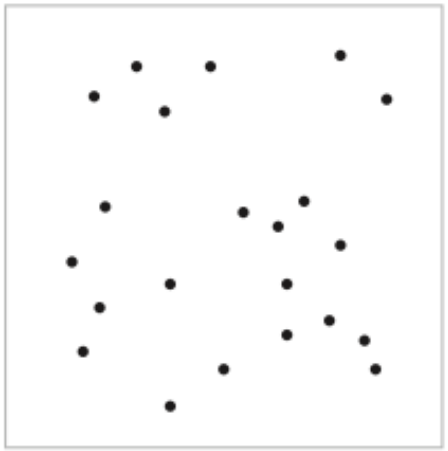
# Feature Matching



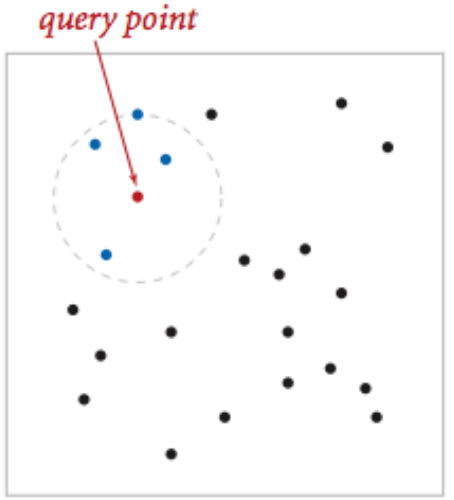

http://www.ubc.ca




Histogram



*set of points*




*4 nearest neighbors*

https://www.cs.princeton.edu

# Feature Matching

- **searching** for the **most similar matches to high-dimensional vectors**

  ← **nearest neighbor matching**

- **no known exact algorithms** for solving these high-dimensional problems that are faster than linear search

  ← **Curse of dimensionality**

- **efficient algorithm** for fast nearest neighbor (NN) matching in large data **sets can bring speed improvements of several orders of magnitude**

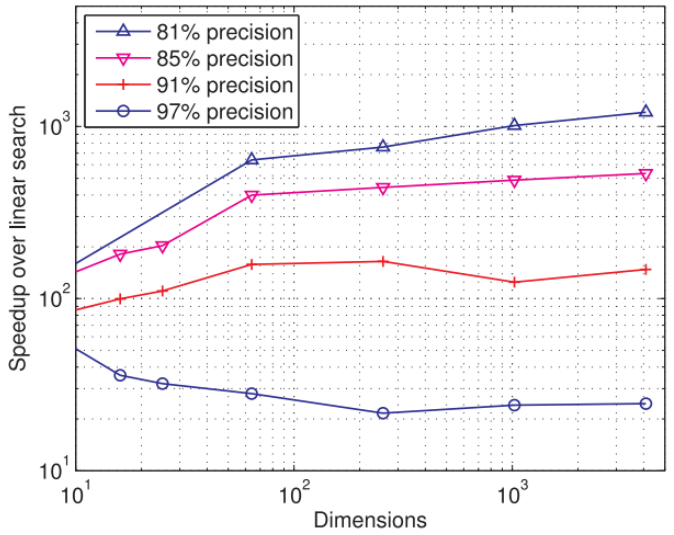
- **FLANN – Fast Library for Approximate Nearest Neighbor**
  - library for performing fast approximate nearest neighbor searches in high dimensional spaces

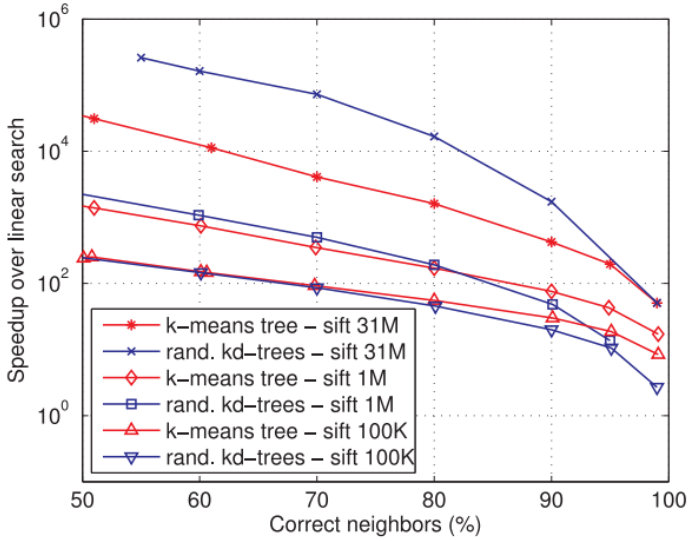
# Feature Matching

- Obtain **speed improvement by allowing an approximate search**
  - not all the neighbors returned are exact
  - some are approximate but still close to the exact neighbors

- **Approximate nearest neighbor** search algorithms provide **~95% correct neighbors**
  - **BUT**: are **two or more orders of magnitude faster** than linear search

http://www.ubc.ca



# Feature Matching Applications

- Nearest neighbor search problem is also of major importance for many **applications**:
  - machine learning
  - image- / document-retrieval
  - data compression
  - bio-informatics
  - data analysis
  - computer vision

# Feature Matching

- Given a set of points in metric space *M* and a query point *q*

$$P = \{p_1, p_2, p_3, ..., p_n\} \qquad q \in M$$

- Find elements that is closest to q with respect to a metric distance *d*

$$NN(q, P) = argmin_{x \in P}\ d(q, x)$$

# Feature Matching

- **How to compare feature vectors in high-dimensional space?**
  - Distance Metric

- How to compare feature vectors in high-dimensional space **efficiently**?
  - Spatial Datastructure
  - Approximate Nearest Neighbor

# Distance Metric
# (Vector Space Norm)

1. $d \geq 0$
2. $d(x, y) = d(y, x)$
3. $d(x, y) \leq d(x, y) + d(y, z)$
4. $d(x, y) = 0 \rightarrow x = y$

# Distance metric

- **Euclidean distance**
  - ‘straight-line’ distance between points ($L^2$-norm)

$$d_{l2}(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

- **Manhatthan distance**
  - L1-norm

$$d_{l1}(p,q) = \sum_{i=1}^{n}|p_i - q_i|$$

- **Maximum norm**
  - $L^\infty$

$$d_{l\infty}(p,q) = \max_i(|p_i - q_i|)$$

# Distance metric

- **Cosine similarity (strictly speaking – this is not a distance metric)**
  - similarity between two vectors (inner product, cos(angle) between vectors)
  - resulting similarity ranges from −1 meaning opposite, to 1 meaning the same, with 0 indicating orthogonality
  - in-between values indicate intermediate similarity or dissimilarity
  - distance between histograms

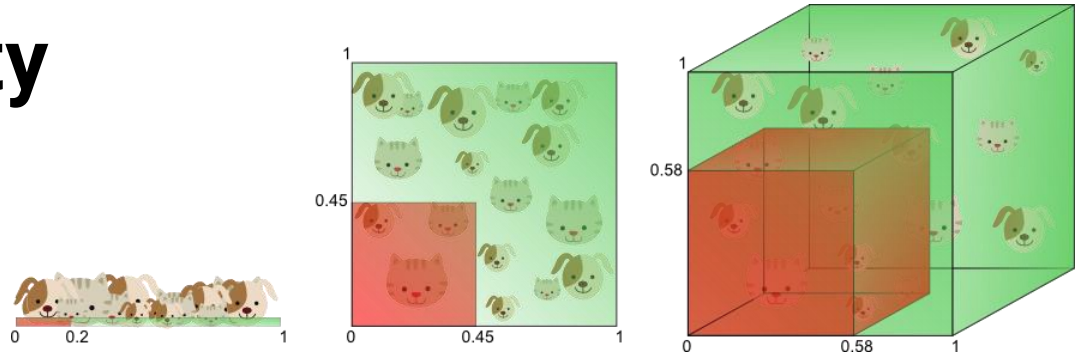
$$a \cdot b = \|a\| \|b\| \cos\theta$$

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

**Something *not* completely unrelated…**

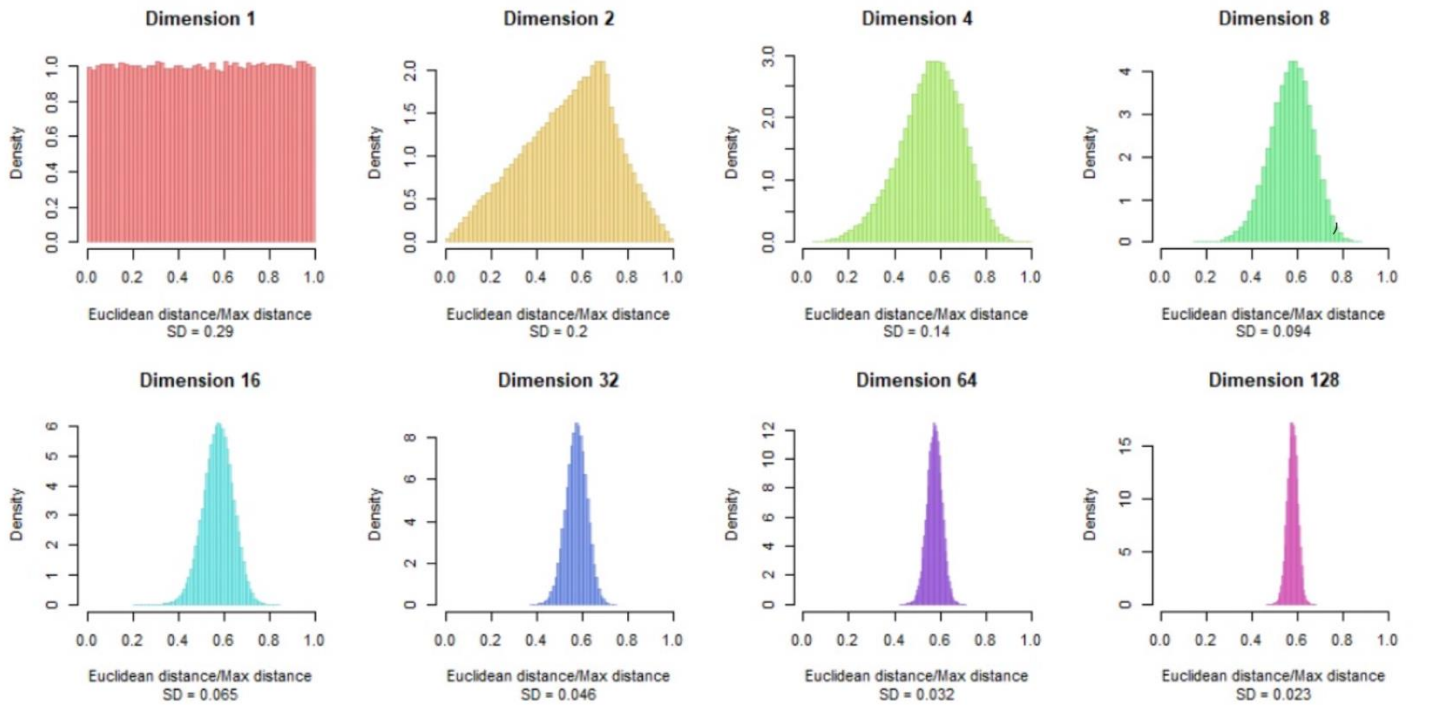
**Curse of …**


**Dimensionality**



# Curse of dimensionality

- Volume grows rapidly with increase of dimensions
- With increasing dimensions the difference in distances decreases




$$\lim_{d\to\infty} E\left(\frac{\mathrm{dist}_{\max}(d) - \mathrm{dist}_{\min}(d)}{\mathrm{dist}_{\min}(d)}\right) \to 0.$$

# Feature Matching

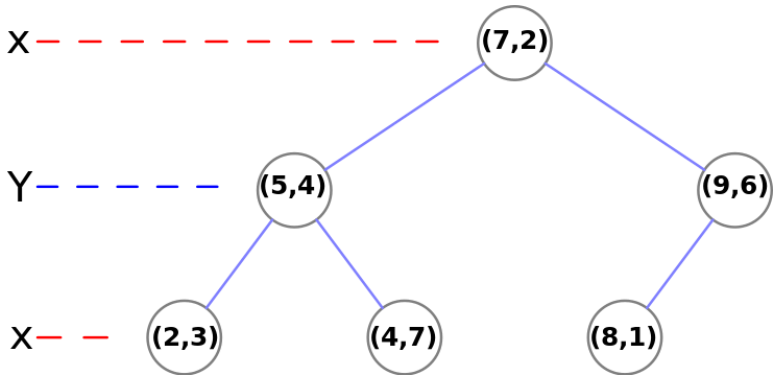
- How to compare feature vectors in high-dimensional space?
  - Distance Metric

- **How to compare feature vectors in high-dimensional space efficiently?**
  - Spatial Datastructure
  - Approximate Nearest Neighbor

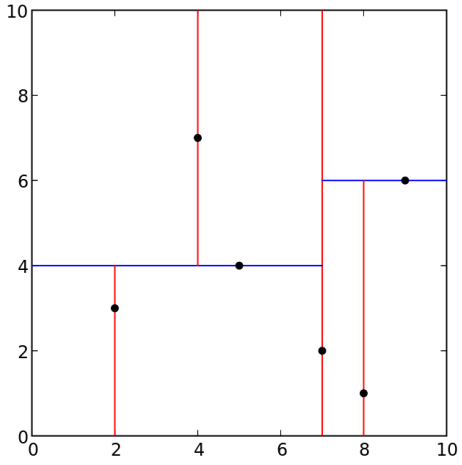
# Multiple Randomized kd-Trees

# kd-tree (Recap)

- **spatial data structure** for organizing points in **high dimensionality spaces**
- **binary tree** where each node is a k-d point
- every **node** can be seen **as a plane dividing space in two parts**
- splitting along changing dimensions / axis
- **results in balanced k-d tree**

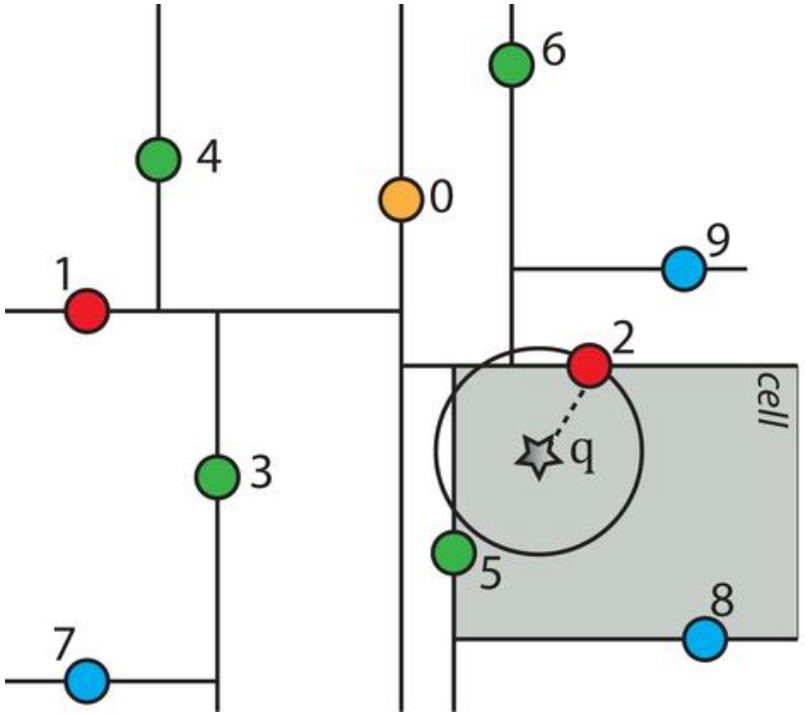



white board example

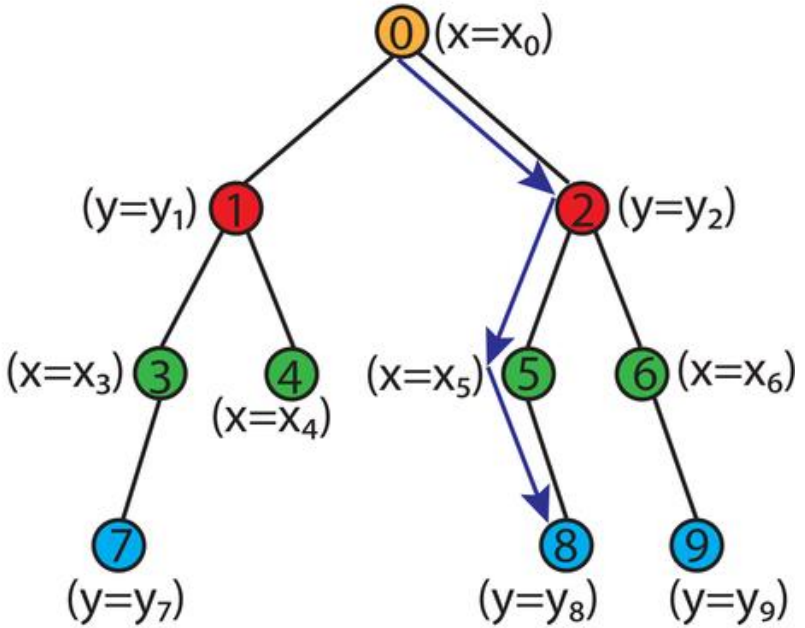



http://wikipedia.org

# Query in kd-trees




http://www.mdpi.com/



# kd-tree

- Very **effective in low dimensional** spaces
- Performance **decreases for high dimensional data**
  - requires searching a large number of nodes

- Speedup by approximating queries:
  - **‘error bound’** approximate search
  - **‘time bound’** approximate search

# Randomized kd-Trees

- **Idea:** multiple randomized k-d trees which are searched in parallel
  - build scheme similar to classic kd-tree

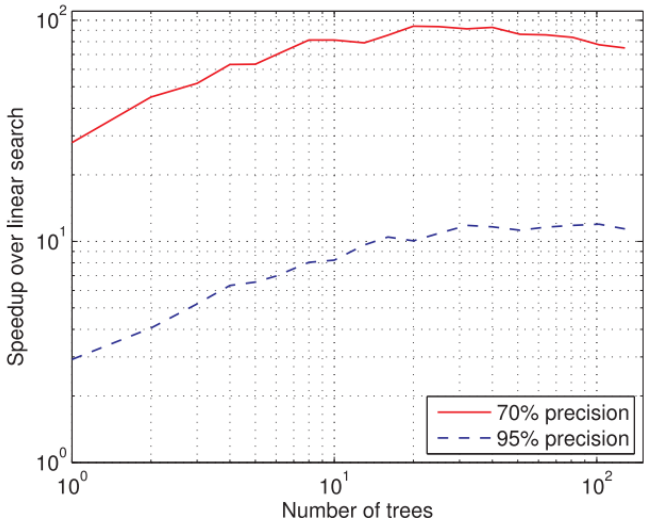
- **Difference to classic kd-tree:**
  - classic – split dimensions are chosen by highest variance
  - randomized – split dimensions are chosen randomly from the **top 5 dimensions highest variance**

- In practice often **20 trees** used

# Searching randomized k-d forest

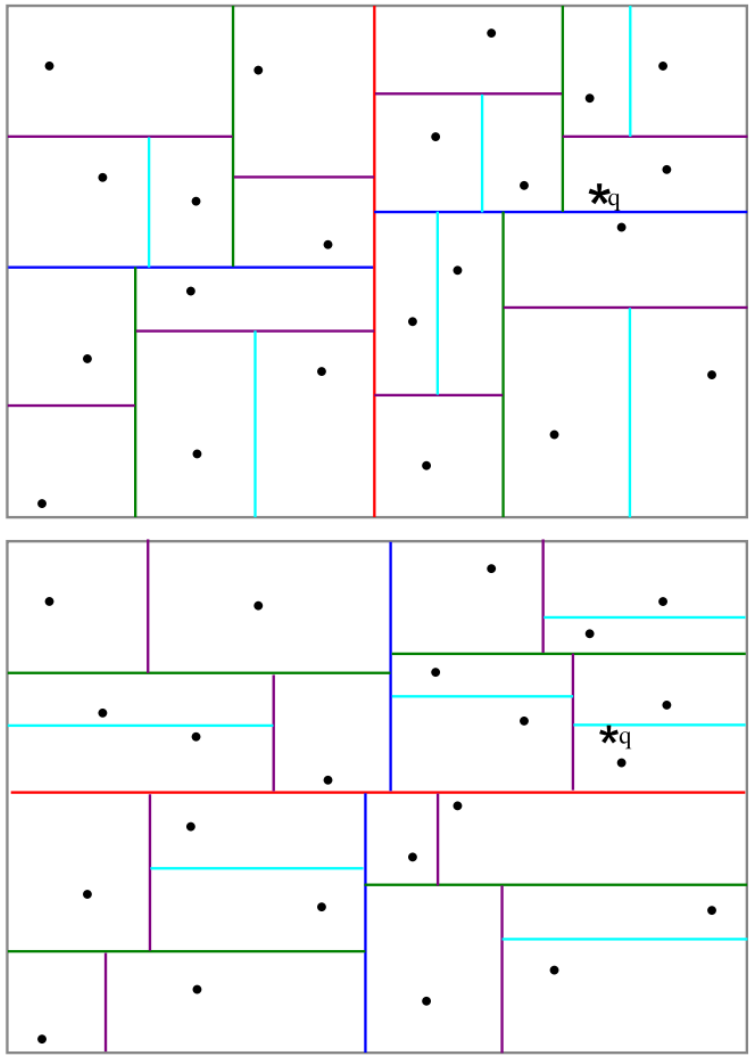
- Maintain **single priority queue across forest**
- PQ is ordered by increasing distance
  - the search will explore first the closest leaves from all the trees
- examined **points are marked as visited across all trees** (not visited in another tree again)
- approximation is determined by **maximum number of visited leaf nodes**

http://www.ubc.ca




100k SIFT feature data set

# Randomized kd-Trees

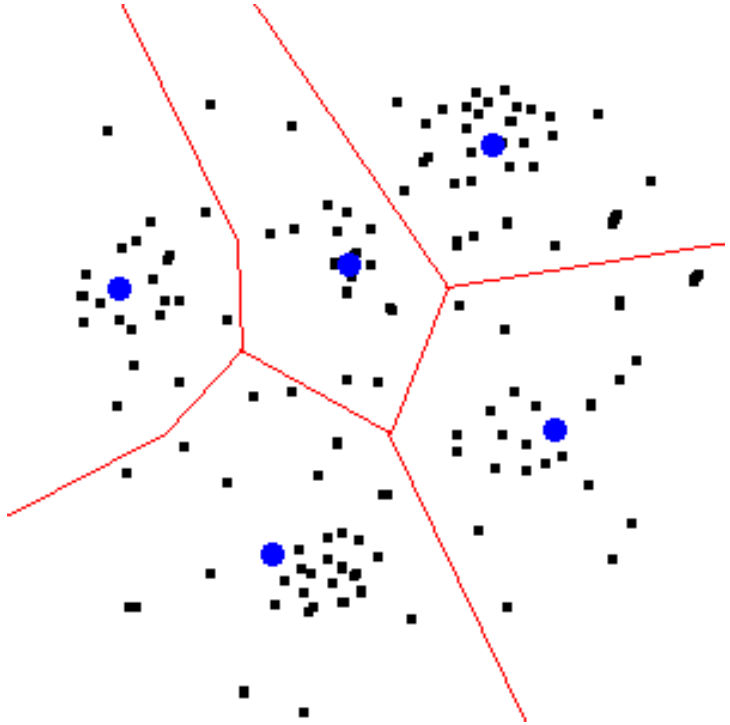


# Priority Search
# k–means Tree

# k-means

- important **clustering method** in CS
- partitions ***N* data points** in ***k* clusters *S* = *{$S_1$, $S_2$, … $S_k$}***
- each **data point belongs to cluster with nearest mean**
  - mean of the cluster is cluster center/ representative

$$\arg\min_{s} \sum_{i=1}^{k} \sum_{x \in s_i} \|x - \mu_i\|^2$$



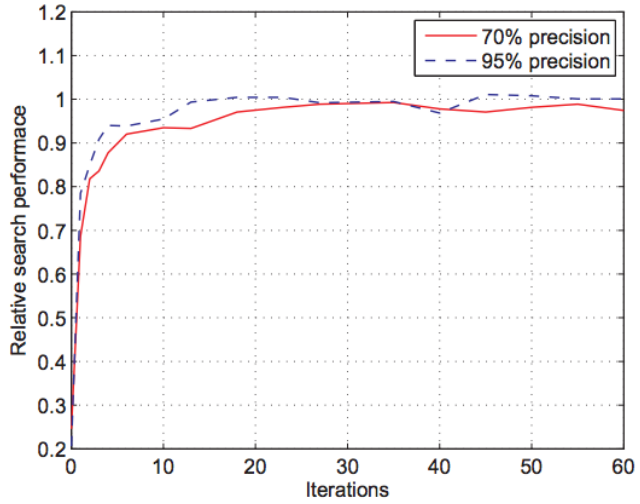
http://mnemstudio.org/

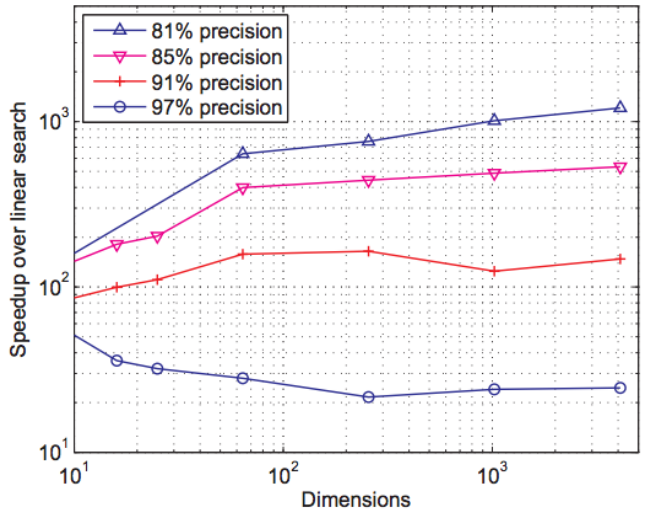
- **results in partitioning of data space ( voronoi cells )**

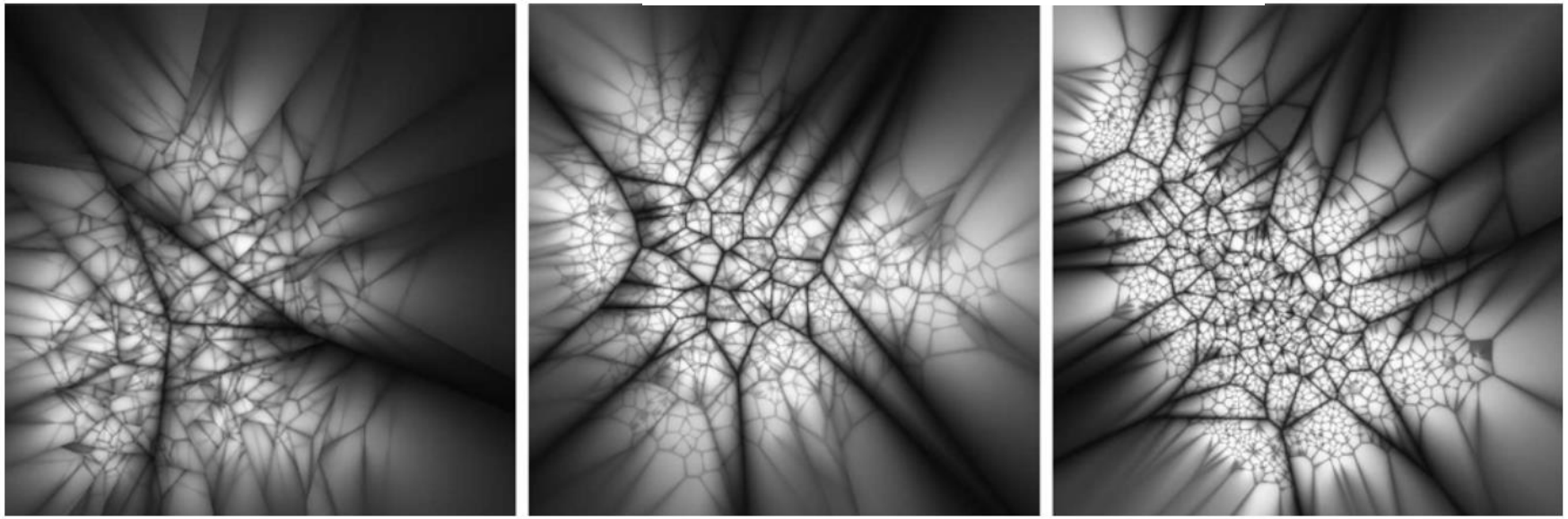
# k-means Trees

- **Contruction:**
  - Tree is constructed by **partitioning the data points** at each **level into *K* distinct regions** using k-means clustering
    - Apply recursively to the points in each region
  - Recursion is stopped when the number of points in a region is smaller than *K*
- **Search:**
  - initially traversing the tree from the root to the closest leaf (following closest cluster centre)
  - add unvisited branches to priority queue
  - use branch that has the closest center to the query point
  - restart the tree traversal from that branch
  - stop after number of visited leaf nodes

# k-means Trees

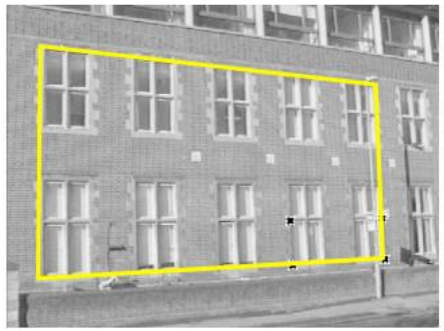


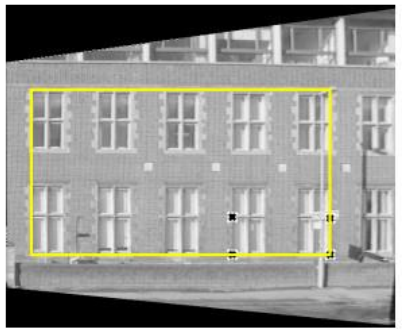



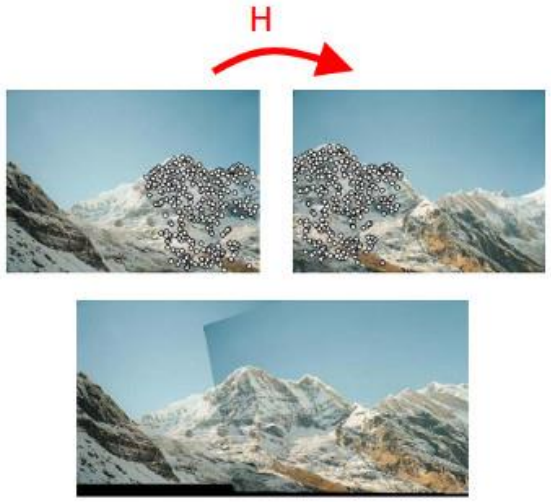
# Feature Matching Application: Homography

- Camera calibration
- 3D reconstruction
- Stereo vision
- Scene understanding

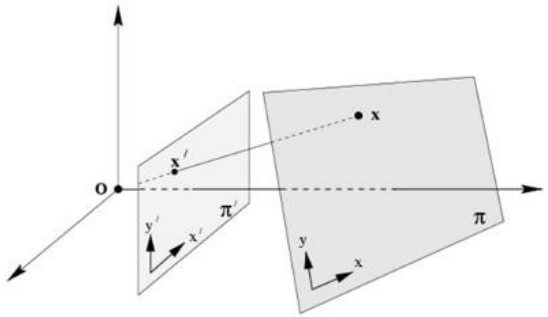


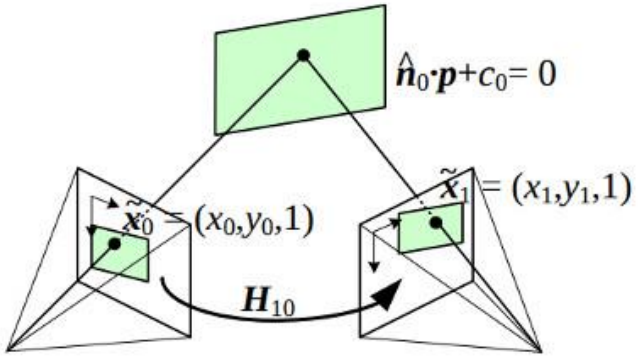


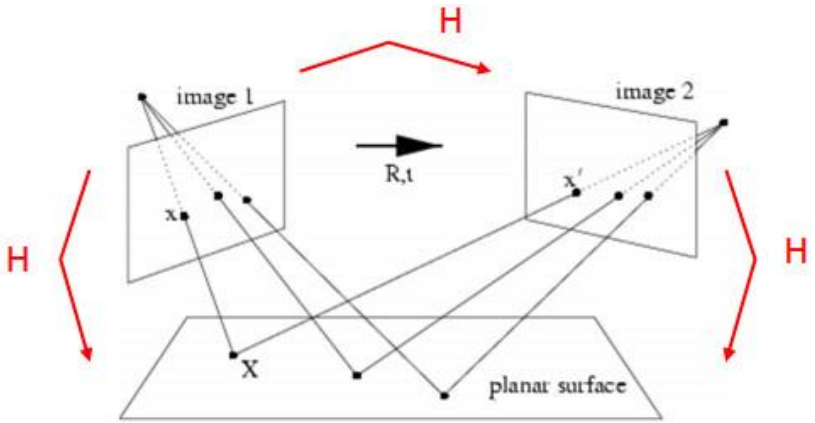
from Hartley & Zisserman



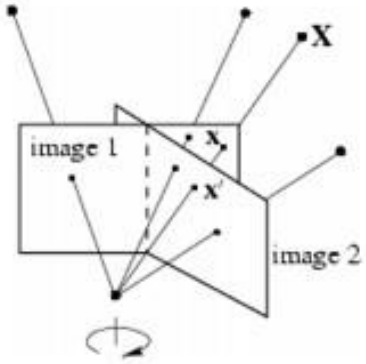
# Homography

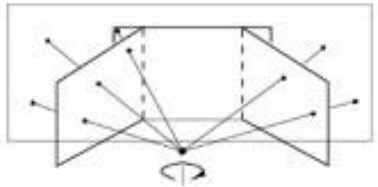



Rotating camera, arbitrary world



# Homography matrix

$$\begin{bmatrix} \mathrm{w}x_i' \\ \mathrm{w}y_i' \\ \mathrm{w} \end{bmatrix} \cong \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$x_i' = \frac{h_{00}x_i + h_{01}y_i + h_{02}}{h_{20}x_i + h_{21}y_i + h_{22}}$$

$$y_i' = \frac{h_{10}x_i + h_{11}y_i + h_{12}}{h_{20}x_i + h_{21}y_i + h_{22}}$$

$$\begin{aligned} x_i'(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{00}x_i + h_{01}y_i + h_{02} \\ y_i'(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{10}x_i + h_{11}y_i + h_{12} \end{aligned}$$

# Homography matrix

**Lets do this in more detail and code**

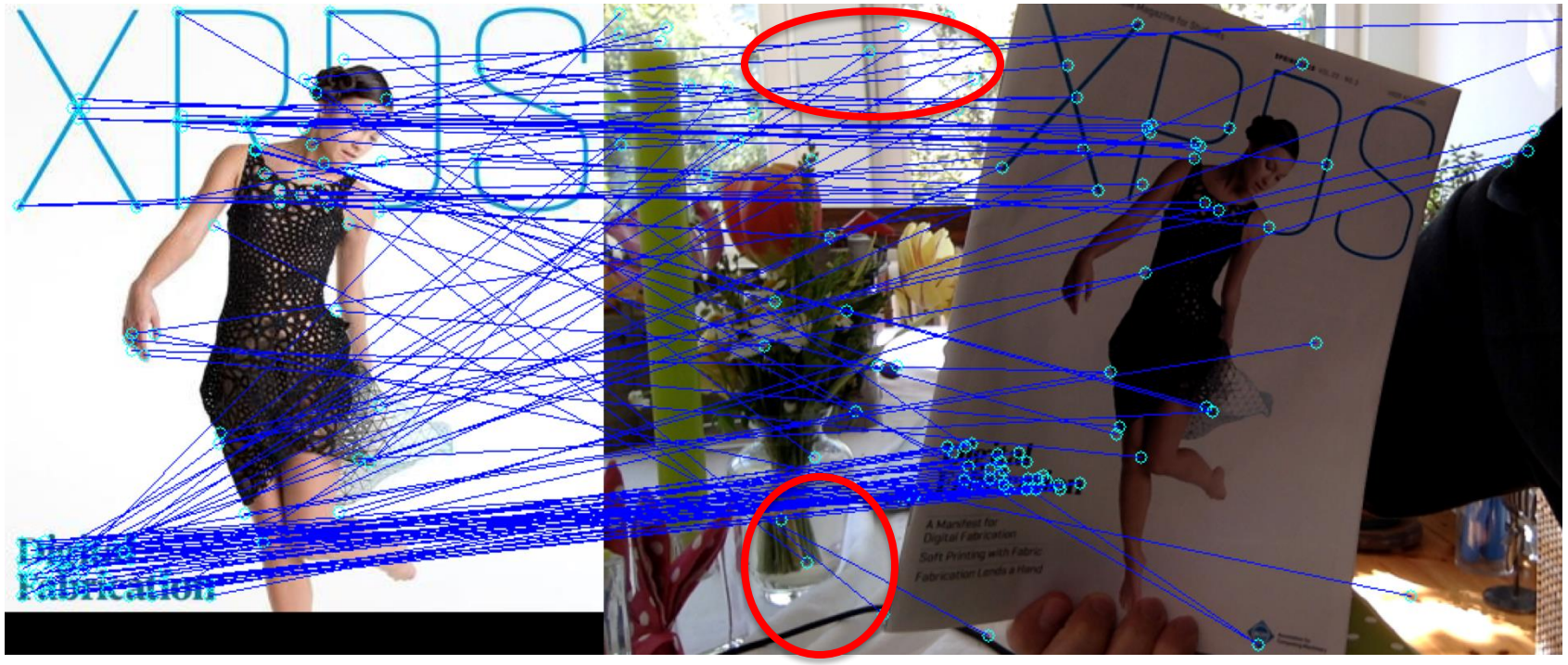
$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 & -y'_1 \\ & & & & & \vdots & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x'_nx_n & -x'_ny_n & -x'_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y'_nx_n & -y'_ny_n & -y'_n \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

| **A** | **h** | **0** |
|---|---|---|
| 2N x 9 | 9 | 2N |

- Defines a least squares problem: $min||Ah - 0||^2$
  - Solution: $\hat{h}$ = eigenvector of $A^TA$ with smallest eigenvalue
  - Works with 4 or more points

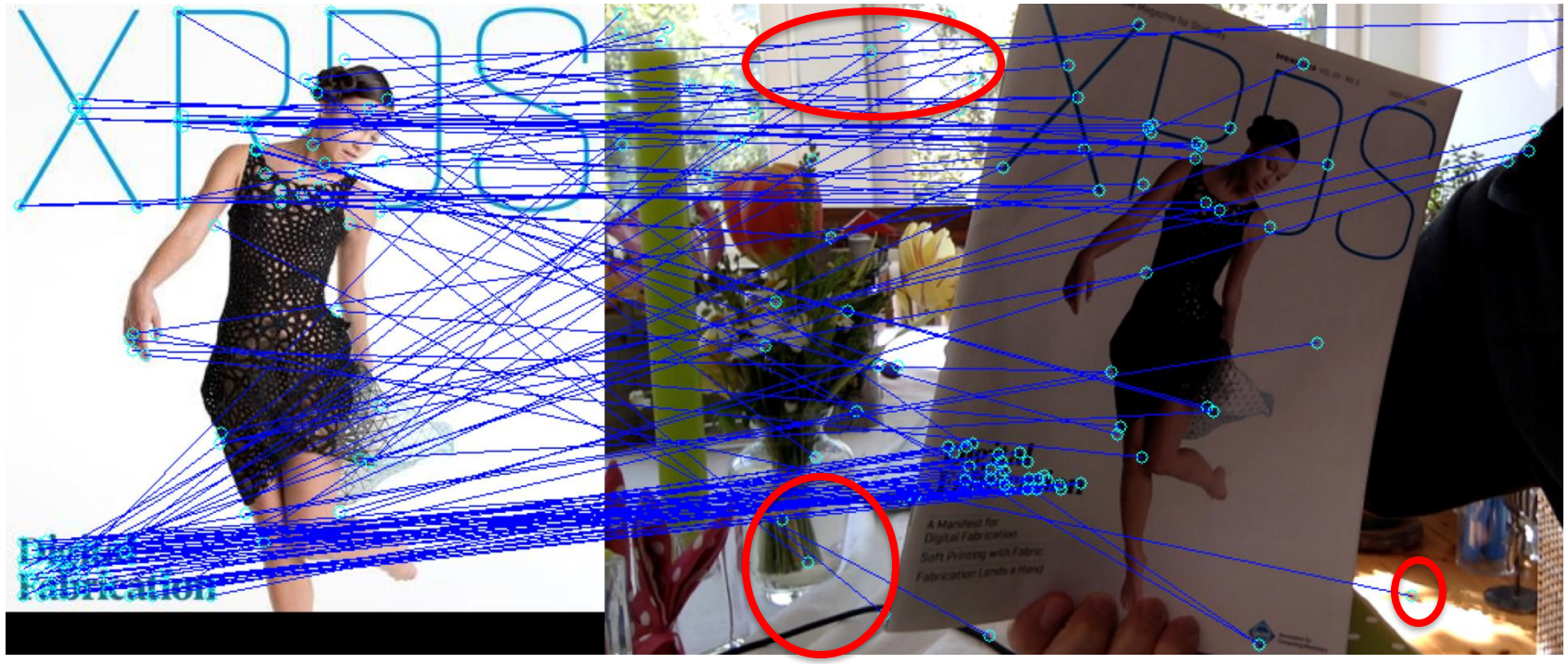
```
H, status = cv2.findHomography(pts_prevFrame, pts_curFrame, cv2.RANSAC, 3.0)
quad = cv2.perspectiveTransform(quad.reshape(1, -1, 2), H).reshape(-1, 2)
```

# RANSAC Motivation



# RANSAC

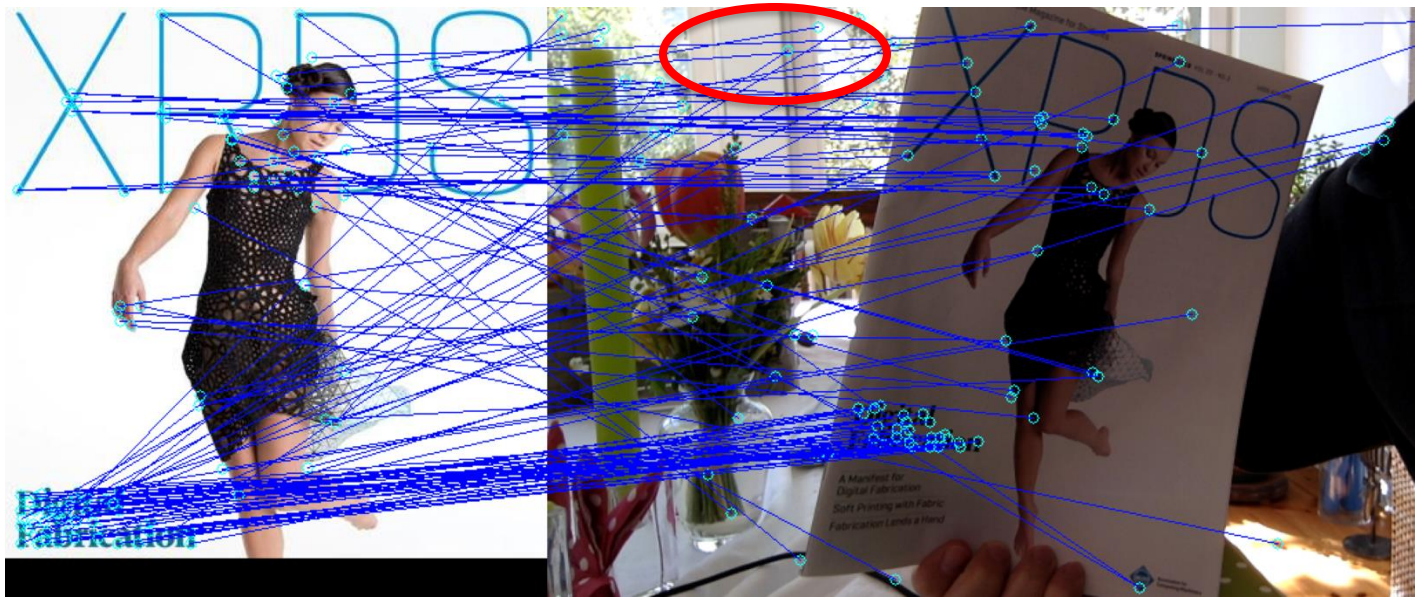
- What do we do about bad matches?




- Note: at this point we don’t know which ones are good/bad

# RANSAC

- Idea:
  - Select one match of at least 4 points and compute homography

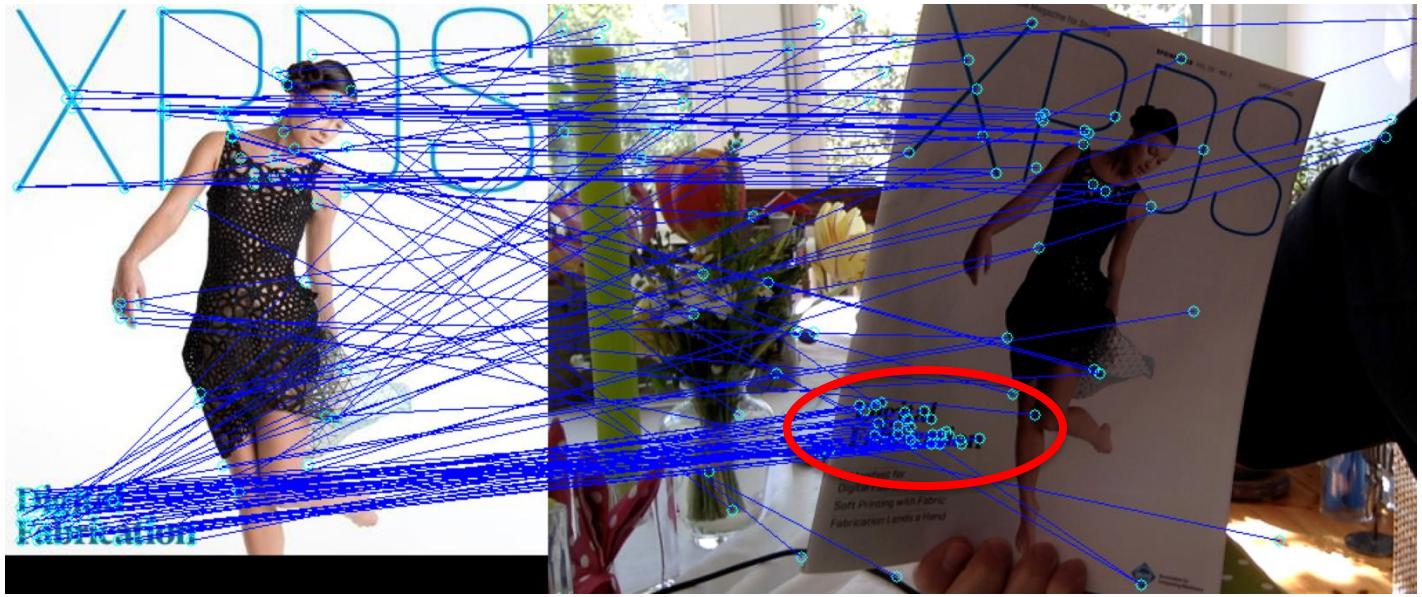



0 inliers

  - Count inliers using the homography matrix

# RANSAC

- Idea:
  - Select one match and count inliers



many inliers

  - **Keep match with largest set of inliers**
  - Find “average” transformation (homography H), but with only inliers