

Reconnaissance d'objets et vision artificielle 2009

Visual search and object recognition using local invariant features

Josef Sivic

<http://www.di.ens.fr/~josef>

Equipe-projet WILLOW, ENS/INRIA/CNRS UMR 8548

Laboratoire d'Informatique, Ecole Normale Supérieure, Paris

With slides from: O. Chum, K. Grauman, B. Leibe, D. Lowe, S. Lazebnik, J. Philbin, D. Nister, N. Snavely, A. Zisserman

The goal of this lecture

Understand how a large scale visual search system works
“local invariant regions in action”

Understand and see in practice some useful techniques

- Approximate nearest neighbour search
- Large scale indexing using inverted files

Other applications using local invariant regions

Problem specification: particular object matching

Example I: Visual search in feature films

Visually defined query

“Groundhog Day” [Rammis, 1993]

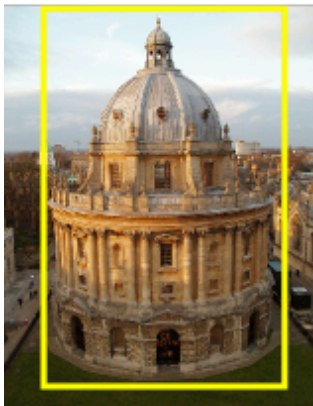
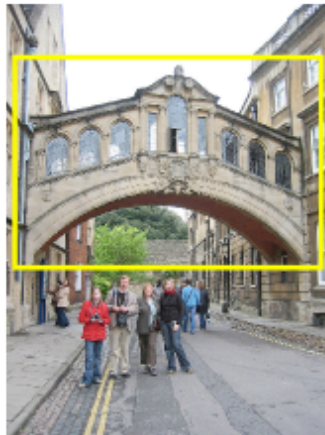
“Find this clock”



“Find this place”



Example II: Search photos on the web for particular places



Find these landmarks

...in these images and 1M more

Review: Why is it difficult?

Want to find the object despite possibly large changes in scale, viewpoint, lighting and partial occlusion



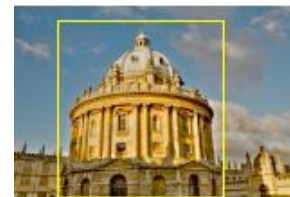
Scale



Viewpoint



Lighting



Occlusion

The need for visual search



Flickr: has 2 billion photographs, more than 1 million added daily

Facebook: has 15 billion images (~27 million added daily)

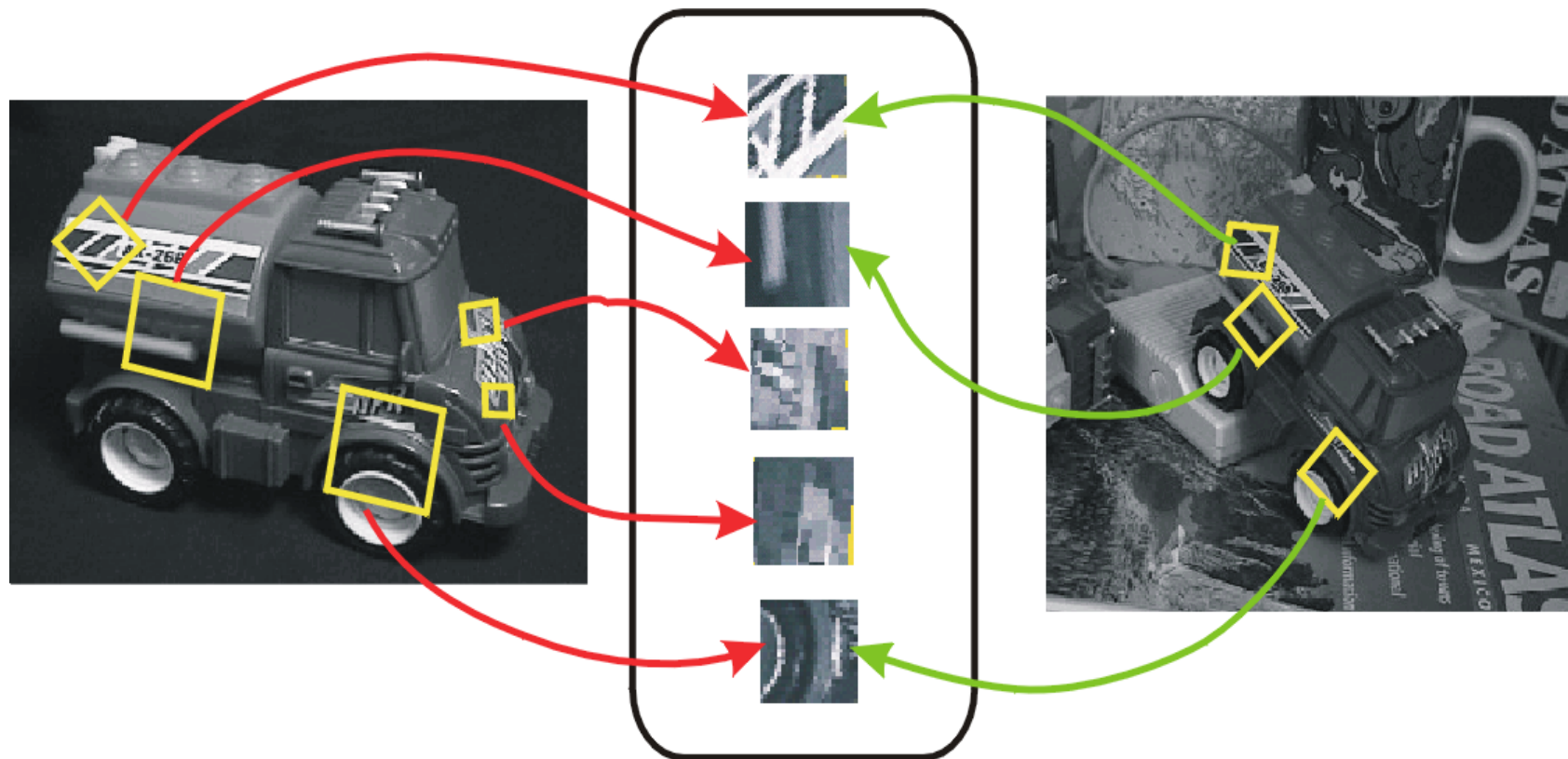
Company collections

Personal collections: 10000s of digital camera photos and mpegs

Vast majority will have minimal, if any, textual annotation. Yet text is the only common way of searching / accessing documents (e.g. Google / Live search)

Review: Image representation

- Image content is transformed into local features that are invariant to geometric and photometric transformations



Slide credit: David Lowe

Review: Affine covariant regions



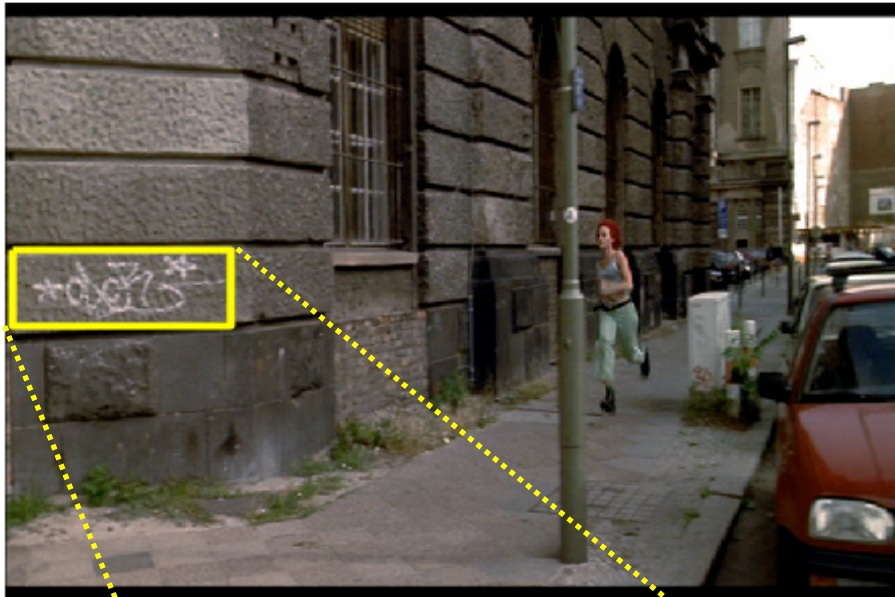
1000+ regions per image



- a region's size and shape are **not** fixed, but
- automatically adapts to the image intensity to cover the same physical surface
- i.e. pre-image is the same surface region

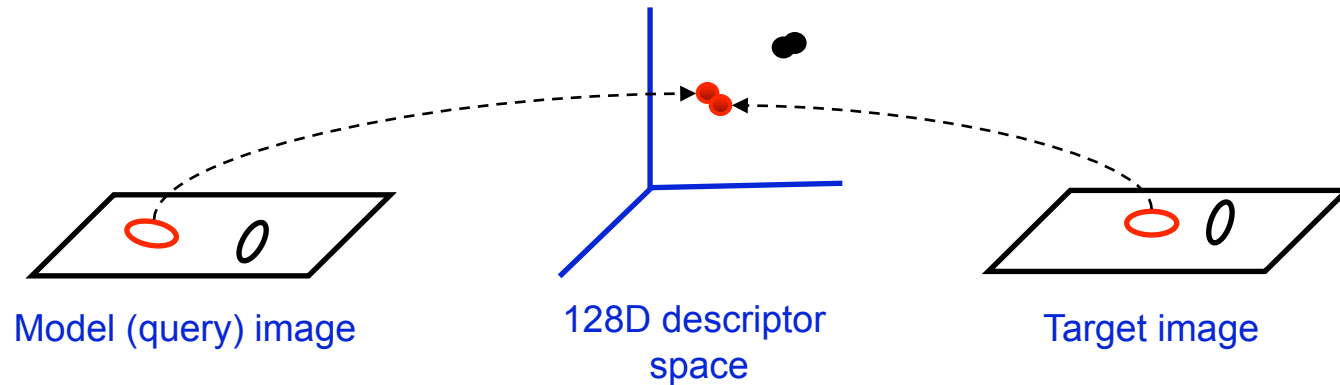
Represent each region by the 128-dimensional SIFT descriptor vector

Example I: Two images - "Where is the Graffiti?"



Review: Object recognition / matching

Establish correspondences between object model image and target image by nearest neighbour matching on SIFT vectors



Solve following problem for all feature vectors, $\mathbf{x}_j \in \mathcal{R}^{128}$, in the query image:

$$\forall j \text{ NN}(j) = \arg \min_i \|\mathbf{x}_i - \mathbf{x}_j\|$$

where, $\mathbf{x}_i \in \mathcal{R}^{128}$, are features in the target image.

Review: problem with matching on descriptors alone



- too much individual invariance
- each region can affine deform independently (by different amounts)
- use semi-local and global spatial relations to verify matches, e.g.:
 - common affine transformation [Lowe '99] (strong requirement)
 - locally similar affine transformation [Ferrari '04]
 - spatial neighbours match spatial neighbours [Schmid '97]

See lectures 1 and 3 for more details.

Example I: Two images - “Where is the Graffiti?”

Initial matches

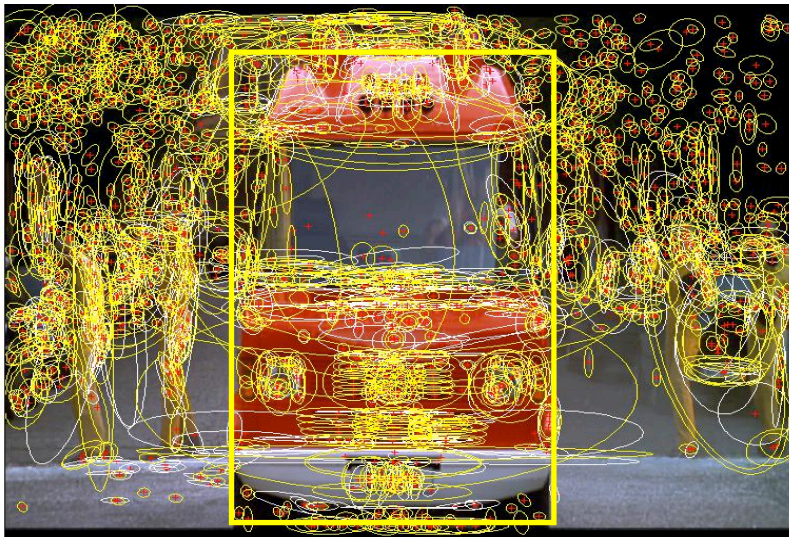
Nearest-neighbor search based on appearance descriptors alone.



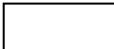
Spatial consistency required



Example II: Two images again

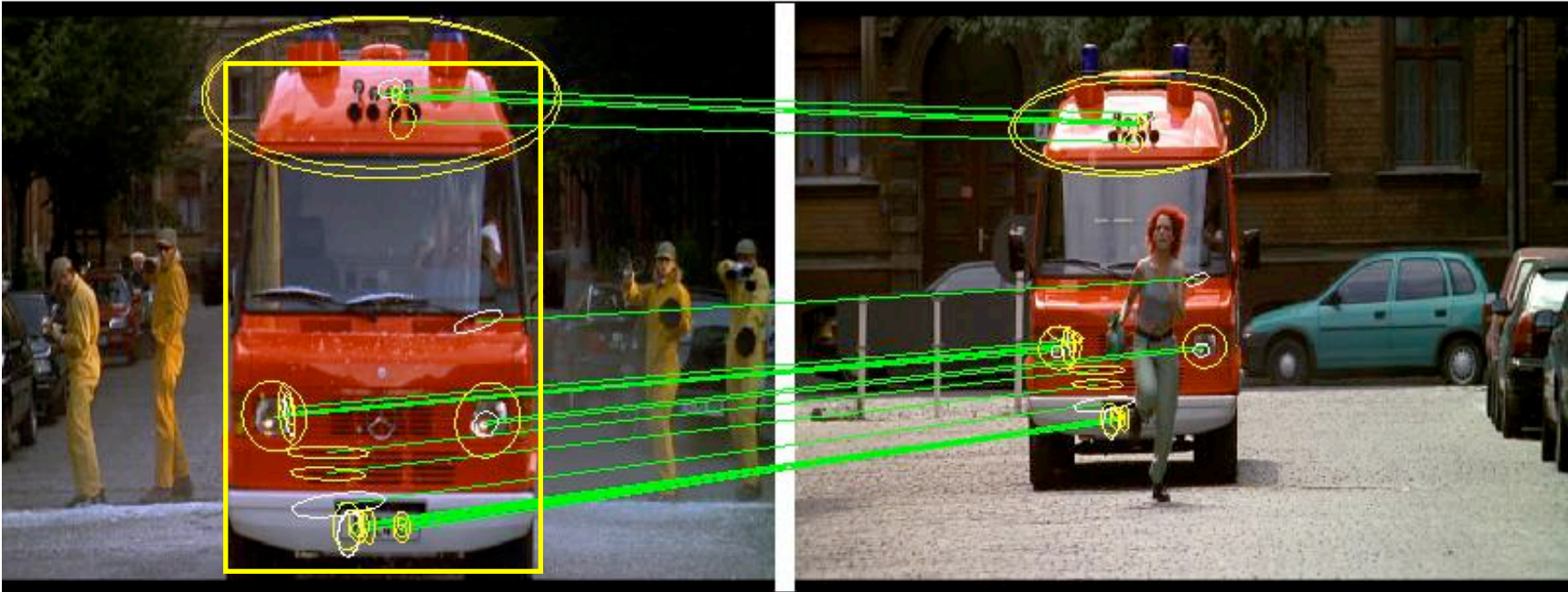


1000+ descriptors per frame

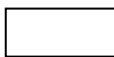

 Shape adapted regions

 Maximally stable regions

Match regions between frames using SIFT descriptors and spatial consistency



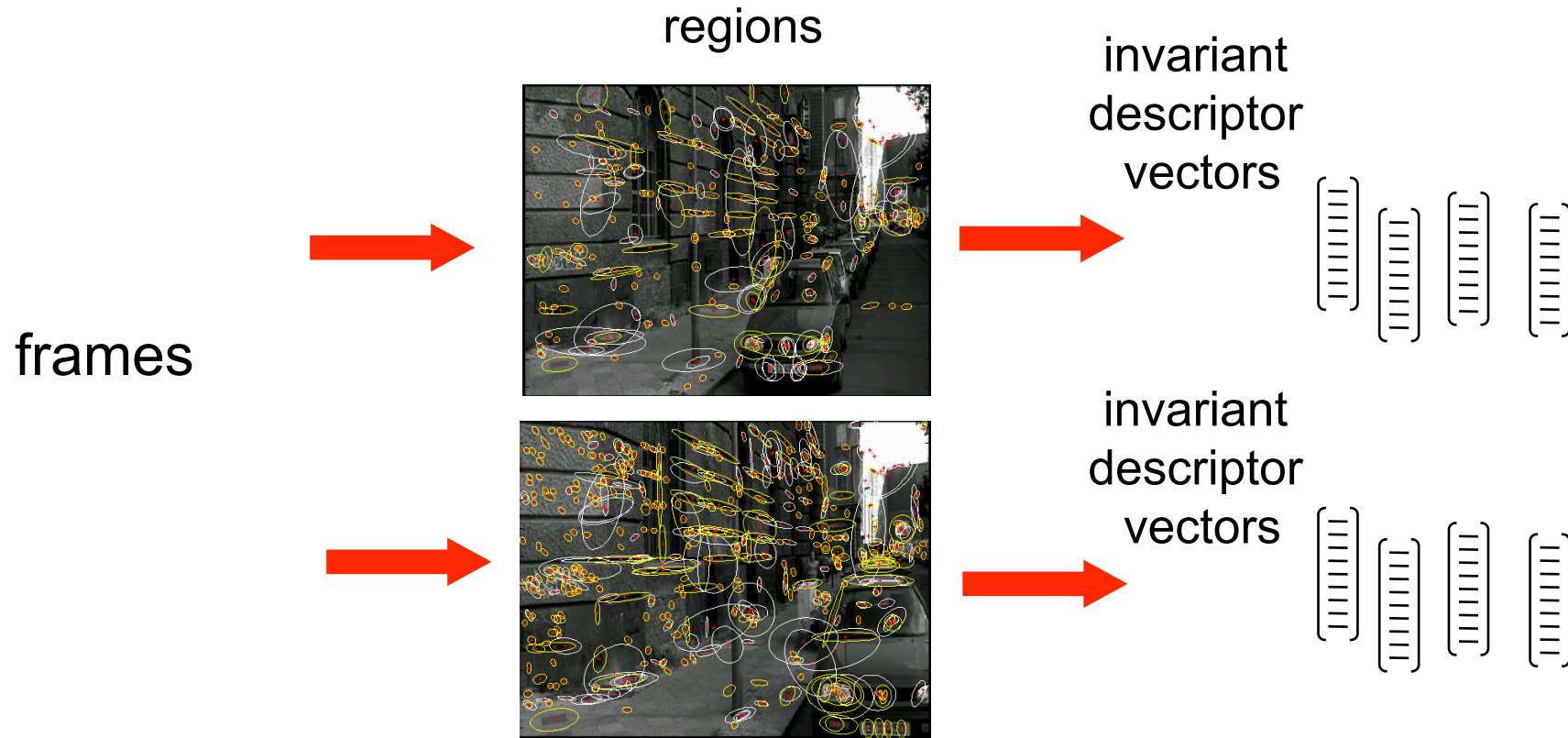
Multiple regions overcome problem of partial occlusion

-  Shape adapted regions
-  Maximally stable regions

What about multiple images?

- Two images work fine (for some types of objects, which ones?, we come back to this).
- How to generalize this strategy to multiple images with reasonable complexity?
 - 10, 10^2 , 10^3 , ..., **10^7** , ... 10^{10} images?

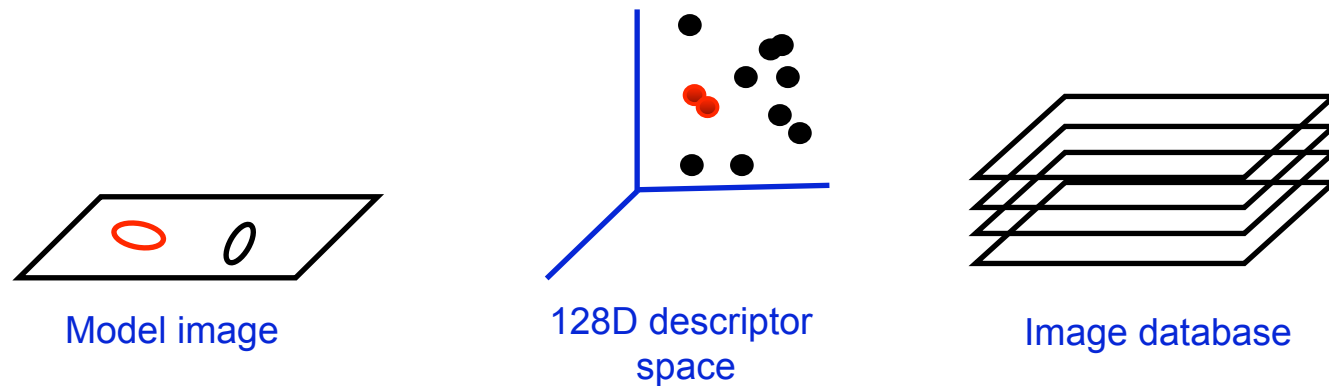
Strategy I: Combine descriptor vectors from all images



1. Compute affine covariant regions in each frame independently (Lecture 2)
2. “Label” each region by a vector of descriptors based on its intensity (Lecture 2)
3. Finding corresponding regions is transformed to **finding nearest neighbour vectors**
4. Rank retrieved frames by number of corresponding regions
5. Verify retrieved frame based on spatial consistency (Lecture 1+3)

Finding nearest neighbour vectors

Establish correspondences between object model image and images in the database by **nearest neighbour matching** on SIFT vectors



Solve following problem for all feature vectors, $\mathbf{x}_j \in \mathcal{R}^{128}$, in the query image:

$$\forall j \text{ NN}(j) = \arg \min_i \|\mathbf{x}_i - \mathbf{x}_j\|$$

where, $\mathbf{x}_i \in \mathcal{R}^{128}$, are features from all the database images.

Quick look at the complexity of the NN-search

N ... images

M ... regions per image (~1000)

D ... dimension of the descriptor (~128)

Exhaustive linear search: $O(M \cdot N \cdot D)$

Example:

- Matching two images ($N=1$), each having 1000 SIFT descriptors
Nearest neighbors search: 0.4 s (2 GHz CPU, implem. in C)
- Memory footprint: $1000 \cdot 128 = 128\text{kB}$ / image

N = 1,000 ... ~7min (~100MB)

N = 10,000 ... ~1h7min (~ 1GB)

...

N = 10^7 ... ~115 days (~ 1TB)

...

All images on Facebook:

N = 10^{10} ... ~300 years (~ 1PB)

History of “large scale” visual search with local regions

Schmid and Mohr '97	– 1k images
Sivic and Zisserman'03	– 5k images
Nister and Stewenius'06	– 50k images (1M)
Philbin et al.'07	– 100k images
Chum et al.'07 + Jegou et al.'07	– 1M images
Chum et al.'08	– 5M images
Jegou et al. '09	– 10M images

All on a single machine in ~ 1 second!

Indexing local features

With potentially thousands of features per image, and hundreds of millions of images to search, how to efficiently find those that are relevant to a new image?

- Low-dimensional descriptors : can use standard efficient data structures for nearest neighbor search
- High-dimensional descriptors: approximate nearest neighbor search methods more practical
- Inverted file indexing schemes

Nearest-neighbor matching

Solve following problem for all feature vectors, \mathbf{x}_j , in the query image:

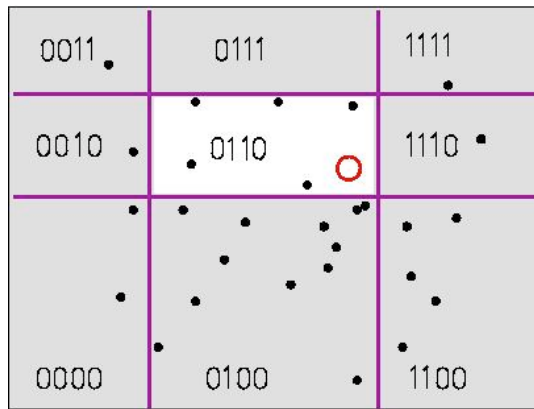
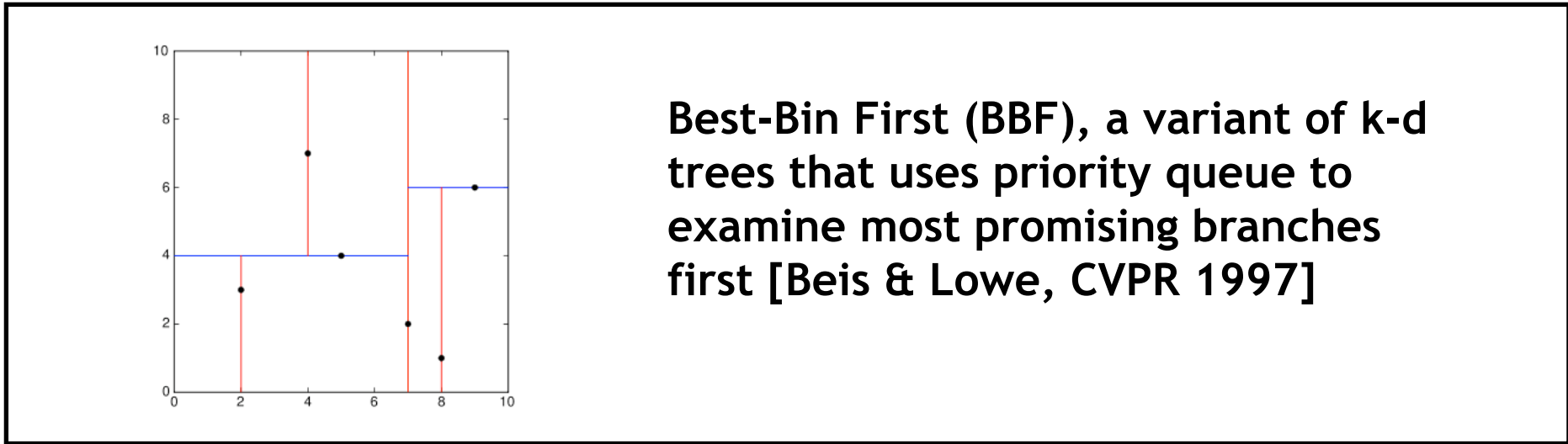
$$\forall j \text{ } NN(j) = \arg \min_i ||\mathbf{x}_i - \mathbf{x}_j||$$

where \mathbf{x}_i are features in database images.

Nearest-neighbour matching is the major computational bottleneck

- Linear search performs dn operations for n features in the database and d dimensions
- No exact methods are faster than linear search for $d > 10$
- Approximate methods can be much faster, but at the cost of missing some correct matches. Failure rate gets worse for large datasets.

Indexing local features: approximate nearest neighbor search



(3)

(4)

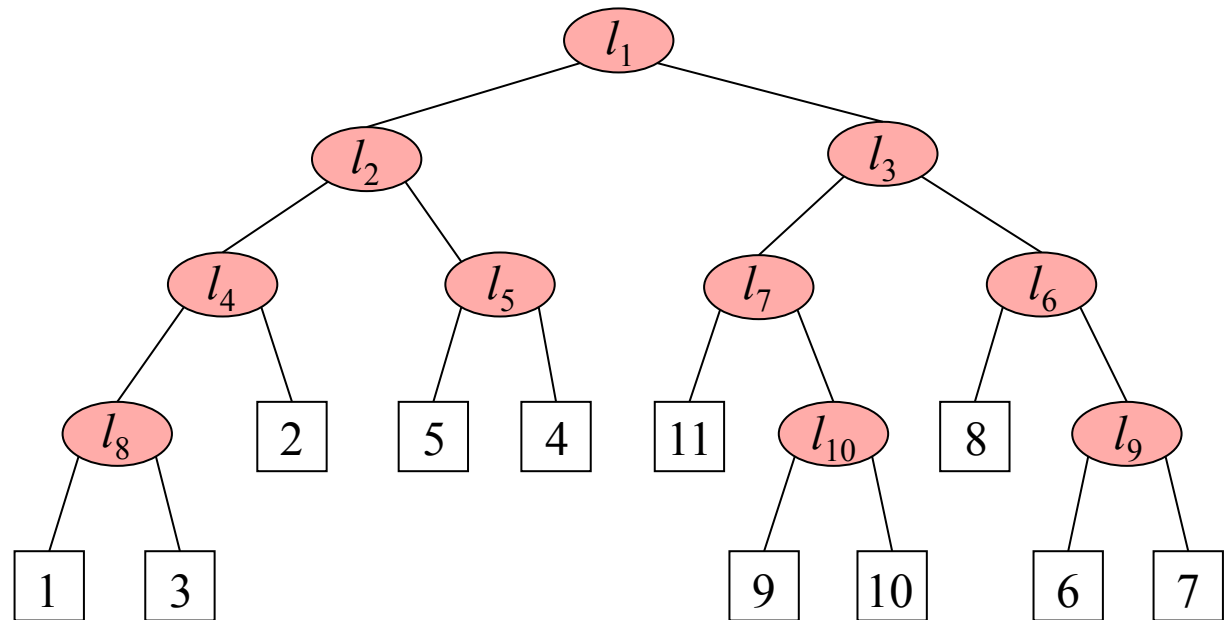
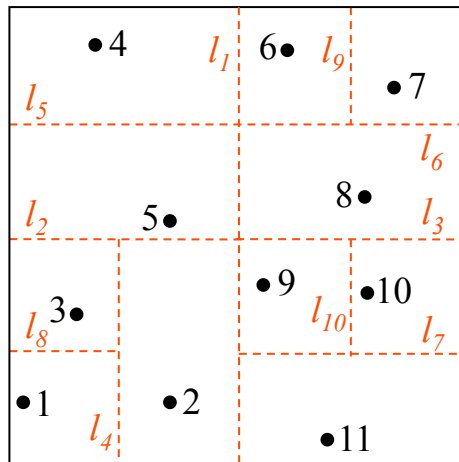
(1)

(2)

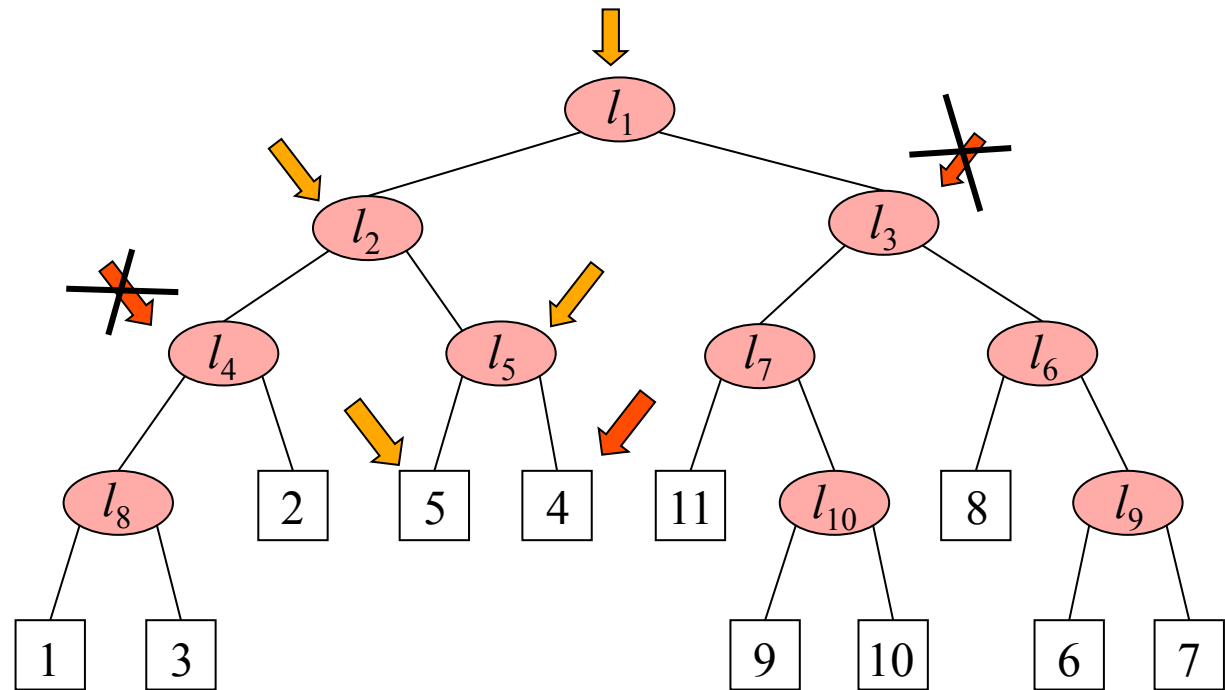
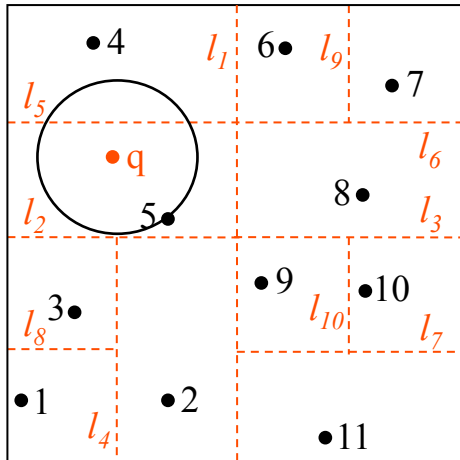
Locality-Sensitive Hashing (LSH), a randomized hashing technique using hash functions that map similar points to the same bin, with high probability [Indyk & Motwani, 1998]

K-d tree construction

Simple 2D example



K-d tree query



K-d tree: Backtracking

Backtracking is necessary as the true nearest neighbor may not lie in the query cell.

But in some cases, almost all cells need to be inspected.

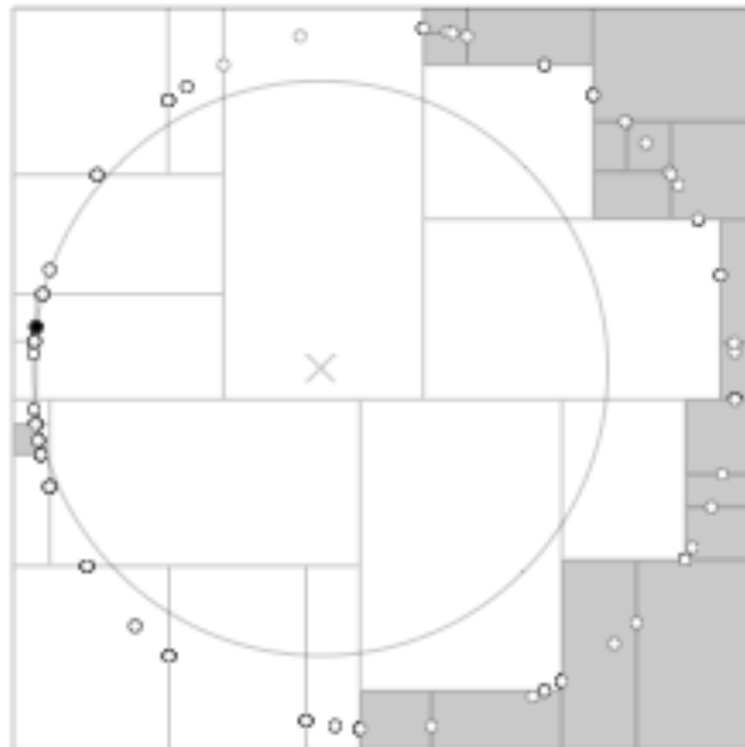


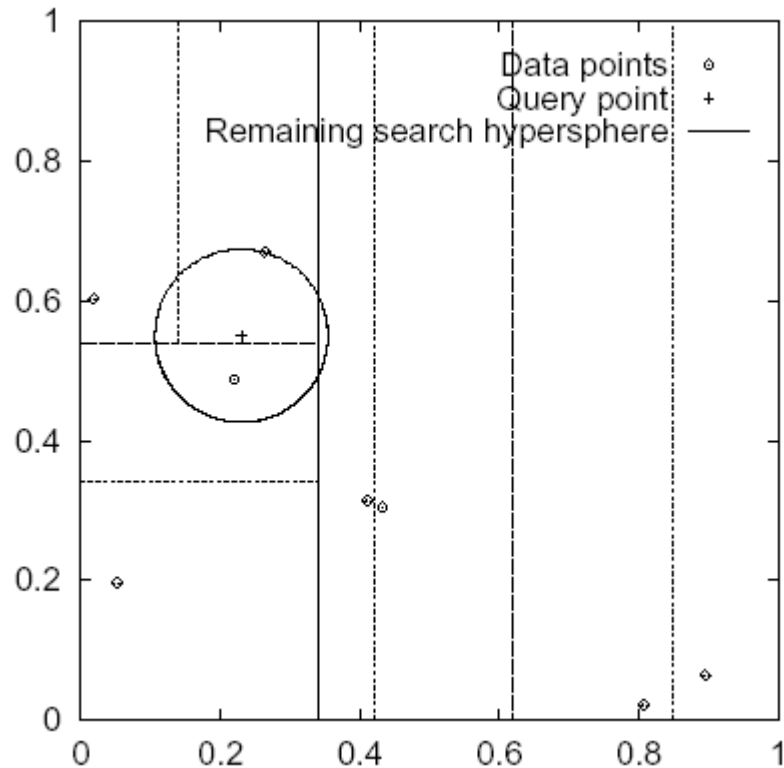
Figure 6.6

A bad distribution which forces almost all nodes to be inspected.

Solution: Approximate nearest neighbor K-d tree

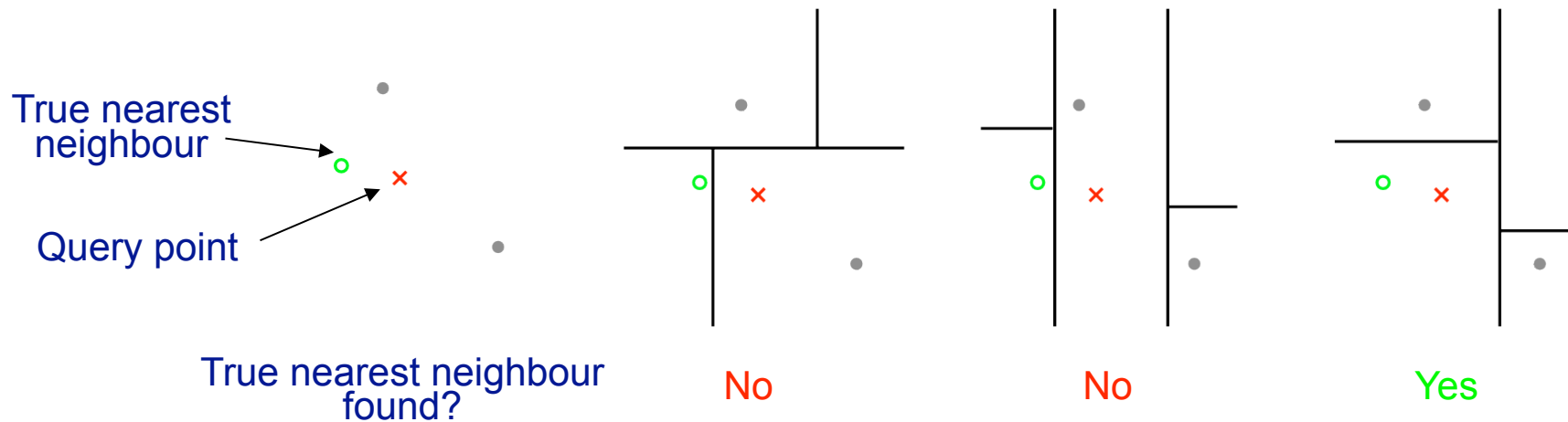
Key ideas:

- Limit the number of neighbouring k-d tree bins to explore
- Search k-d tree bins in order of distance from query
- Requires use of a priority queue
- Randomization



Randomized K-d trees

- How to choose the dimension to split and the splitting point?
 - Pick dimension with the highest variance
 - Split at the mean/median
- Multiple randomized trees increase the chances of finding nearby points



Approximate nearest neighbour search using a randomized forest of K-d trees: Algorithm summary

1. Descent all (typically 8) trees to the leaf node
2. Search k-d tree bins in order of distance from query
 - Distance between the query and the bin is defined as the minimum distance between the query and any point on the bin boundary
 - Requires the use of a priority queue:
 - > During lookup an entry is added to the priority queue about the option not taken
 - > For multiple trees, the queue is shared among the trees
 - Limit the number of neighbouring K-d tree bins to explore (parameter of the algorithm, typically set to 512)

Experimental evaluation for SIFT matching

<http://www.cs.ubc.ca/~lowe/papers/09muja.pdf>

FAST APPROXIMATE NEAREST NEIGHBORS WITH AUTOMATIC ALGORITHM CONFIGURATION

Marius Muja, David G. Lowe

Computer Science Department, University of British Columbia, Vancouver, B.C., Canada

mariusm@cs.ubc.ca, lowe@cs.ubc.ca

Keywords: nearest-neighbors search, randomized kd-trees, hierarchical k-means tree, clustering.

Abstract: For many computer vision problems, the most time consuming component consists of nearest neighbor matching in high-dimensional spaces. There are no known exact algorithms for solving these high-dimensional problems that are faster than linear search. Approximate algorithms are known to provide large speedups with only minor loss in accuracy, but many such algorithms have been published with only minimal guidance on selecting an algorithm and its parameters for any given problem. In this paper, we describe a system that answers the question, “What is the fastest approximate nearest-neighbor algorithm for my data?” Our system will take any given dataset and desired degree of precision and use these to automatically determine the best algorithm and parameter values. We also describe a new algorithm that applies priority search on hierarchical k-means trees, which we have found to provide the best known performance on many datasets. After testing a range of alternatives, we have found that multiple randomized k-d trees provide the best performance for other datasets. We are releasing public domain code that implements these approaches. This library provides about one order of magnitude improvement in query time over the best previously available software and provides

Randomized K-d trees

Performance w.r.t. the number of trees

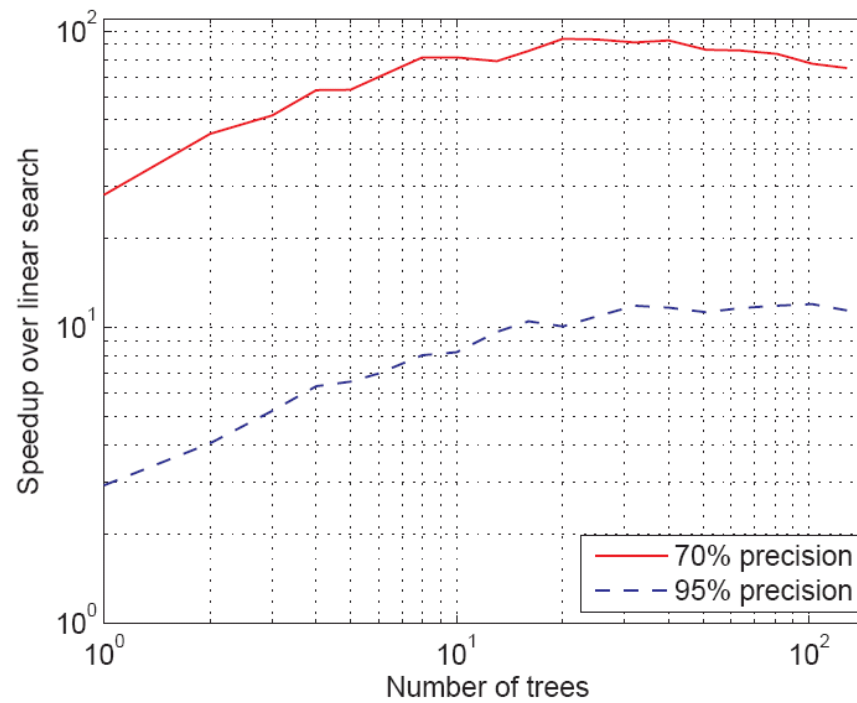
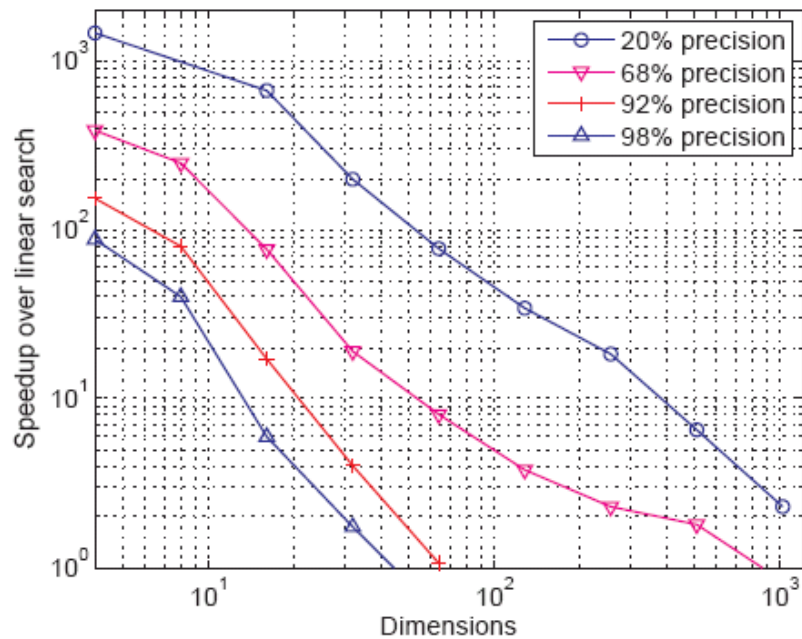


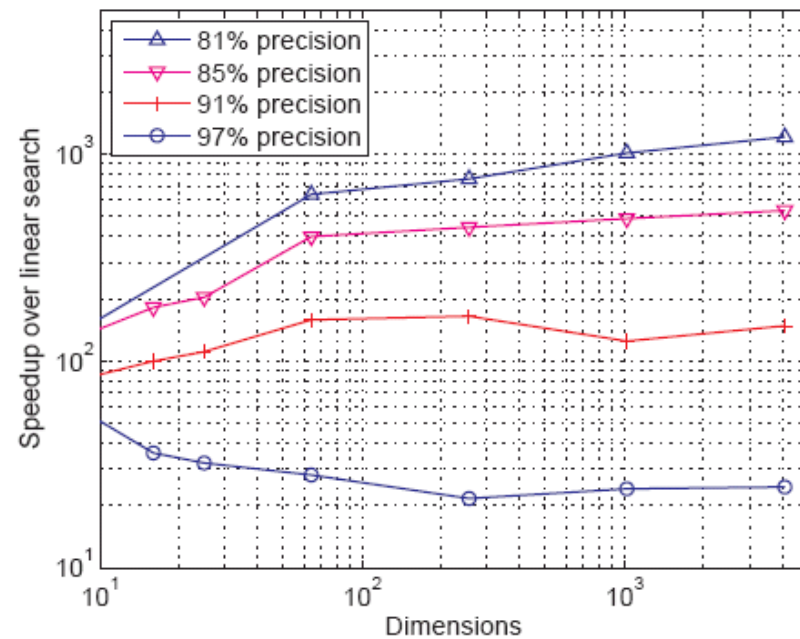
Figure 2: Speedup obtained by using multiple random kd-trees (100K SIFT features dataset)

Randomized K-d trees

Performance w.r.t. the number of dimensions



(a) Random vectors



(b) Image patches

Figure 4: Search efficiency for data of varying dimensionality. The random vectors (a) represent the hardest case in which dimensions have no correlations, while most real-world problems behave more like the image patches (b)

Randomized K-d trees: discussion

- Find approximate nearest neighbor in $O(\log N)$ time, where N is the number of data points.
- Increased memory requirements: needs to store multiple (~ 8) trees
- Good performance in practice for recognition problems (NN-search for SIFT descriptors and image patches).
- Code available online:
<http://people.cs.ubc.ca/~mariusm/index.php/FLANN/FLANN>

Variation: K-means tree

- Partition of the space is determined by recursive application of k-means clustering.
- Cell boundaries are not axis aligned, but given by the set of cluster centers.
- Also called “tree structured vector quantization”.
- Finding nearest neighbor to a query point involves recursively finding nearest cluster center.
- Look-up complexity $O(\log N)$

Example

Tree construction:

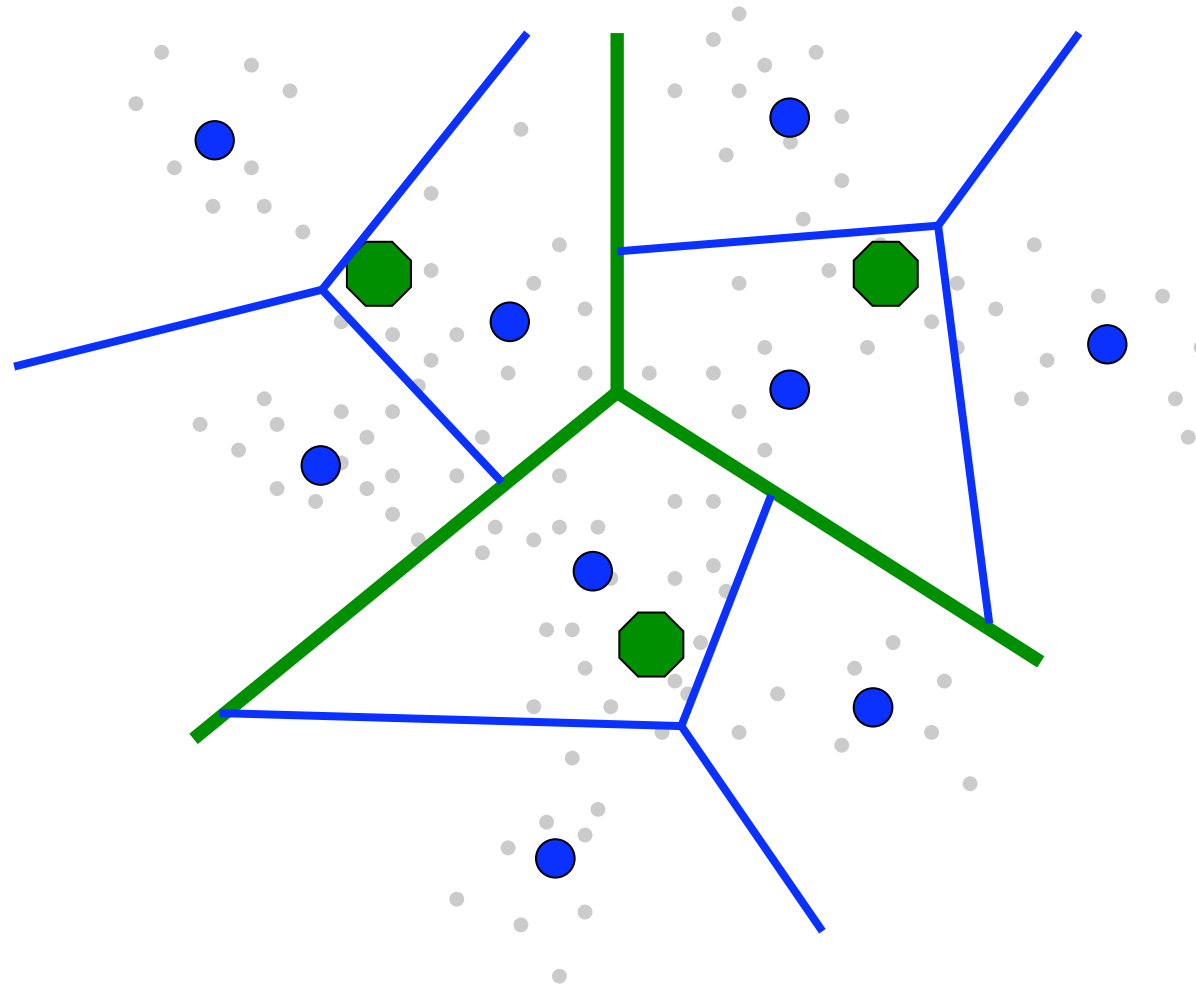


Figure credit: David Nister

Example

Query look-up:

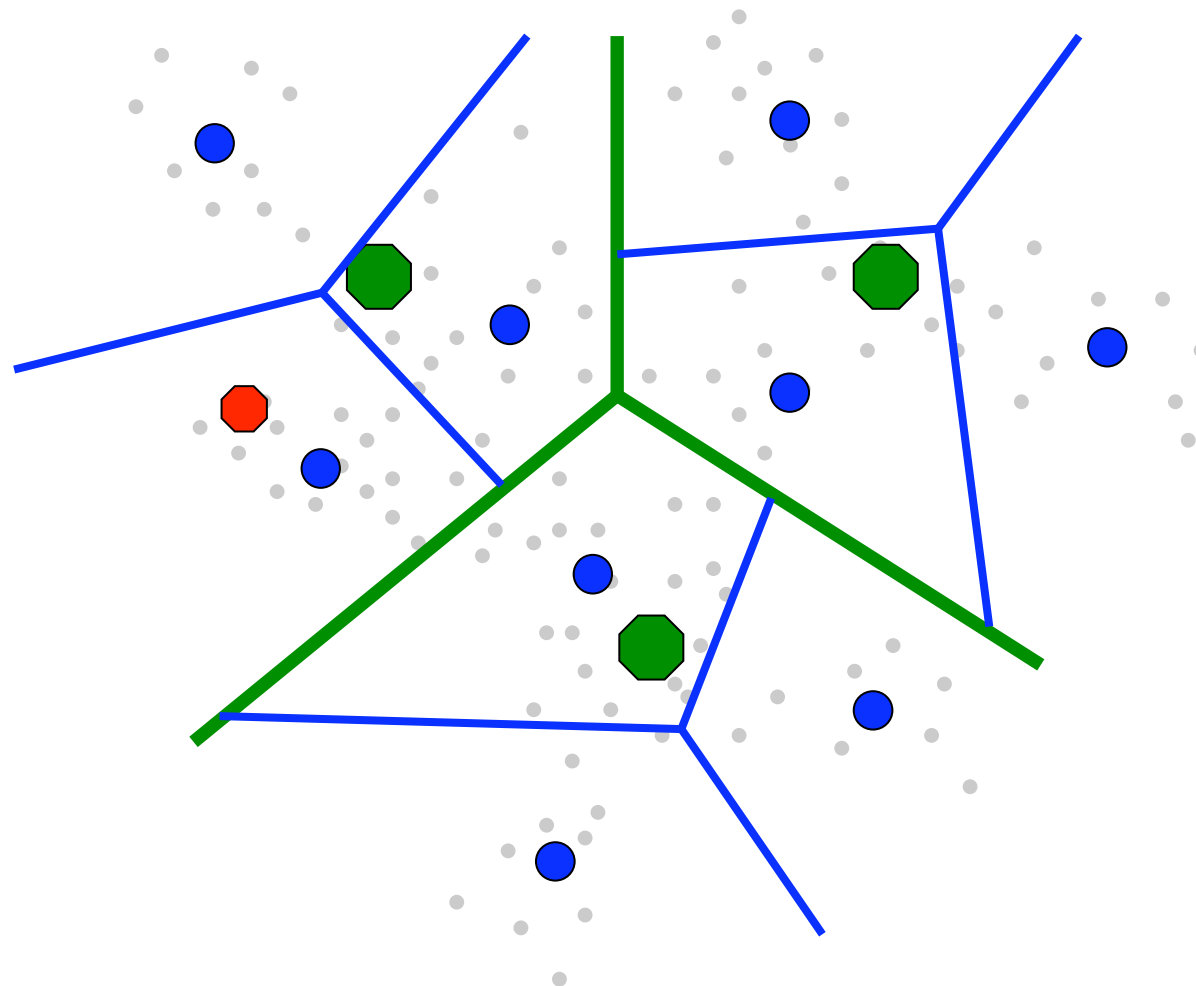
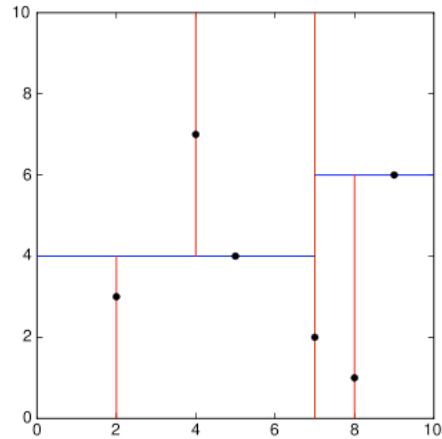
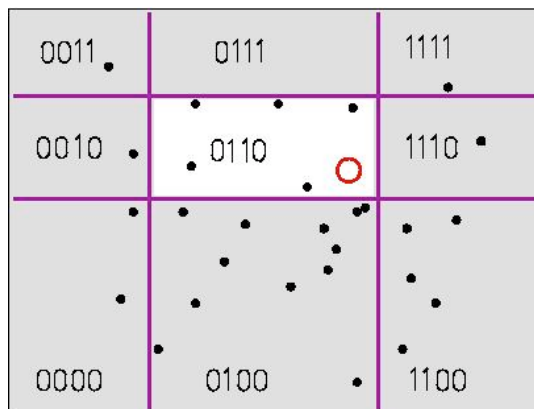


Figure credit: David Nister

Indexing local features: approximate nearest neighbor search



Best-Bin First (BBF), a variant of k-d trees that uses priority queue to examine most promising branches first [Beis & Lowe, CVPR 1997]



(1)
(2)

Locality-Sensitive Hashing (LSH), a randomized hashing technique using hash functions that map similar points to the same bin, with high probability [Indyk & Motwani, 1998]

(3) (4)

Locality Sensitive Hashing (LSH)

Idea: construct hash functions $g: \mathbb{R}^d \rightarrow \mathbb{Z}^k$ such that

for any points p, q :

If $\|p-q\| \leq r$, then $\Pr[g(p)=g(q)]$ is “high” or “not-so-small”

If $\|p-q\| > cr$, then $\Pr[g(p)=g(q)]$ is “small”

Example of g : linear projections

$g(p) = \langle h_1(p), h_2(p), \dots, h_k(p) \rangle$, where $h_{x,b}(p) = \lfloor (p \cdot X + b) / w \rfloor$

$\lfloor \cdot \rfloor$ is the “floor” operator.

X_i are sampled from a Gaussian.

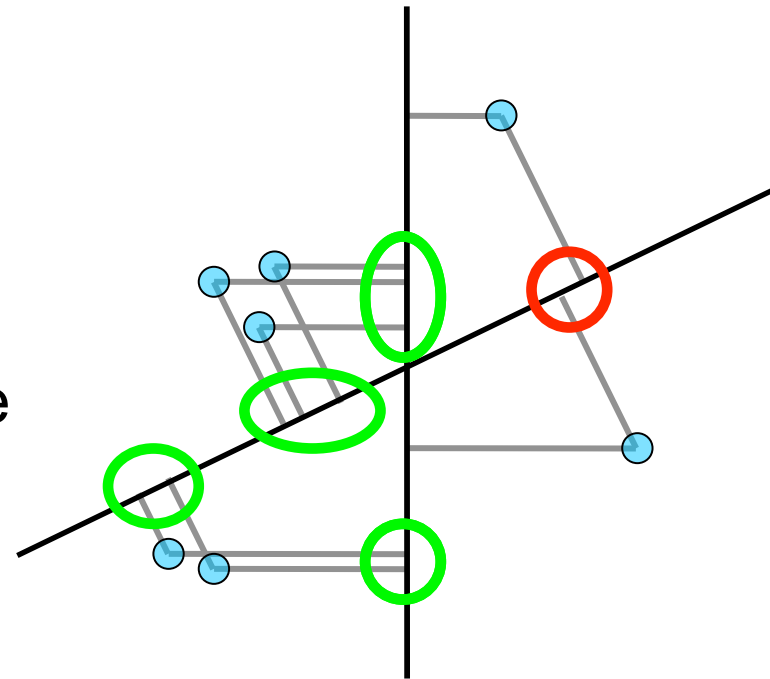
w is the width of each quantization bin.

b is sampled from uniform distr. $[0, w]$.

[Datar-Immorlica-Indyk-Mirroknii'04]

Locality Sensitive Hashing (LSH)

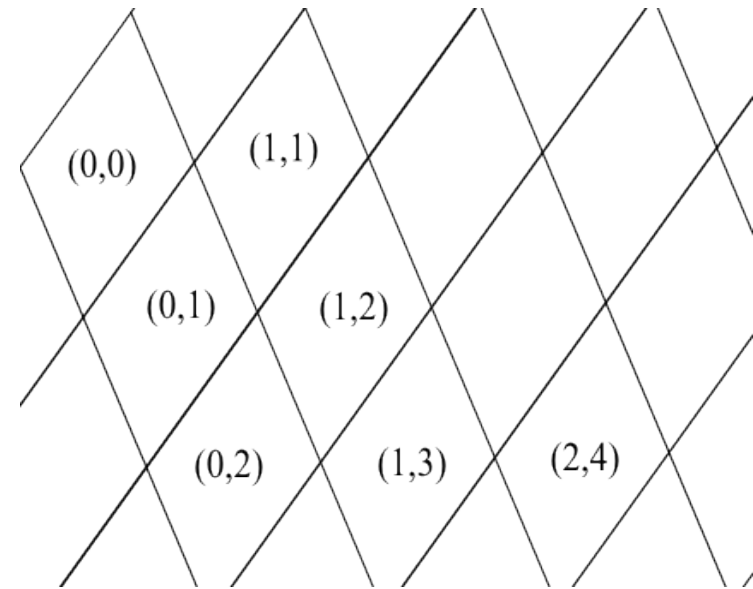
- Choose a random projection
- Project points
- Points close in the original space remain close under the projection
- Unfortunately, converse not true



- Answer: use multiple quantized projections which define a high-dimensional “grid”

Locality Sensitive Hashing (LSH)

- Cell contents can be efficiently indexed using a hash table
- Repeat to avoid quantization errors near the cell boundaries



- Point that shares at least one cell = potential candidate
- Compute distance to all candidates

LSH: discussion

In theory, query time is $O(kL)$, where k is the number of projections and L is the number of hash tables

I.e. independent of the number of points, N .

In practice, LSH has high memory requirements as large number of projections/hash tables are needed.

Code and more materials available online:

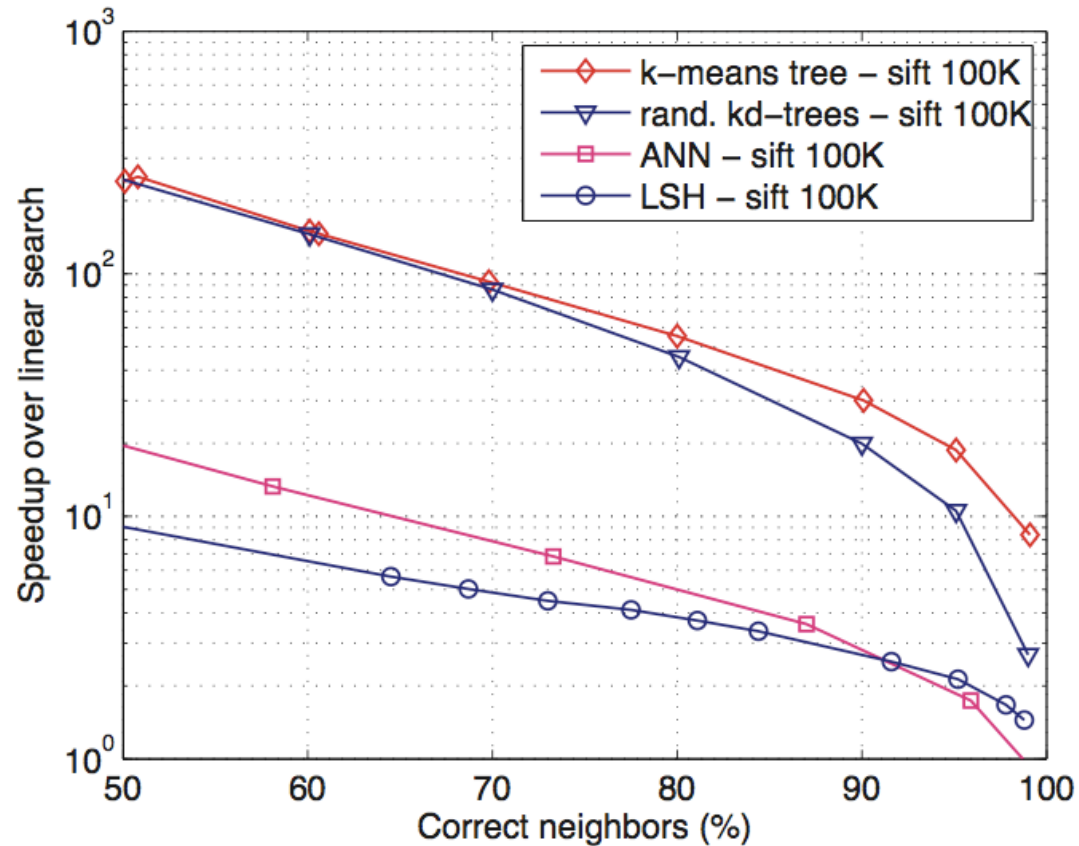
<http://www.mit.edu/~andoni/LSH/>

See also:

[http://cobweb.ecn.purdue.edu/~malcolm/yahoo/Slaney2008\(LSHTutorialDraft\).pdf](http://cobweb.ecn.purdue.edu/~malcolm/yahoo/Slaney2008(LSHTutorialDraft).pdf)

Comparison of approximate NN-search methods

Dataset: 100K SIFT descriptors

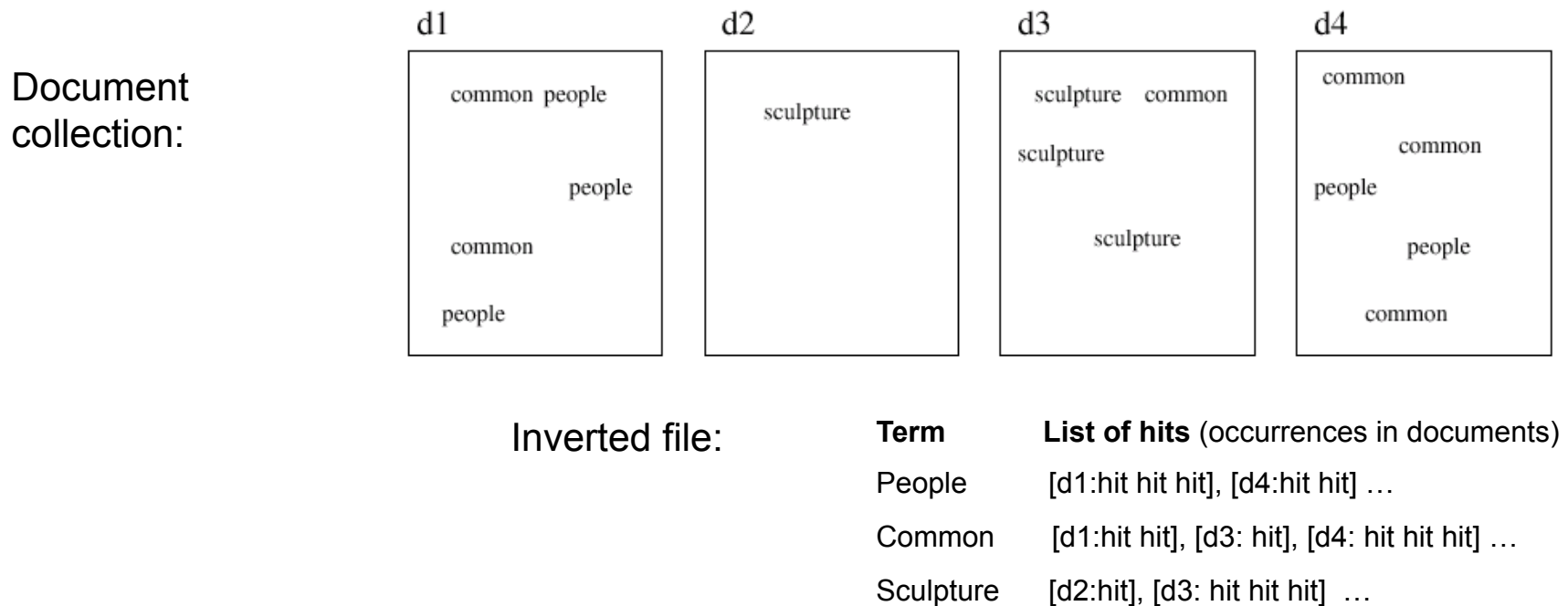


Code for all methods available online, see [Muja&Lowe'09](#)

Figure: Muja&Lowe'09

Another idea: visual indexing using inverted files

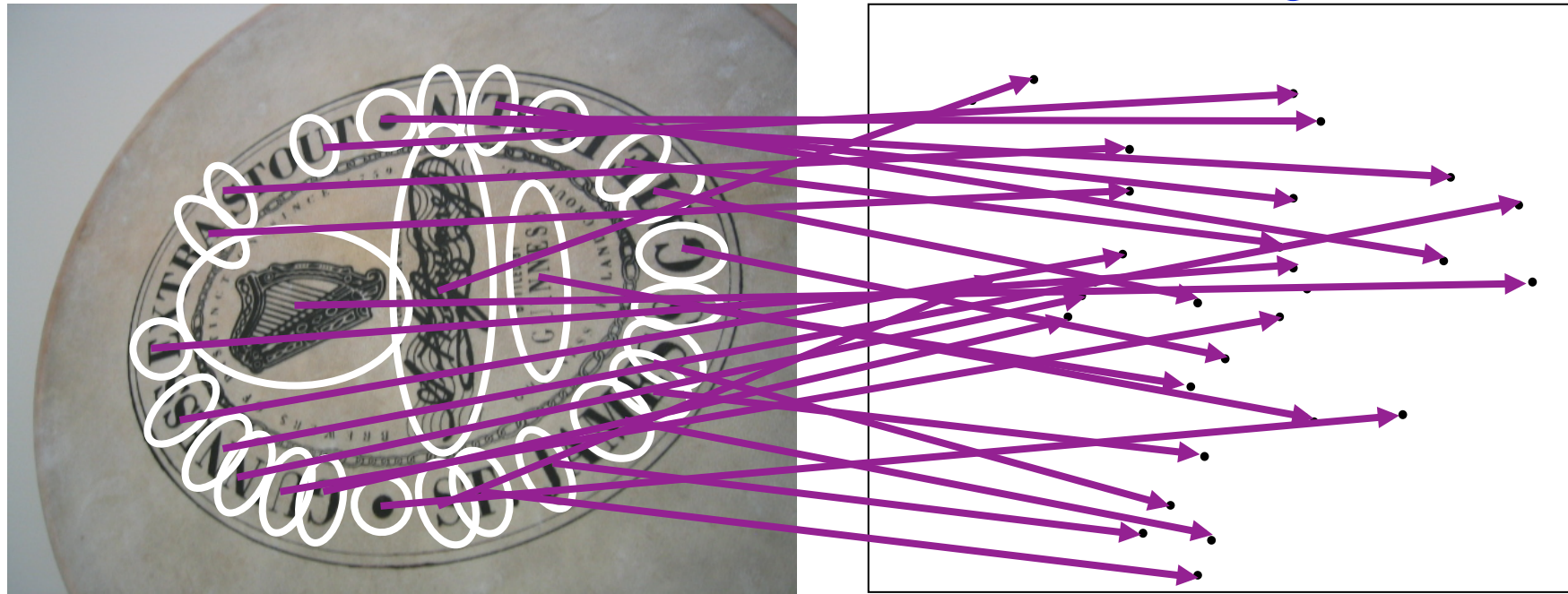
- Is storing all feature descriptors in memory feasible?
Example: $N = 10^7$ images, 10^{10} features, 1TB memory?
- Look how text-based search engines (Google) index documents – **inverted files**.



Need to map feature descriptors to “visual words”.

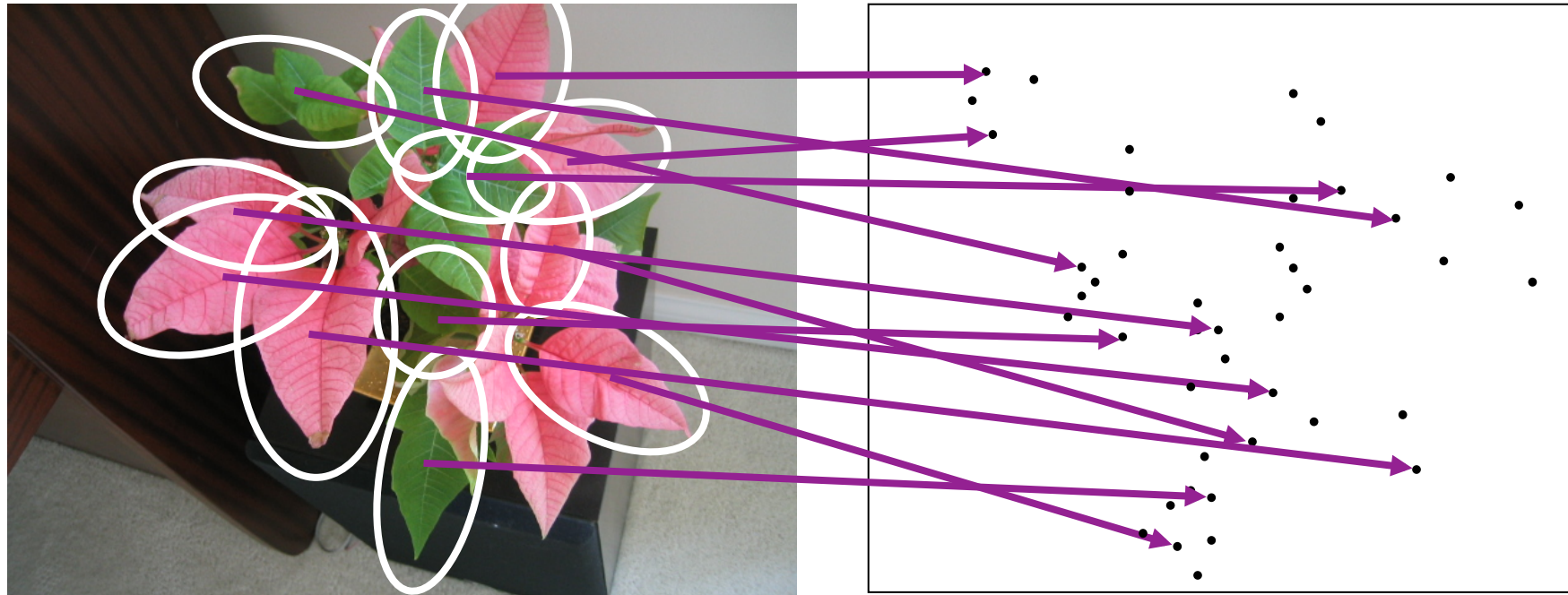
Visual words: main idea

Extract some local features from a number of images ...

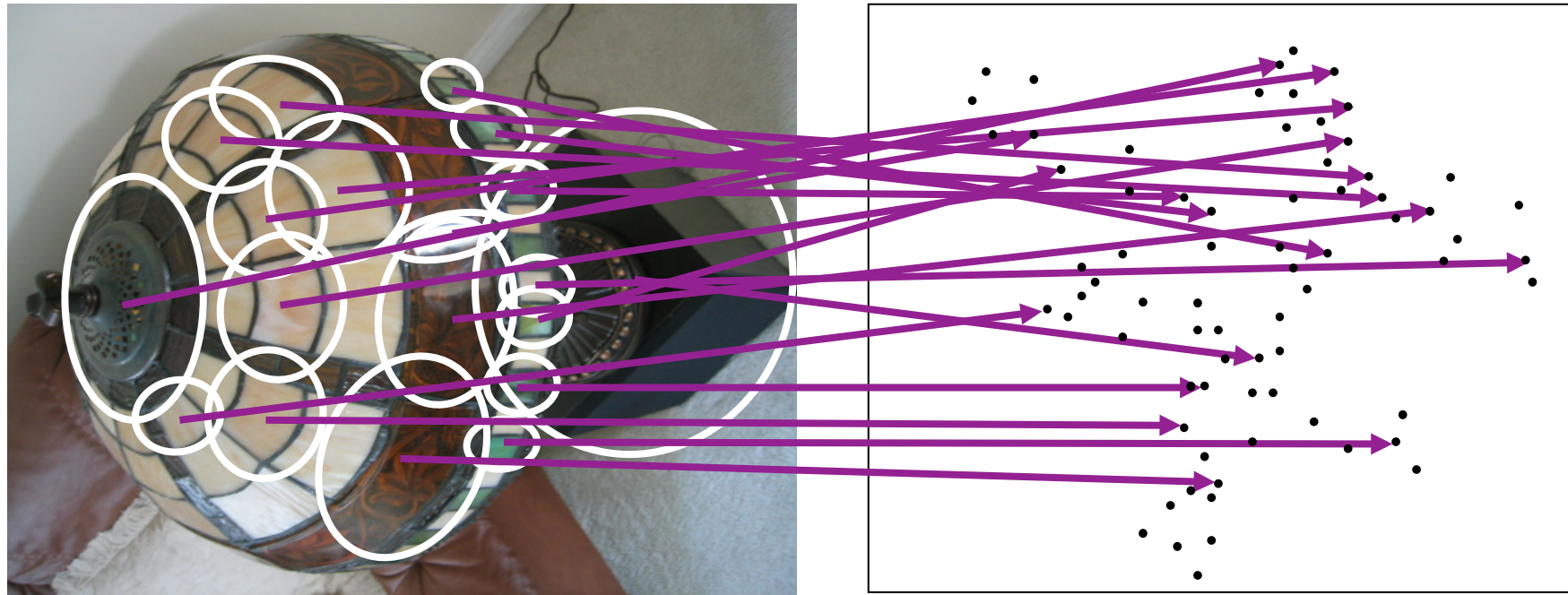


e.g., SIFT descriptor space: each point is 128-dimensional

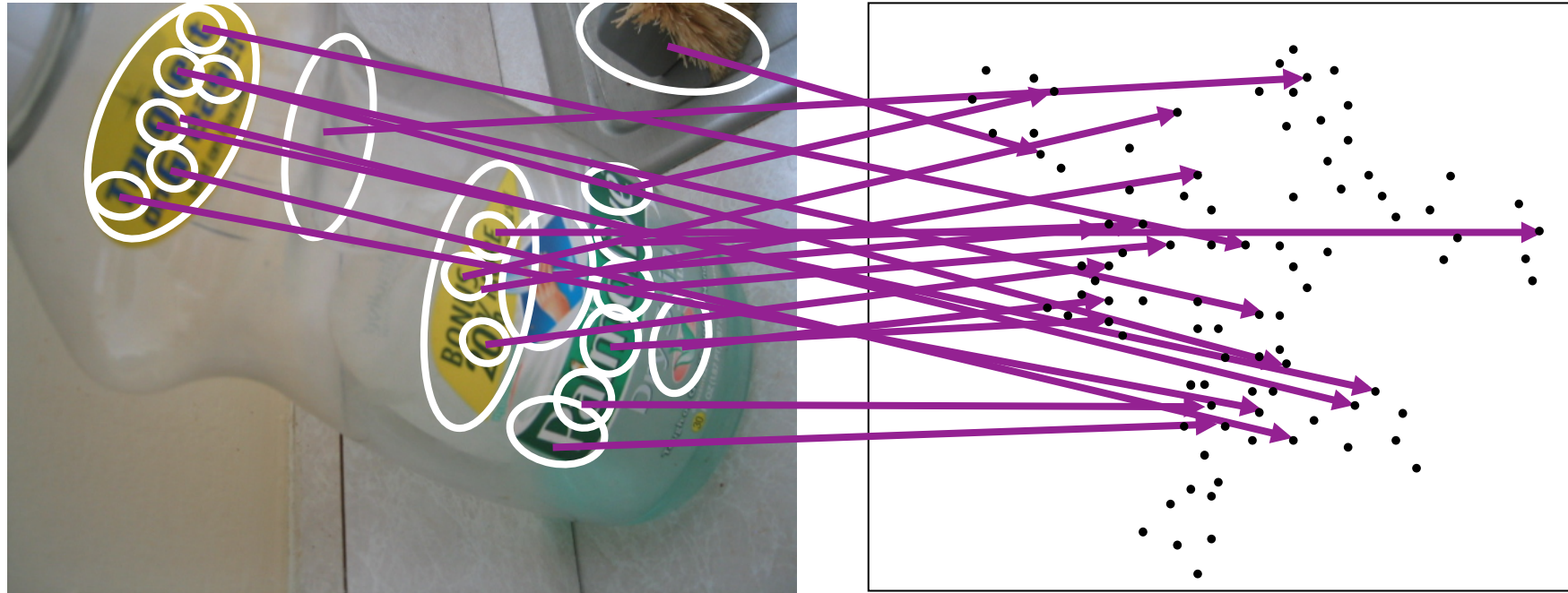
Visual words: main idea

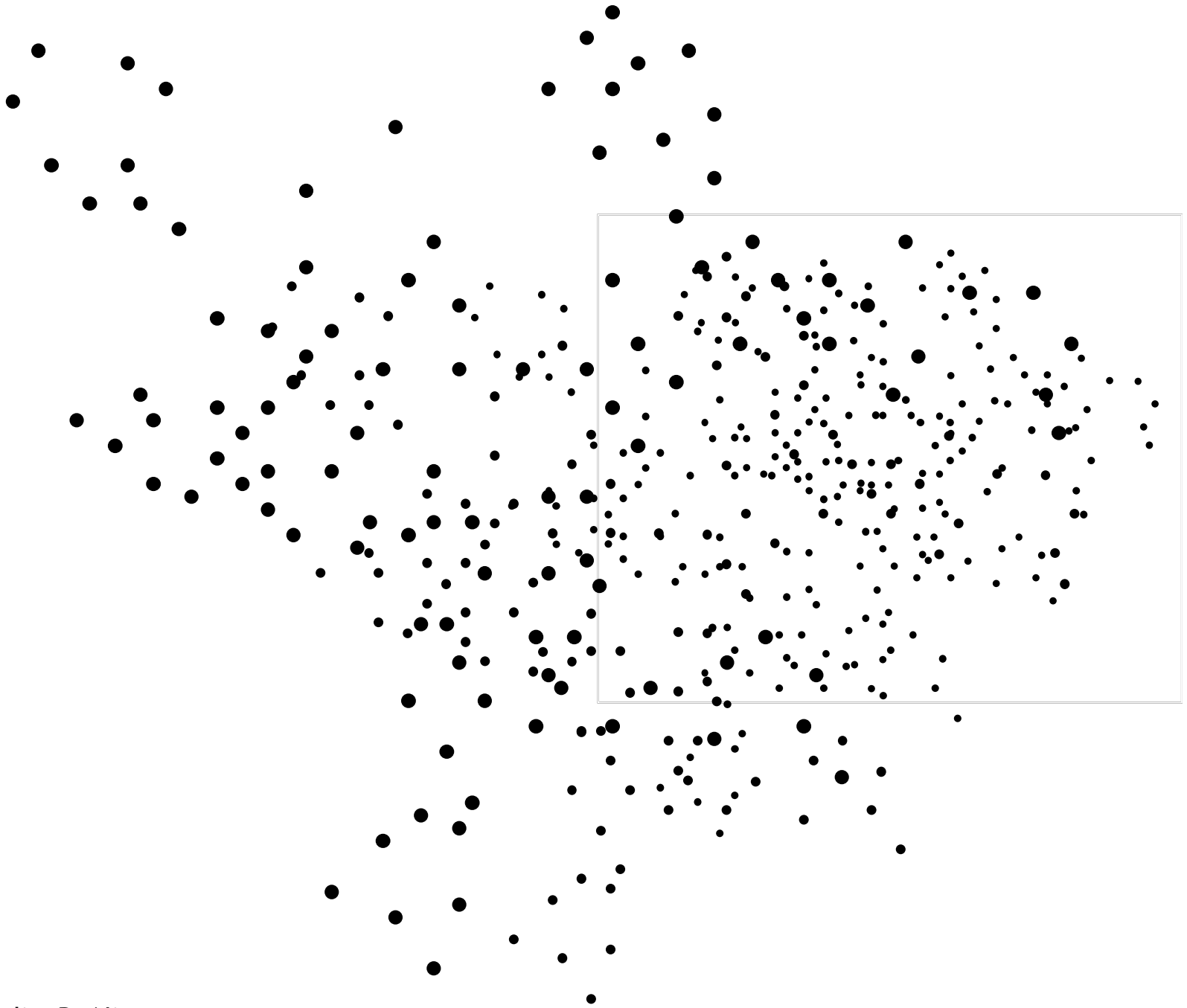


Visual words: main idea

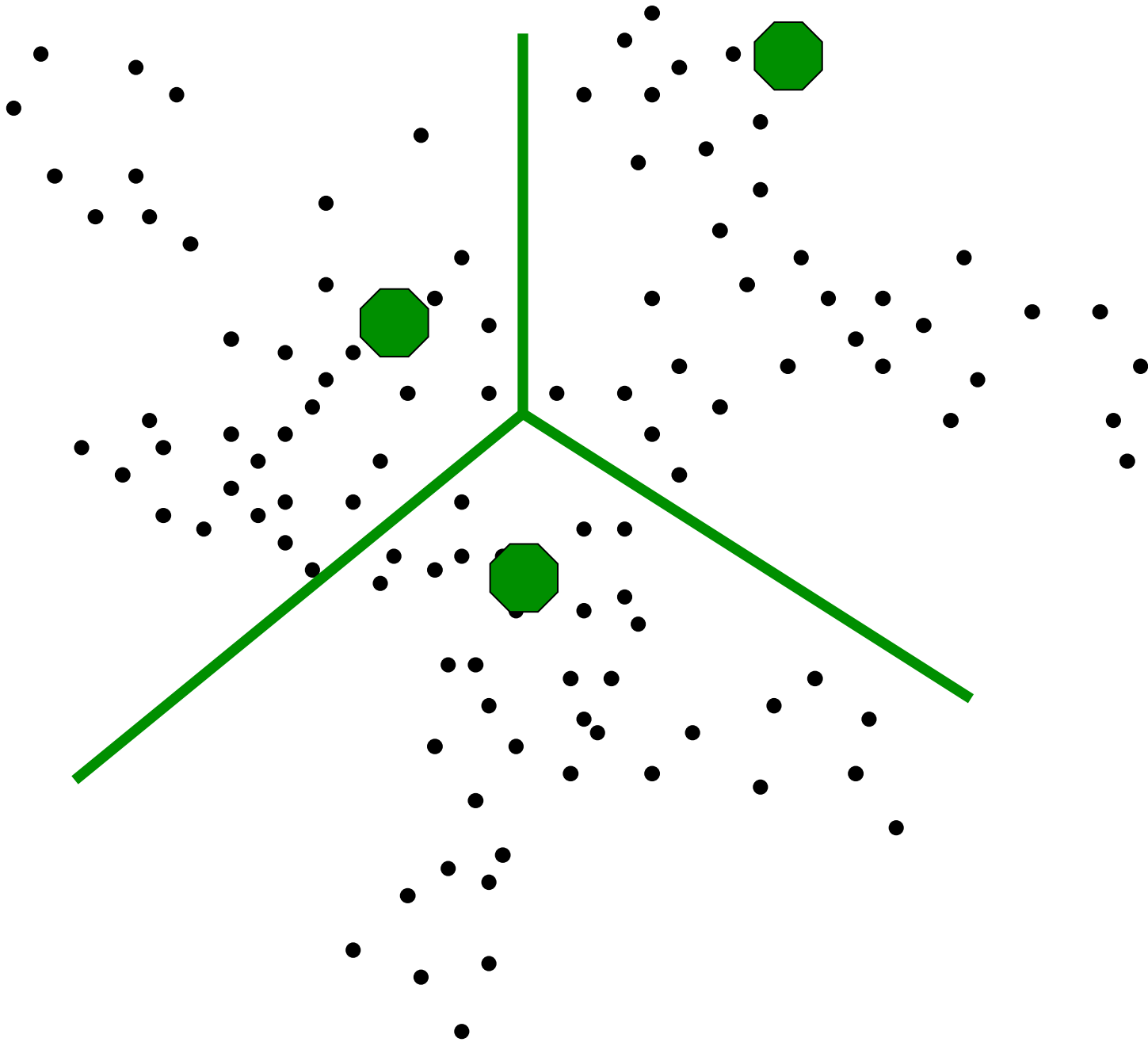


Visual words: main idea





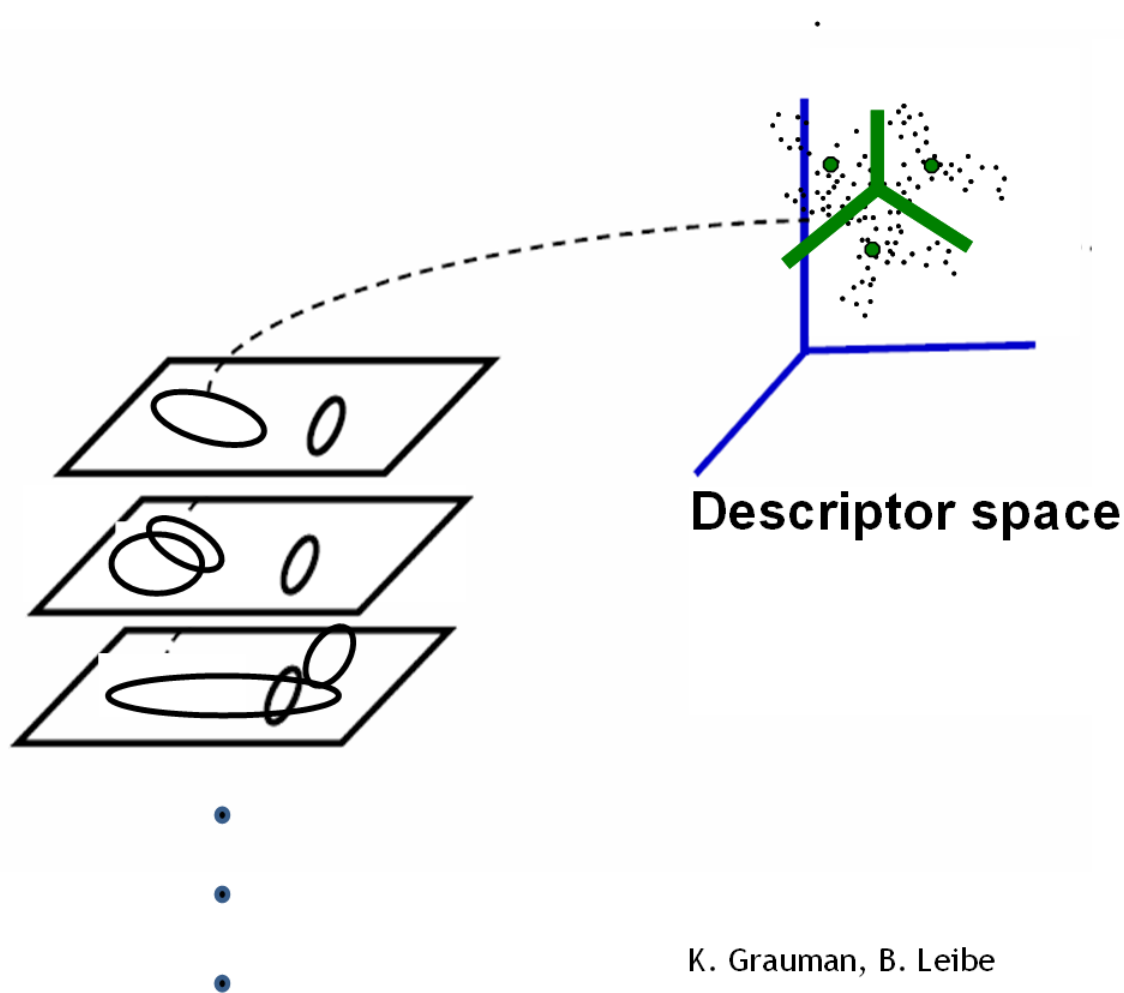
Slide credit: D. Nister



Slide credit: D. Nister

Visual words: main idea

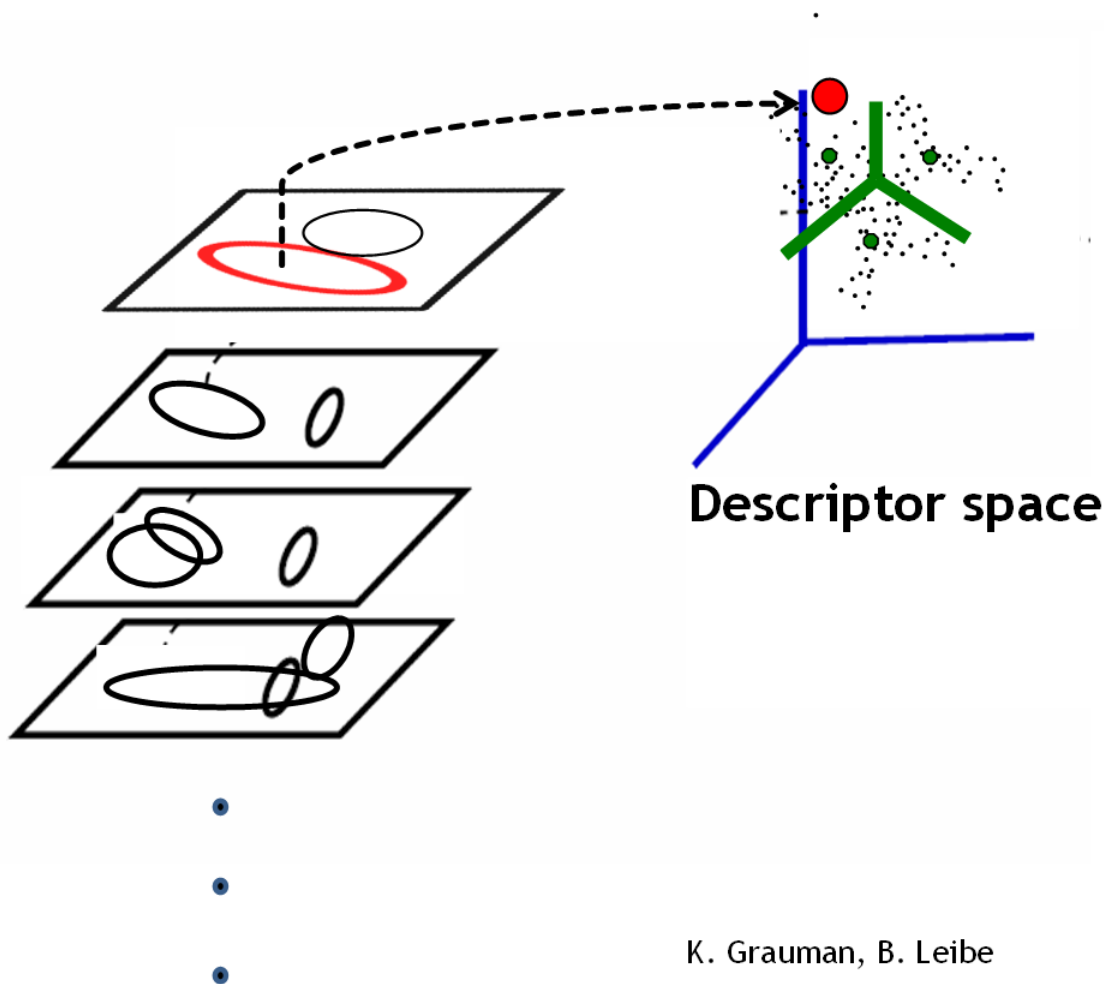
Map high-dimensional descriptors to tokens/words by quantizing the feature space



- Quantize via clustering, let cluster centers be the prototype “words”

Visual words: main idea

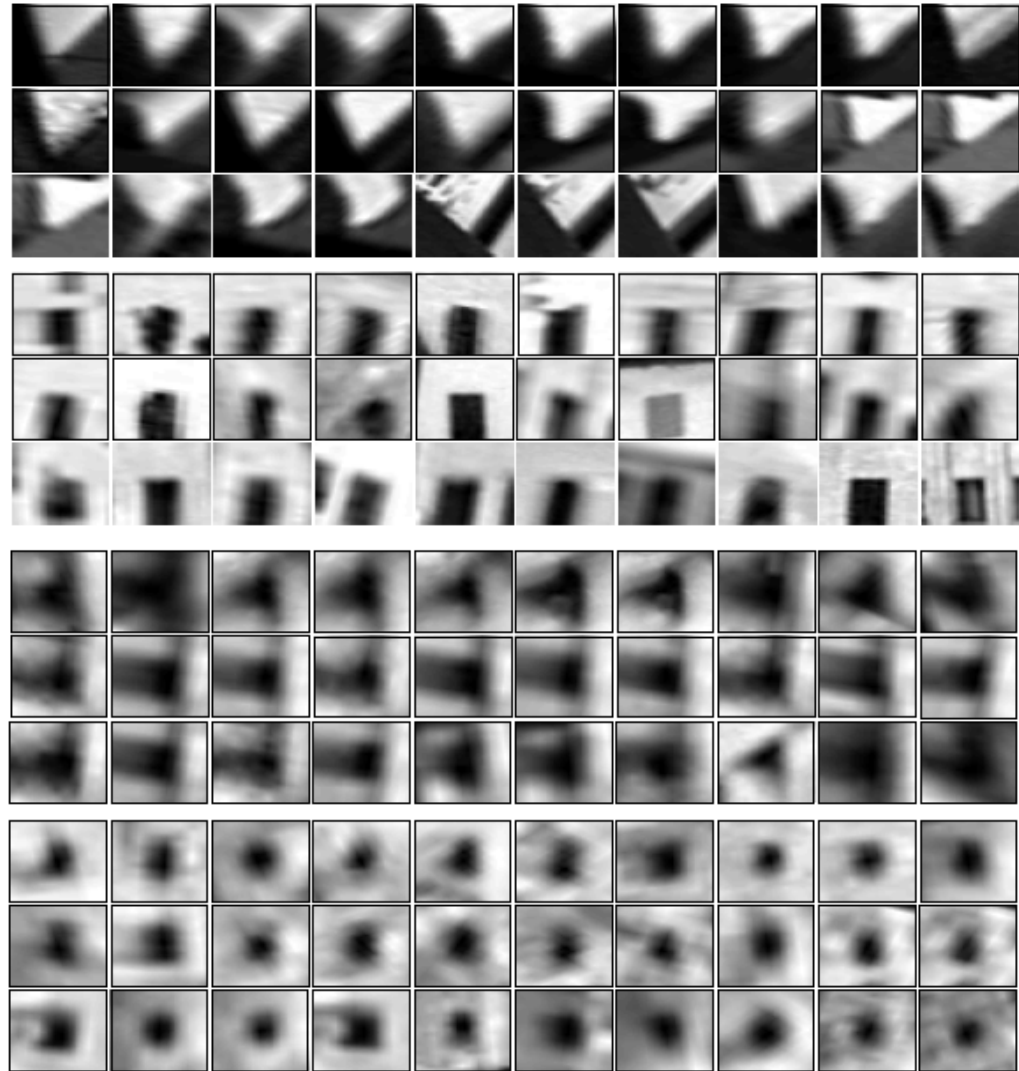
Map high-dimensional descriptors to tokens/words by quantizing the feature space



- Determine which word to assign to each new image region by finding the closest cluster center.

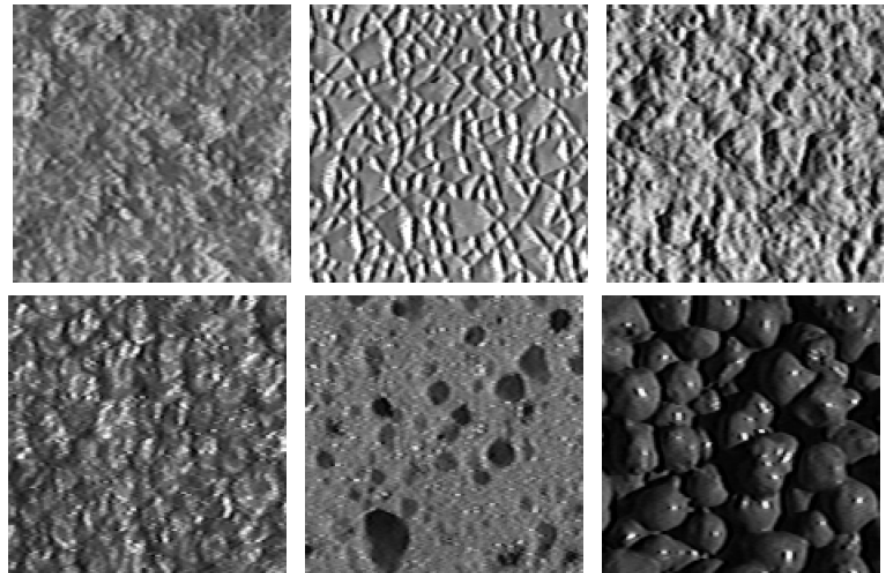
Visual words

Example: each group of patches belongs to the same visual word



Visual words

- First explored for texture and material representations
- *Texton* = cluster center of filter responses over collection of images
- Describe textures and materials based on distribution of prototypical texture elements.



Leung & Malik 1999; Varma & Zisserman, 2002; Lazebnik, Schmid & Ponce, 2003;

Inverted file index for images comprised of visual words



frame #5



frame #10

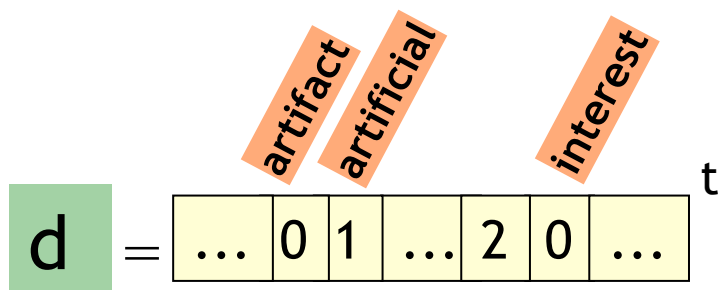
Word number	List of image numbers
1	5, 10, ...
2	10, ...
...	...

- Score each image by the number of common visual words (tentative correspondences)

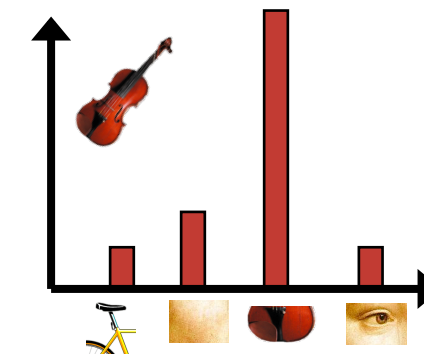
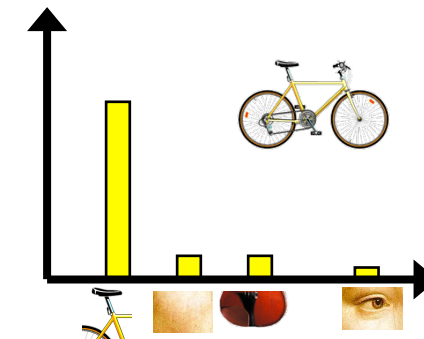
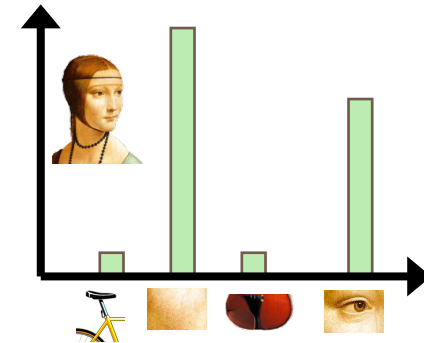
Another interpretation: Bags of visual words

Summarize entire image based on its distribution (histogram) of visual word occurrences.

Analogous to bag of words representation commonly used for documents.



Hofmann 2001



Another interpretation: the bag-of-words model

For a vocabulary of size K , each image is represented by a K -vector

$$\mathbf{v}_d = (t_1, \dots, t_i, \dots, t_K)^\top$$

where t_i is the number of occurrences of visual word i .

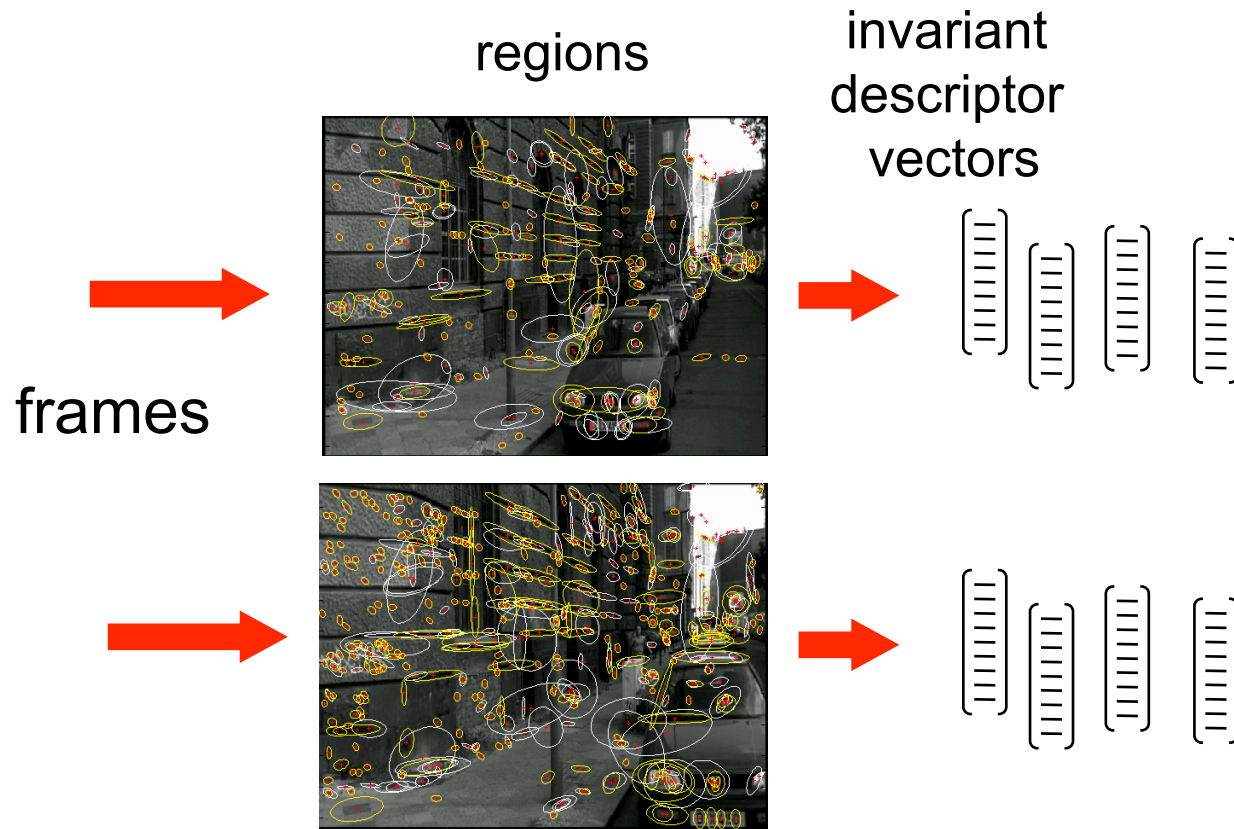
Images are ranked by the normalized scalar product between the query vector \mathbf{v}_q and all vectors in the database \mathbf{v}_d :

$$f_d = \frac{\mathbf{v}_q^\top \mathbf{v}_d}{\|\mathbf{v}_q\|_2 \|\mathbf{v}_d\|_2}$$

Scalar product can be computed efficiently using inverted file.

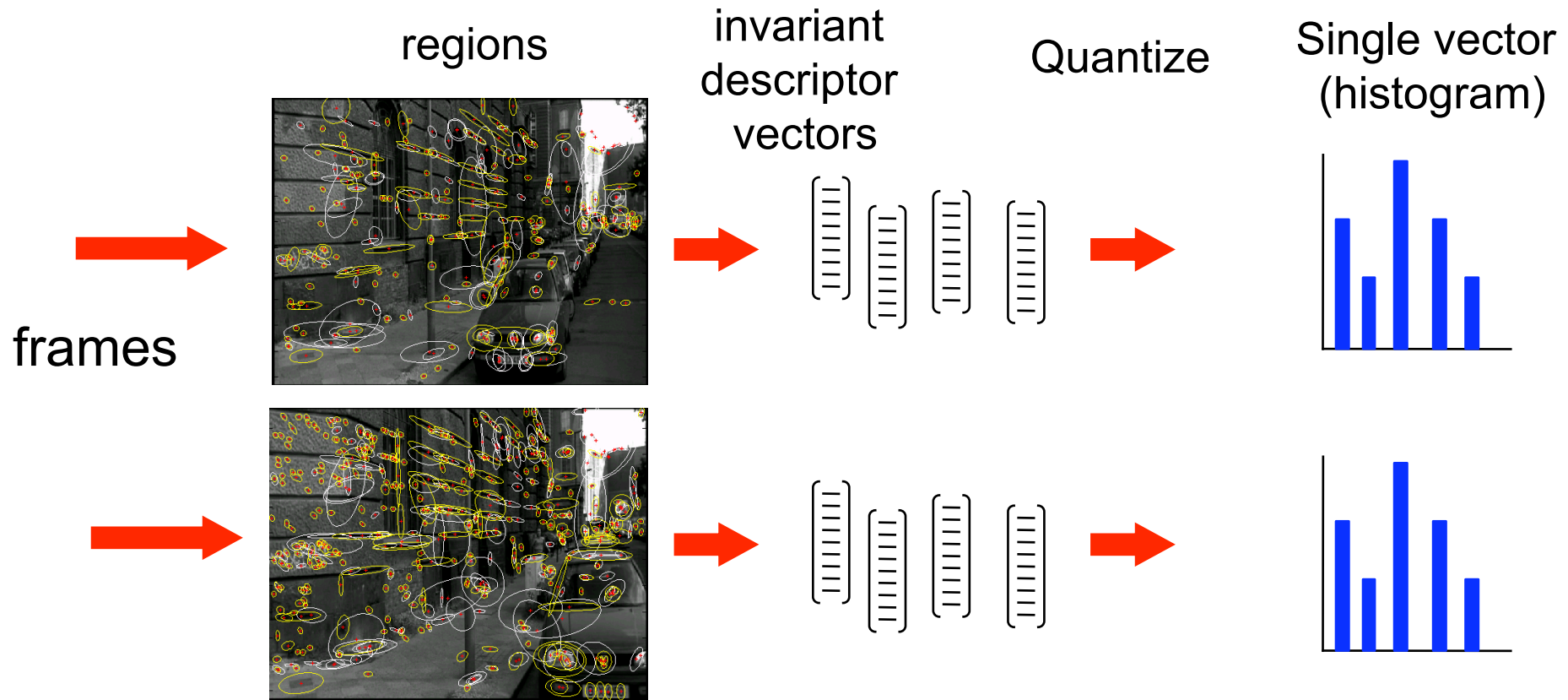
What if vectors are binary? What is the meaning of $\mathbf{v}_q^\top \mathbf{v}_d$?

Strategy I: Match descriptor vectors from all images



1. Compute affine covariant regions in each frame independently
2. “Label” each region by a vector of descriptors based on its intensity
3. Finding corresponding regions is transformed to **finding nearest neighbour vectors**
4. Rank retrieved frames by number of corresponding regions
5. Verify retrieved frame based on spatial consistency

Strategy II: Match histograms of visual words



1. Compute affine covariant regions in each frame independently
2. "Label" each region by a vector of descriptors based on its intensity
3. **Build histograms of visual words by descriptor quantization**
4. **Rank retrieved frames by matching vis. word histograms using inverted files.**
5. Verify retrieved frame based on spatial consistency

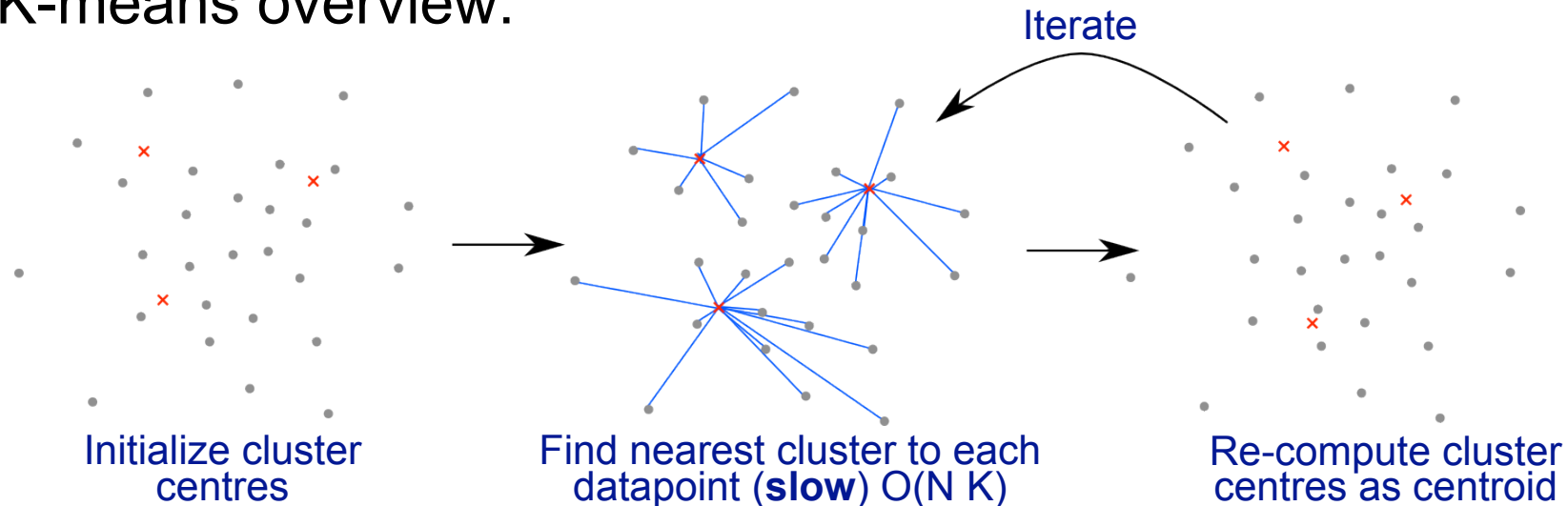
Visual words: discussion I.

Efficiency – cost of quantization

- Need to still assign each local descriptor to one of the cluster centers. Could be prohibitive for large vocabularies (K=1M)
- Approximate NN-search still needed.
- True also for building the vocabulary.

Vocabulary building: Quantization using K-means

- K-means overview:



- K-means provably locally minimizes the sum of squared errors (SSE) between a cluster centre and its points
- Note also that the quantizer depends on the initialization.
- The nearest neighbour search is the bottleneck

Vocabulary building: Efficient K-means

- Use the **approximate nearest neighbour search** (randomized forest of kd-trees or k-means trees) to determine the closest cluster centre for each data point.
- **Original K-means complexity: $O(N K)$**
- **Approximate K-means / k-means tree complexity: $O(N \log K)$**
- Can be scaled to very large K.

Visual words: discussion II.

Generalization

- Is vocabulary/quantization learned on one dataset good for searching another dataset?
- Not really – need to build vocabulary for each dataset.

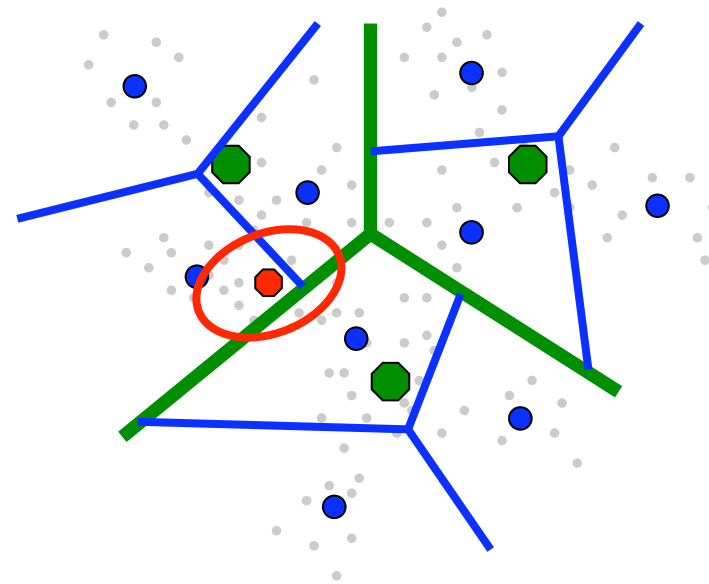
But, see recent work by Jegou et al.:

Hamming Embedding and Weak Geometry Consistency for Large Scale Image Search, ECCV'2008

<http://lear.inrialpes.fr/pubs/2008/JDS08a/>

Visual words: discussion III.

- What about quantization effects?
- Visual word assignment can change due to e.g. noise in region detection, descriptor computation or non-modeled image variation (3D effects, lighting)



See also:

Jegou et al., ECCV'2008, <http://lear.inrialpes.fr/pubs/2008/JDS08a/>

Philbin et al. CVPR'08, <http://www.robots.ox.ac.uk/~vgg/publications/html/philbin08-bibtex.html>

Visual words: discussion IV.

- Need to determine the size of the vocabulary, K .
- Other algorithms for building vocabularies, e.g. agglomerative clustering / mean-shift, but typically more expensive.
- Supervised quantization?: also give examples of images / descriptors which should and should not match.

Demo

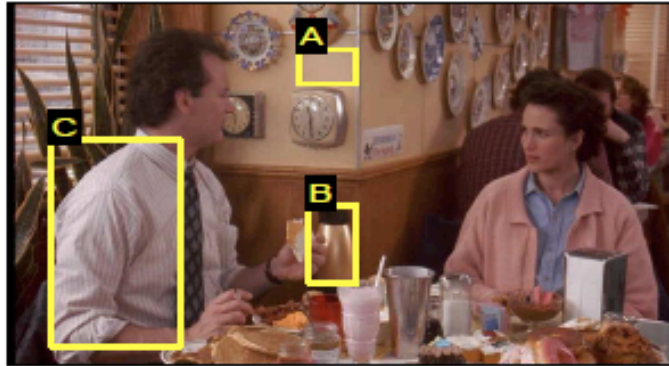
Oxford Buildings Search

[http://www.robots.ox.ac.uk/~vgg/research/oxbuildings/
index.html](http://www.robots.ox.ac.uk/~vgg/research/oxbuildings/index.html)

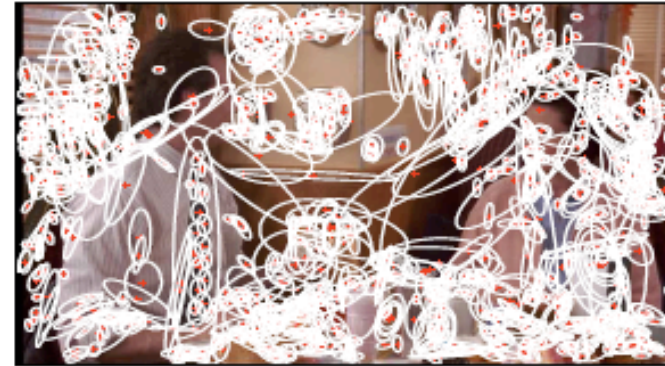
What objects/scenes local regions do not work on?



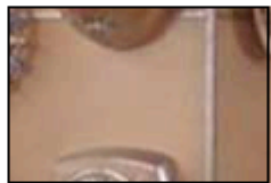
What objects/scenes local regions do not work on?



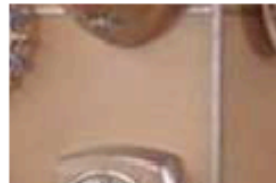
(a)



(b)



(c)



(d)



(e)



(f)



(g)



(h)

E.g. texture-less objects, objects defined by shape, deformable objects, wiry objects.

Other applications of large scale visual search

Sony Aibo (Evolution Robotics)

SIFT usage

- Recognize docking station
- Communicate with visual cards

Other uses

- Place recognition
- Loop closure in SLAM

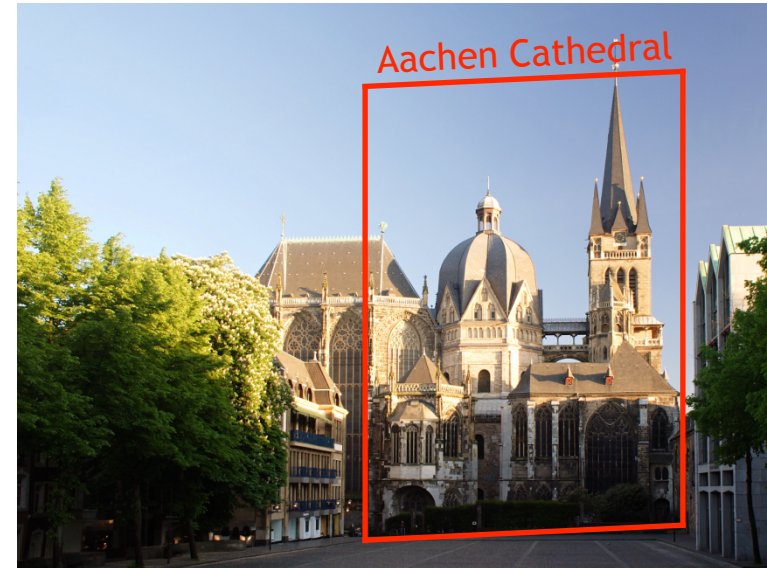


Example Applications



Mobile tourist guide

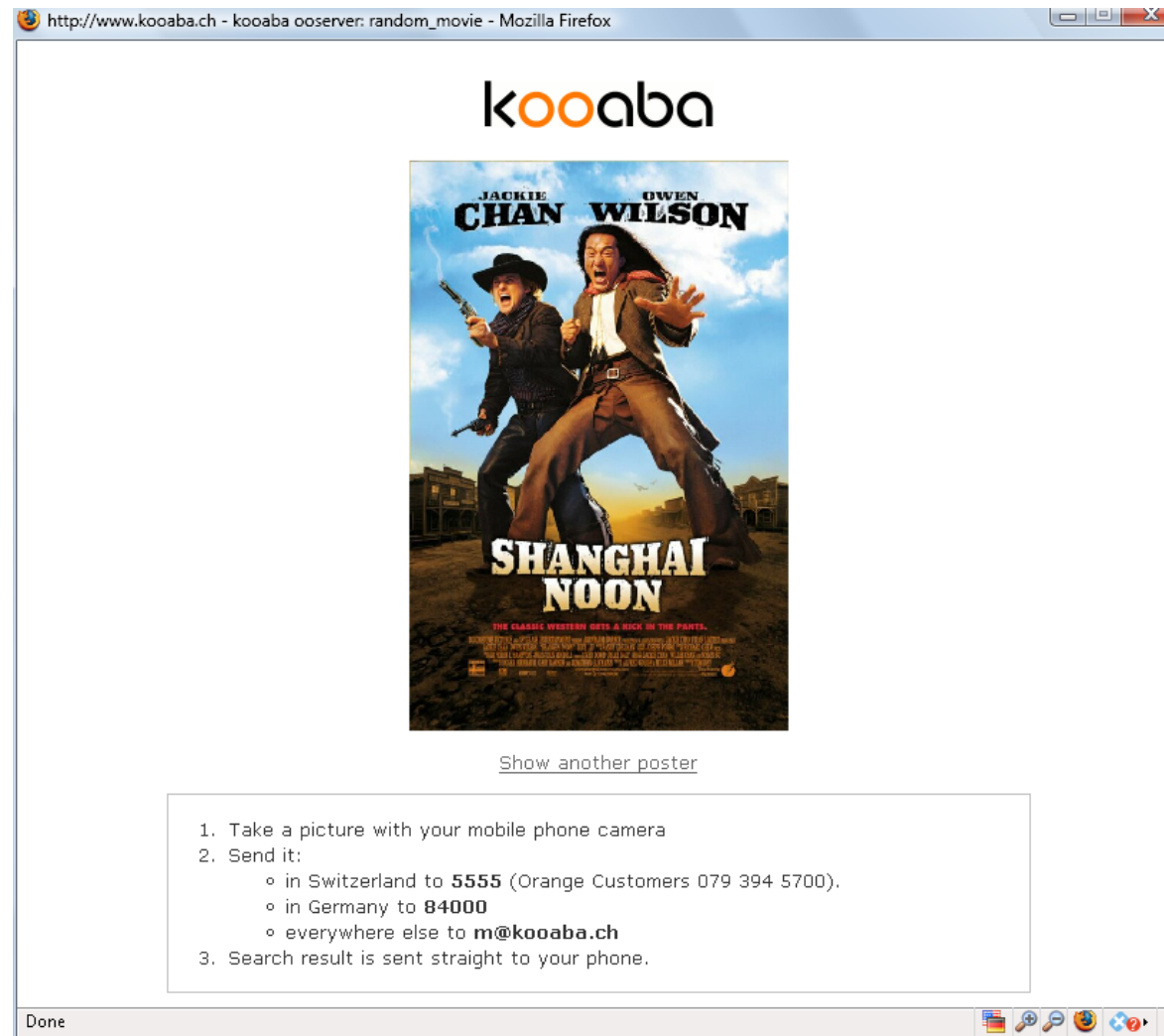
- Self-localization
- Object/building recognition
- Photo/video augmentation



Web Demo: Movie Poster Recognition


50'000 movie posters indexed

Query-by-image from mobile phone available in Switzerland



http://www.kooaba.ch - kooaba ooserver: random_movie - Mozilla Firefox

kooaba



SHANGHAI NOON

THE CLASSIC WESTERN GETS A KICK IN THE PANTS.

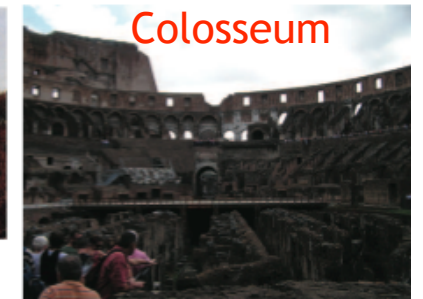
Show another poster

1. Take a picture with your mobile phone camera
2. Send it:
 - in Switzerland to **5555** (Orange Customers 079 394 5700).
 - in Germany to **84000**
 - everywhere else to **m@kooaba.ch**
3. Search result is sent straight to your phone.

Done

http://www.kooaba.com/en/products_engine.html#

Image Auto-Annotation



Left: Wikipedia image
Right: closest match from Flickr

[Quack CIVR'08]



Building Rome in a Day – or –

matching and 3D reconstruction in large unstructured datasets.

Goal: Build a 3D model of a city from a large collection of images downloaded from the Internet

Use a cluster with 500 CPU cores.

Building Rome in a Day, Sameer Agarwal, Noah Snavely, Ian Simon, Steven M. Seitz and Richard Szeliski,
International Conference on Computer Vision, 2009
<http://grail.cs.washington.edu/rome/>



15,464



37,383



76,389

Flickr: Creative Commons - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://www.flickr.com/creativecommons/by-nc-nd-2.0/

Home | Tags | Groups | People | Invite

Logged in as Jimantha | Your Account | Help | Sign Out




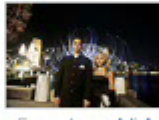











Photos: Yours · Upload · Organize · Your Contacts · Explore

flickr BETA

Creative Commons / Attribution-NonCommercial-NoDerivs License

(Or, [browse popular tags](#))

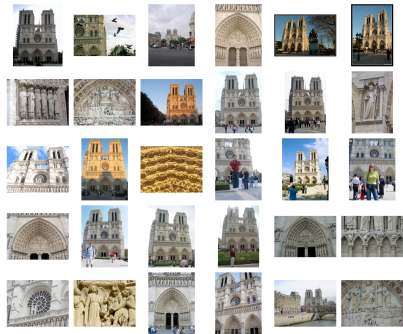
Here are the **100 most recent** licensed photos:

 From mctarnell	 From mugsy1274	 From darren 'djp' paine	 From darren 'djp' paine	 From dizz
 From alpha_zone	 From darren 'djp' paine	 From 331	 From mugsy1274	 From dalekg
				

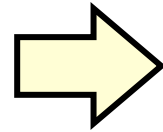
Done Adblock

Reproduced with permission of Yahoo! Inc. © 2005 by Yahoo! Inc.
YAHOO! and the YAHOO! logo are trademarks of Yahoo! Inc.

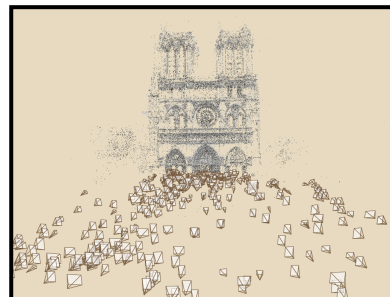
Photo Tourism overview



Input photographs



Scene reconstruction



Relative camera positions and orientations
Point cloud
Sparse correspondence

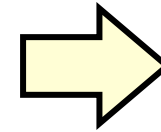
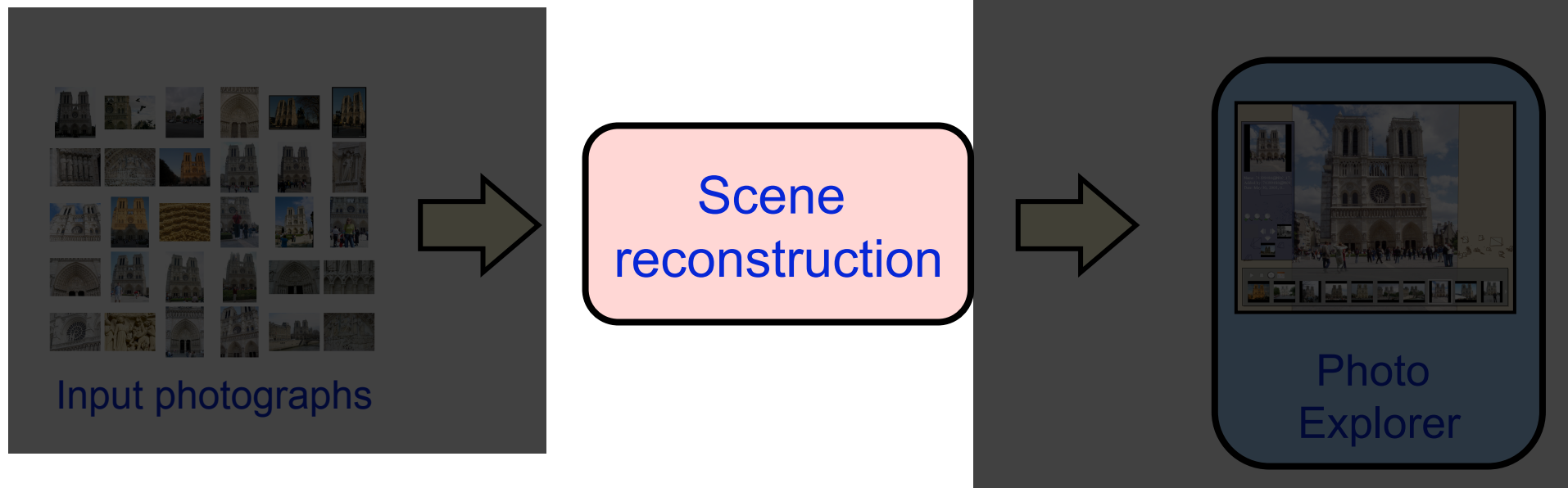


Photo Explorer

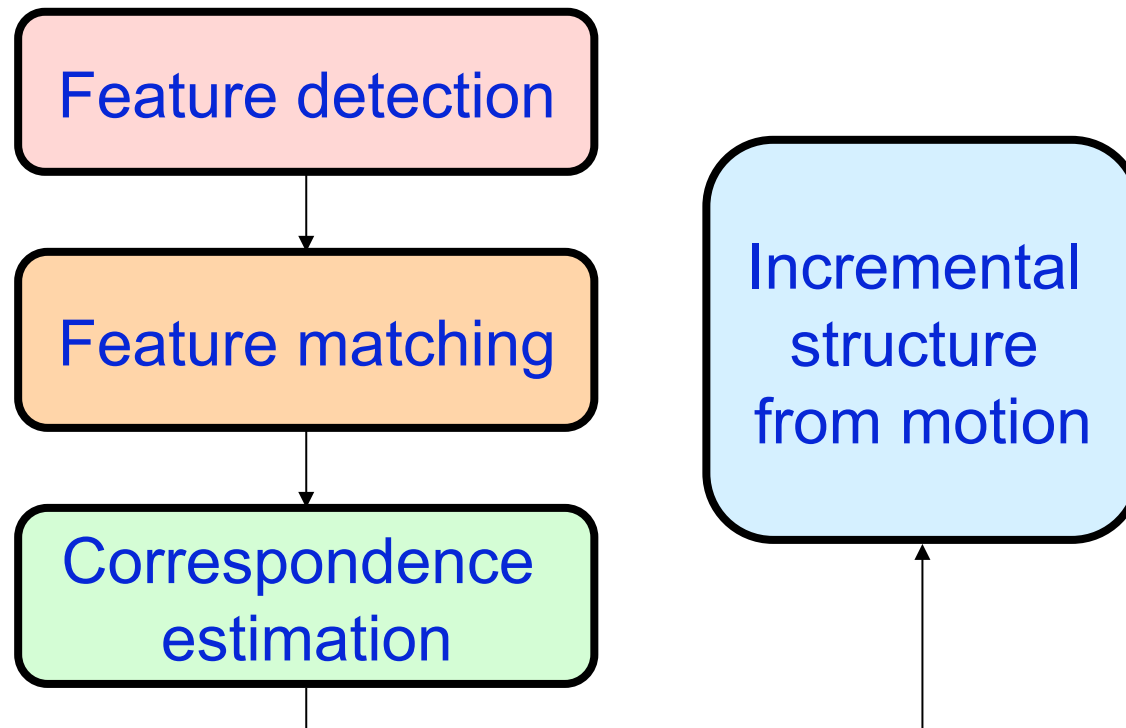
Photo Tourism overview



Scene reconstruction

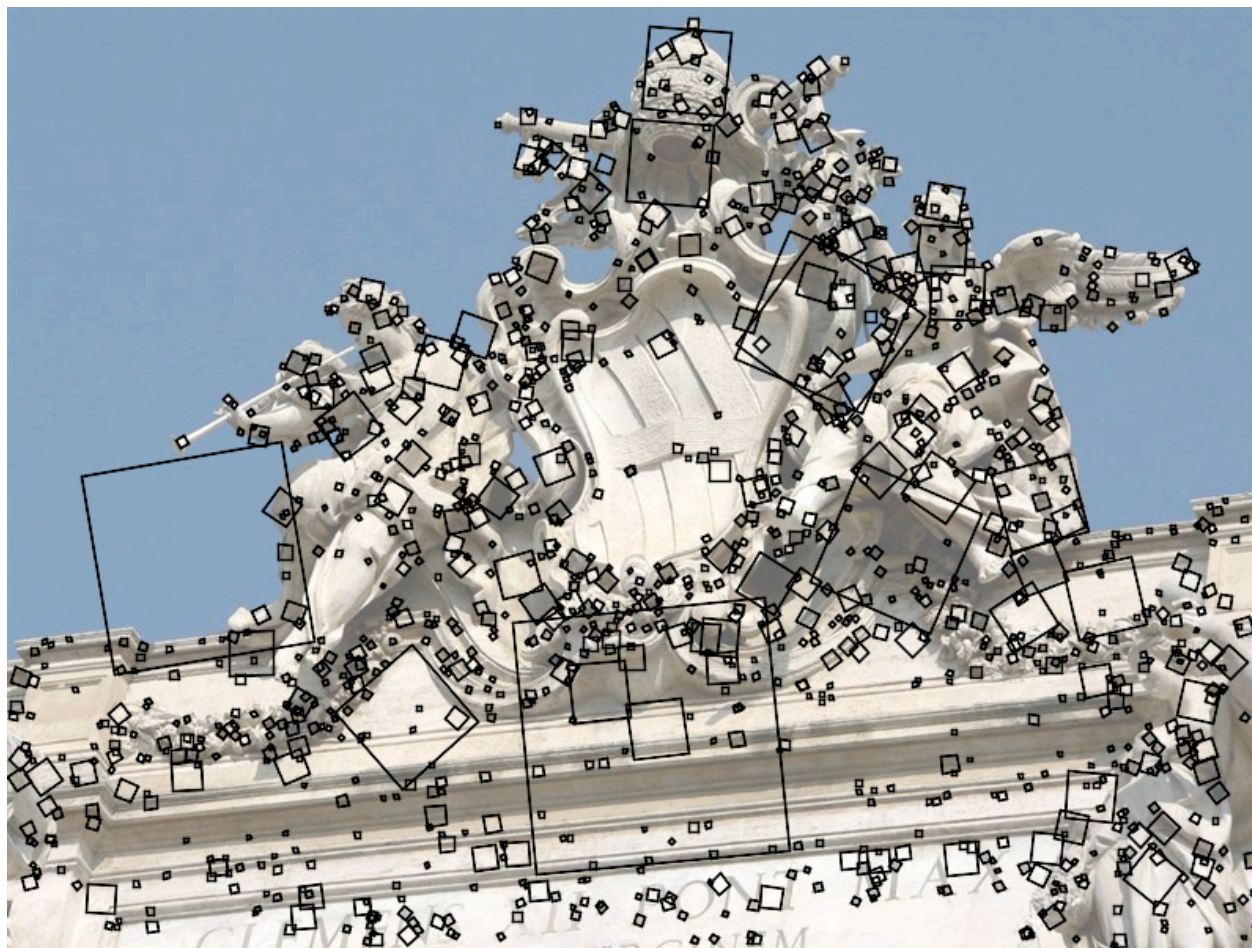
Automatically estimate

- position, orientation, and focal length of cameras
- 3D positions of feature points



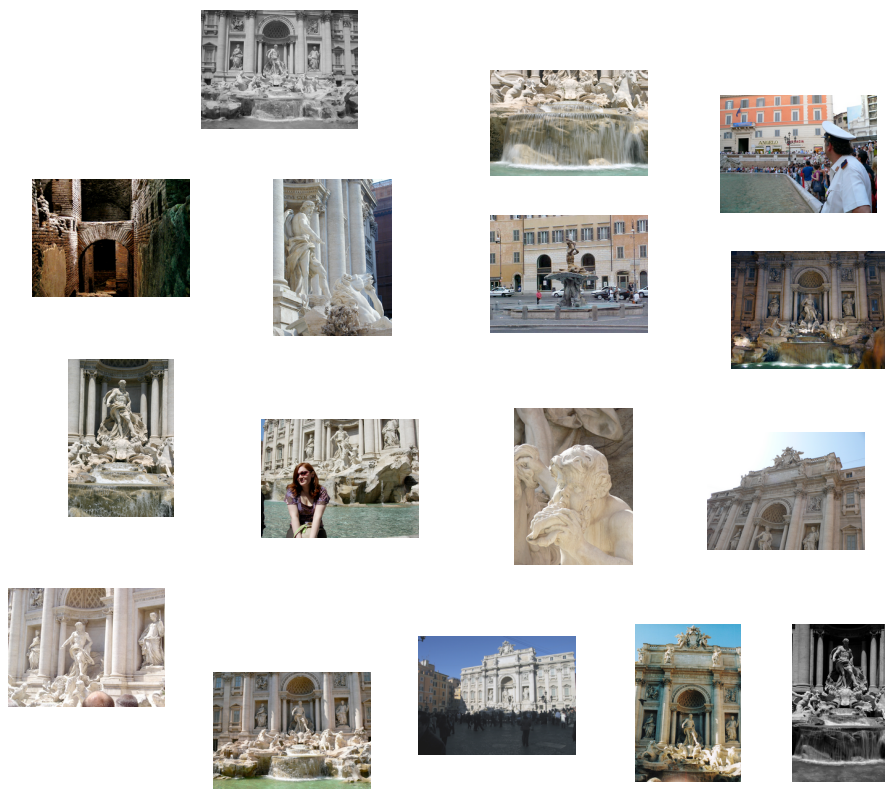
Feature detection

Detect features using SIFT [Lowe, IJCV 2004]



Feature detection

Detect features using SIFT [Lowe, IJCV 2004]



Feature detection

Detect features using SIFT [Lowe, IJCV 2004]



Feature matching

Complexity of matching:

Unfortunately, even with a well optimized implementation of the matching procedure described above, it is not practical to match all pairs of images in our corpus. For a corpus of 100,000 images, this translates into 5,000,000,000 pairwise comparisons, which with 500 cores operating at 10 image pairs per second per core would require about 11.5 days to match. Furthermore, this does not even take into account the network transfers required for all cores to have access to all the SIFT feature data for all images.

From Agarwal et al. “Building Rome in a Day”, ICCV’09

Feature matching

Obtain candidate pairs of images to match using visual vocabulary matching based on k-means tree

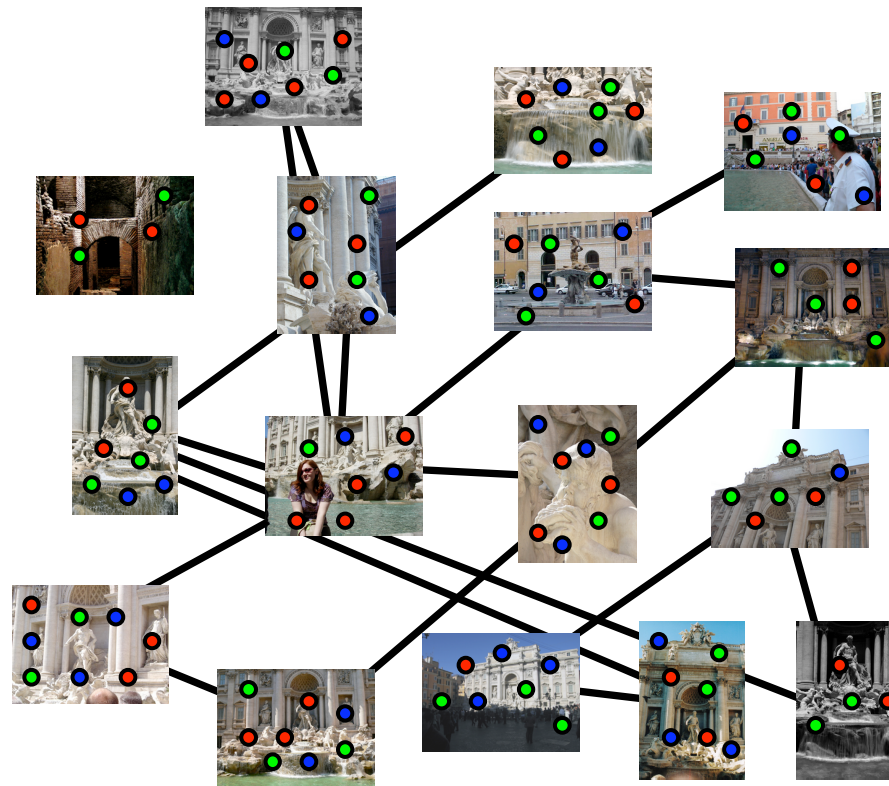


Figure: N. Snavely

Feature matching

Match features between candidate pairs using K-d trees built on SIFT descriptors.

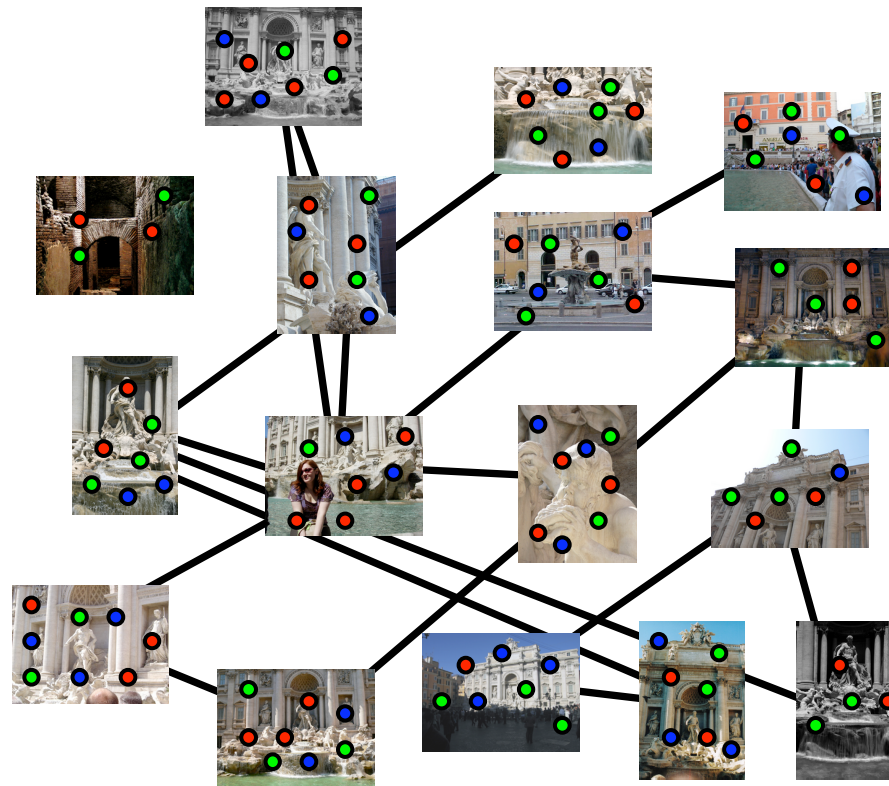
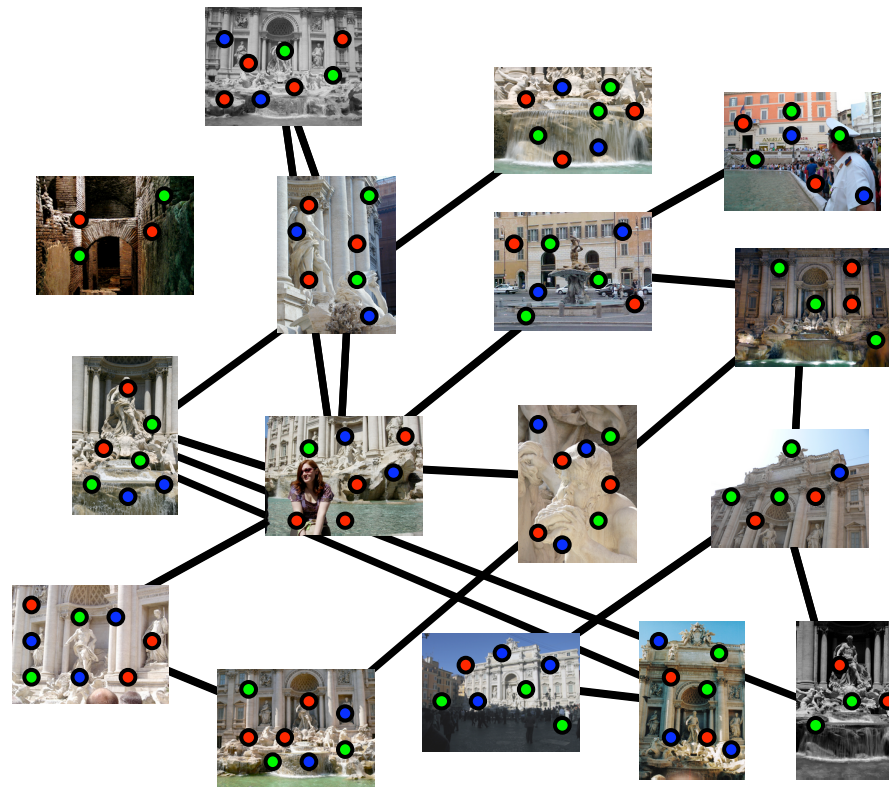


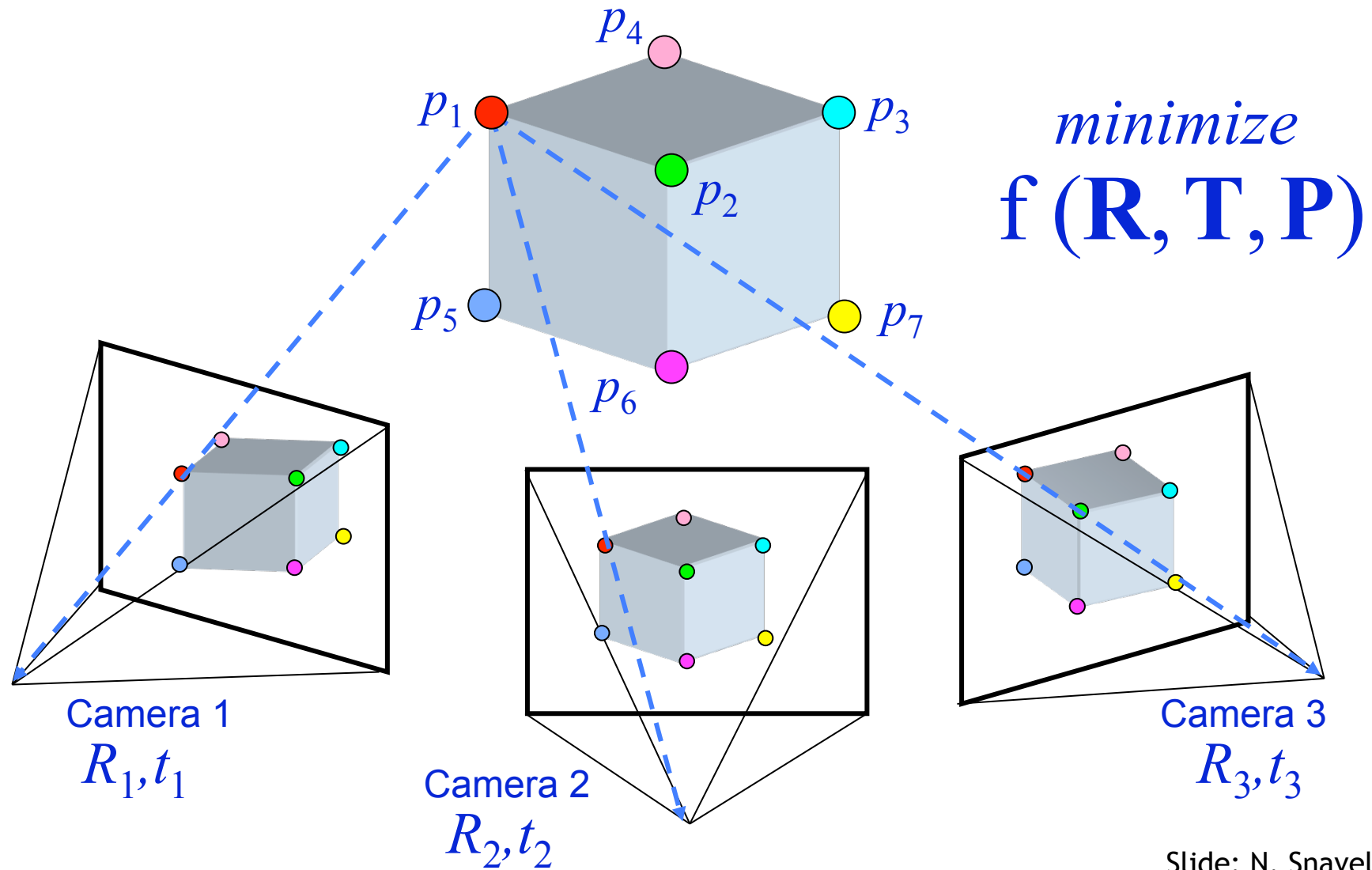
Figure: N. Snavely

Feature matching

Refine matching using RANSAC [Fischler & Bolles 1987]
to estimate fundamental matrices between pairs



Structure from motion (R. Keriven's class)



Example of the final 3D point cloud and cameras

57,845 downloaded images, 11,868 registered images. This video: 4,619 images.

