

Efficient visual search of local features

Cordelia Schmid

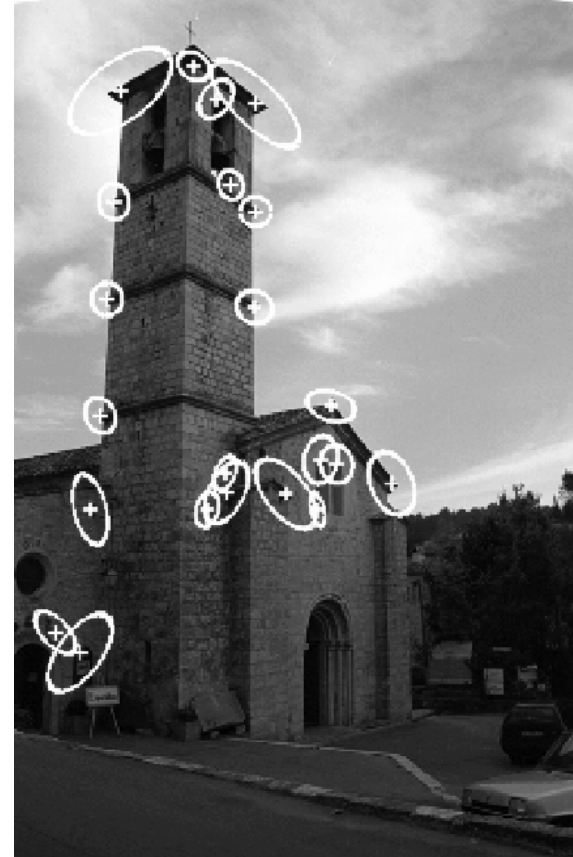
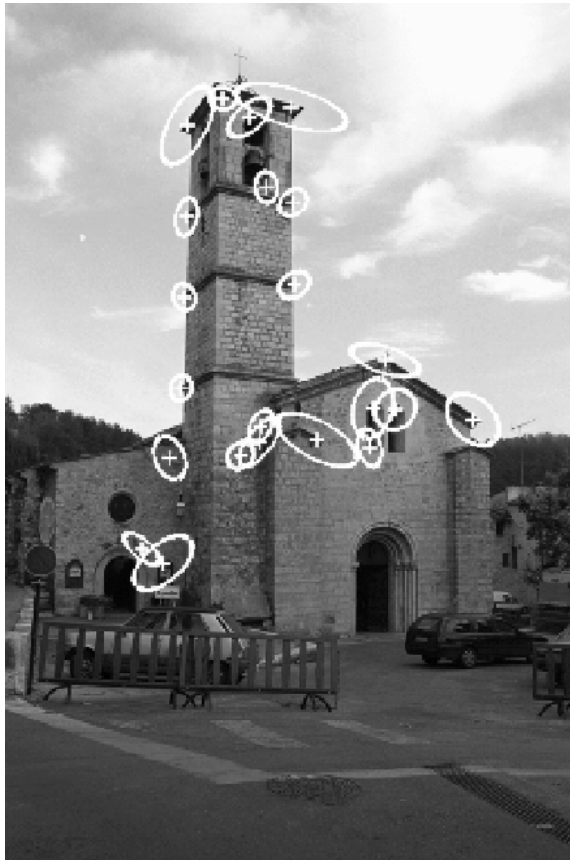
Visual search



change in viewing angle

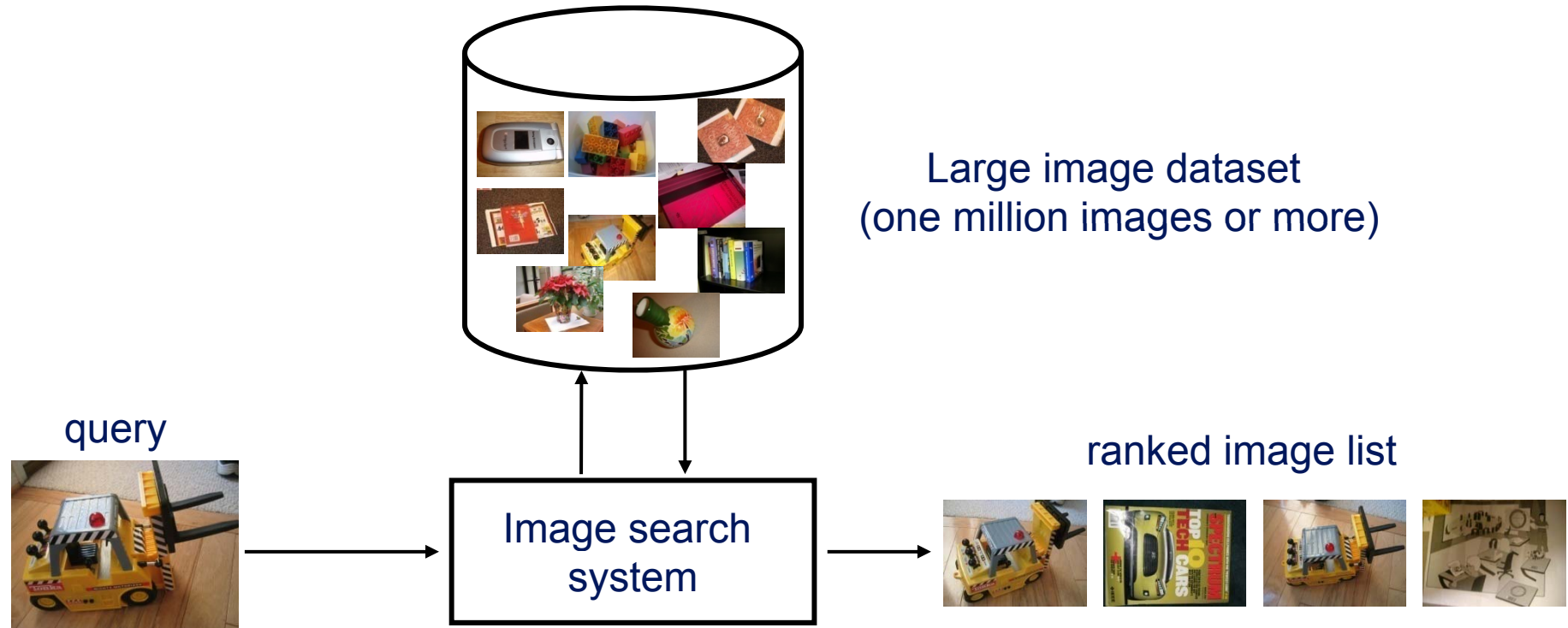


Matches



22 correct matches

Image search system for large datasets

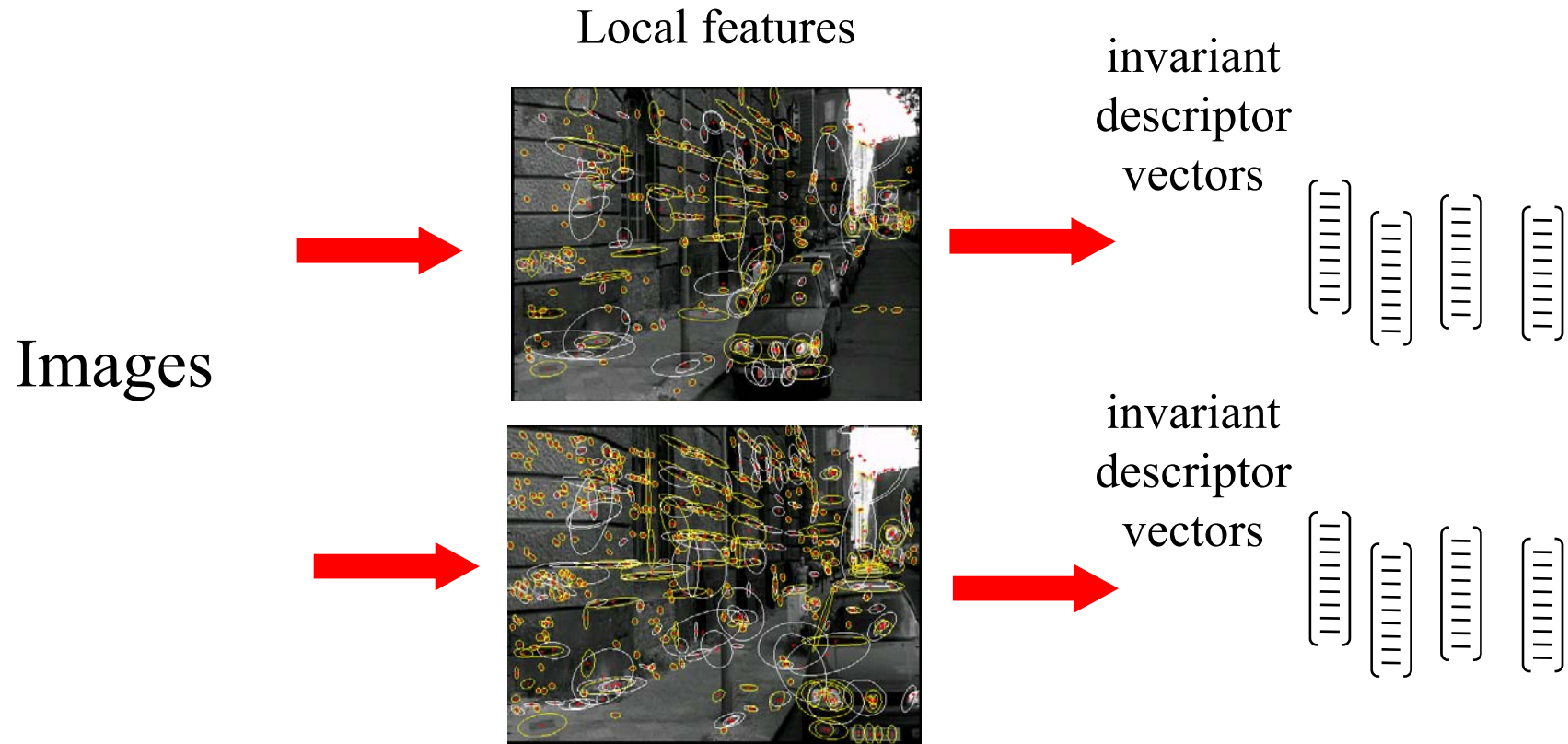


- **Issues** for very large databases
 - to reduce the query time
 - to reduce the storage requirements

Two strategies

1. Efficient approximate nearest neighbour search on local feature descriptors.
2. Quantize descriptors into a “visual vocabulary” and use efficient techniques from text retrieval.
(Bag-of-words representation)

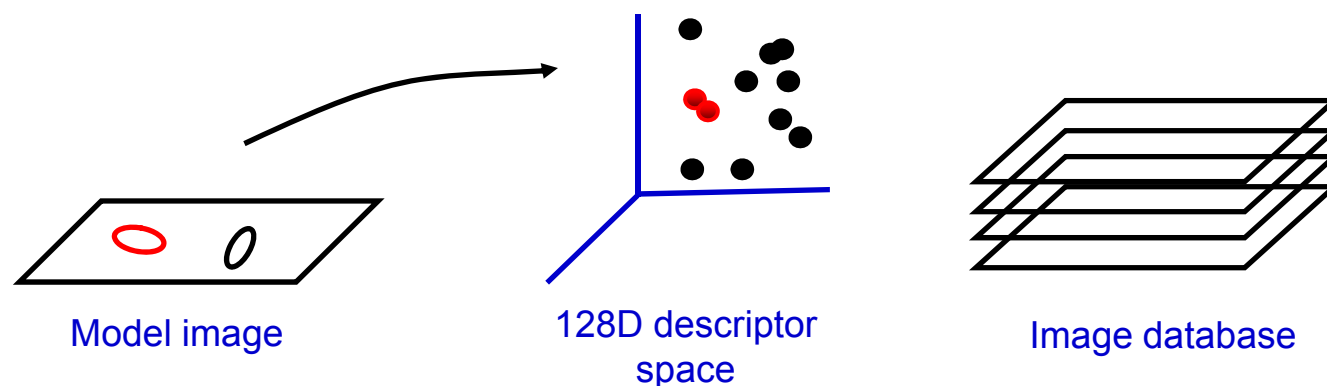
Strategy 1: Efficient approximate NN search



1. Compute local features in each image independently
2. Describe each feature by a descriptor vector
3. Find nearest neighbour vectors between query and database
4. Rank matched images by number of (tentatively) corresponding regions
5. Verify top ranked images based on spatial consistency

Finding nearest neighbour vectors

Establish correspondences between query 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, implementation in C)
- Memory footprint: $1000 \cdot 128 = 128\text{kB}$ / image

# of images	CPU time	Memory req.
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)

Nearest-neighbor matching

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

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

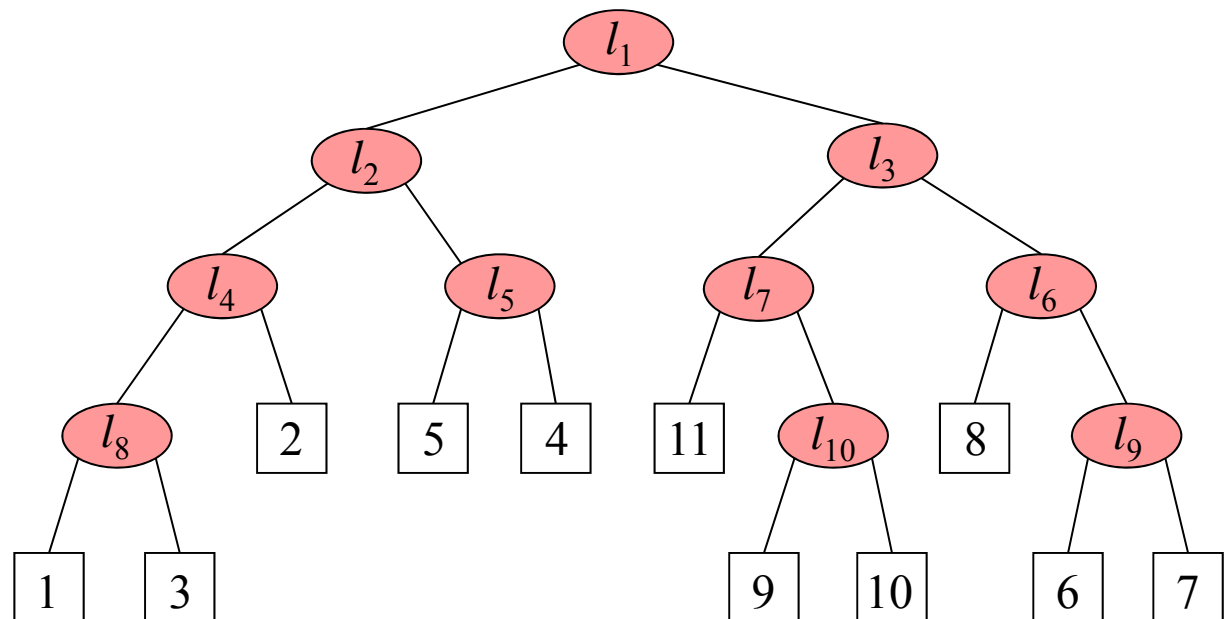
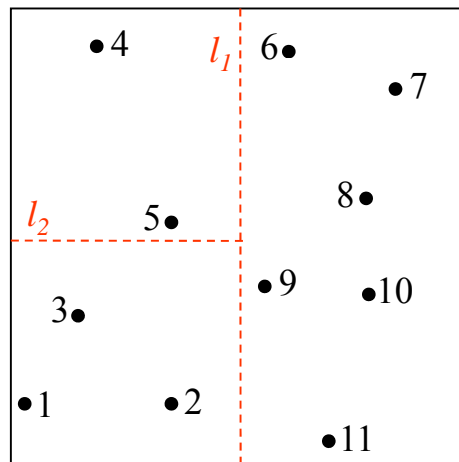
where \mathbf{x}_i are features in database images.

Nearest-neighbour matching is the major computational bottleneck

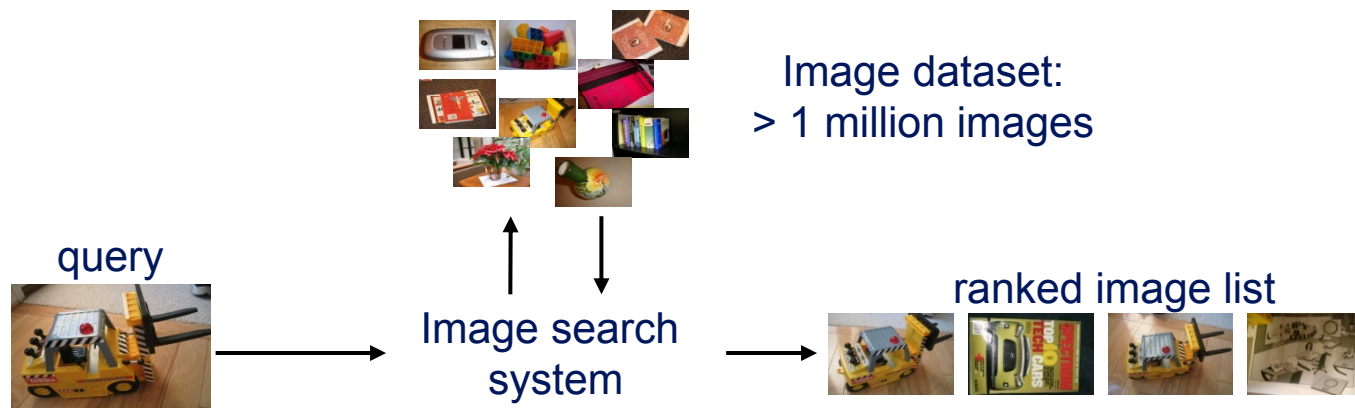
- Linear search performs dn operations for n features in the database and d dimensions
- No exact methods are faster than linear search for $d > 10$
- Approximate methods can be much faster, but at the cost of missing some correct matches. Failure rate gets worse for large datasets.

K-d tree

- K-d tree is a **binary tree** data structure for organizing a set of points
- Each internal node is associated with an **axis aligned hyper-plane** splitting its associated points into two sub-trees.
- Dimensions with high variance are chosen first.
- Position of the splitting hyper-plane is chosen as the mean/median of the projected points – balanced tree.

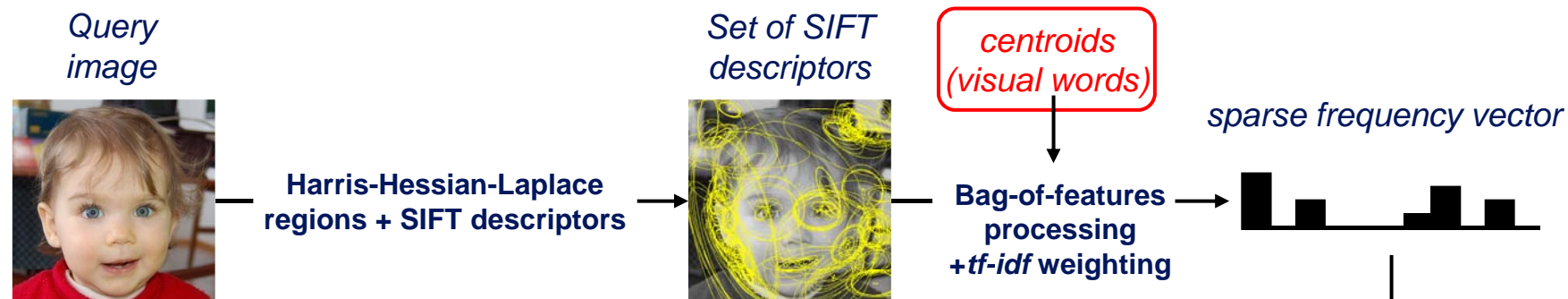


Large scale object/scene recognition



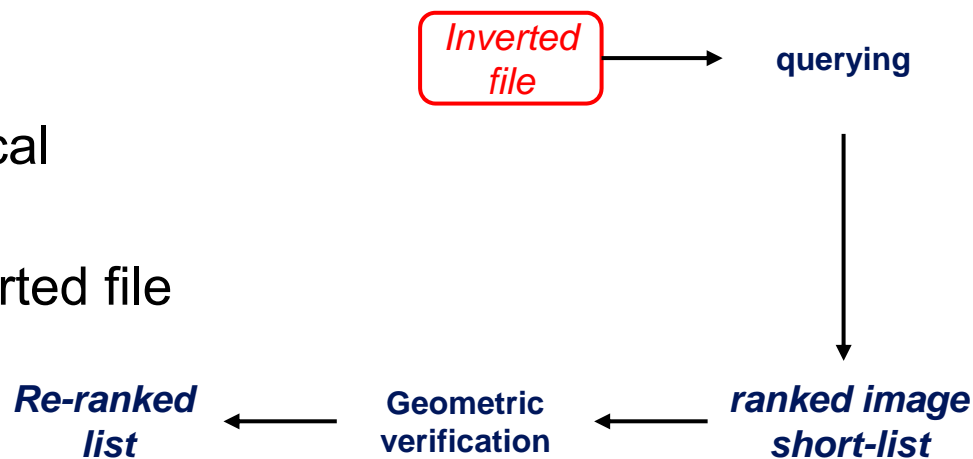
- Each image described by approximately 2000 descriptors
 - $2 * 10^9$ descriptors to index for one million images!
- Database representation in RAM:
 - Size of descriptors : 1 TB, search+memory intractable

Bag-of-features [Sivic&Zisserman'03]



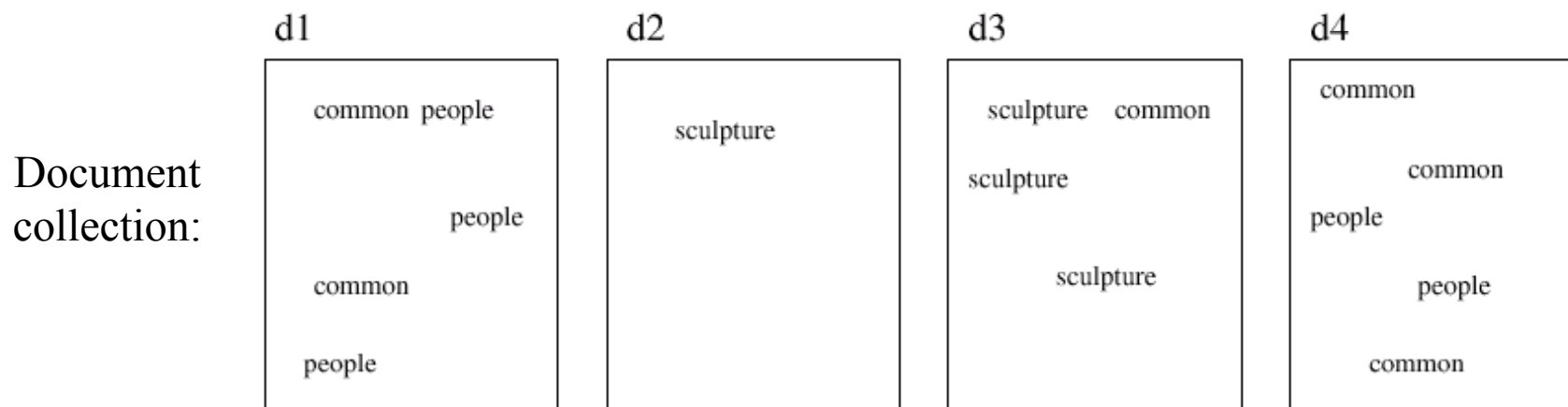
- “visual words”:
 - 1 “word” (index) per local descriptor
 - only images ids in inverted file

=> 8 GB fits!



[Chum & al. 2007]

Indexing text with inverted files

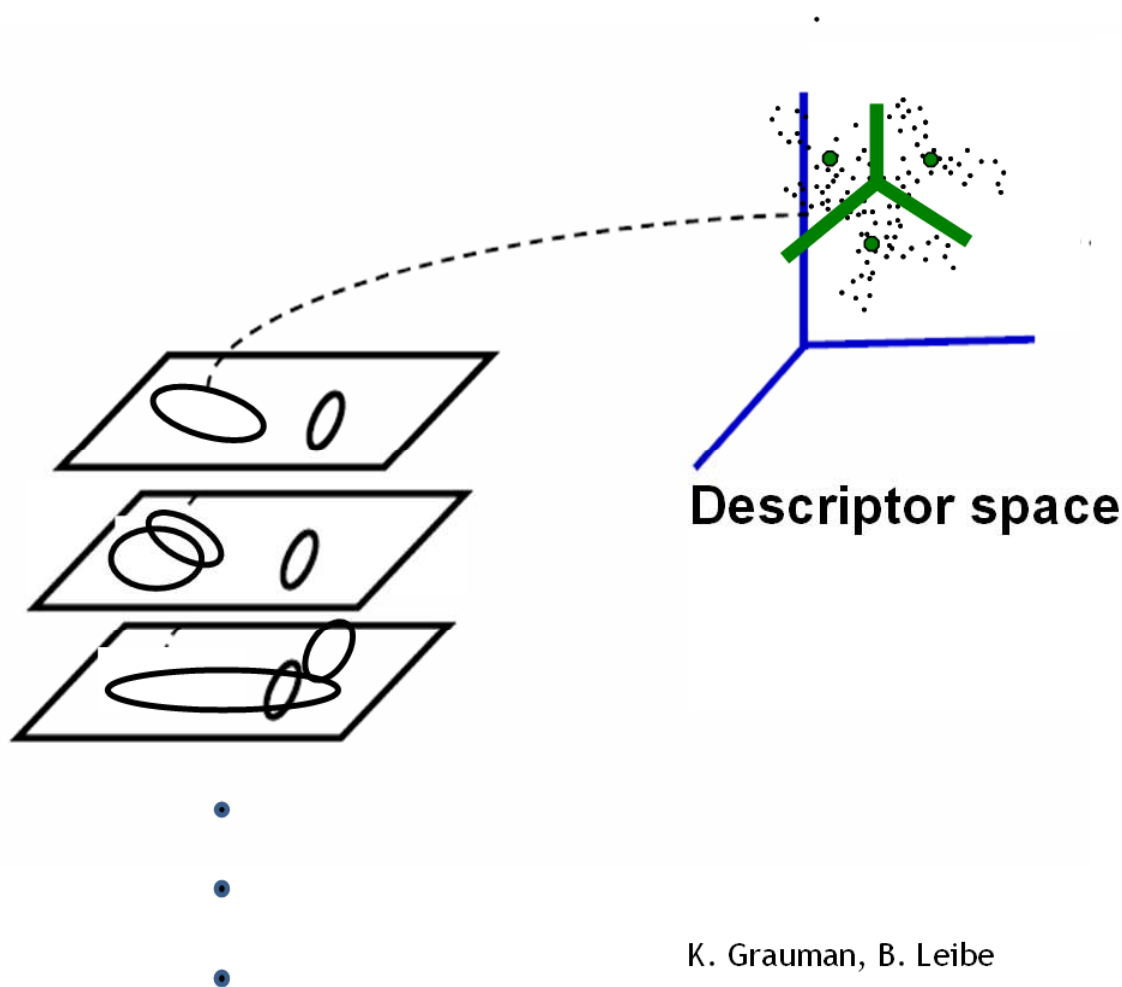


Inverted file:	Term	List of hits (occurrences in documents)
	People	[d1:hit hit hit], [d4:hit hit] ...
	Common	[d1:hit hit], [d3: hit], [d4: hit hit hit] ...
	Sculpture	[d2:hit], [d3: hit hit hit] ...

Need to map feature descriptors to “visual words”

Visual words: main idea

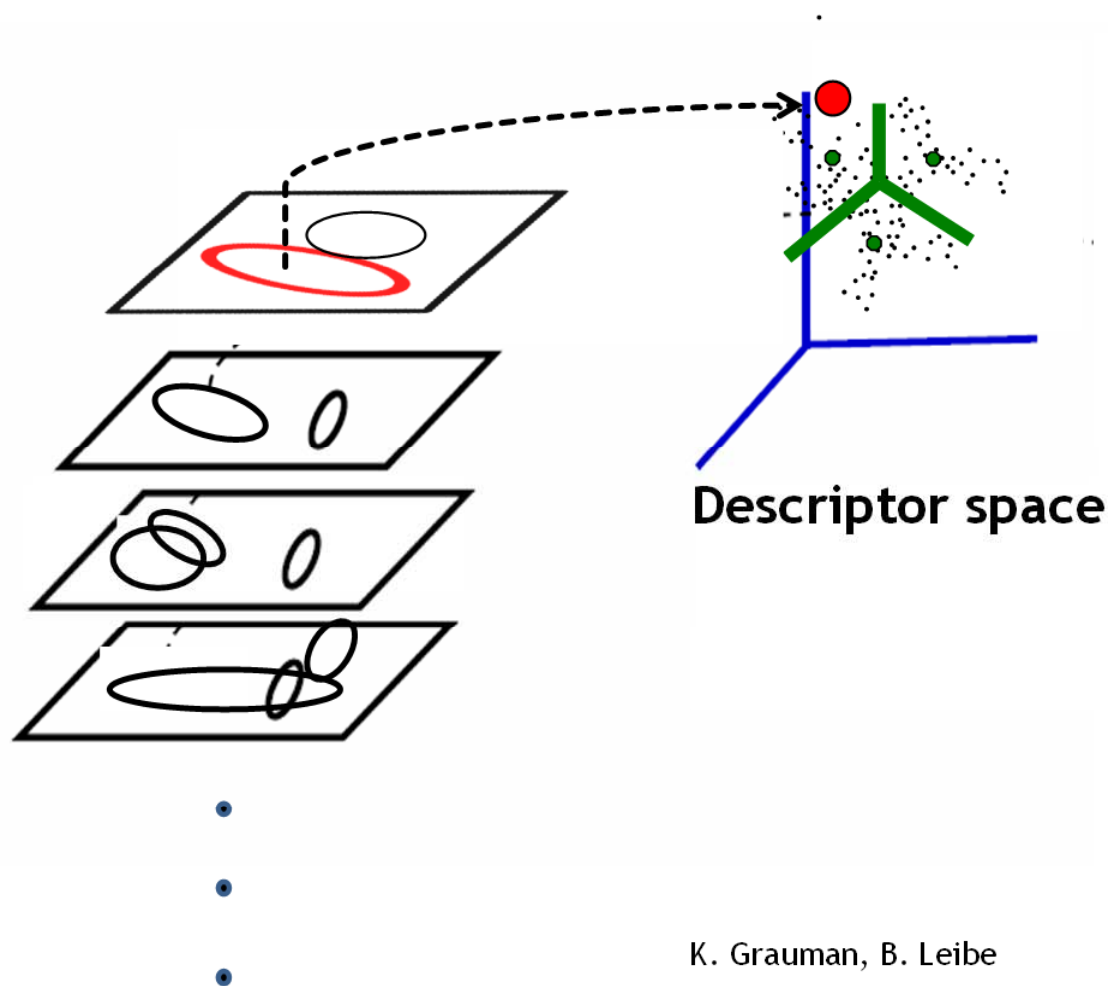
Map high-dimensional descriptors to tokens/words by quantizing the feature space



- 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

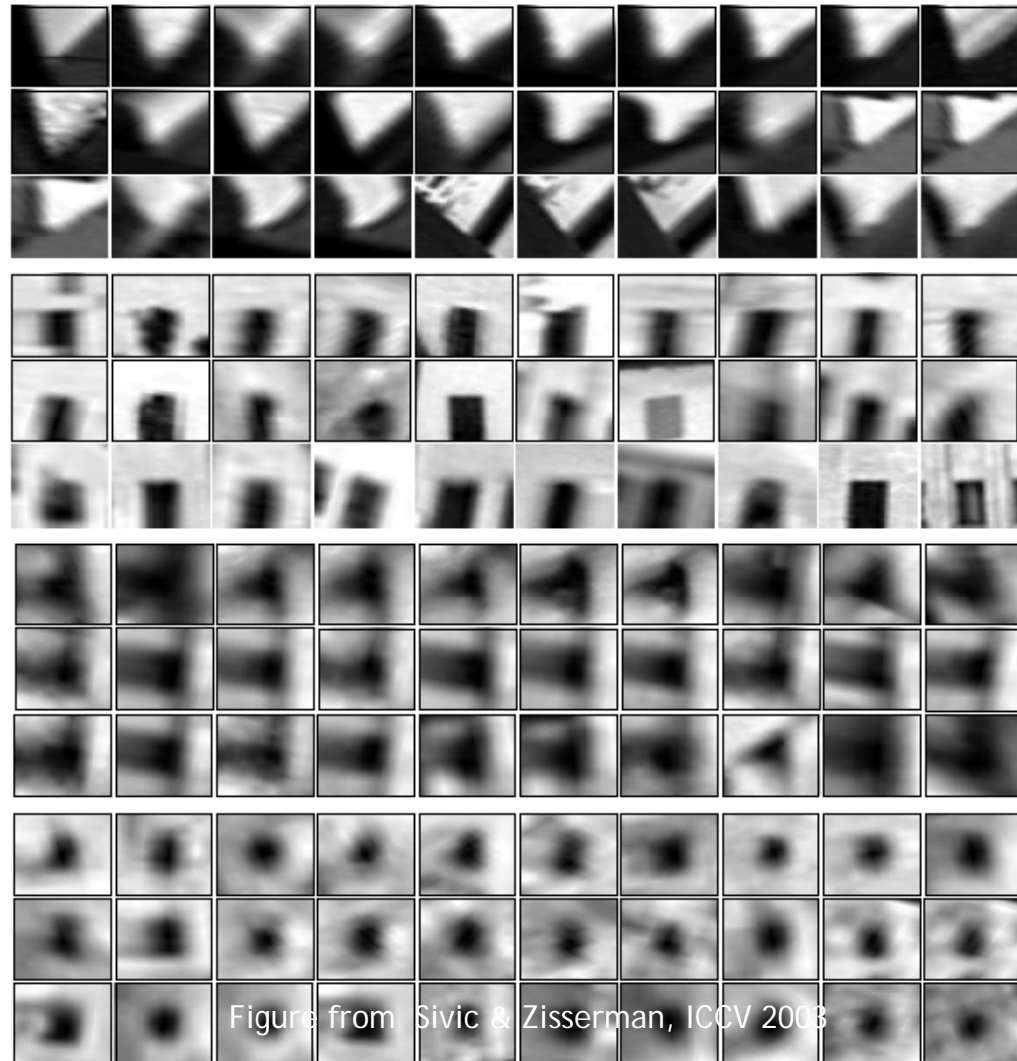


Figure from Sivic & Zisserman, ICCV 2003

Inverted file index for images comprised of visual words



frame #5



frame #10

Word List of image
number numbers

1	→	5, 10, ...
2	→	10, ...
...		...

- Score each image by the number of common visual words (tentative correspondences)
- Dot product between bag-of-features
- Fast for sparse vectors !

Inverted file index for images comprised of visual words

- Weighting with tf-idf score: weight visual words based on their frequency
 - Tf: normalized term (word) t_i frequency in a document d_j

$$tf_{ij} = n_{ij} / \sum_k n_{kj}$$

- Idf: inverse document frequency, total number of documents divided by number of documents containing the term t_i

$$idf_i = \log \frac{|D|}{|\{d : t_i \in d\}|}$$

Tf-Idf: $tf - idf_{ij} = tf_{ij} \cdot idf_i$

Visual words

- Map descriptors to words by quantizing the feature space
 - Quantize via k-means clustering to obtain visual words
 - Assign descriptor to closest visual word
- Bag-of-features as approximate nearest neighbor search

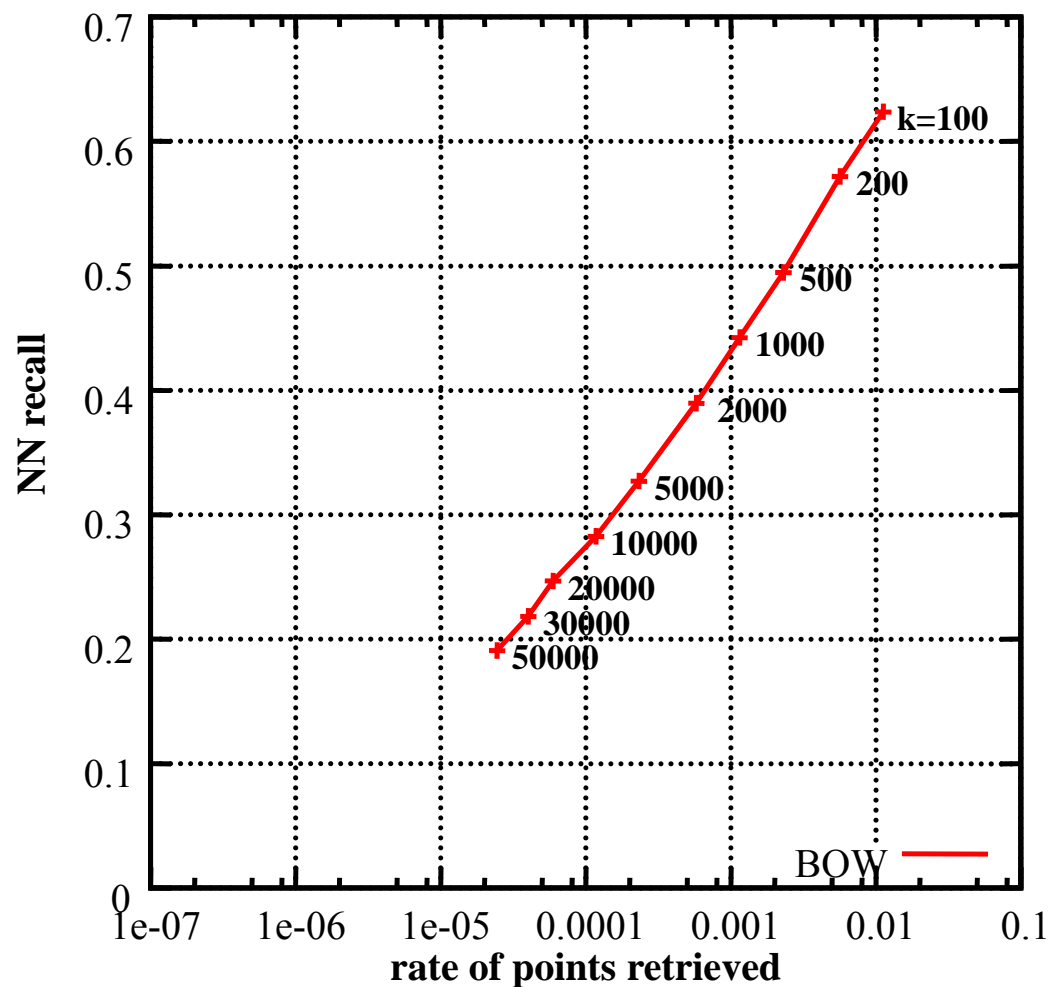
Bag-of-features matching function $f_q(x, y) = \delta_{q(x), q(y)}$

where $q(x)$ is a quantizer, i.e., assignment to visual word and $\delta_{a,b}$ is the Kronecker operator ($\delta_{a,b}=1$ iff $a=b$)

Approximate nearest neighbor search evaluation

- ANN algorithms usually returns a short-list of nearest neighbors
 - this short-list is supposed to contain the NN with high probability
 - exact search may be performed to re-order this short-list
- Proposed quality evaluation of ANN search: trade-off between
 - **Accuracy: NN recall** = probability that *the* NN is in this list
against
 - **Ambiguity removal** = proportion of vectors in the short-list
 - the lower this proportion, the more information we have about the vector
 - the lower this proportion, the lower the complexity if we perform exact search on the short-list
- ANN search algorithms usually have some parameters to handle this trade-off

ANN evaluation of bag-of-features



- ANN algorithms returns a list of potential neighbors

- Accuracy: NN recall** = probability that *the* NN is in this list

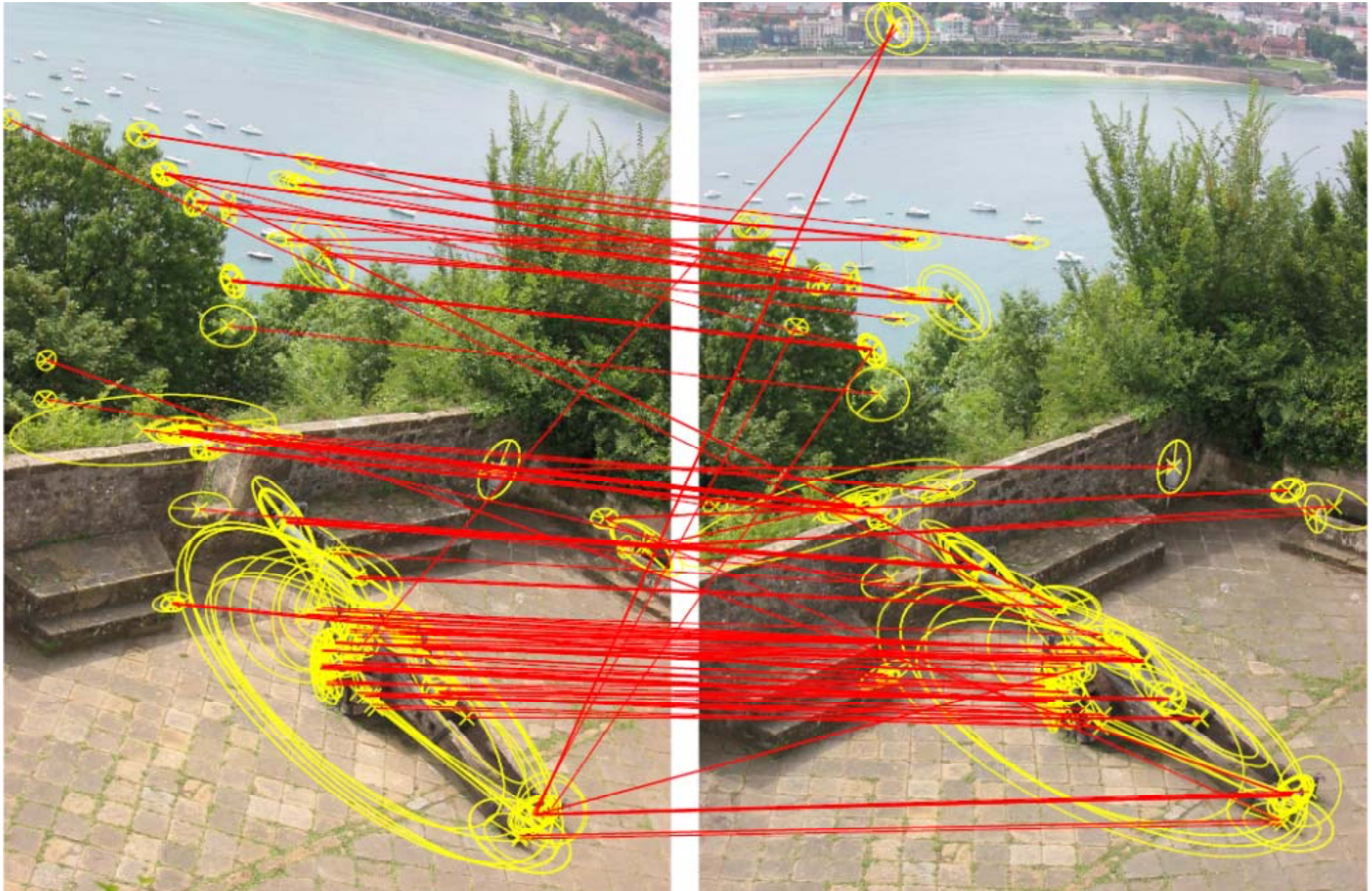
- Ambiguity removal:** = proportion of vectors in the short-list

- In BOF, this trade-off is managed by the number of clusters k

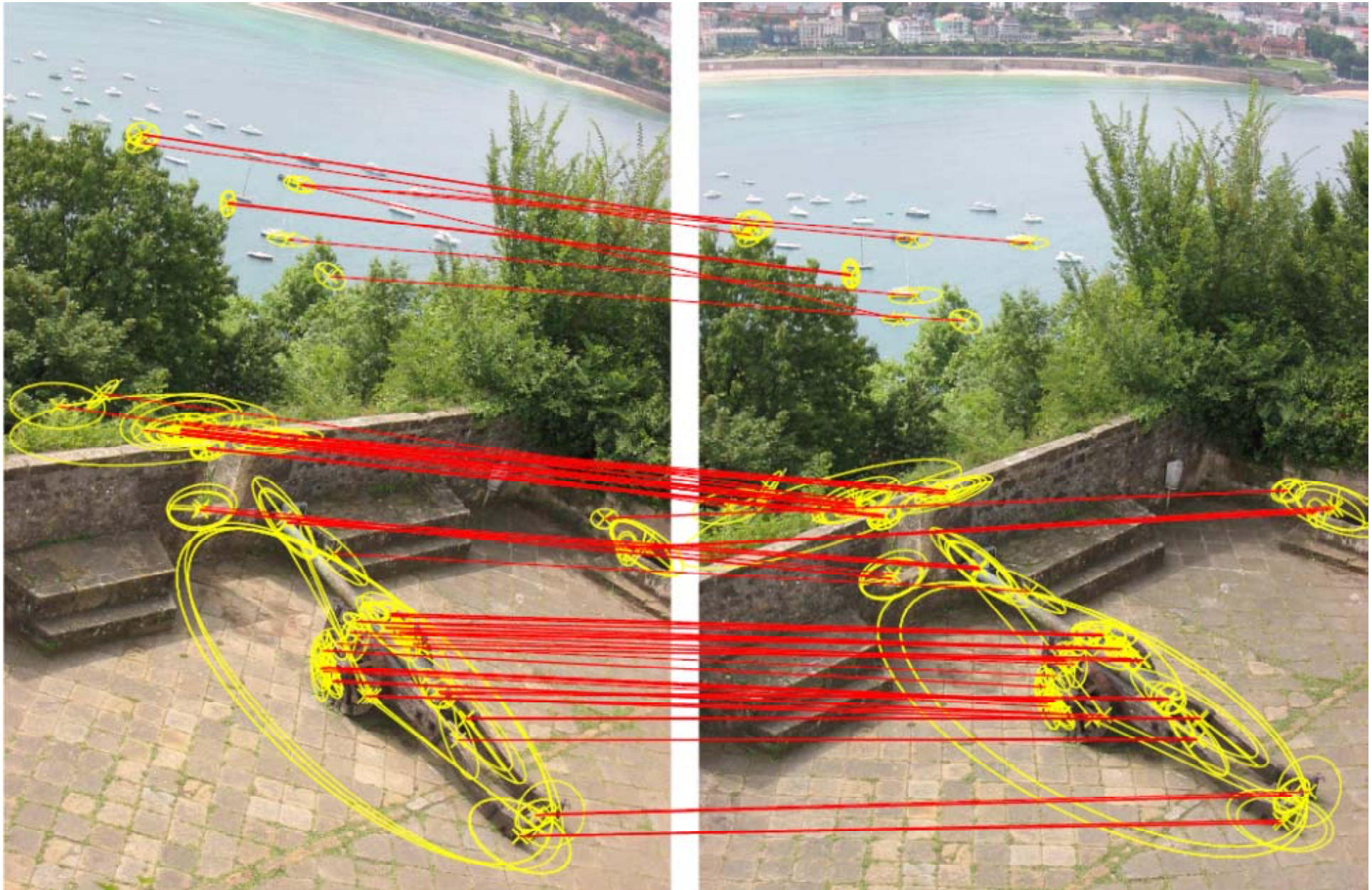
Vocabulary size

- The intrinsic matching scheme performed by BOF is weak
 - for a “small” visual dictionary: too many false matches
 - for a “large” visual dictionary: complexity, true matches are missed
- No good trade-off between “small” and “large” !
 - either the Voronoi cells are too big
 - or these cells can’t absorb the descriptor noise
 - intrinsic approximate nearest neighbor search of BOF is not sufficient

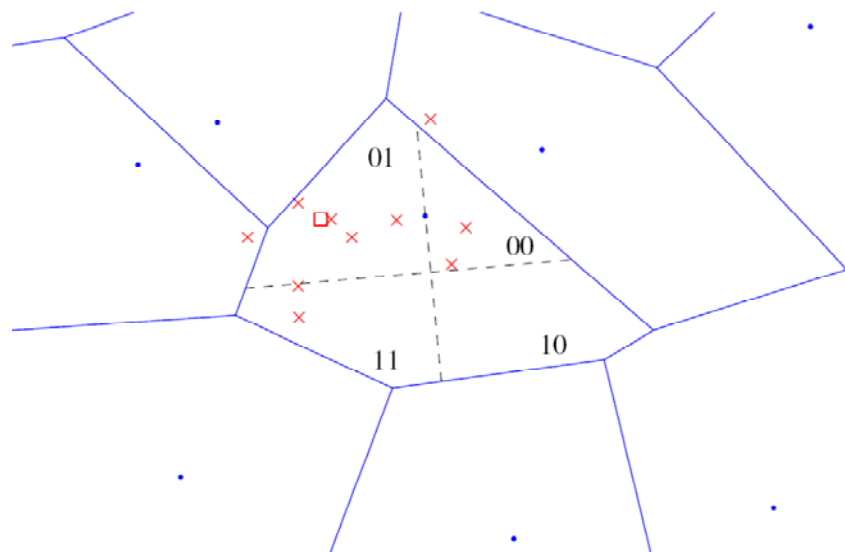
20K visual word: false matches



200K visual word: good matches missed



Hamming Embedding [Jegou et al. ECCV'08]



Representation of a descriptor x

- Vector-quantized to $q(x)$ as in standard BOF
- + **short binary vector $b(x)$** for an additional localization in the Voronoi cell

Two descriptors x and y match iff

$$f_{\text{HE}}(x, y) = \begin{cases} (\text{tf-idf}(q(x)))^2 & \text{if } q(x) = q(y) \\ & \text{and } h(b(x), b(y)) \leq h_t \quad \text{where } h(a, b) \text{ Hamming distance} \\ 0 & \text{otherwise} \end{cases}$$

Hamming Embedding

- Nearest neighbors for Hamming distance \approx those for Euclidean distance
→ a metric in the embedded space reduces dimensionality curse effects
- Efficiency
 - Hamming distance = very few operations
 - Fewer random memory accesses: 3 x faster than BOF with same dictionary size!

Hamming Embedding

- **Off-line** (given a quantizer)

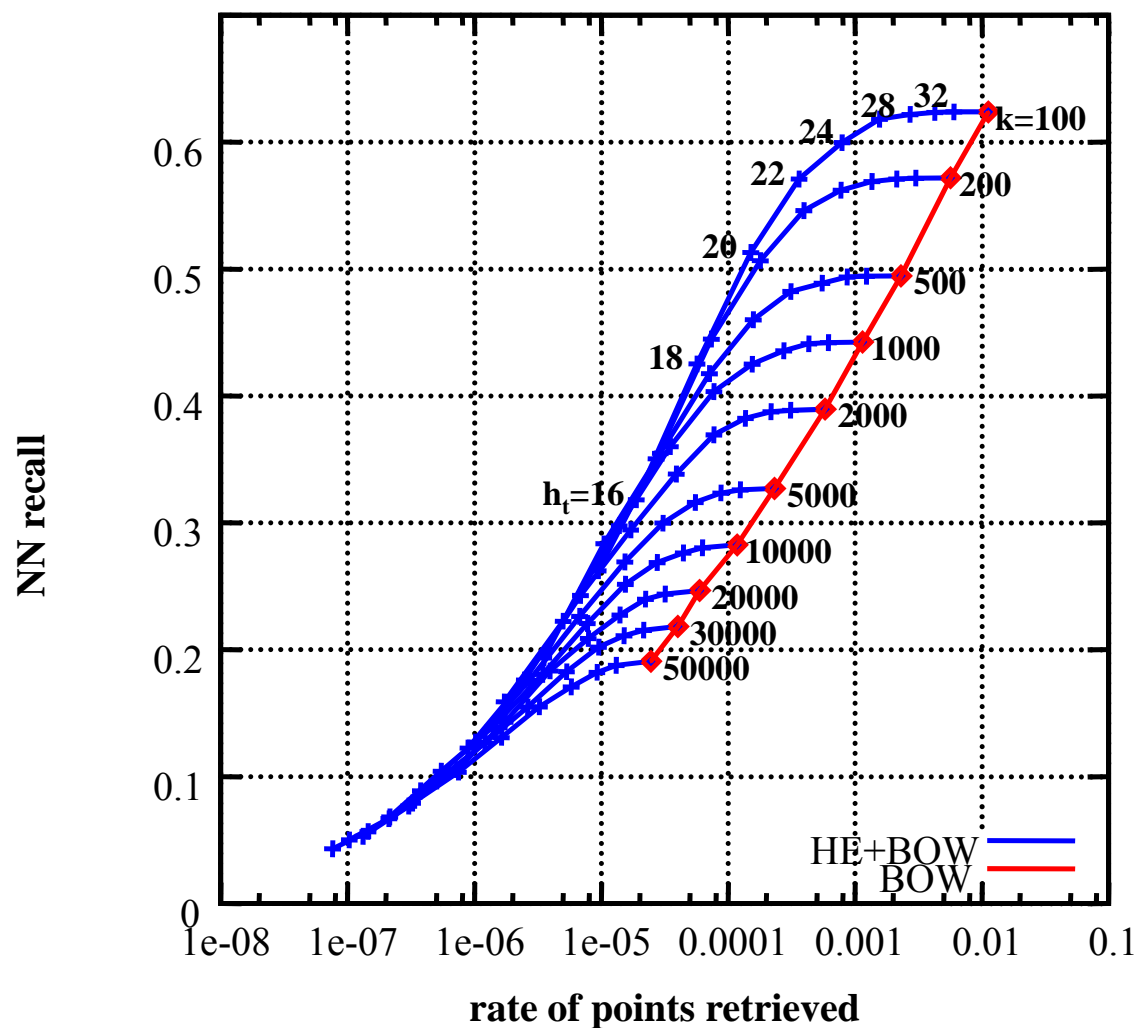
- draw an orthogonal projection matrix P of size $d_b \times d$
- this defines d_b random projection directions
- for each Voronoi cell and projection direction, compute the median value for a learning set

- **On-line**: compute the binary signature $b(x)$ of a given descriptor

- project x onto the projection directions as $z(x) = (z_1, \dots, z_{d_b})$
- $b_i(x) = 1$ if $z_i(x)$ is above the learned median value, otherwise 0

ANN evaluation of Hamming Embedding

0.7



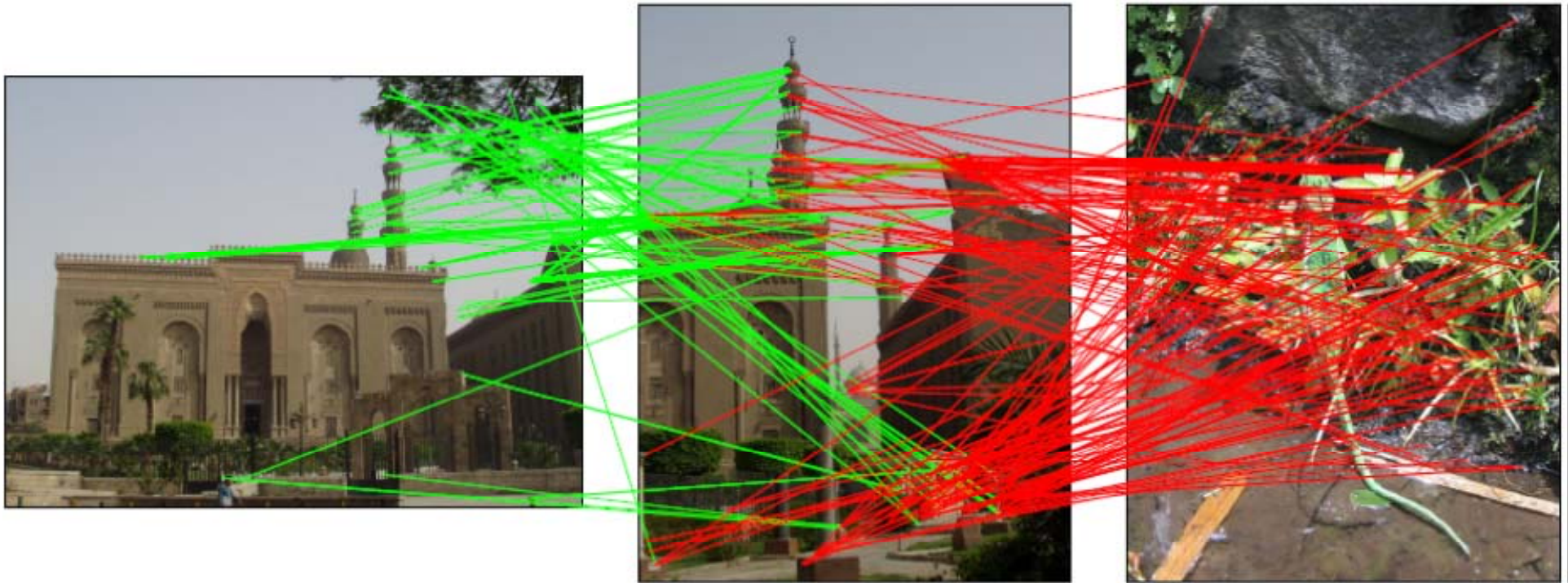
compared to BOW: at least 10 times less points in the short-list for the same level of accuracy

Hamming Embedding provides a much better trade-off between recall and ambiguity removal

Matching points - 20k word vocabulary

201 matches

240 matches



Many matches with the non-corresponding image!

Matching points - 200k word vocabulary

69 matches

35 matches

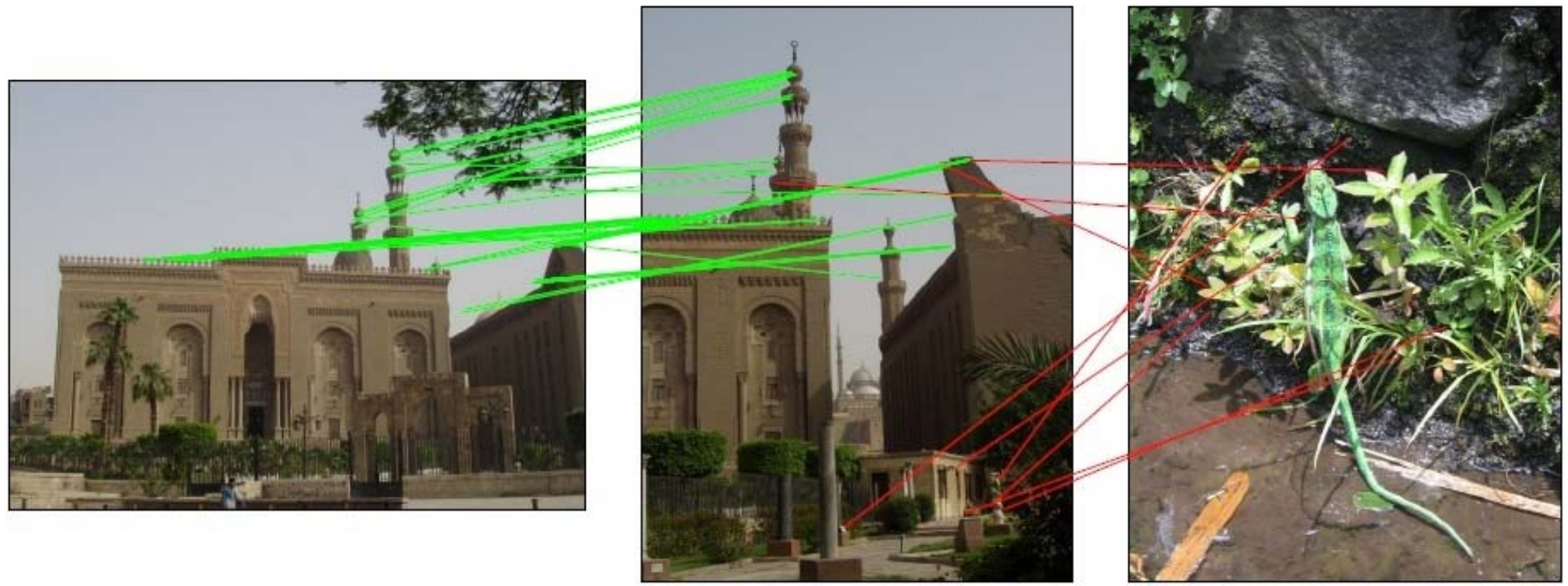


Still many matches with the non-corresponding one

Matching points - 20k word vocabulary + HE

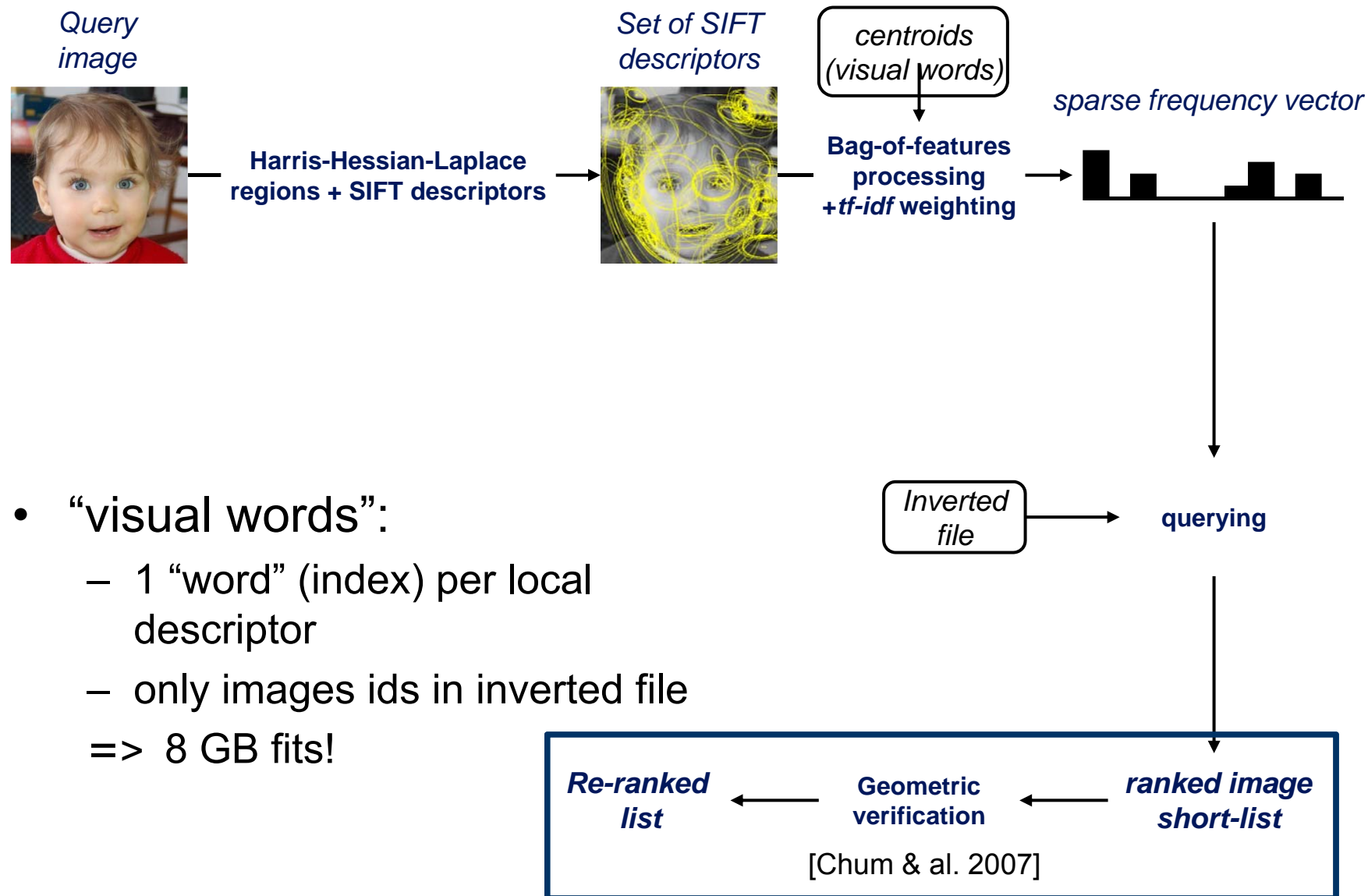
83 matches

8 matches



10x more matches with the corresponding image!

Bag-of-features [Sivic&Zisserman'03]



- “visual words”:
 - 1 “word” (index) per local descriptor
 - only images ids in inverted file
- => 8 GB fits!

Geometric verification

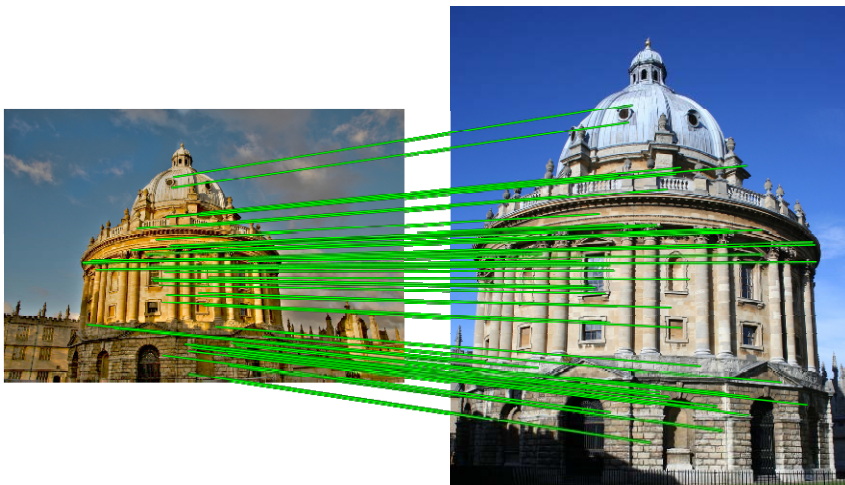
Use the **position** and **shape** of the underlying features to improve retrieval quality



Both images have many matches – which is correct?

Geometric verification

We can measure **spatial consistency** between the query and each result to improve retrieval quality



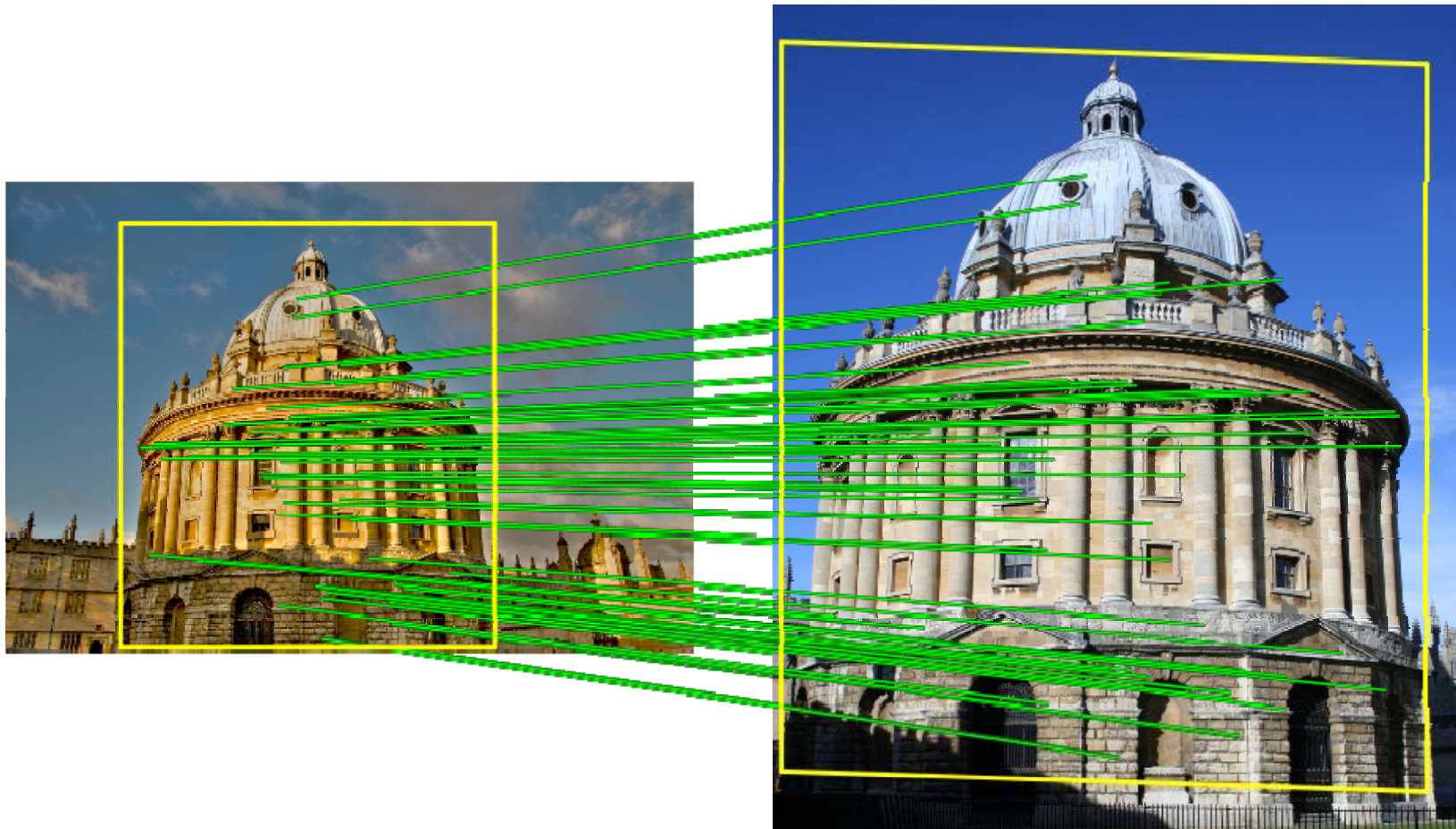
Many spatially consistent matches – **correct result**



Few spatially consistent matches – **incorrect result**

Geometric verification

Gives **localization** of the object

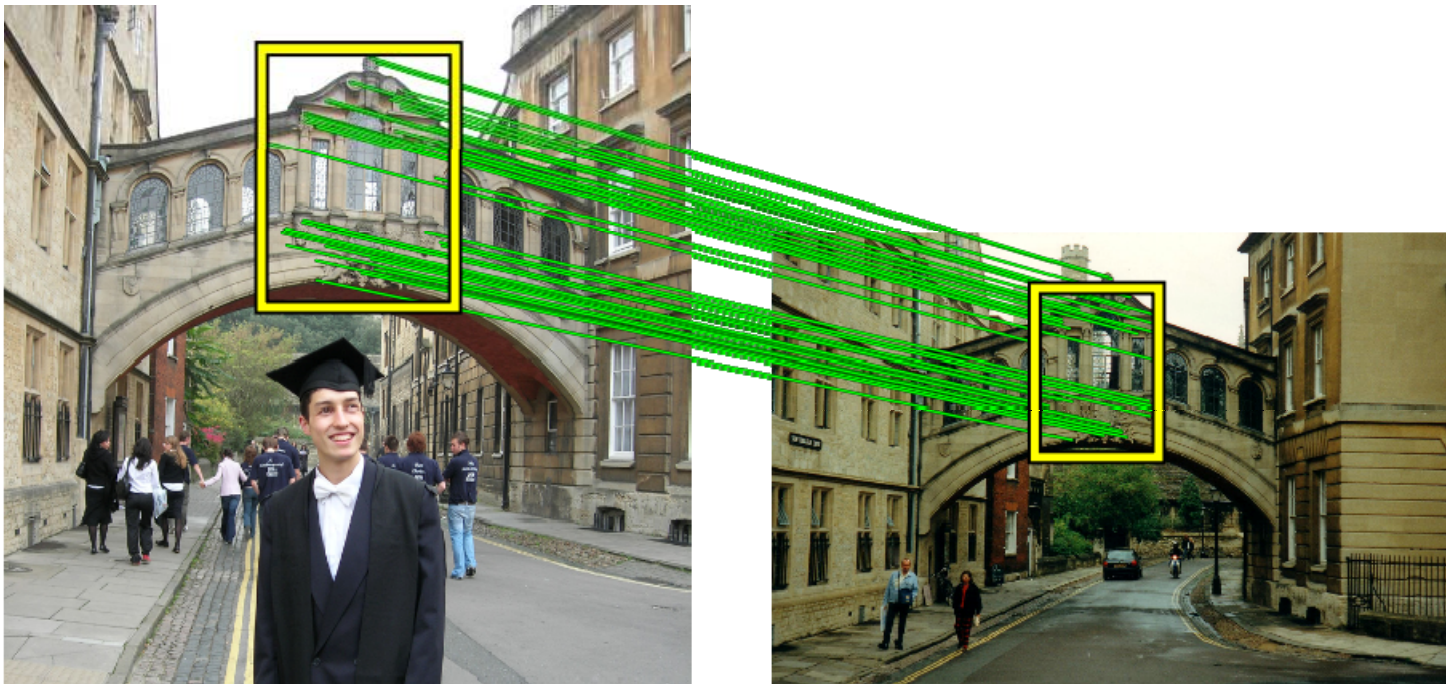


Geometric verification

- Remove outliers, matches contain a high number of incorrect ones
- Estimate geometric transformation
- Robust strategies
 - RANSAC
 - Hough transform

Example: estimating 2D affine transformation

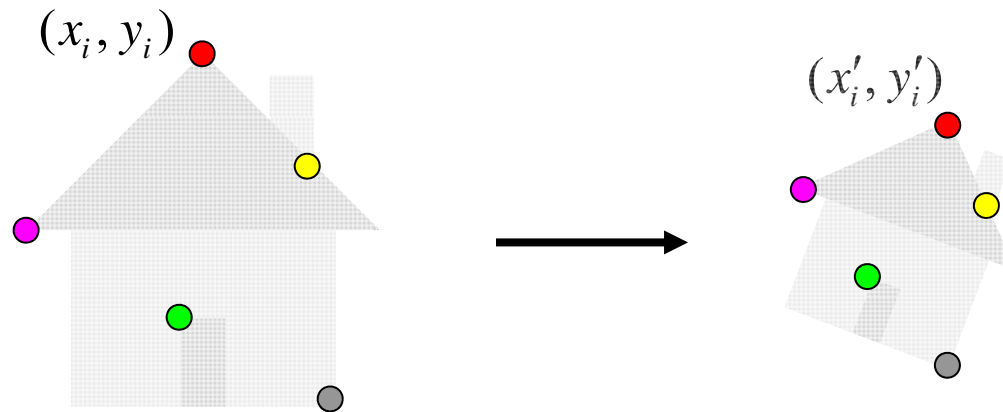
- Simple fitting procedure (linear least squares)
- Approximates viewpoint changes for roughly planar objects and roughly orthographic cameras
- Can be used to initialize fitting for more complex models



Matches consistent with an affine transformation

Fitting an affine transformation

Assume we know the correspondences, how do we get the transformation?



$$\begin{bmatrix} x'_i \\ y'_i \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

$$\begin{bmatrix} \dots & \dots & \dots & \dots & \dots & \dots \\ x_i & y_i & 0 & 0 & 1 & 0 \\ 0 & 0 & x_i & y_i & 0 & 1 \\ \dots & \dots & \dots & \dots & \dots & \dots \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} \dots \\ x'_i \\ y'_i \\ \dots \end{bmatrix}$$

Fitting an affine transformation

$$\begin{bmatrix} & & \text{L} & & & & m_1 \\ x_i & y_i & 0 & 0 & 1 & 0 & m_2 \\ 0 & 0 & x_i & y_i & 0 & 1 & m_3 \\ & & \text{L} & & & & m_4 \\ & & & & & & t_1 \\ & & & & & & t_2 \end{bmatrix} = \begin{bmatrix} \text{L} \\ x'_i \\ y'_i \\ \text{L} \end{bmatrix}$$

Linear system with six unknowns

Each match gives us two linearly independent equations: need at least three to solve for the transformation parameters

Comparison

Hough Transform

Advantages

- Can handle high percentage of outliers (>95%)
- Extracts groupings from clutter in linear time

Disadvantages

- Quantization issues
- Only practical for small number of dimensions (up to 4)

Improvements available

- Probabilistic Extensions
- Continuous Voting Space
- Can be generalized to arbitrary shapes and objects

RANSAC

Advantages

- General method suited to large range of problems
- Easy to implement
- “Independent” of number of dimensions

Disadvantages

- Basic version only handles moderate number of outliers (<50%)

Many variants available, e.g.

- PROSAC: Progressive RANSAC [Chum05]
- Preemptive RANSAC [Nister05]

Geometric verification – example

1. Query



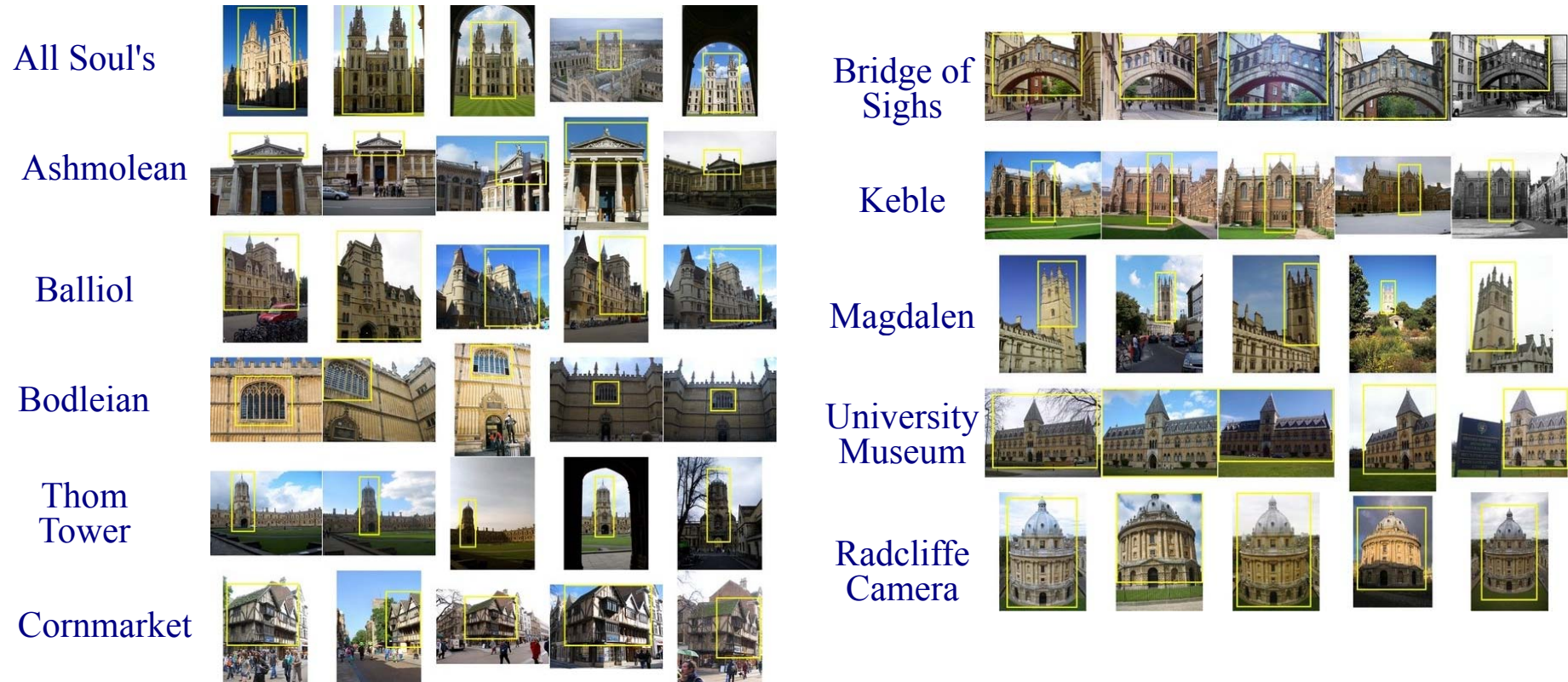
2. Initial retrieval set (bag of words model)



3. Spatial verification (re-rank on # of inliers)



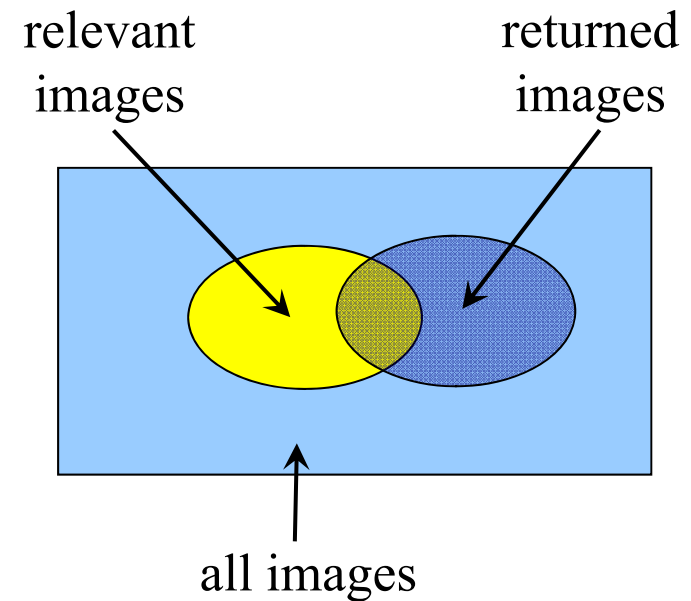
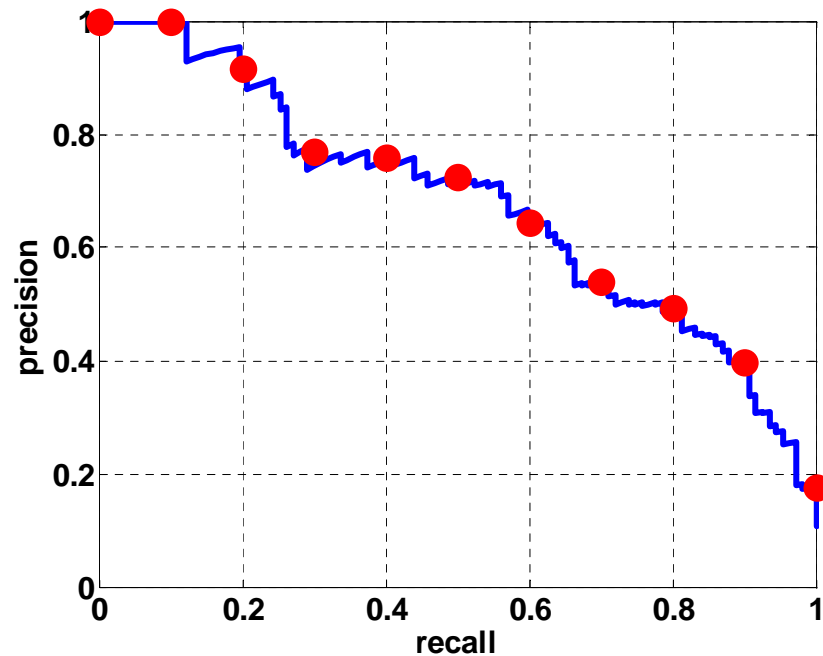
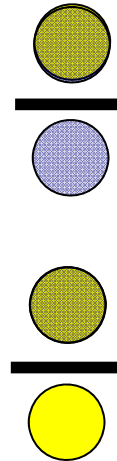
Evaluation dataset: Oxford buildings



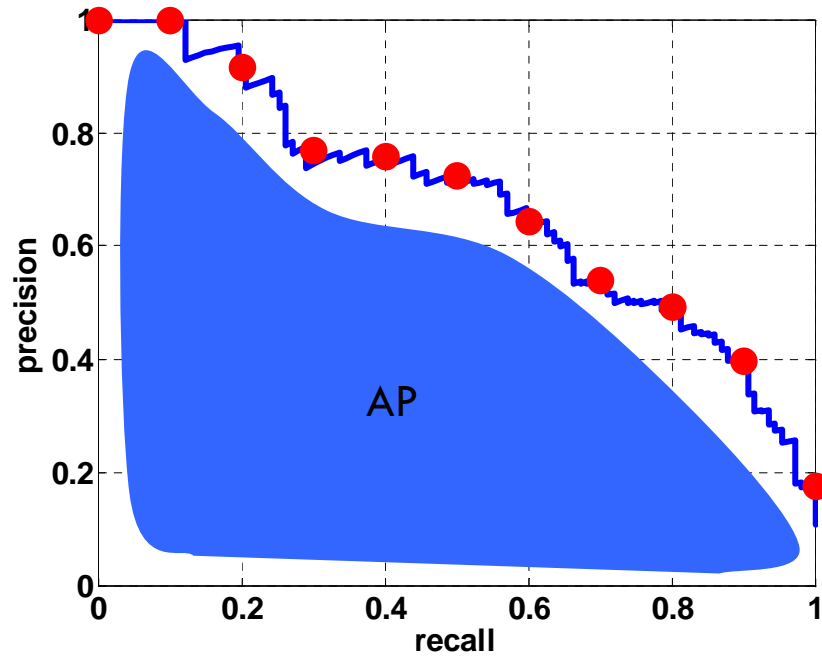
- Ground truth obtained for 11 landmarks
- Evaluate performance by mean Average Precision

Measuring retrieval performance: Precision - Recall

- Precision: % of returned images that are relevant
- Recall: % of relevant images that are returned

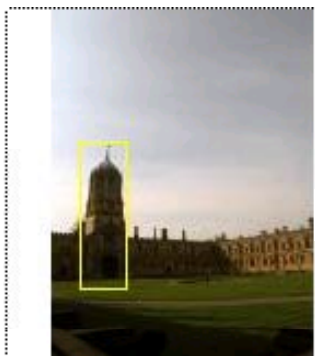


Average Precision

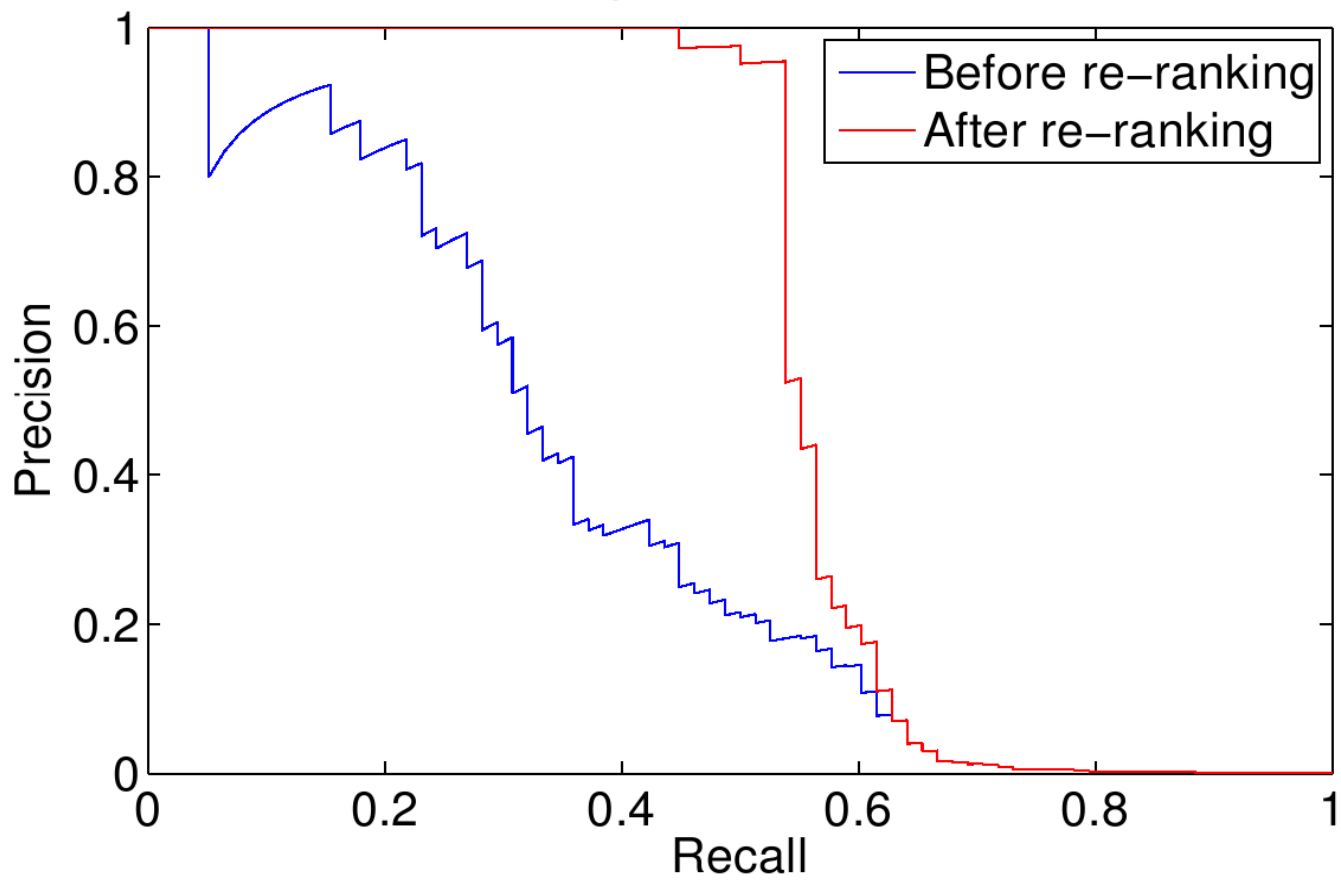


- A good AP score requires both high recall **and** high precision
- Application-independent

Performance measured by mean Average Precision (mAP)
over 55 queries on 100K or 1.1M image datasets



Query: ChristChurch3



INRIA holidays dataset

- Evaluation for the INRIA holidays dataset, 1491 images
 - 500 query images + 991 annotated true positives
 - Most images are holiday photos of friends and family
- 1 million & 10 million distractor images from Flickr
- Vocabulary construction on a different Flickr set

- Evaluation metric: mean average precision (in $[0,1]$, bigger = better)
 - Average over precision/recall curve

Holiday dataset – example queries



Dataset : Venice Channel



Dataset : San Marco square

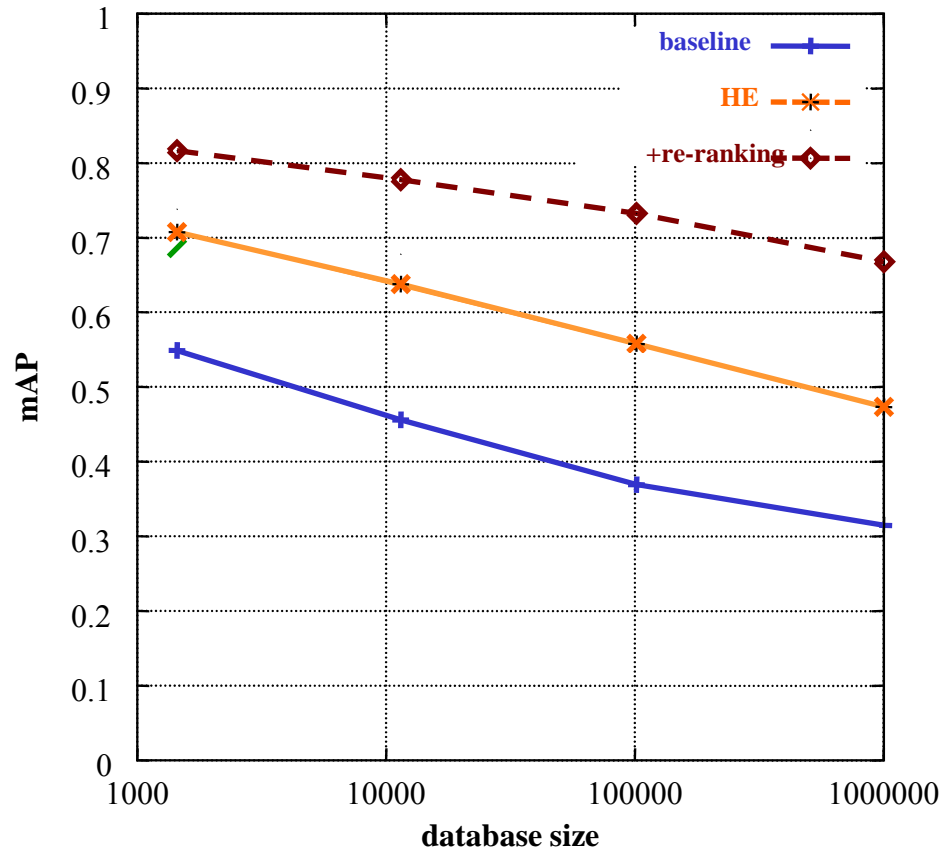


Example distractors - Flickr



Experimental evaluation

- Evaluation on our holidays dataset, 500 query images, 1 million distracter images
- Metric: mean average precision (in [0,1], bigger = better)



Results – Venice Channel



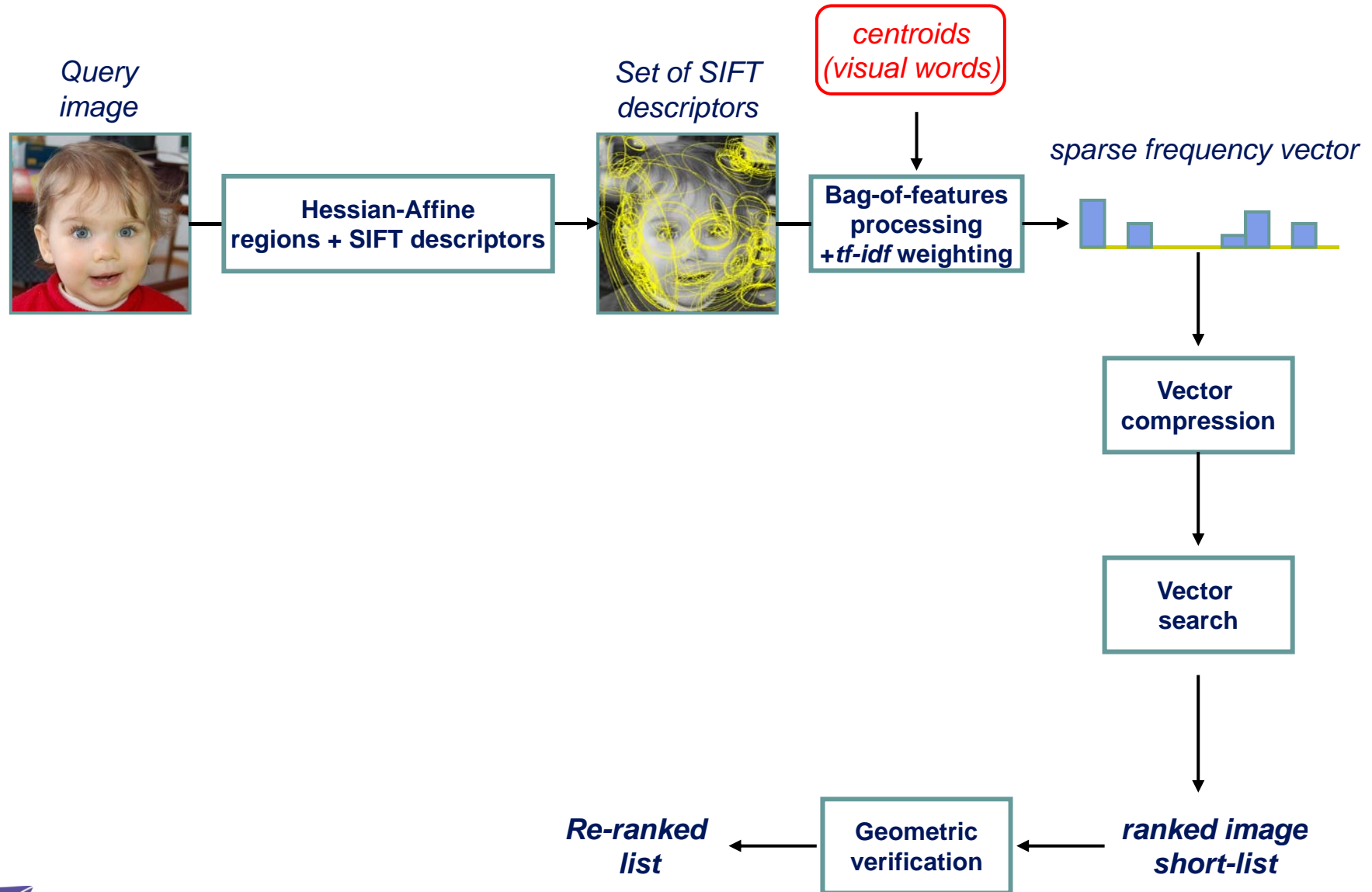
Demo at <http://bigimbaz.inrialpes.fr>

Towards larger databases?

- BOF can handle up to ~10 M d'images
 - ▶ with a limited number of descriptors per image
 - ▶ 40 GB of RAM
 - ▶ search = 2 s

- Web-scale = billions of images
 - ▶ With 100 M per machine
 - search = 20 s, RAM = 400 GB
 - not tractable!

Recent approaches for very large scale indexing



Related work on very large scale image search

- GIST descriptors with Spectral Hashing [Torralba et al. '08]
- Compressing the BoF representation (miniBof) [Jegou et al. '09]
- Aggregating local desc into a compact image representation [Jegou et al. '10]
- Efficient object category recognition using classemes [Torresani et al.'10]