Lecture 23:

A Systems View of Large-Scale 3D Reconstruction

Visual Computing Systems
CMU 15-869, Fall 2013

Goals and motivation

- Construct a detailed 3D model of the world from unstructured photographs (e.g., Flickr, Facebook)
  - Organize the world's photographs in 3D space

- <u>Leverage</u> the organization to perform tasks
  - Allow navigation/browsing of 3D environments (better maps, "virtual tourism")
  - Given a picture, where was it taken?
  - Find canonical views of scenes
  - Differentiate transient objects in scene from stationary ones
  - Many more uses…

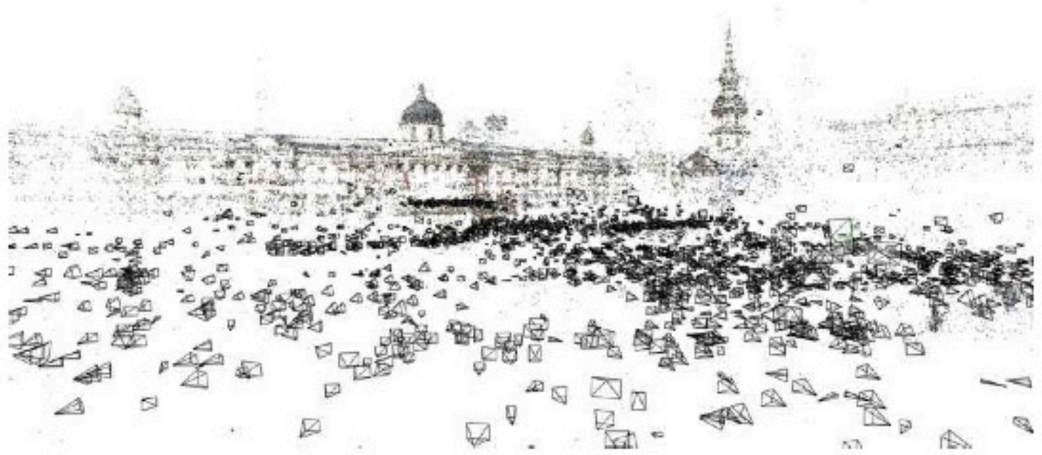


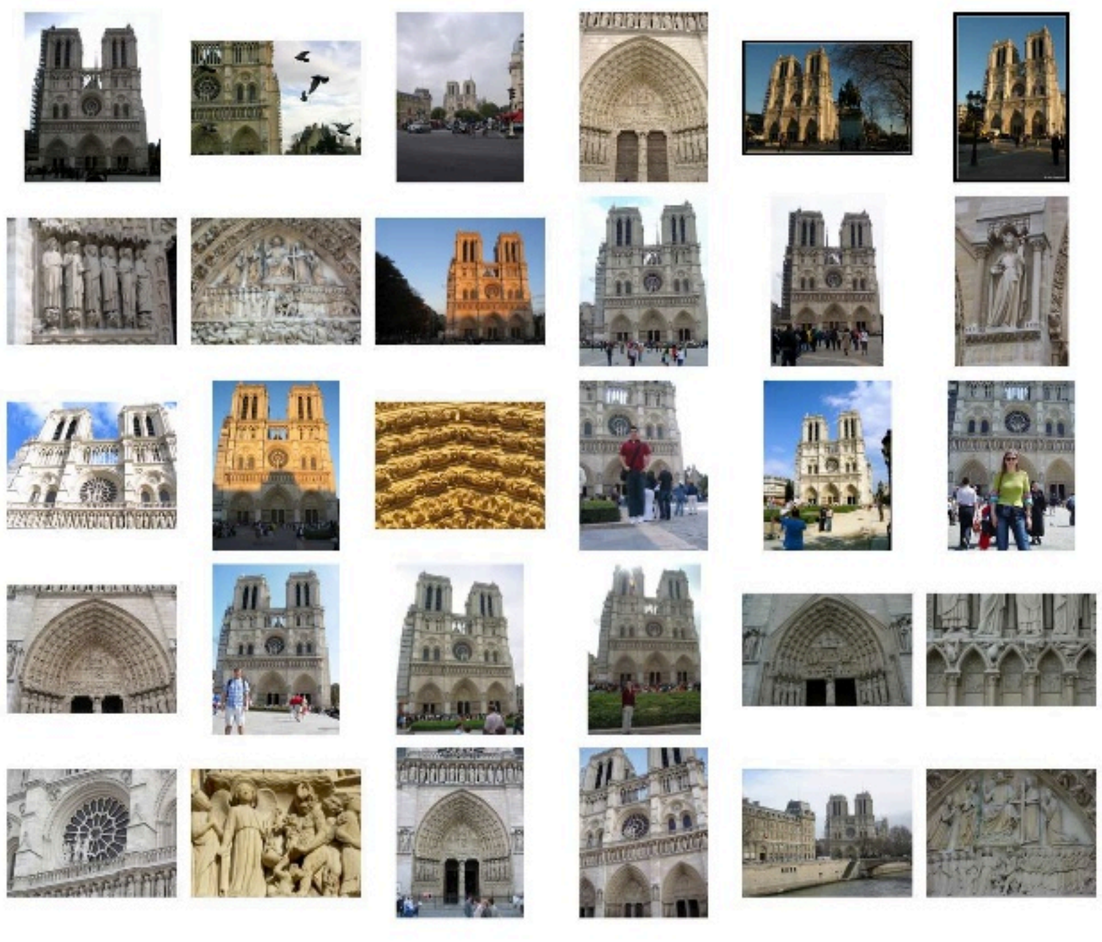
Image credits: Snavely et al. CVPR 2008, Agarwal et al. ICCV 2009

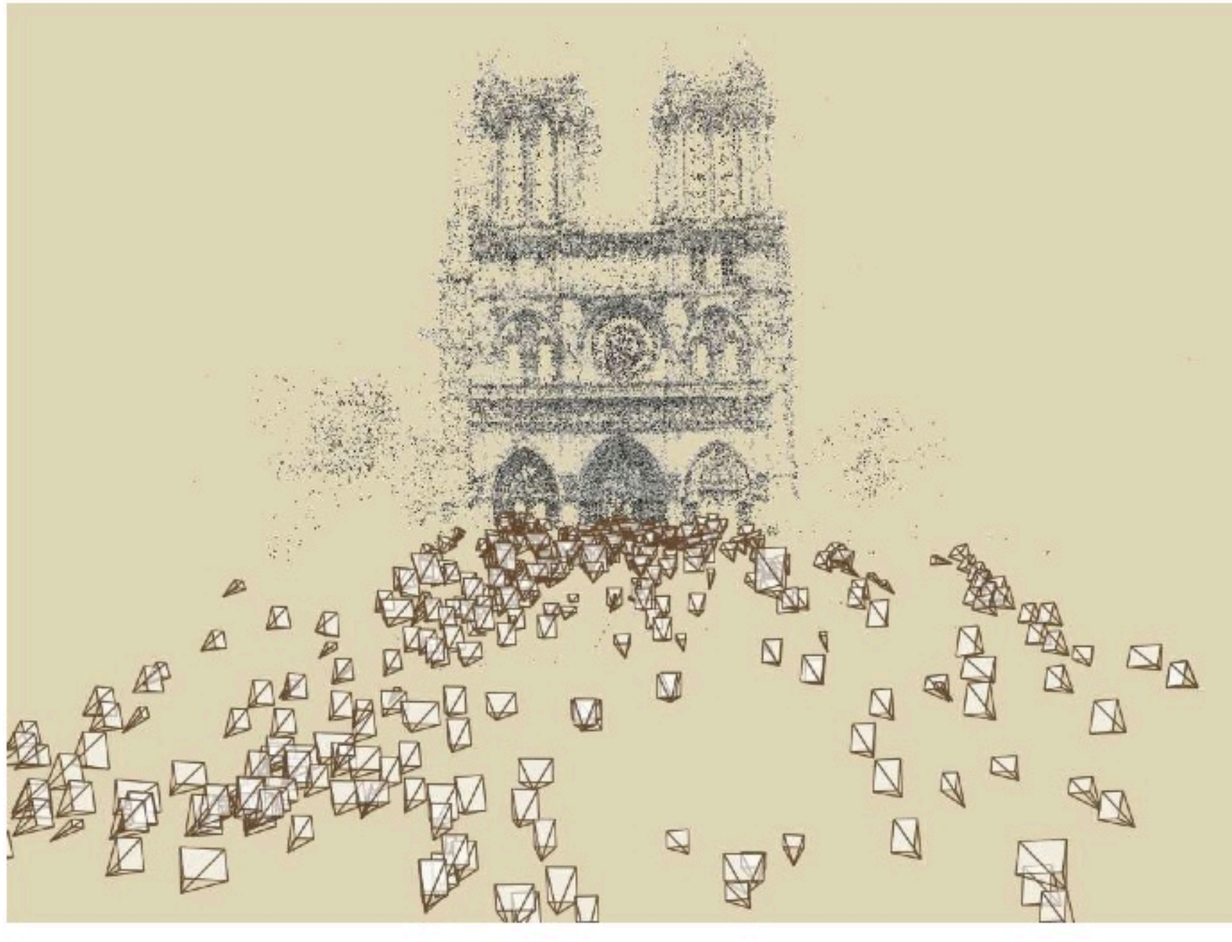
Today

- A good example of large-scale systems problem
- Arriving at efficient solutions involved combination of
  - Parallel execution
  - Algorithmic innovation
- We will "black box" key computer vision techniques to focus on overall algorithm design/systems issues
  - e.g., F-matrix estimation, homography estimation, pose estimation

Reconstructing scenes

- **Input:**
  - **Unstructured collection of photos**
- **Output:**
  - **Sparse 3D representation of scene (point cloud)**
  - **Position of camera for each photo**





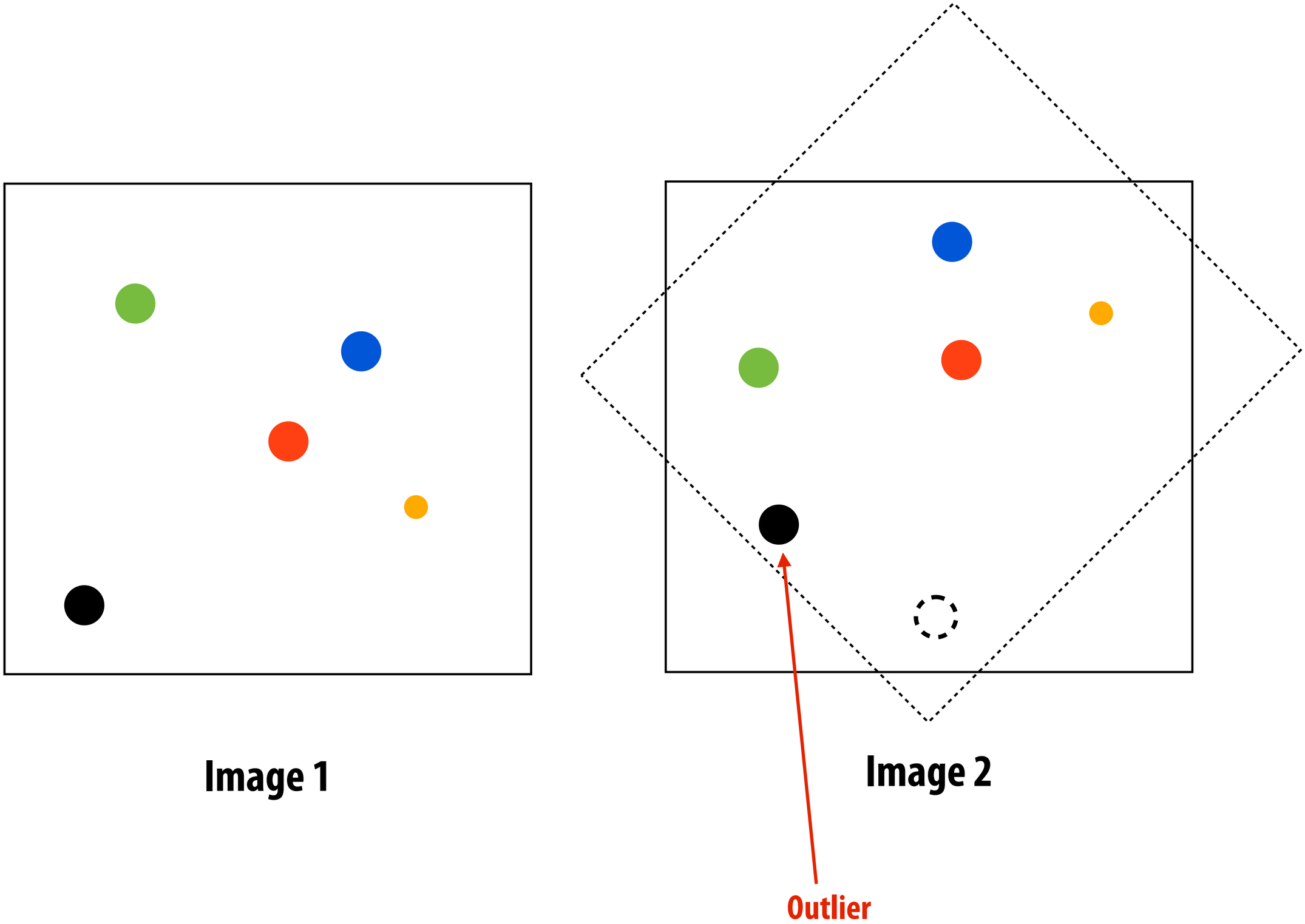
Step 1: find matching images

- Goal: find pairs of images containing the same object
- Step 1: compute feature points for all images (SIFT keypoint descriptors)
  - ~ thousands of keypoints per image
- Step 2: for each pair of images (I,J), determine if a match exists
  - Find matching keypoints

```
Compute K-D tree for all keypoints in J
for each keypoint i in I:
   // d1, d2 are distance to first and second nearest neighbor
   (d1, d2) = perform approximate nearest neighbor (ANN) lookup for i
   if (d1/d2 < threshold)
      closest NN is match for i
```

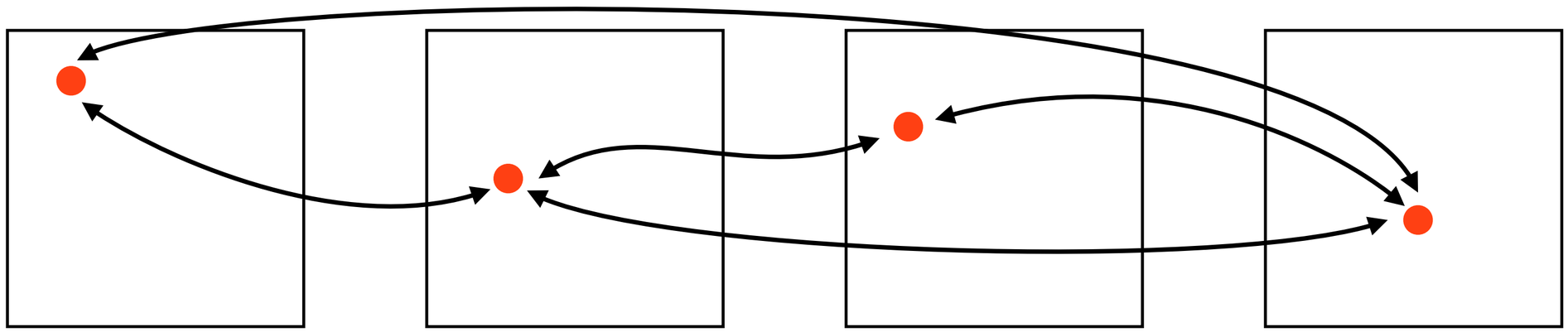
  - Verify matching keypoints: attempt to find geometric relationship between the two viewpoints: estimate a fundamental matrix for the image pair
    - RANSAC:
      - Select 8 matching keypoints at random, estimate F-matrix
      - If there are not at least 20 inlier keypoints, repeat

Geometric verification example (here: in 2D)

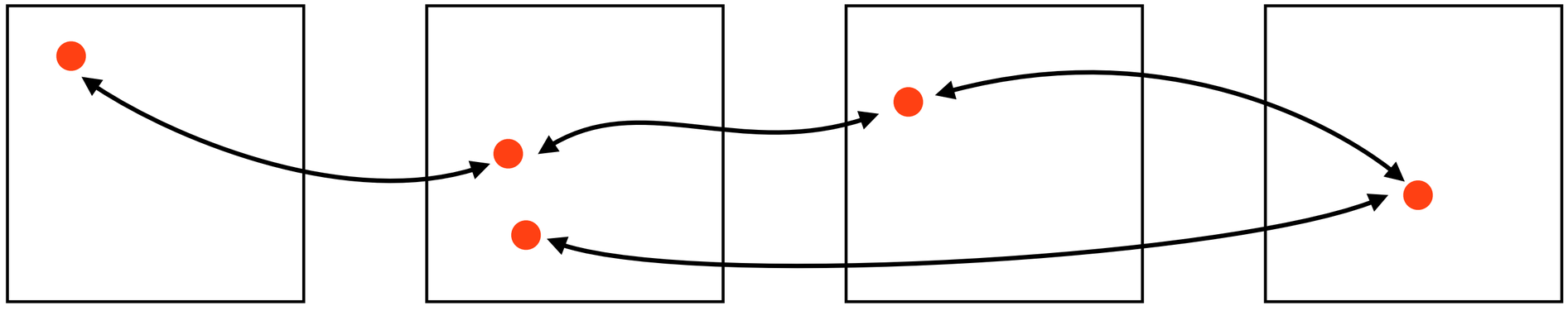


Step 2: organize matches into tracks

- Track = connected set of matching keypoints
  - Idea: a track corresponds to a single point in the scene
  - Track must contain at least two keypoints



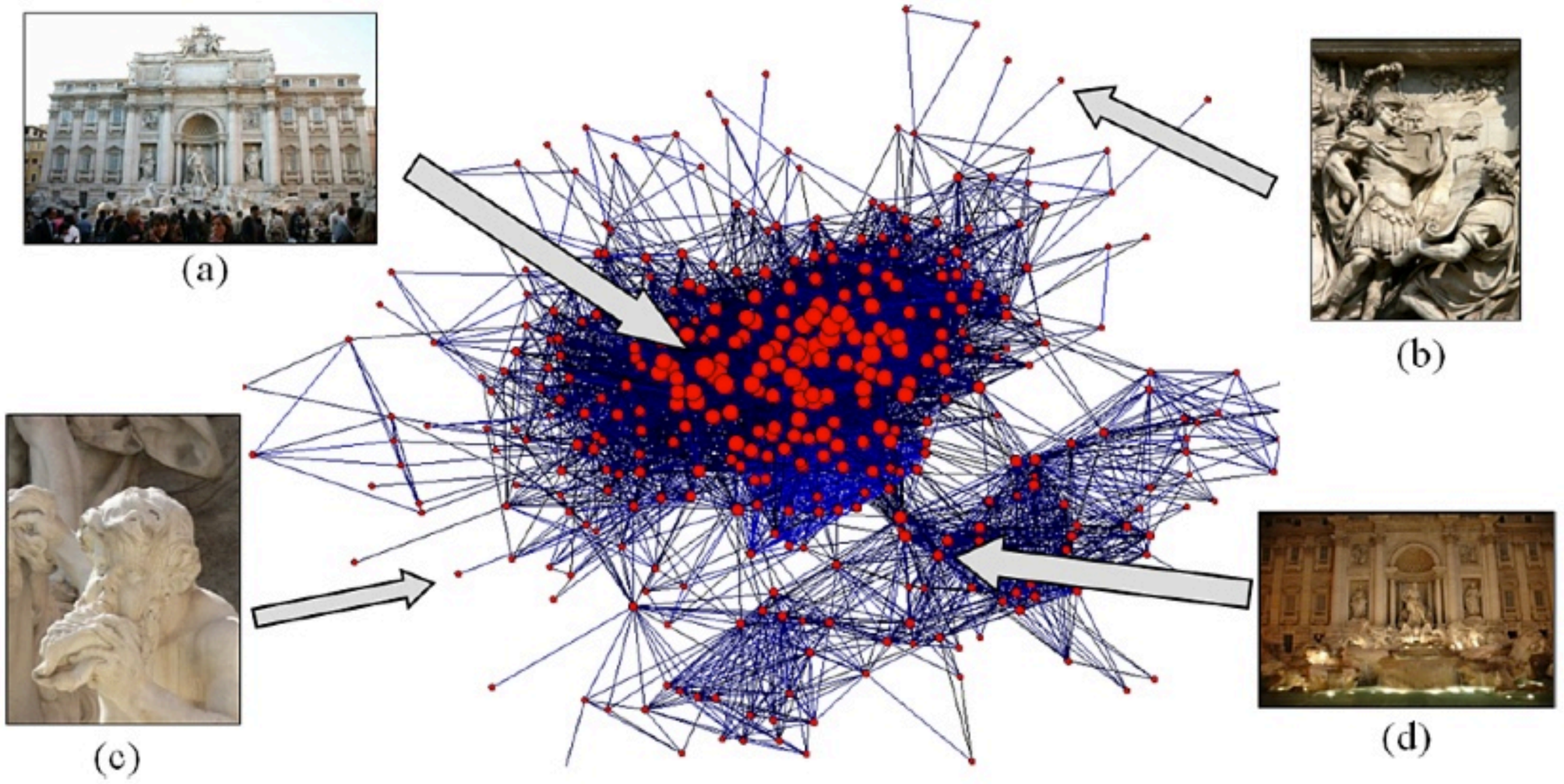
Consistent track: black arrows indicate matching keypoints in difference images



Inconsistent track: contains two keypoints in one image
(keypoints in track cannot correspond to same scene point)

Image connectivity graph

- Nodes = images
- Edges = images that contain matching keypoints

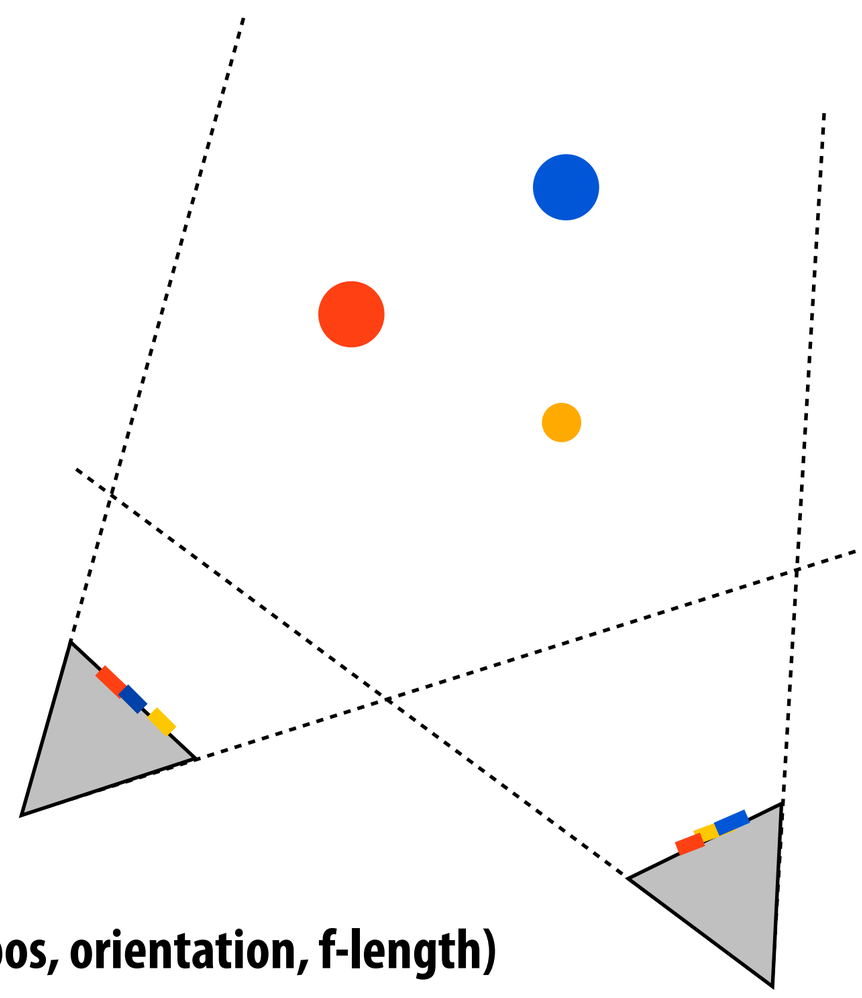


In this example, the two densely connected regions correspond to daytime and nighttime photos

Image credit: Snavely et al. 2007

Step 3: structure from motion (SfM)

- Given image match graph and a set of tracks, estimate:
  - Camera parameters for each image
    - Position, orientation, focal length
  - 3D scene position of each track



- Goal: minimize track reprojection error:
  - Error = SSDs between projection of each track and the corresponding feature in the image.

$$\underset{\theta, X}{\operatorname{argmin}} \sum w_{ij} \left\| P_{\theta_i} X_j - f_{ij} \right\|$$

  - Non-linear problem: solved via bundle adjustment

Where:

P_{θ_i} is the projection matrix into the *i*'th image (depends on camera pos, orientation, f-length)

X_j is the 3D scene position of track *j*

f_{ij} is the 2D keypoint location of track *j* in image *i*

w_{ij} is a binary indicator: designating whether a keypoint for track *j* exists in image *i*

Incremental SfM approach

- Incrementally solve for camera positions, one camera at a time:
  - Begin with data that we are most confident in (avoid local minima)
- Initialization:
  - Pick a pair of images with large number of feature matches and also wide baseline, estimate camera pose from these matches *
  - Triangulate shared tracks to estimate 3D position
  - Run two-frame bundle adjustment to refine camera poses and track position
- Add next camera:
  - Choose camera that observes most tracks with known positions
  - Estimate camera pose from track matches using DLT/RANSAC
  - Run bundle adjustment to refine only new camera and positions of tracks it observes
  - Add new tracks to scene (observed by new camera but not yet in scene)
    - Triangulate positions of new tracks using two cameras with maximum angle of separation
  - Run bundle adjustment to globally refine all camera and track position estimates

* Snavely et al. initialize with image pair that has at least 100 matches, and for which the smallest percentage of matches are inliers to an estimated homography relating the two images

Algorithm summary

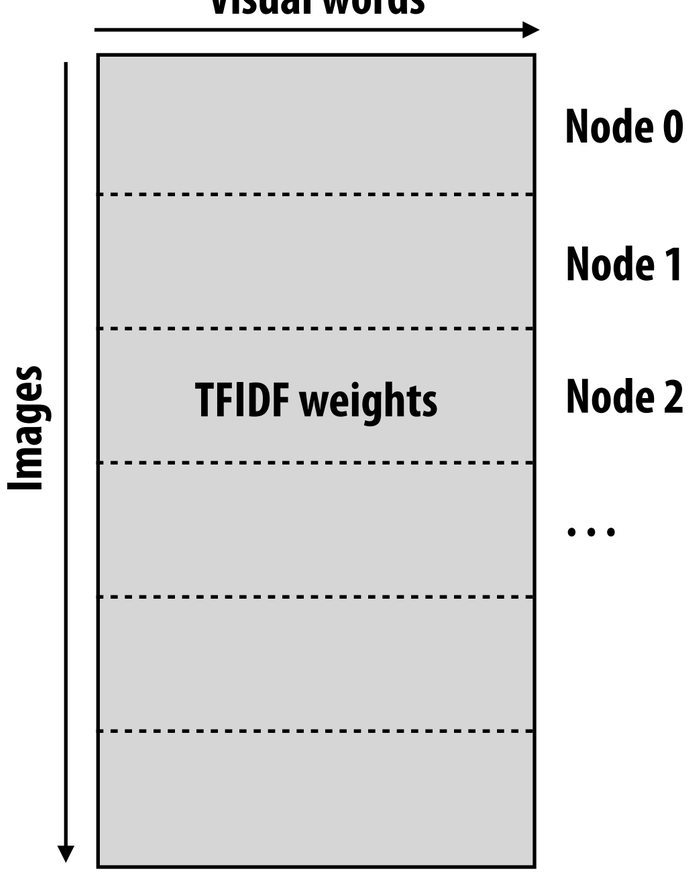
- For each image, compute matching images
- Organize matching keypoints intro tracks
- While no new cameras can be estimated
  - Pick next camera to estimate
  - Refine estimate globally using bundle adjustment

Accelerating match finding

- A naive formulation of match finding is $O(N^2)$
  - Recall, we care about large values of N
  - Match process itself is expensive (finding a geometric fit for keypoint matches via homography or fundamental matrix): ~ a few matches per core per second
  - N=1,000,000, 10 matches per second per core = 3100 CPU years
- Must avoid performing expensive check on all possible matches!
- This is a retrieval problem!

Accelerating match finding

- Step 1: use fast retrieval techniques to find candidate matching images
  - e.g., use inverted index with TF-IDF weighting as discussed in last class
  - Result: obtain k-NN for query image
- For each of the *k* candidates, perform expensive geometric verification step
  - Reduce complexity of expensive operations to O(*kN*), where k << N



Parallelization on a distributed system:

1. Partition images across nodes, compute features/BOW + term-frequencies for all images
   - SIFT features for 1M images: ~ 1-2 TB
   - BOW representation for 1M images: ~ 13 GB
2. Global reduction to compute IDF for each visual word
3. Broadcast IDF information to all nodes
4. Broadcast TFIDF table to all nodes (13 GB)
5. Each node computes kNN for the images it owns.

Improving match finding for 3D scene reconstruction

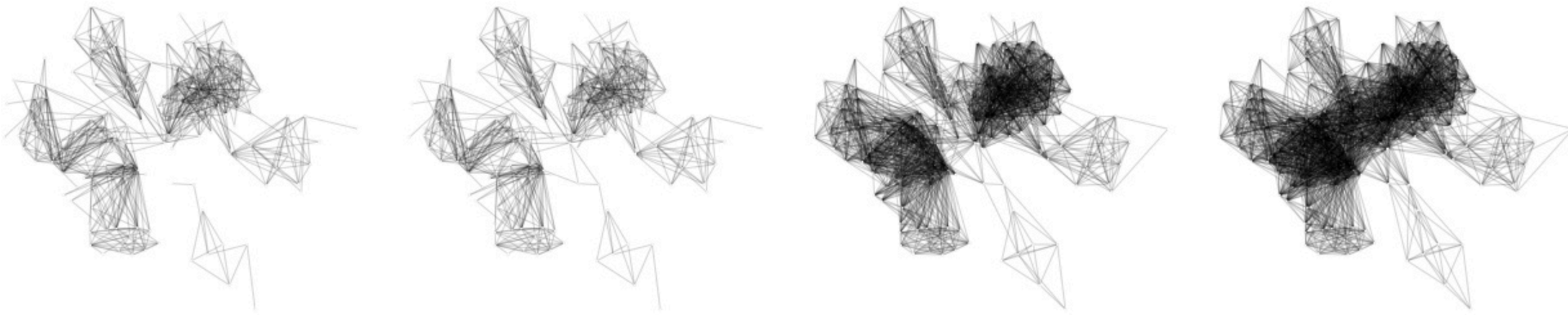
- Assume primary goal is to produce a high-quality 3D scene reconstruction (not to compute position of camera for every image in the database)
- Want a match graph that is sufficiently dense to enable 3D reconstruction:
  - Want as few connected components in match graph as possible (each connected component will be it's own 3D scene after reconstruction)
    - Prefer a single, large scene reconstruction, not many "pieces" of scene
  - Want multiple views of the same track (several images containing the same features to aid robustness of reconstruction)

Building a match graph

[Agarwal 2009]

- Step 1: Compute k nearest neighbors using acceleration structure, $k = k1 + k2$
- Step 2: Perform geometric verification of top *k1* matches, add graph edge is verification succeeds
- Step 3: Verify next *k2* matches, but only verify pair (*I*,*J*) if image *I* and image *J* are in different components of the graph
- Step 4: Densify the graph using several rounds of "query expansion"

```
For each image I
  For each neighbor J of I in graph
    For each neighbor K of J in graph
      Verify (I, K)
```



Initial Matches | CC Merge | Query Expansion 1 | Query Expansion 4

Putting it all together (distributed implementation)

- In parallel across all nodes, compute features
- Compute IDF weights via reduction, broadcast to all nodes
- Broadcast TFIDF information (weight table) to all nodes
- Independently compute *K=k1+k2* NN on all nodes
- For each image *i*, verify top *k1* candidates (parallelized dynamically via shared work queue across nodes)
- Compute match graph connected components (sequentially on one node is easiest)
- For each image *i*, verify next *k2* candidates if pair is not in same graph connected component (dynamic parallelization)
- For each image i, verify further matches based on candidates returned from query expansion
  - Repeat for N rounds, or until convergence
- Generate tracks:
  - Each node generates tracks for the images it owns (in parallel across nodes)
  - Then merge tracks across nodes (parallel reduction, or sequentially on home node)
- Compute graph skeletal set

Match graph sparsification

[Snavely 2008]

- All images do not contribute accurately to coverage/accuracy of 3D reconstruction
- For efficiency, we'd like to compute SfM using a minimal set of images (the "skeletal set") that yields similar reconstruction quality as the full match graph

Match graph	Skeletal set (black nodes are skeletal set images, gray nodes are remaining images)	Reconstruction from skeletal set	Adding addition images with pose estimation	final result post bundle adjustment

Result: 2 to 50x improvement in reconstruction performance

Image credit: Snavely et al. 2008

Systems problems, algorithmic solutions

- Desire to work at scale triggered innovation in algorithms
  - Scale imposes new constraints

- Iterative approach to SfM (to avoid local minimal)
- New algorithm for removing redundant images from match graph
  - Redundant = doesn't improve reconstruction quality
- Improved algorithm for bundle adjustment at scale
  - Not discussed today
  - See "Bundle Adjustment in the Large", Agarwal et al. ECCV 2010

Results

"Building Rome in a Day" Agarwal et al. 2009

						Time (hrs)		
Data set	Images	Cores	Registered	Pairs verified	Pairs found	Matching	Skeletal sets	Reconstruction
Dubrovnik	57,845	352	11,868	2,658,264	498,982	5	1	16.5
Rome	150,000	496	36,658	8,825,256	2,712,301	13	1	7
Venice	250,000	496	47,925	35,465,029	6,119,207	27	21.5	16.5












Building Rome on a Cloudless Day

2.8M images on a single PC in one day (Frahm et al. ECCV 2010)

Dataset	Gist & Clustering	SIFT & Geom. verification	Local iconic scene graph	Dense	total time
Rome & geo	1:35 hrs	11:36 hrs	8:35 hrs	1:58 hrs	23:53 hrs
Berlin & geo	1:30 hrs	11:46 hrs	7:03 hrs	0:58 hrs	21:58 hrs

Dataset	#images total	LSBC clusters	iconics	verified	3D models	largest model
Rome & geo	2,884,653	100, 000	21,651	306788	63905	5671
Berlin & geo	2,771,966	100, 000	14664	124317	31190	3158

Key ideas:

- Represent images using 512-bit binary codes (using locality-sensitive hash of GIST+4x4 RGB image descriptor)
- Cluster binary codes by Hamming distance
- Verify clusters by finding N images near center that can be geometrically verified using SIFT keypoints (reject clusters than cannot be verified)
- Compute "iconic" image for each cluster (image with most inliers)
- Compute matches between iconics, limiting matches to images within 150 meters of each other (as given by image geotags)
- Use high-performance plane-sweep 3D reconstruction
- Use a single PC with four GPUs

Application: location recognition

- Given a new image, how can we leverage an existing 3D reconstruction to accelerate location recognition?

- First-thought solution: find matches for query image:
  - For each SIFT feature in query image, finding matching tracks in scene database (recall: tracks = scene features)
  - Then attempt pose estimation from matches

Observation

- Not all scene features are equally useful in matching images
- Many scene features appear in many images
  - Example below: clock face on tower is most frequently observed point in database

- Idea: compute scene point set that is a K-cover of all images in DB
  - K-cover: set of points such that at least K points are present in each image
- Simply greedy algorithm: select point P that covers largest number of not-yet-full covered images

K-coverings of scene images

- Compute two K-coverings
  - P^s: 5-covering, capped at 2000 points
  - P^c: 100-covering

Localization

- Scene point-to-feature matching: rather than match query image to DB, match database to feature points in the query image
  - Idea: we know a lot about scene points, we know very little about the query image
- Simple: algorithm:
  - Test scene points again query image in priority order
    - Initialization:
      - Highest priority: Ps
      - Next highest priority: Pc
      - All other points: priority = number of images they are visible in
- If match is found, for all images containing point:
  - Increase priority of all scene points in these images

How it works

- Test the most likely to match images in the database first
  - Only need a few matches to localize
- Once a match is found, leverage co-orrurence of points to predict new matching points
- Intuitive:
  - Common images get found very quickly
  - Uncommon images take longer to localize
  - Optimize for the common case!