Lecture 21:

A Systems View of Large-Scale 3D Reconstruction

Visual Computing Systems
CMU 15-869, Fall 2014

Goals and motivation

- Construct a detailed 3D model of the world from collections of photographs
  - Organize the world's photographs by their position in 3D space
- Leverage the organization to perform tasks
  - Allow navigation/browsing of 3D environments (better maps, "virtual tourism")
  - Geolocation: given a picture, where was it taken?
  - Find canonical views of scenes
  - Differentiate transient objects in scene from stationary ones
  - Many more uses…

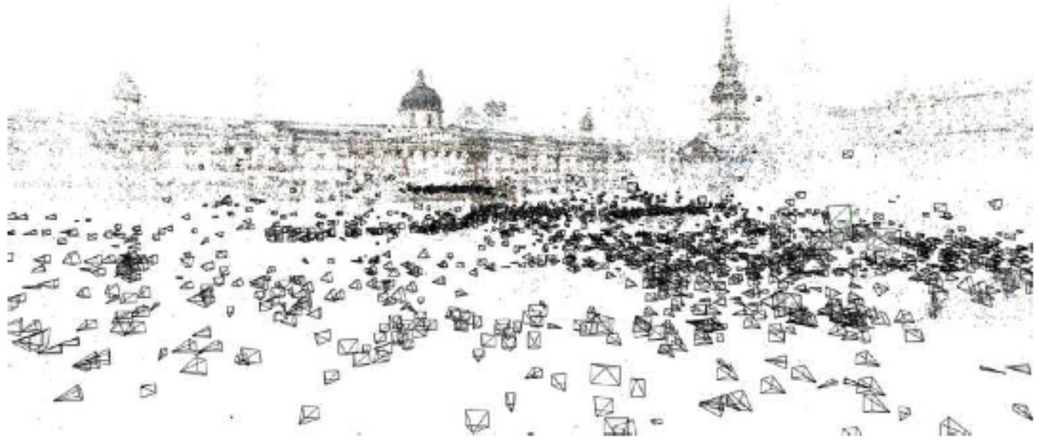


Image credits: Snavely et al. CVPR 2008, Agarwal et al. ICCV 2009

Preliminaries and background

1. Image similarity / retrieval basics
2. Nearest neighbor search and approximate nearest neighbor search (ANN) using a KD-tree
3. RANSAC algorithm overview

Background part 1: image retrieval basics

Are these images similar?






Photographs of same backyard, over six-month period.

Pixel differences

Image 1

Pixel differences

Image 2

Pixel differences

diff(x,y) = image1(x,y) - image 2(x,y)

Are these two web pages similar?

Visual Computing Systems

CMU 15-869 | Fall 2014

Visual computing tasks such as 2D/3D graphics, image processing, and image understanding are important responsibilities of modern computer systems ranging from sensor-rich smart phones to large datacenters. These workloads demand exceptional system efficiency and this course examines the key ideas, techniques, and challenges associated with the design of parallel (and heterogeneous) systems that serve to accelerate visual computing applications. This course is intended for graduate and advanced undergraduate-level students interested in architecting efficient future graphics and image processing platforms and for students seeking to develop scalable algorithms for these platforms.

When We Meet

Time: Mon/Wed 1:30 - 2:50pm
Location: GHC 4102
Instructor: **Kayvon Fatahalian**

Click here for Course Logistics and Details

Schedule

Note, please consult the **suggested readings page** for a list of readings relevant to lecture topics.

Sep 8	**Course Introduction + the Real-Time Graphics Pipeline** Real-time rendering from a systems perspective
Sep 10	**Graphics Pipeline Parallelization and Scheduling** Characteristics of the pipeline workload, Molnar's scheduling taxonomy, trade-offs between parallelism, communication, and locality
Sep 15	**Geometry Processing and the Scheduling Challenges of Data Amplification** Clipping, tessellation, challenges of parallel scheduling
Sep 17	**Visibility: High-Performance Rasterization and Occlusion** Visibility algorithms and their fixed-function implementation, occlusion culling, anti-aliasing, frame-buffer compression
Sep 22, 24	**Texturing Part I: Basic Algorithms and Cache-Efficient Data Layout** Anti-aliasing using the mip-map, locality-friendly texture data layouts
Sep 24, 29	**Texturing Part II: Texture Compression and GPU Latency Hiding Mechanisms** Hardware-accelerated texture decompression techniques, prefetching, and multi-threading
Oct 1	**Shading Language Design** Level of abstraction decisions, mapping shader logic to GPU processing cores
Oct 6	**Compute-Mode GPU Programming Models and Emerging "So-Called-Low-Level" APIs** Motivation for alternative abstractions, interoperability issues with graphics pipeline, AMD's Mantle, Apple's Metal
Oct 8	**Deferred Shading, and Why it's Now So Popular in Games** Motivation for use, impact on global renderer scheduling decisions
Oct 13	**High-Performance Ray Tracing and Emerging Ray Tracing Hardware** Workload characteristics, coherence optimizations, potential for hardware acceleration

Parallel Computer Architecture and Programming (CMU 15-418)

From smart phones, to multi-core CPUs and GPUs, to the world's largest supercomputers and web sites, parallel processing is ubiquitous in modern computing. The goal of this course is to provide a deep understanding of the fundamental principles and engineering trade-offs involved in designing modern parallel computing systems as well as to teach parallel programming techniques necessary to effectively utilize these machines. Because writing good parallel programs requires an understanding of key machine performance characteristics, this course will cover both parallel hardware and software design.

[Our Self-Made Online Reference]

[Policies, Logistics, and Details]

When We Meet

Tues/Thurs 9:00 - 10:20am
Baker Hall A51 (Giant Eagle Auditorium)
Instructor: **Kayvon Fatahalian**

Spring 2013 Schedule

Jan 15	**Why Parallelism?**
Jan 17	**A Modern Multi-Core Processor: Forms of Parallelism + Understanding Latency and BW** Assignment 1 out
Jan 22	**Parallel Programming Models and Their Corresponding HW/SW Implementations**
Jan 24	**Parallel Programming Basics (the parallelization thought process)** Assignment 1 due
Jan 29	**GPU Architecture and CUDA Programming** Assignment 2 out
Jan 31	**Performance Optimization I: Work Distribution**
Feb 5	**Performance Optimization II: Locality, Communication, and Contention**
Feb 7	**Parallel Application Case Studies**
Feb 12	**Workload-Driven Performance Evaluation** Assignment 2 due Assignment 3 out

Another example: which web page is most similar to the search query...

Are these two web pages similar?

Visual Computing Systems
CMU 15-869 | Fall 2014

Visual computing tasks such as 2D/3D graphics, image processing, and image understanding are important responsibilities of modern computer systems ranging from sensor-rich smart phones to large datacenters. These workloads demand exceptional system efficiency and this course examines the key ideas, techniques, and challenges associated with the design of parallel (and heterogeneous) systems that serve to accelerate visual computing applications. This course is intended for graduate and advanced undergraduate-level students interested in architecting efficient future graphics and image processing platforms and for students seeking to develop scalable algorithms for these platforms.

When We Meet

Time: Mon/Wed 1:30 - 2:50pm
Location: GHC 4102
Instructor: **Kayvon Fatahalian**

Click here for Course Logistics and Details

Schedule

Note, please consult the **suggested readings page** for a list of readings relevant to lecture topics.

Sep 8	**Course Introduction + the Real-Time Graphics Pipeline** Real-time rendering from a systems perspective
Sep 10	**Graphics Pipeline Parallelization and Scheduling** Characteristics of the pipeline workload, Molnar's scheduling taxonomy, trade-offs between parallelism, communication, and locality
Sep 15	**Geometry Processing and the Scheduling Challenges of Data Amplification** Clipping, tessellation, challenges of parallel scheduling
Sep 17	**Visibility: High-Performance Rasterization and Occlusion** Visibility algorithms and their fixed-function implementation, occlusion culling, anti-aliasing, frame-buffer compression
Sep 22, 24	**Texturing Part I: Basic Algorithms and Cache-Efficient Data Layout** Anti-aliasing using the mip-map, locality-friendly texture data layouts
Sep 24, 29	**Texturing Part II: Texture Compression and GPU Latency Hiding Mechanisms** Hardware-accelerated texture decompression techniques, prefetching, and multi-threading
Oct 1	**Shading Language Design** Level of abstraction decisions, mapping shader logic to GPU processing cores
Oct 6	**Compute-Mode GPU Programming Models and Emerging "So-Called-Low-Level" APIs** Motivation for alternative abstractions, interoperability issues with graphics pipeline, AMD's Mantle, Apple's Metal
Oct 8	**Deferred Shading, and Why it's Now So Popular in Games** Motivation for use, impact on global renderer scheduling decisions
Oct 13	**High-Performance Ray Tracing and Emerging Ray Tracing Hardware** Workload characteristics, coherence optimizations, potential for hardware acceleration

Parallel Computer Architecture and Programming (CMU 15-418)

From smart phones, to multi-core CPUs and GPUs, to the world's largest supercomputers and web sites, parallel processing is ubiquitous in modern computing. The goal of this course is to provide a deep understanding of the fundamental principles and engineering trade-offs involved in designing modern parallel computing systems as well as to teach parallel programming techniques necessary to effectively utilize these machines. Because writing good parallel programs requires an understanding of key machine performance characteristics, this course will cover both parallel hardware and software design.

[Our Self-Made Online Reference]

[Policies, Logistics, and Details]

When We Meet

Tues/Thurs 9:00 - 10:20am
Baker Hall A51 (Giant Eagle Auditorium)
Instructor: **Kayvon Fatahalian**

Spring 2013 Schedule

Jan 15	**Why Parallelism?**
Jan 17	**A Modern Multi-Core Processor: Forms of Parallelism + Understanding Latency and BW** Assignment 1 out
Jan 22	**Parallel Programming Models and Their Corresponding HW/SW Implementations**
Jan 24	**Parallel Programming Basics (the parallelization thought process)** Assignment 1 due
Jan 29	**GPU Architecture and CUDA Programming** Assignment 2 out
Jan 31	**Performance Optimization I: Work Distribution**
Feb 5	**Performance Optimization II: Locality, Communication, and Contention**
Feb 7	**Parallel Application Case Studies**
Feb 12	**Workload-Driven Performance Evaluation** Assignment 2 due Assignment 3 out

Another example: which web page is most similar to the search query...

Google cmu visual computing systems

About 318,000 results (0.69 seconds)

Kayvon Fatahalian - Carnegie Mellon University
www.cs.**cmu**.edu/~kayvonf/ ▾ Carnegie Mellon University ▾
Assistant Professor of Computer Science. **Carnegie Mellon University**. 412-268-1234 (Smith Hall 225). I architect new **visual computing systems** that enable ...
You've visited this page many times. Last visit: 10/27/14

Visual Computing Systems - Carnegie Mellon Graphics
graphics.cs.**cmu**.edu/courses/15869/fall2013/ ▾ Carnegie Mellon University ▾
Visual computing tasks such as 2D/3D graphics, image processing, and image understanding are important responsibilities of modern **computer systems** ...

[PDF] Visual Computing Systems CMU 15-869, Fall 2013 Lectur...
graphics.cs.**cmu**.edu/.../gfxpipeline_slides.pd... ▾ Carnegie Mellon University ▾
A **systems** architect must meet challenging application goals within specifc design ...
The characteristics/requirements of important **visual computing** workloads.

15-869: Visual Computing Systems - Piazza
https://piazza.com/**cmu**/fall2013/15869/home ▾ Piazza ▾
Visual computing tasks such as graphics, image processing, and image ... 15-869: **Visual Computing Systems** is a course taught at **Carnegie Mellon University** ...

One simple definition of similarity to query

Given query words: w_1 *and* w_2

for each document *d* in database:

score(d, w_1, w_2) = number of occurrences of w_1 *and* w_2 in d

Return top 20 results in sorted order based on score

Ways to improve above approach:

- **Improve accuracy of score function (return better results *)**
- **Improve query execution time: above solution is O(N) for database of N documents**

*** The quality of the returned results is referred to as the "performance" of the algorithm. "An algorithm performs better if it returns better results". Clearly, using the term "performance" in this way going to cause problems in this class.**

Improving the solution: use an index

To simplify, let:

`score(d,w1,w2) = 1 if d contains w1 and w2, 0 otherwise`

```
Document 0: Kayvon is teaching 15-869 today. Yay 15-869!
Document 1: 15-869 is awesome, Kayvon claims.
Document 2: Kayvon is occasionally awesome.
```

Index: maps words to documents

- Kayvon: 0, 1, 2
- is: 0, 1, 2
- teaching: 0
- 15-869: 0, 1
- yay: 0
- thinks: 1
- today: 0
- awesome: 1, 2
- occasionally: 2

Query: `kayvon awesome`

Partial result set:

`kayvon: {0, 1,,2}`
`awesome: {1, 2}`

Result:

`{0,1,2} ∩ {1, 2} = {1,2}`

Full inverted index

Inverted index contains one entry per word occurrence:

```
score(d,w1,w2) =
     if d contains w1 and w2, number of occurrences of w1 or w2
     0 otherwise
```

```
Document 0: Kayvon is teaching 15-869 today. Yay 15-869!
Document 1: 15-869 is awesome, Kayvon claims.
Document 2: Kayvon is occasionally awesome.
```

Index: maps words to (document, position)

- Kayvon: (0,0), (1,3) (2,0)
- is: (0,1), (1,1), (2,1)
- teaching: (0, 2)
- 15-869: (0,3), (0,6), (1, 0)
- yay: (0, 5)
- claims: (1,4)
- today: (0, 4)
- awesome: (1,2), (2,3)
- occasionally: (2,2)

Query: kayvon 15-869

Partial result set:

```
kayvon: {(0,0), (1,3), (2,0)}
15-869: {(0,3), (0,6), (1,0)}
```

Result:

```
{0 (1), 1 (1), 2 (1)} ∩
{0 (2), 1 (0)}
= {0 (3), 1 (2)}
```

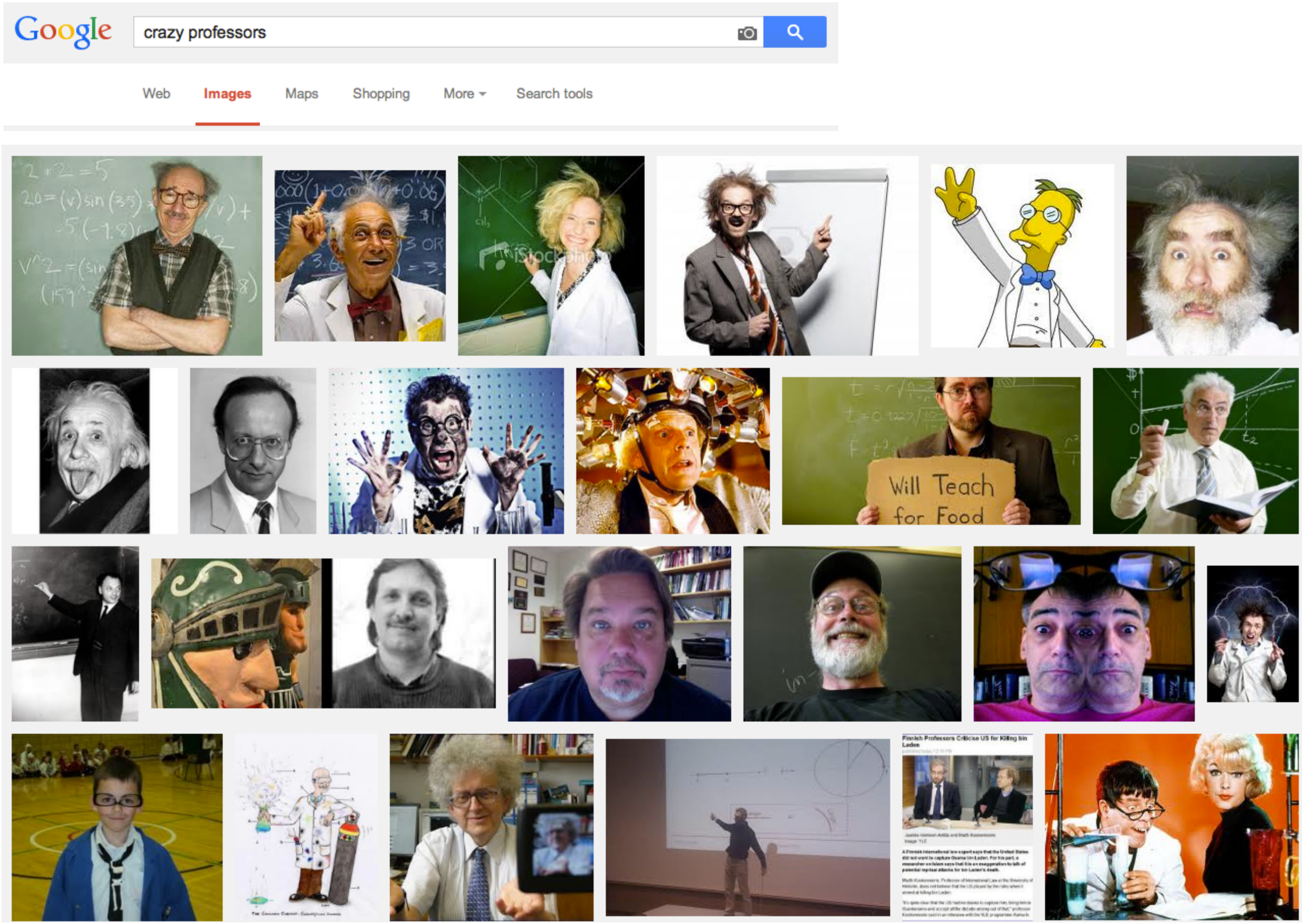
Ranking:

```
0, 1
```

TF-IDF weighting

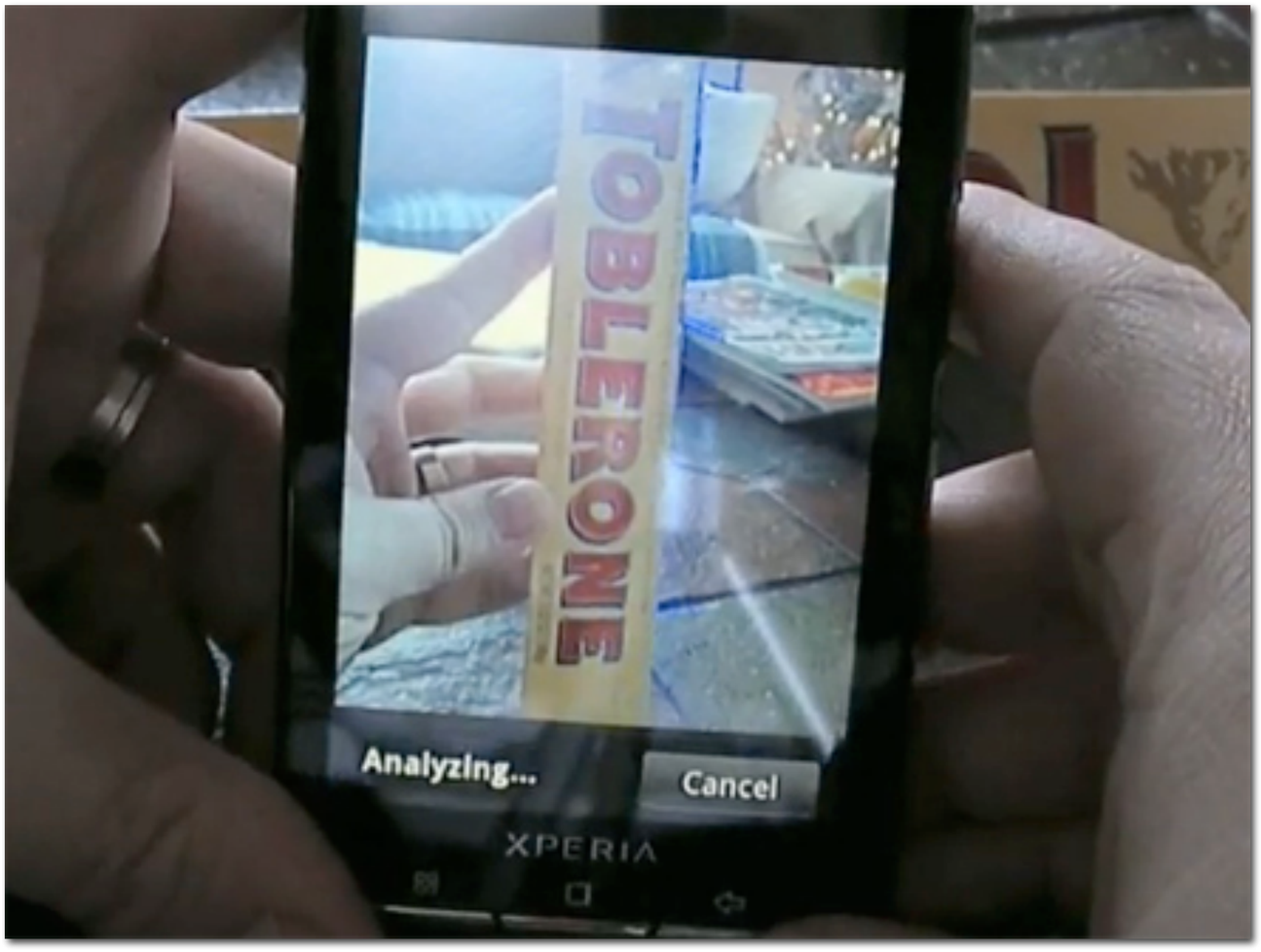
- Term frequency:
  - TF(w,d) = the number of occurrences of word w in document d
  - Measure of how relevant a document is for a given query word
- Inverse document frequency:
  - $\text{IDF}(w, D) = \log \frac{|D|}{|\{d \in D : w \in d\}|}$
    - $|D|$ ← Number of documents in database D
    - $|\{d \in D : w \in d\}|$ ← Number of documents containing w
  - Measure of how discriminative a word is (idf is small for common words)
    - Depends on number of occurrences in entire document collection
    - Idea: words that appear in most documents should influence score less
- $\text{tfidf\_score}(w, d, D) = \text{TF}(w,d) \times \text{IDF}(w, D)$
- Many variants on how to compute TF(w,d)
  - Binary: 1 of 0, depending on whether word is in document
  - Normalized frequency: number of occurrences normalized by document size

Searching for images (via text query)



Content-based image retrieval

- Search for images, based on a query images
  - Take a photo, find similar looking photos
  - Take a photo, find information about (objects, people, etc.) in photo



Text-based document retrieval

- Key idea was the breakdown of document into words
  - Documents that have the same words are likely to be similar
  - Words are a meaningful granularity of text to latch on to

Content-based image retrieval

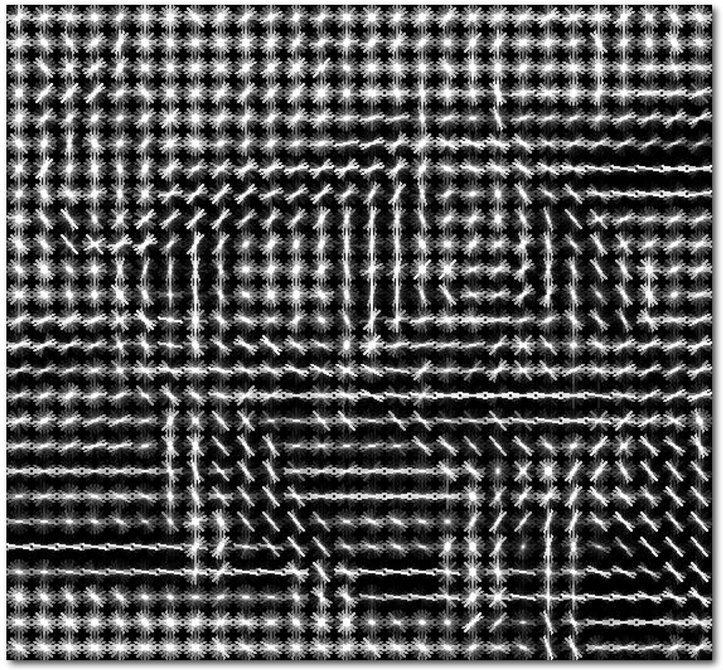
- If we wanted to follow the text analogy, what are the words?
  - Pixels?
  - Blocks of pixels?
  - Descriptors/features computed from images?

Correspondence

- Defining similarity requires us to quantify the notion of correspondence
  - Example: pictures of the same place are similar
  - Example: pictures containing the same/similar objects are similar
- Seek image representations ("descriptors") such that numerically similar descriptors correspond to meaningful correspondences
  - Example: similar descriptor value corresponds to same object in the scene: descriptor's value is invariant to noise, lighting, affine object transformation (rotation, translation, scale)
  - Of course, good descriptors should also be distinctive… shouldn't take on same value for every image!

Histogram of oriented gradients (HOG) [Dalal and Triggs 05]

- Idea: local object appearance/shape is well characterized by distribution of local intensity gradients
- Gradient orientation is less sensitive to illumination change than gradient magnitude

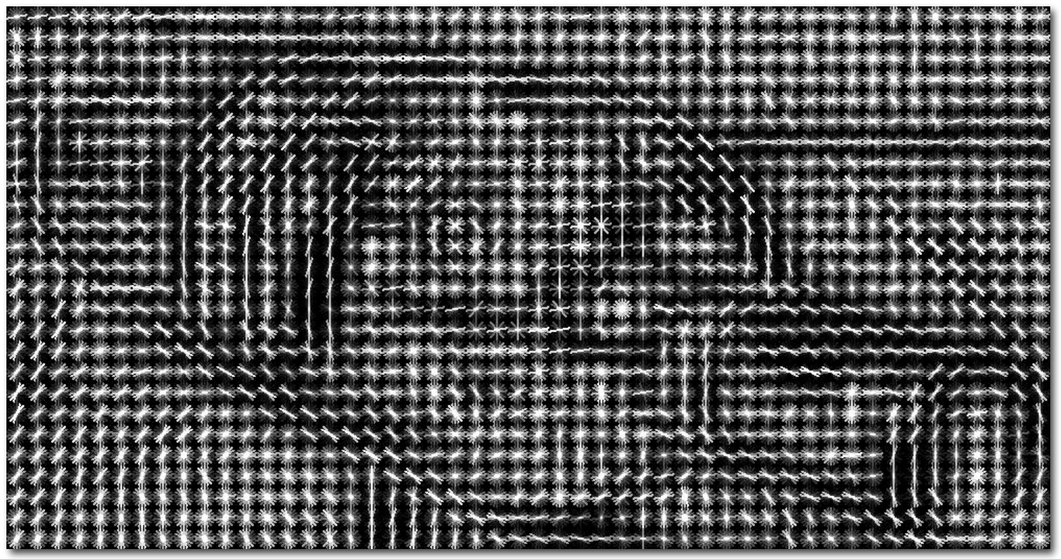


For each pixel *p* in block:

Compute local gradient

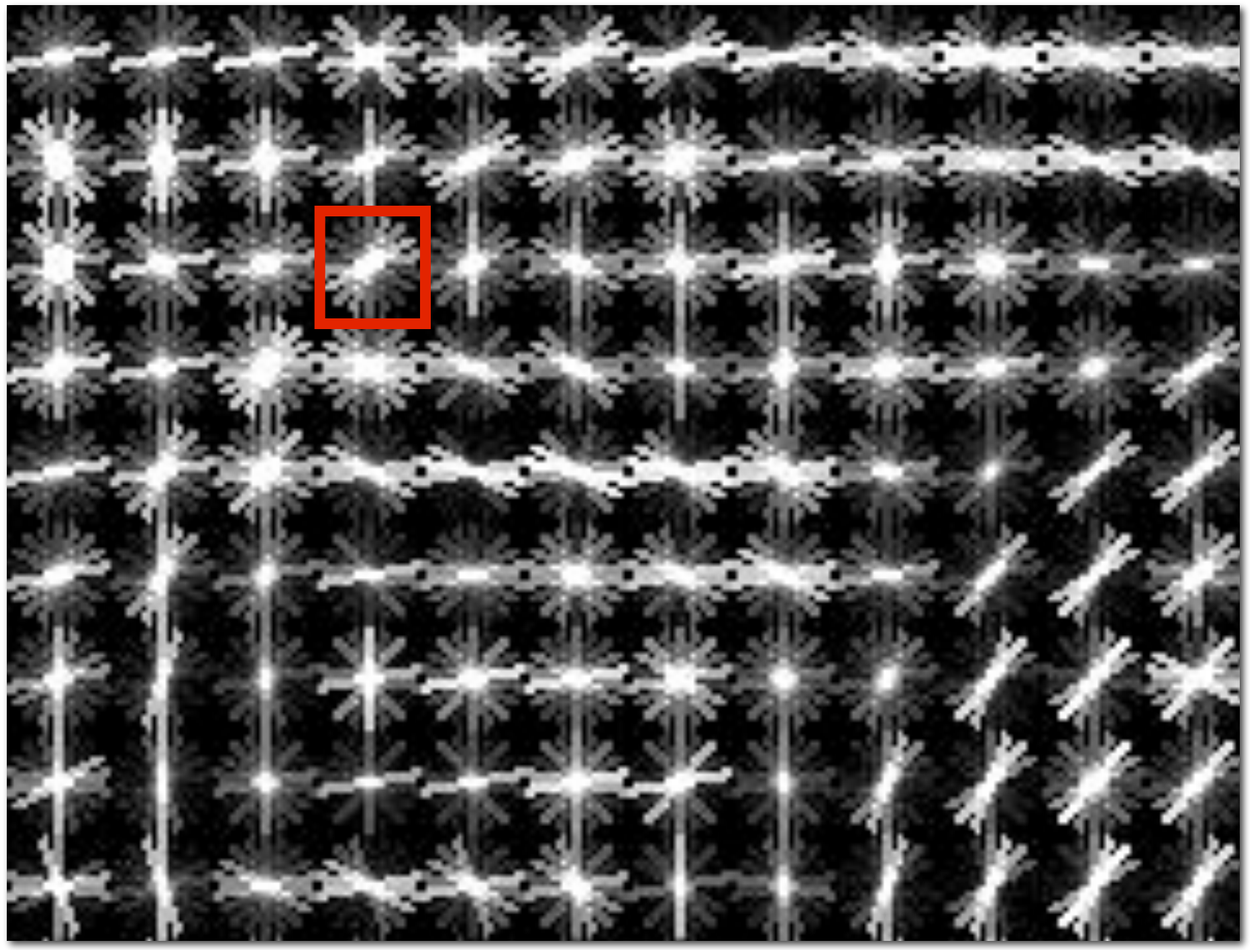
Add vote to histogram cell based on gradient orientation

(vote is weighted based on gradient magnitude and distance between p and block center)



[Image credit: Vondrick et al. ICCV 13]

HOG visualization close up

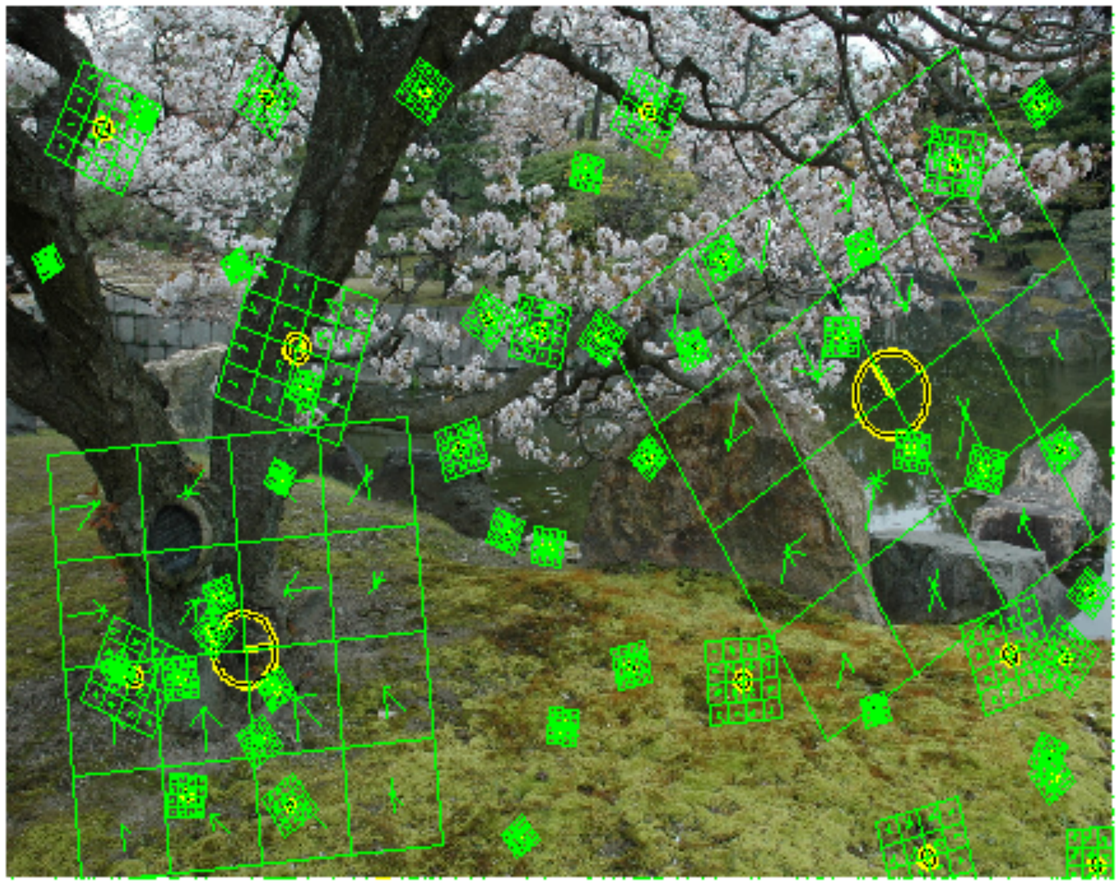


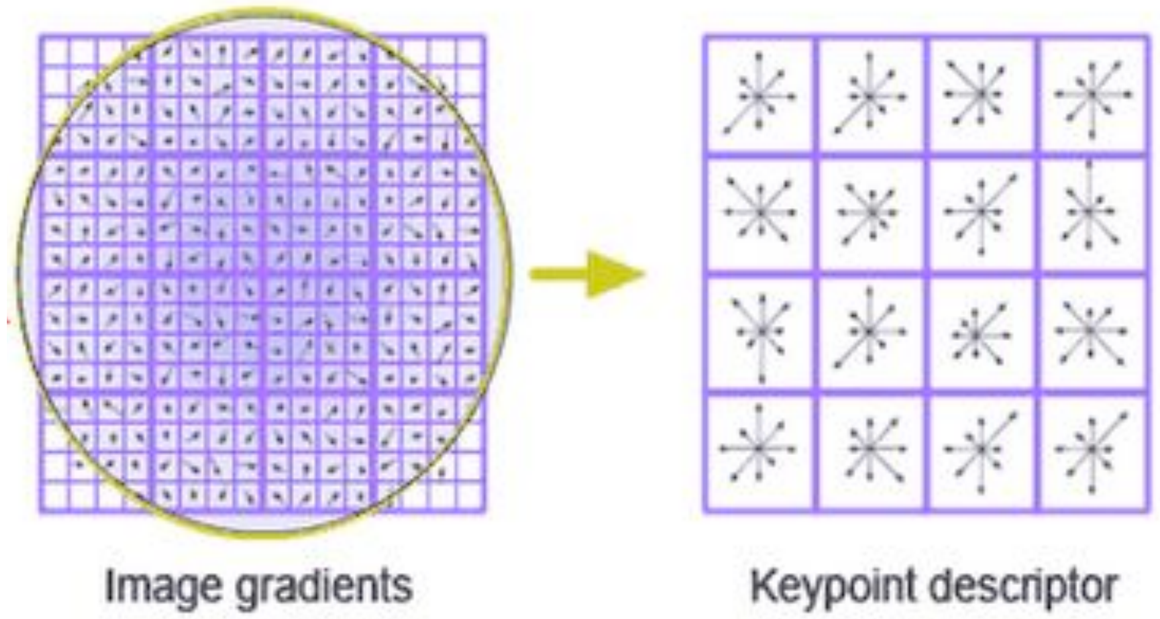
Visualizing magnitude of each histogram cell as a line

(Direction of line is at a right angle to the corresponding gradient orientation)

Sparse SIFT descriptors

- Interest-point-based, orientation of gradients descriptor
- Find interest points (locations in image, with support region scale and orientation)
- Compute 128-element descriptor for interest points





Pool gradient samples from 4x4 window into 8-bin histogram

Concatenate 4x4 grid of histograms to get full descriptor (8 x 16 = 128)

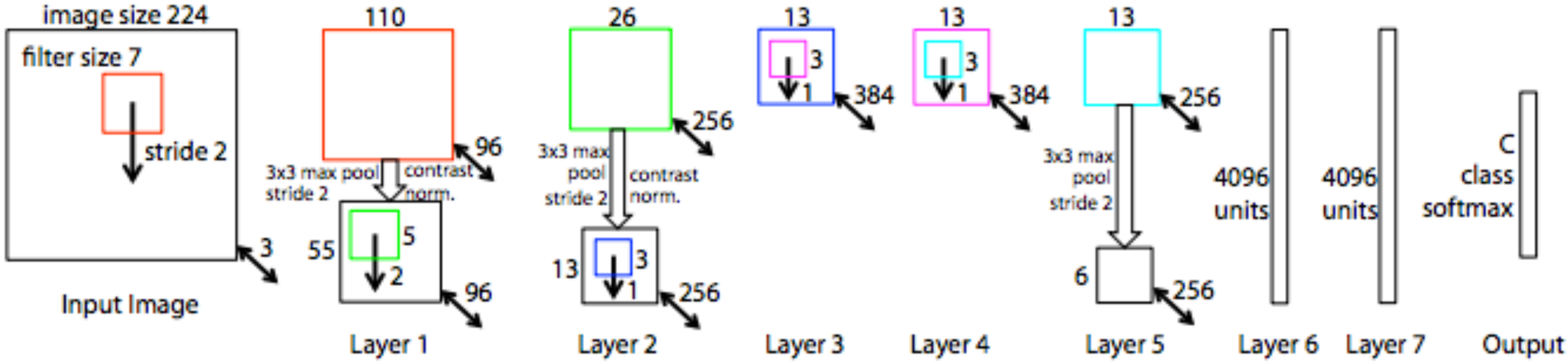
Figure credits:

R. Bandara, Codeproject

Chen, Kong, Oh, Sanan, Wohlberk 09

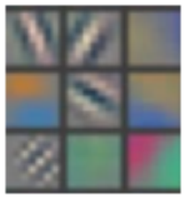
State-of-the-art: learn good features

Obtain feature representations by training deep neural networks

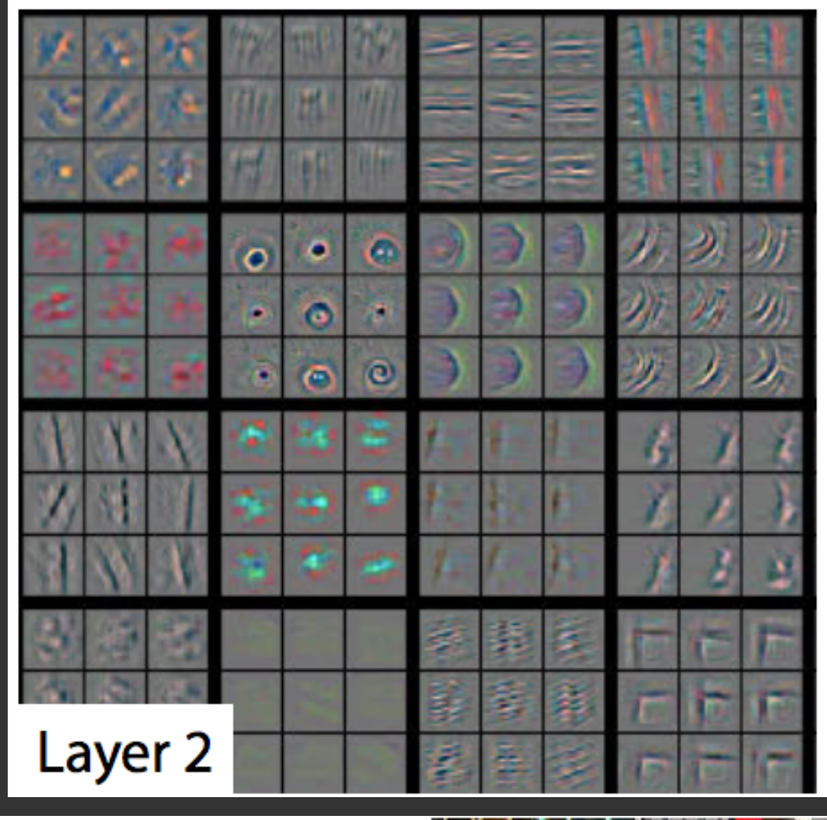


Krizhevsky 2012 classifier ("AlexNet"): trained to recognize objects in 1000 categories
First seven layers compute 4096-dimensional descriptor from image
Soft-max classifier performs classification on this descriptor.

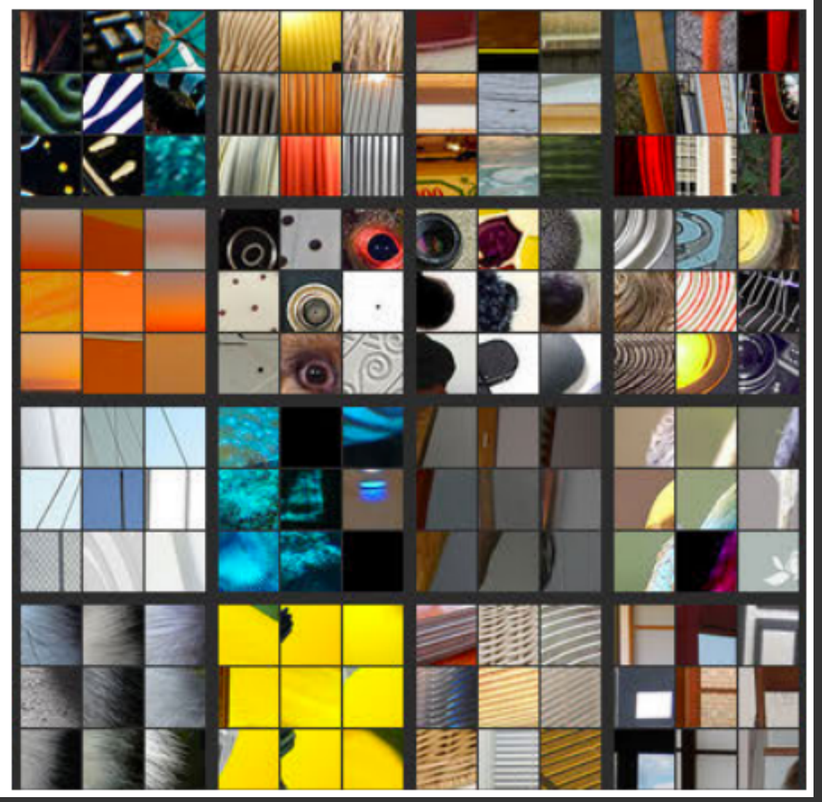
Visualizing responses of filters in network

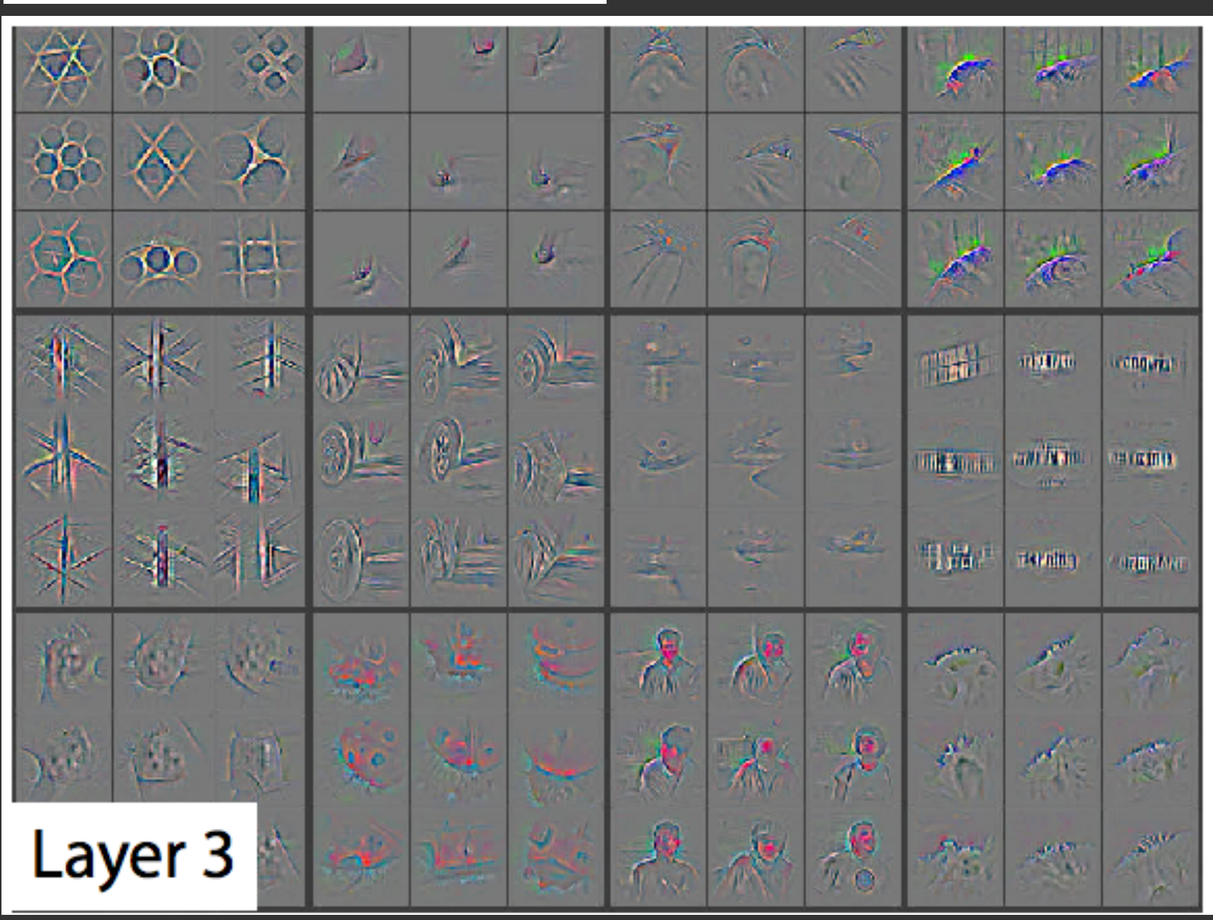


Layer 1

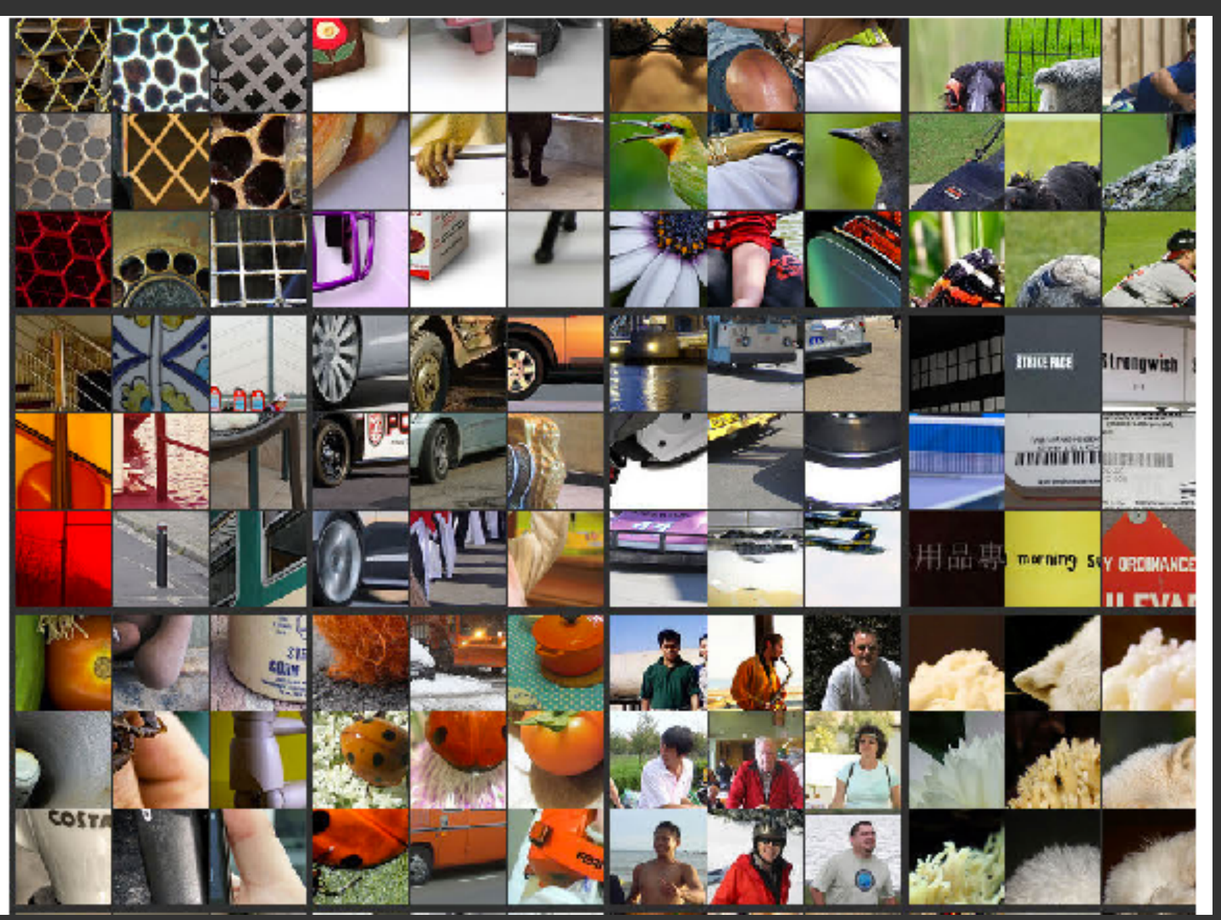


Layer 2

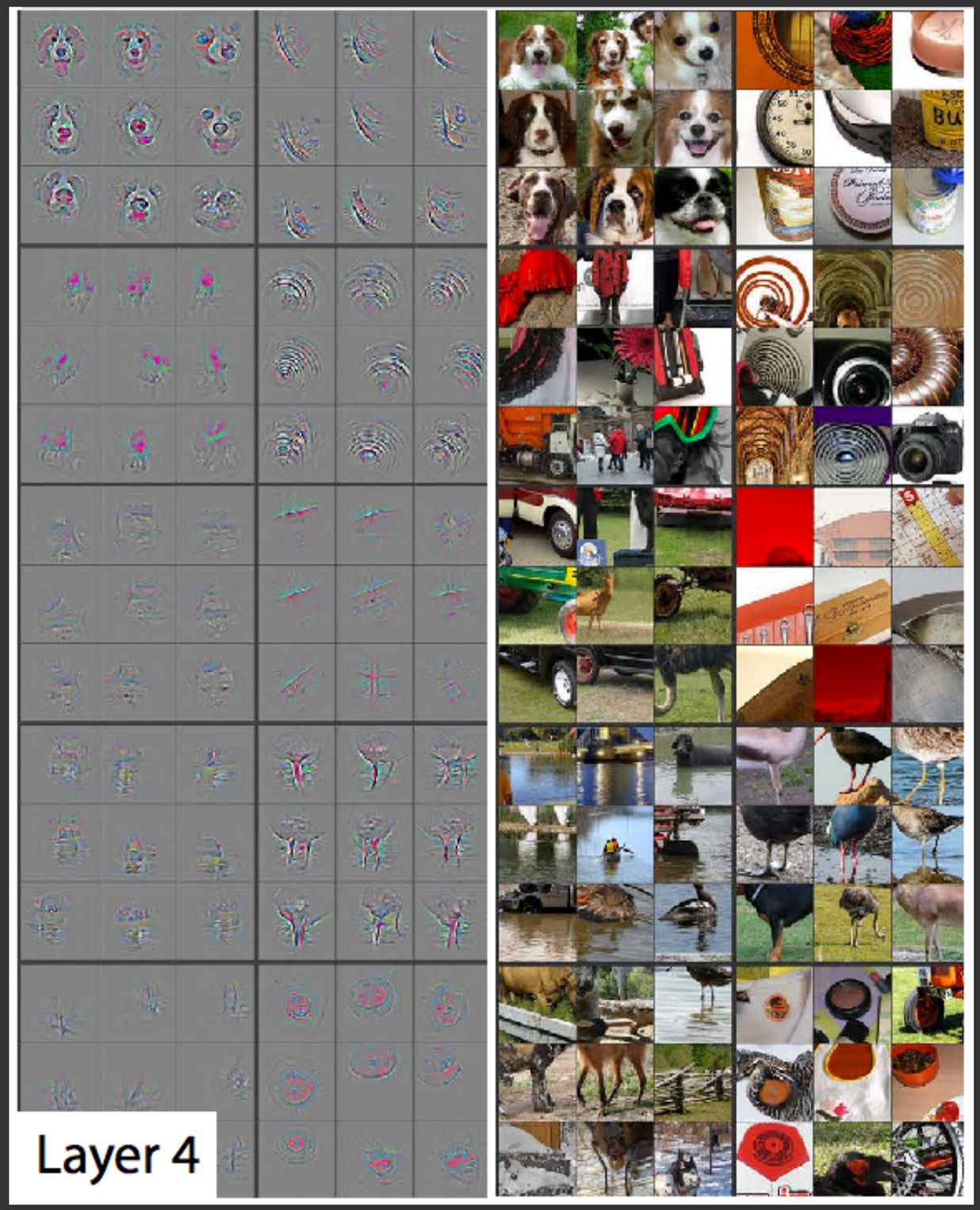




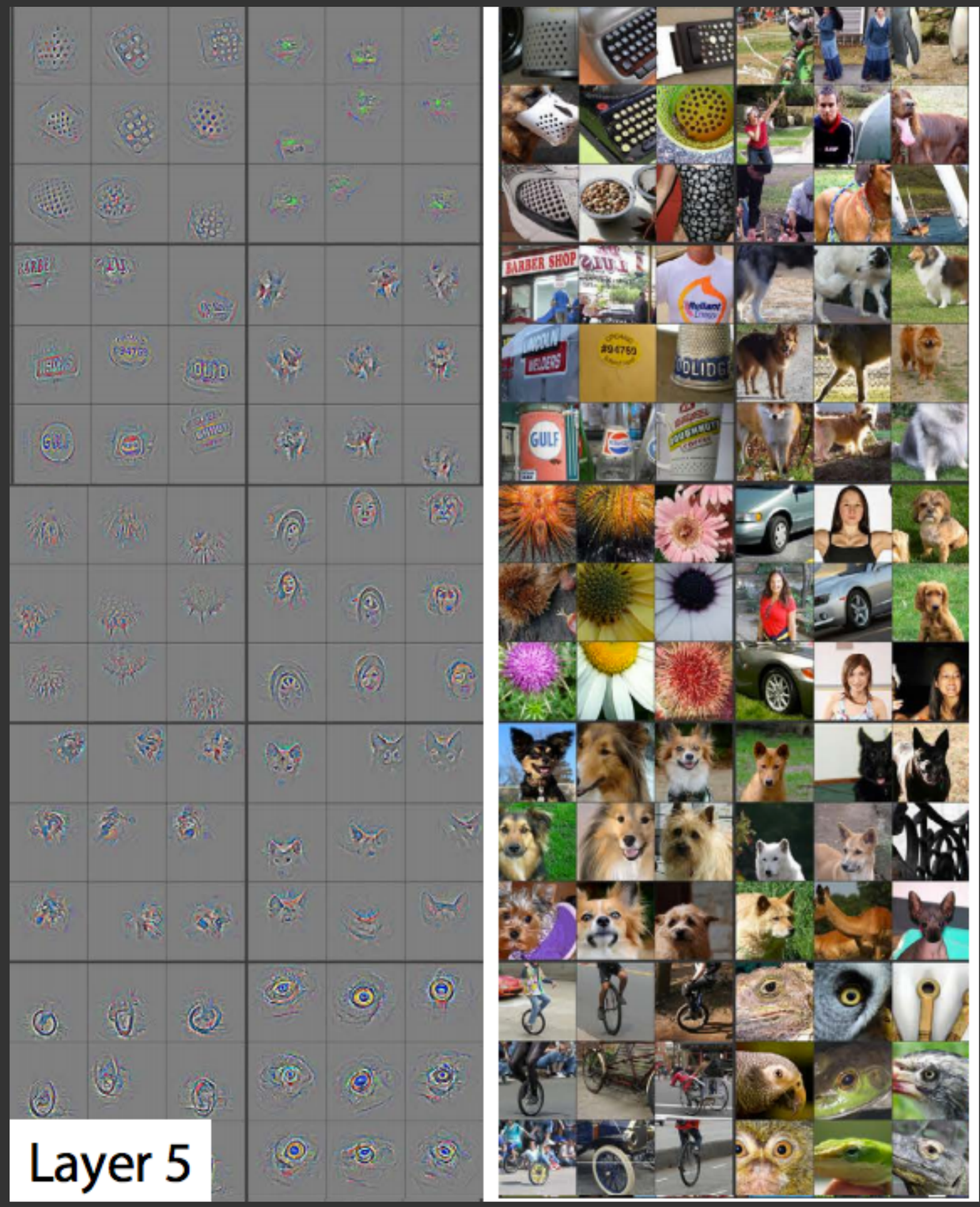
Layer 3



Visualizing responses of filters in network



Layer 4



Layer 5

Visual words

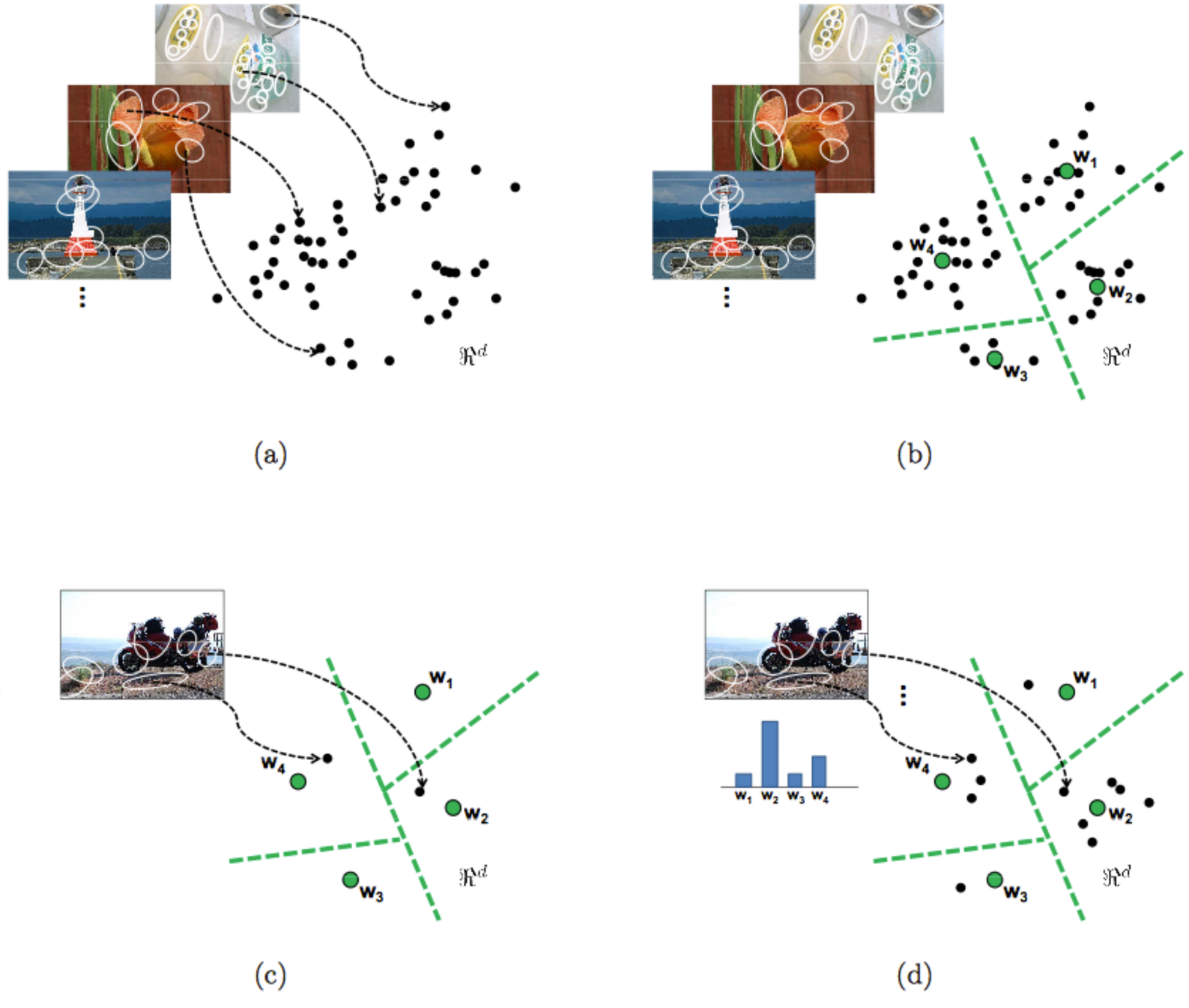
- Text document is made up of words (discrete values in a vocabulary)
- Descriptors are points in continuous high-dimensional descriptor space
- Idea: construct "visual words" from descriptors

(A) Features in images

(B) Compute "vocabulary" for dataset by clustering all features across all images: represent each cluster by its mean (or median) feature

(C) Bin (discretize) all image features by assigning feature to closest cluster in vocabulary

(D) Represent image by its histogram of visual word counts



Bag of words (BOW) image descriptor:

- **Bag of words (BOW) descriptor:**
  - **Image descriptor is a histogram of word occurrences**
  - **Very sparse vector**

0	0	0	1	0	1	0	4	0	0	0	0	0	8	9	3	0	0	0	. . .

- **Given query image descriptor *q*, compute score for database image *d*:**
  - **Example: dot product of normalized query descriptor and DB image descriptor:**

$$\text{score}(q,d) = \frac{q \cdot d}{\|q\|\|d\|}$$

  - **Improvement: weight descriptor elements by visual word IDF values**
  - **Many alternative distance functions:**
    - **e.g., histogram intersection: $\min(q_i, d_i)$ rather than inner product**

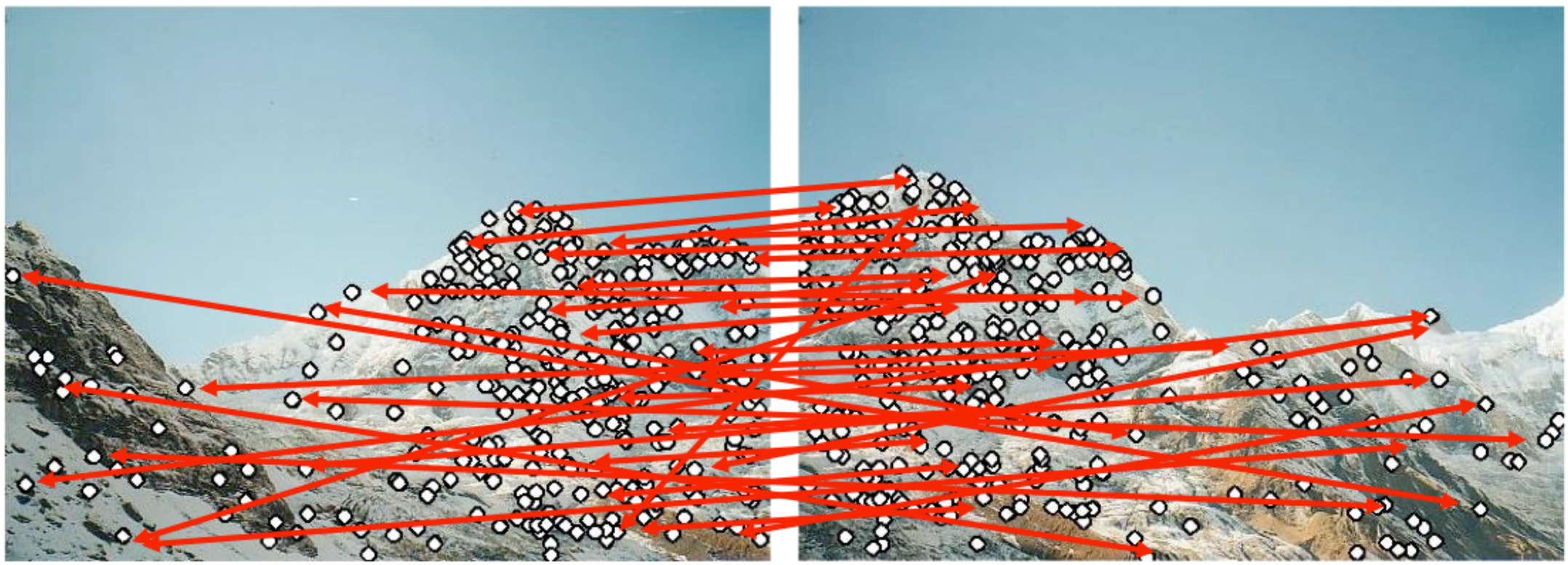
Summary

- **Image search using bag of words descriptors and an inverted index acceleration structure:**
  1. **Compute features for image collection**
  2. **Build vocabulary (visual words) by clustering features in collection**
  3. **Compute inverted index:**
     - **For each visual word, index stores list of images with word, plus the tf-idf weight for that word in that image:** $tfidf\_score(w, d, D) = tf(w,d) * idf(w, D)$
  4. **For each query image:**
     - **Compute BOW descriptor**
     - **Use inverted index to find candidate set of similar images**
     - **Compute score between query and candidate images (e.g., dot product of descriptors)**
     - **Rank results by score**

Background part 2: nearest neighbor search using a KD-tree

Search application: establish feature correspondence

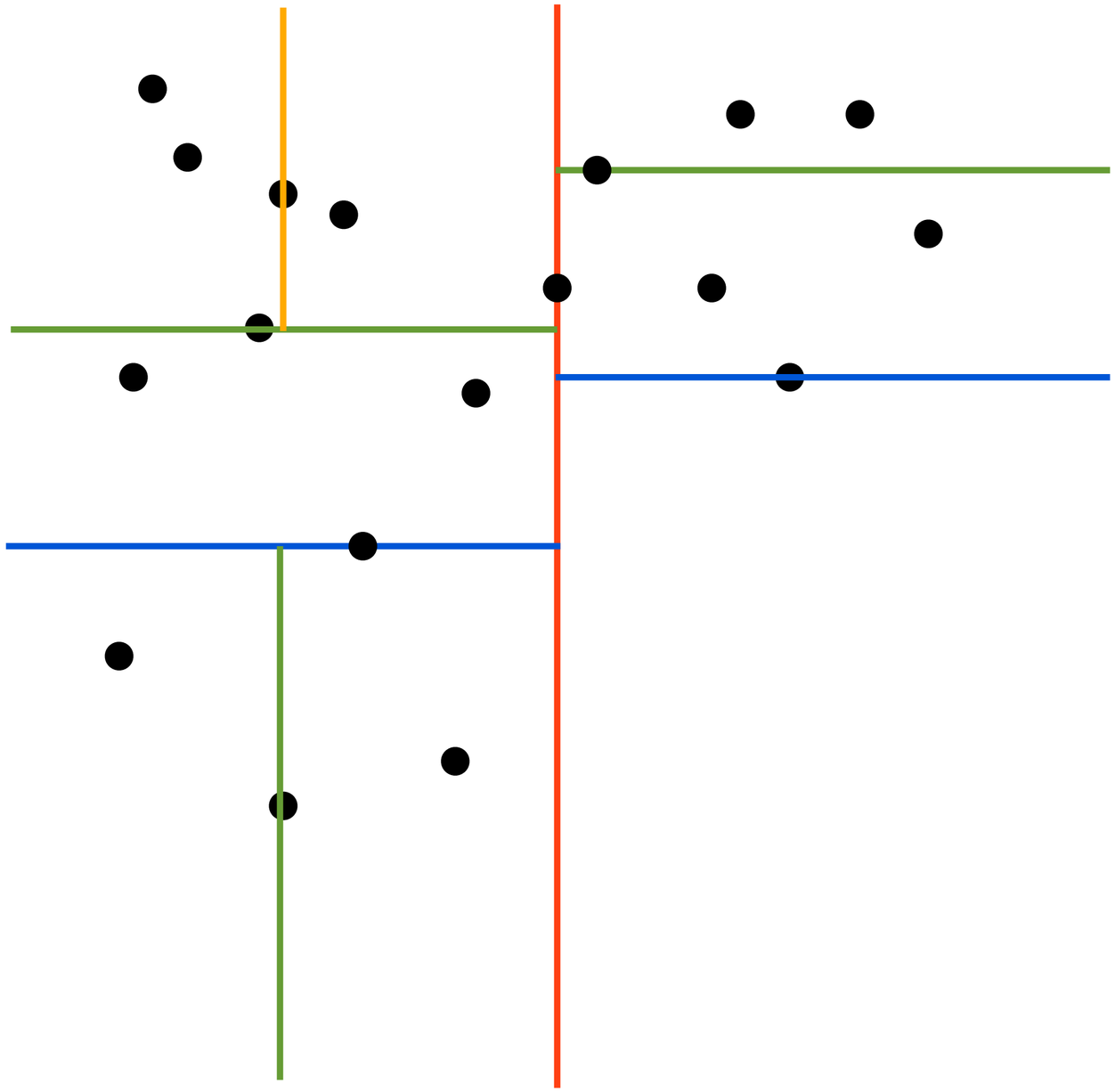
- Example: SIFT descriptor (length-128 vector)
- For all descriptors in image 1, find nearest neighbor descriptor in image 2

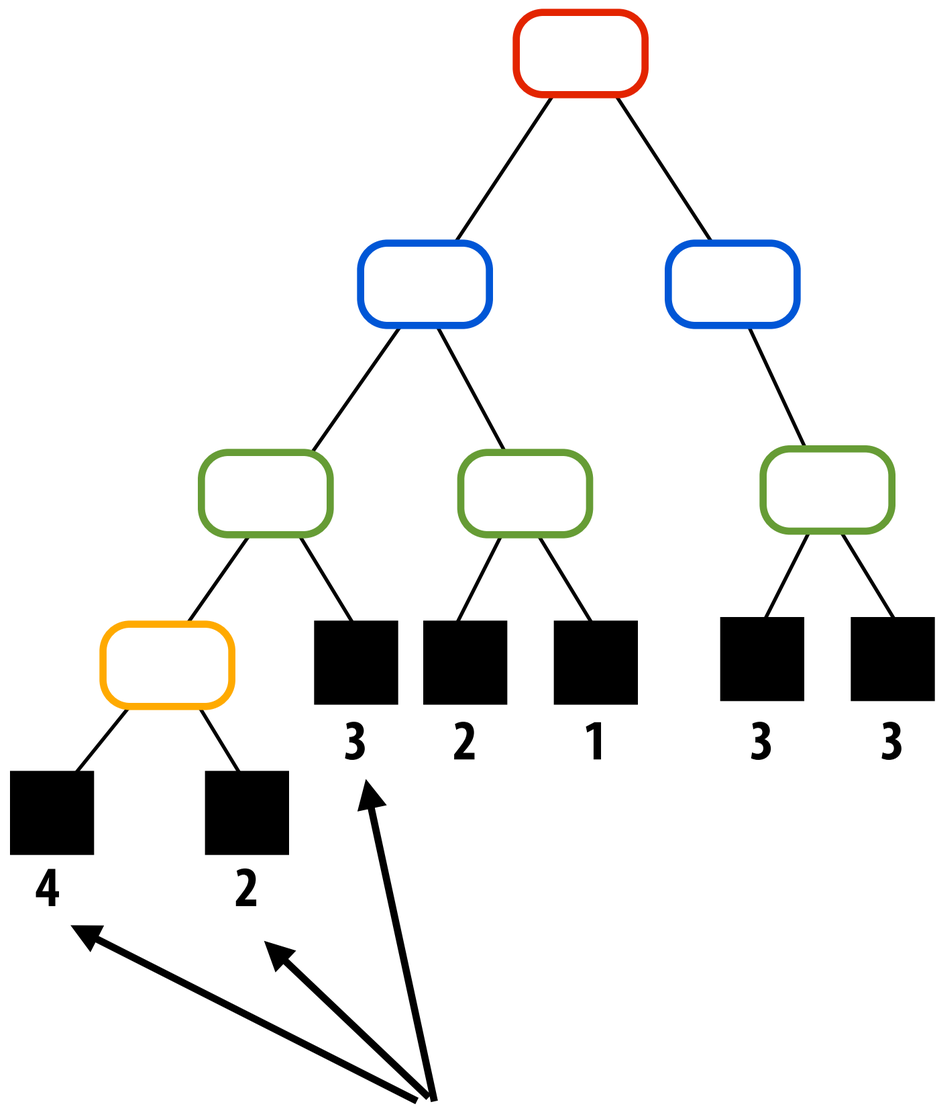


1 2

Review: K-D tree

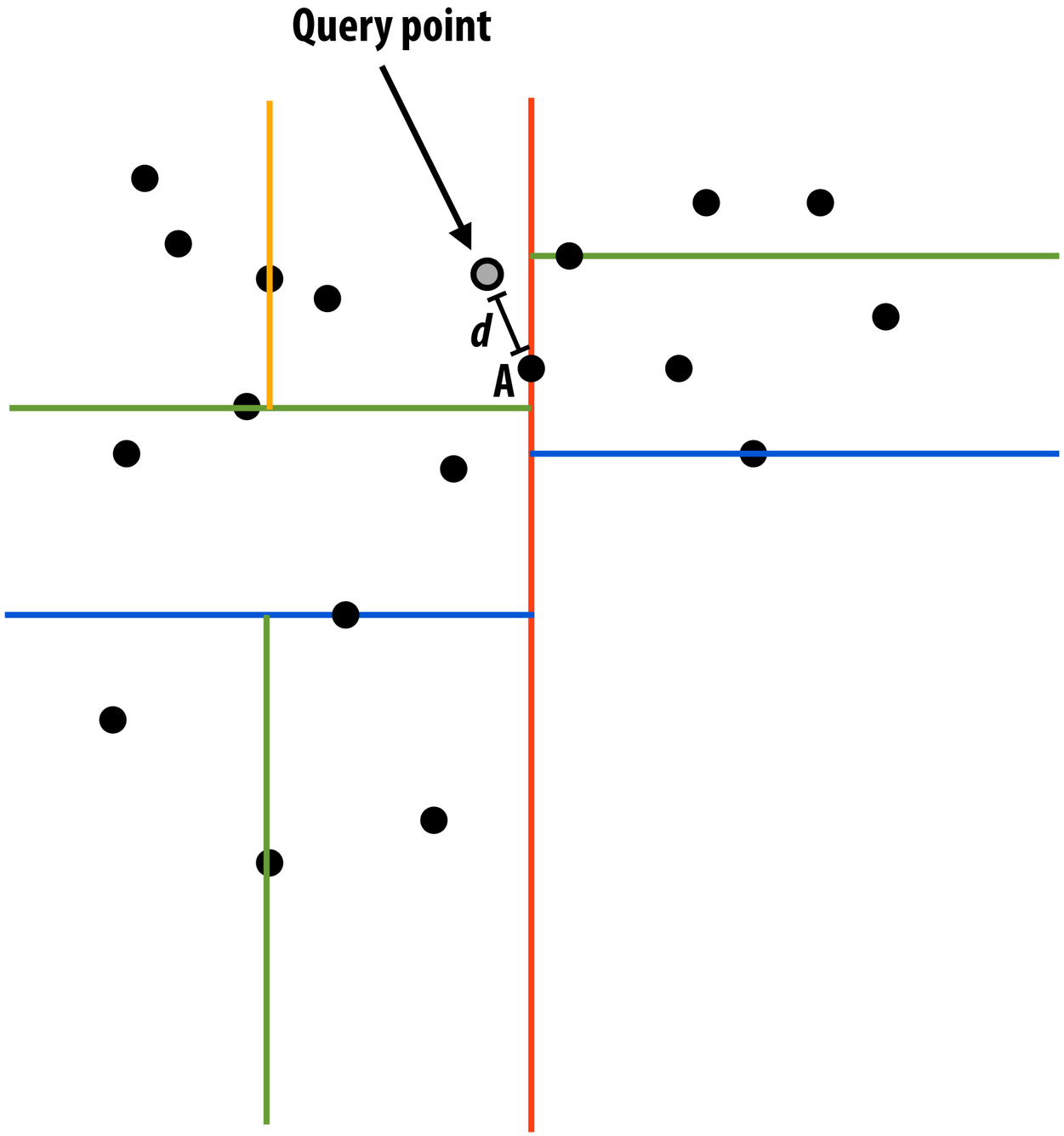
- Spatial partitioning hierarchy
- K = dimensionality of space (below: K = 2)

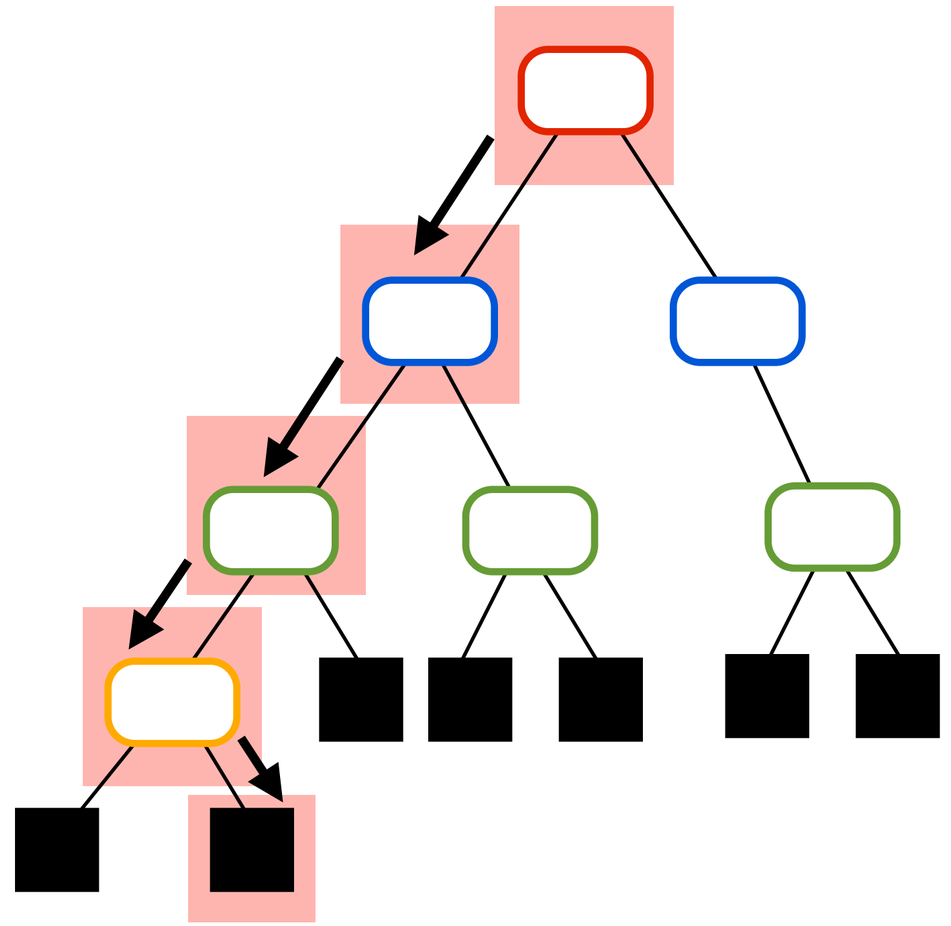




Nearest neighbor search with K-D tree

Step 1: traverse to leaf cell containing query: compute closest point in this cell to the query.

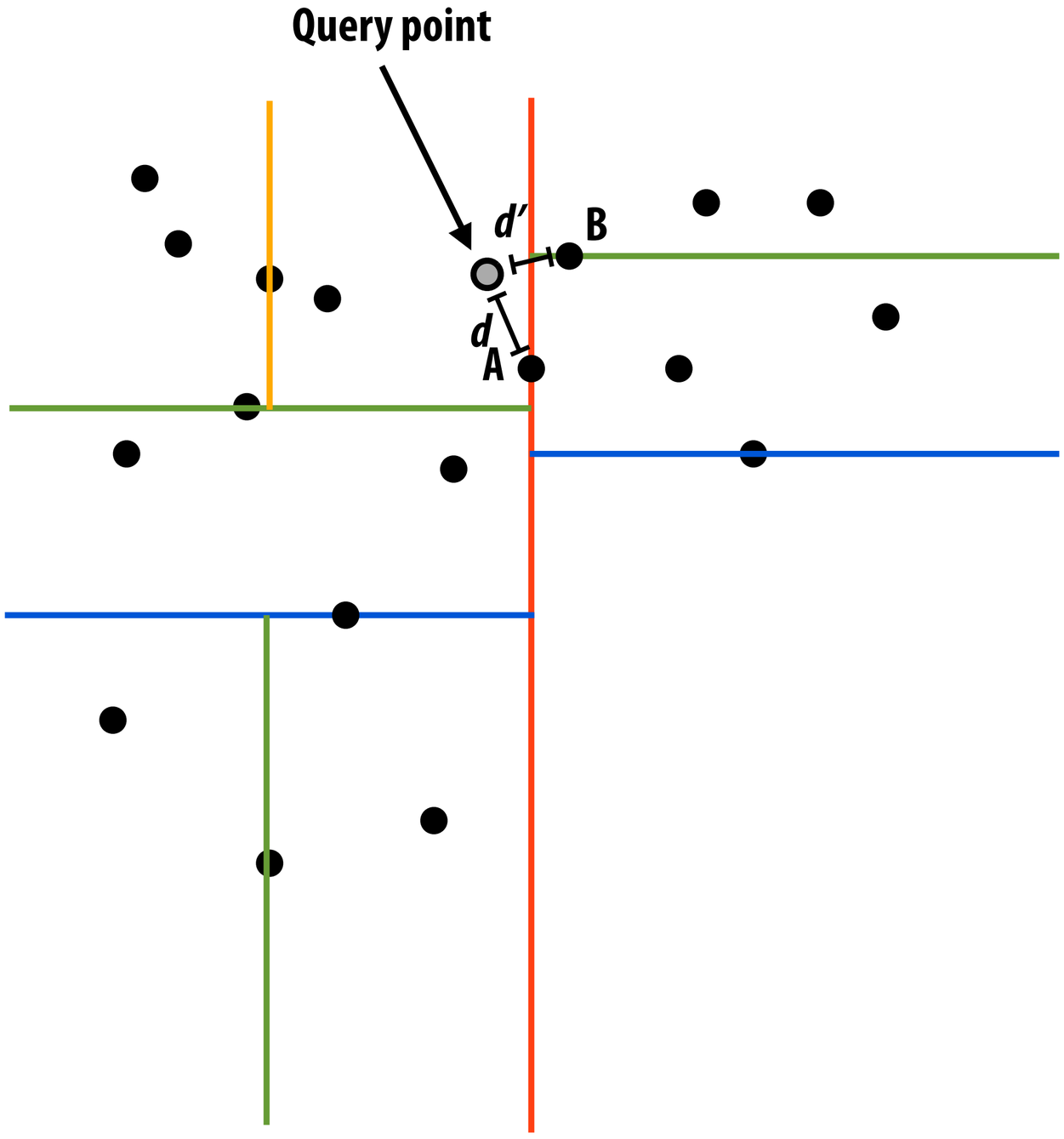


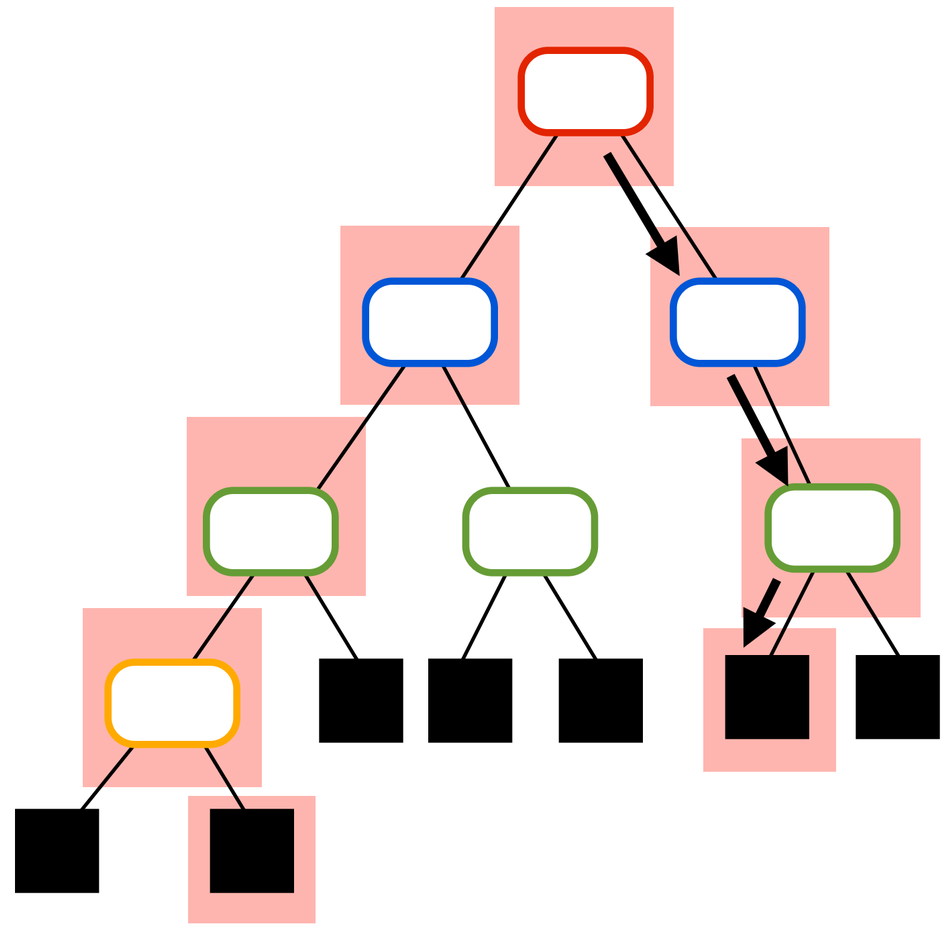


Closest so far: A (at distance *d*)

Nearest neighbor search with K-D tree

Step 2: backtrack: if distance to other cells is closer than distance to closest point found so far, must check points in this cell

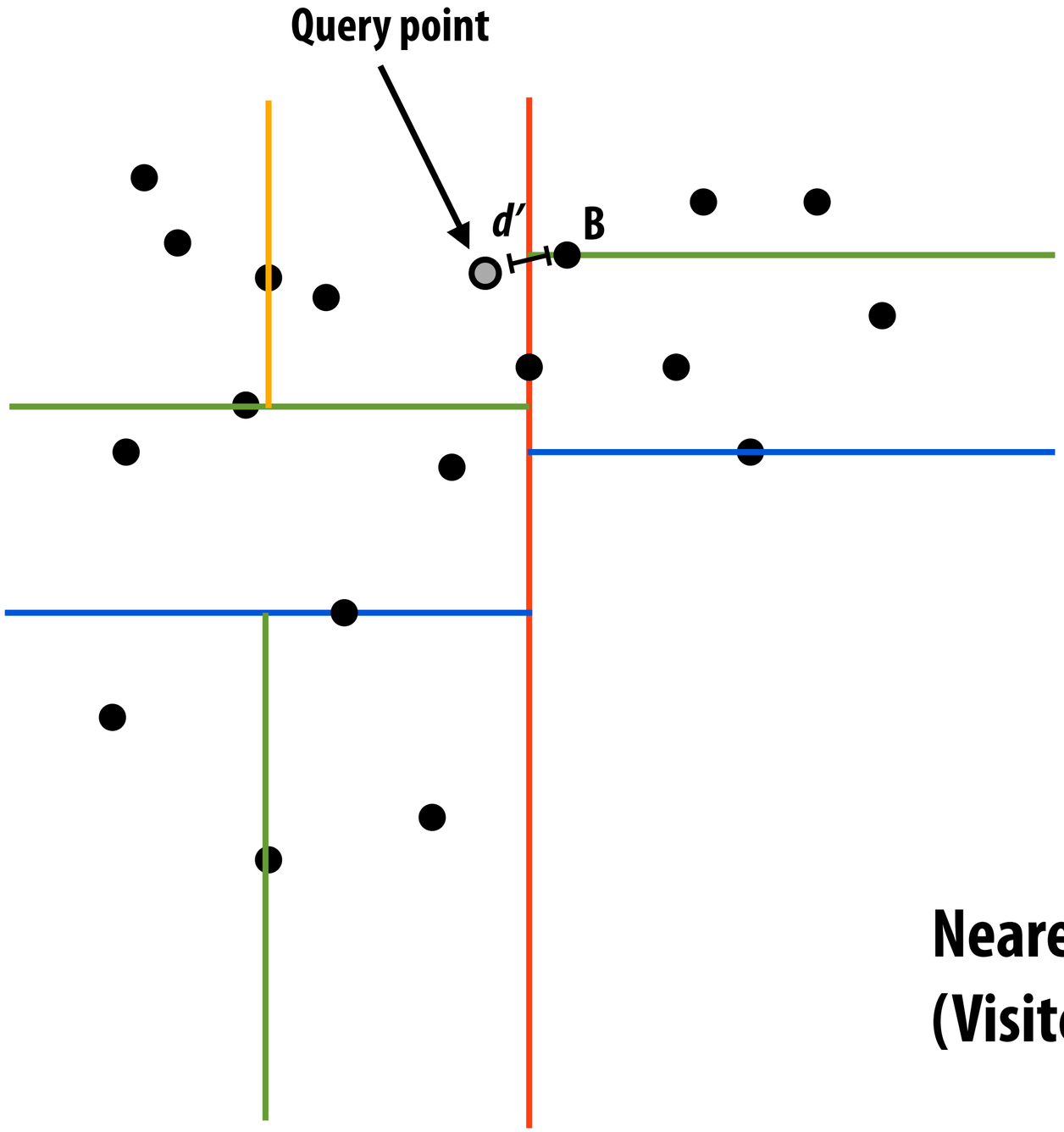


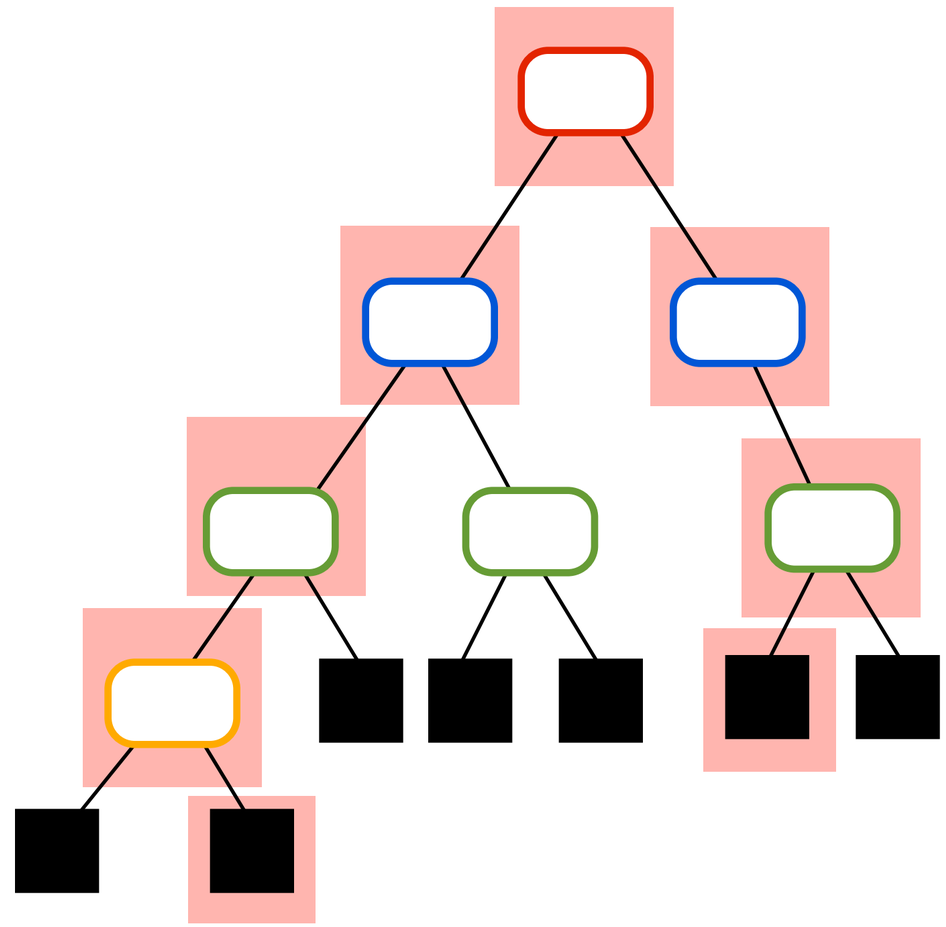


Closest so far: B (at distance *d′*)

Nearest neighbor search with K-D tree

Step 2: backtrack: if distance to other cells is closer than distance to closest point found so far, must check points in this cell

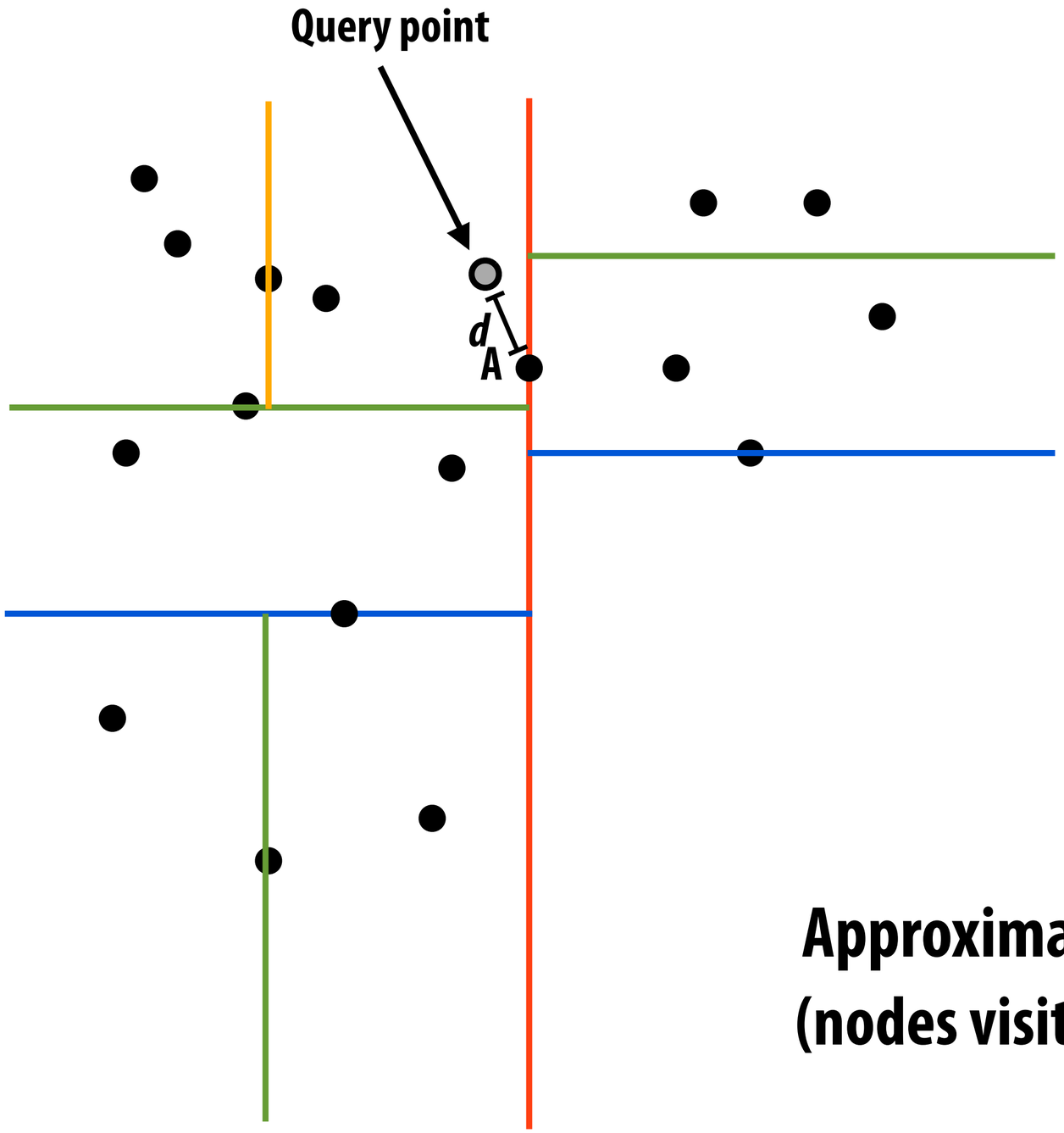


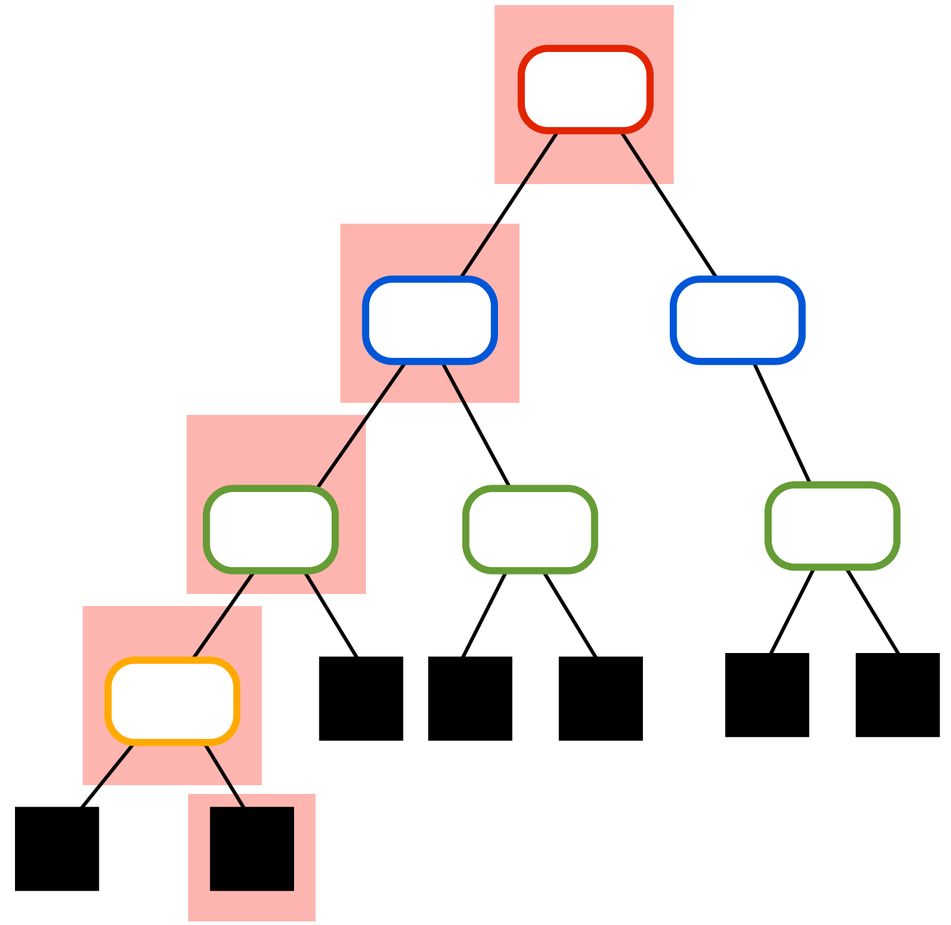


Nearest neighbor result: B (at distance *d'*)
(Visited nodes during query shown in pink)

Approximate nearest neighbor (ANN) search

One simple answer: just take closest point in leaf node containing query



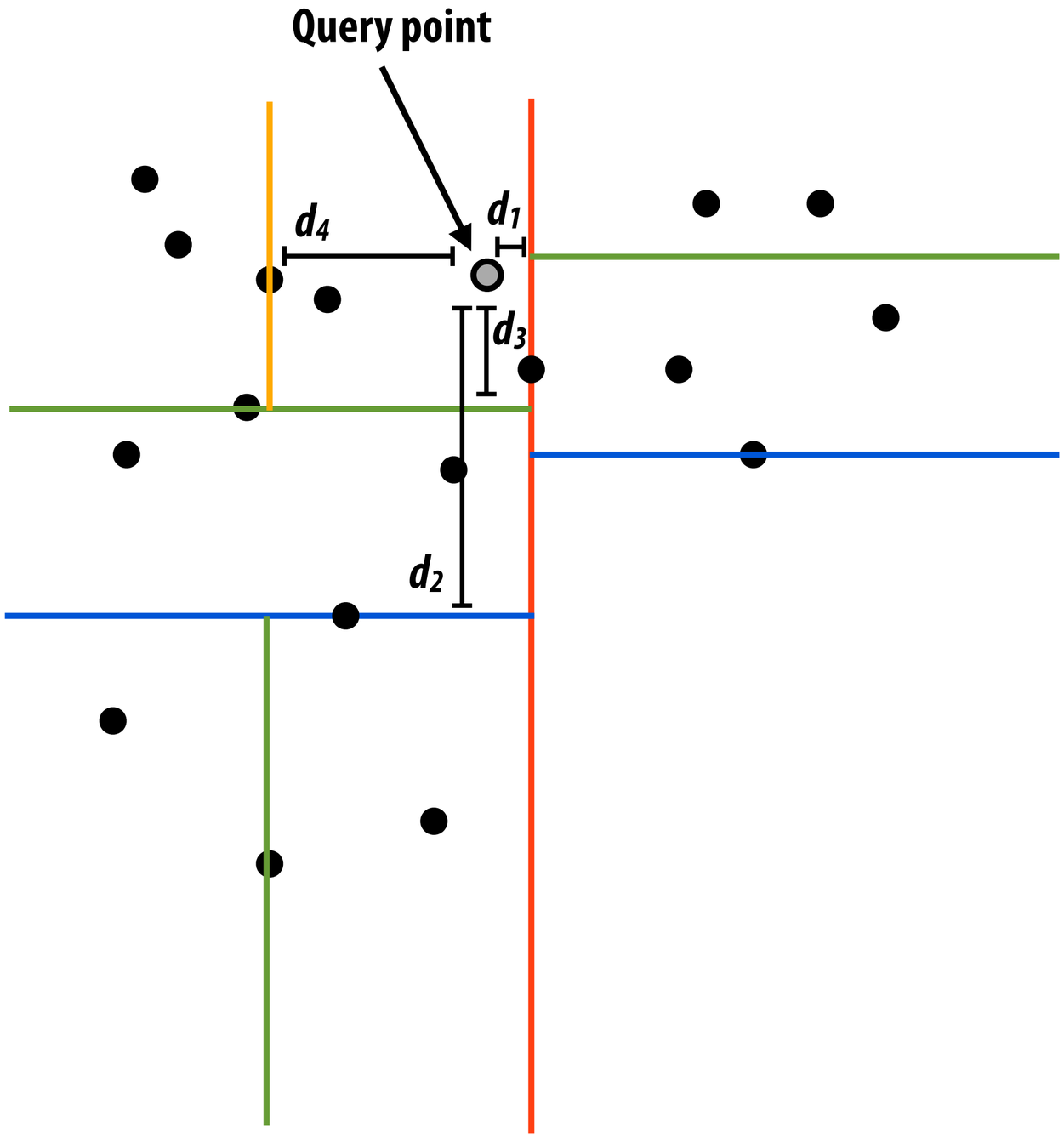


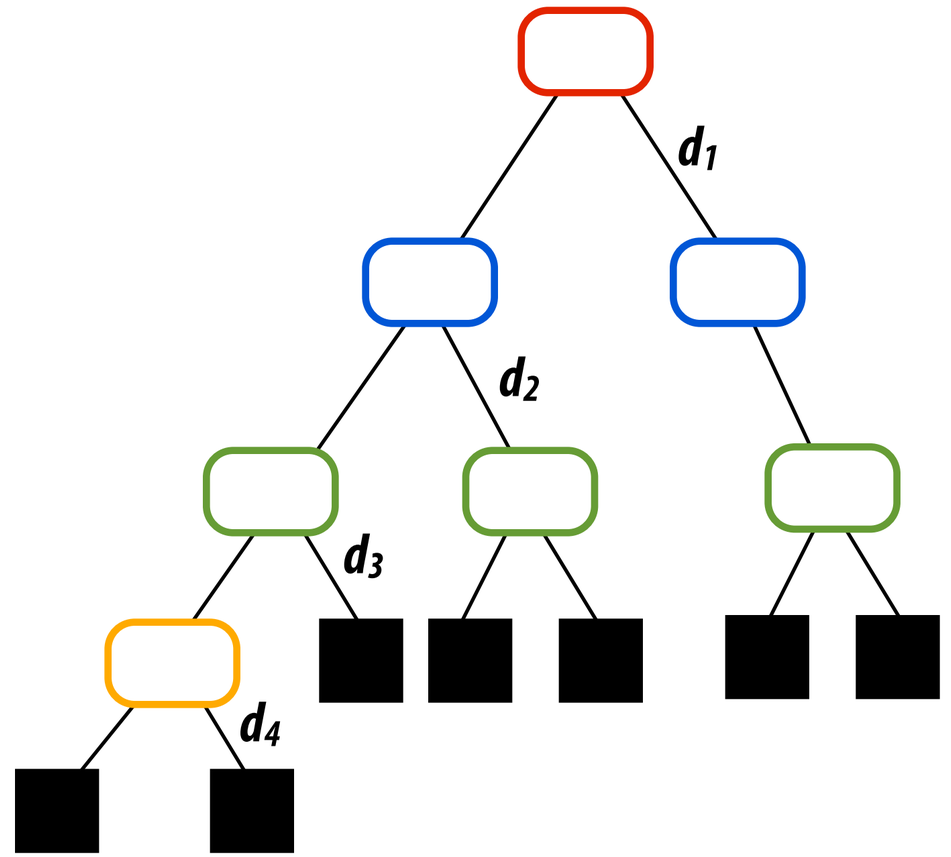
Approximate nearest neighbor: A (at distance *d*)
(nodes visited during query shown in pink)

Approximate nearest neighbor search

Improvement: place nodes in priority queue during downward traversal

Resume downward traversal from closest N nodes to query





Basic K-D tree build

- To find a partition for a node:
  - Partition axis for which the variance of current data points is the highest
  - Split at the median of the current data points

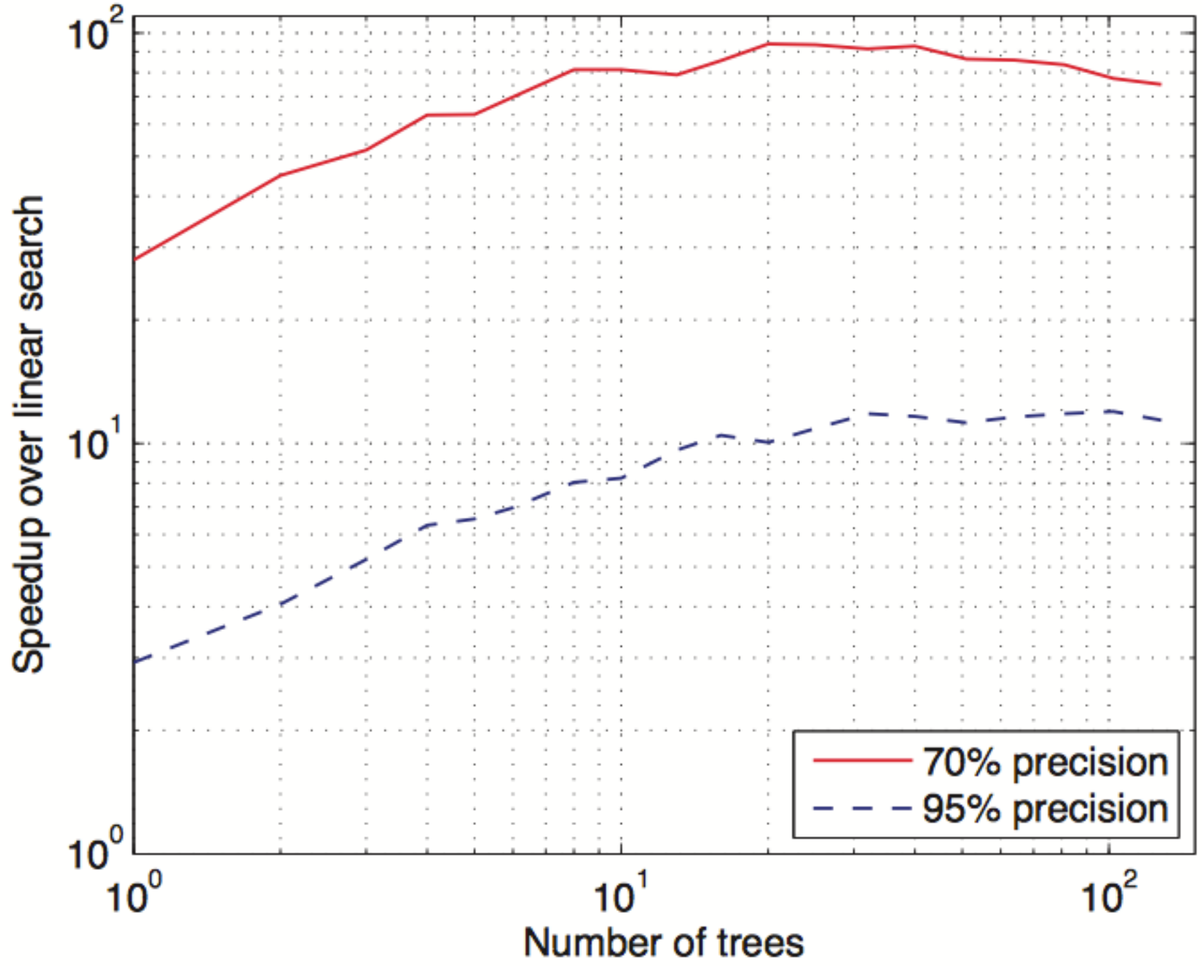
- You implemented nearest neighbor search in 3-D using a K-D tree when implementing photon mapping in 15-462 (there, K=3)

Randomized K-D tree

- To find a partition for a node:
  - Randomly choose axis to partition
    - Draw from distribution weighted proportionally with variance of current data points is the highest
    - Simple solution: pick partition axis by uniformly sampling from top N axes with highest variance
  - Randomly choose partition point
    - Draw from distribution heavily weighted at the median of the current data points (make it likely to split near the median of the data points)

Approximate nearest-neighbor (ANN) search using a forest of randomized K-D trees

- Construct a set ("forest") of random K-D trees
- For each tree, find NN in leaf cell containing query
  - Add all nodes (across all trees) traversed along the way to a priority queue (node priority = distance from query to node)
- Take closest of all answers across all trees as an initial ANN
- For top D nodes in queue, resume downward search from that node (D = 5 in figure [Muja et al. 2009])
- Solution for approximate k-NN as well

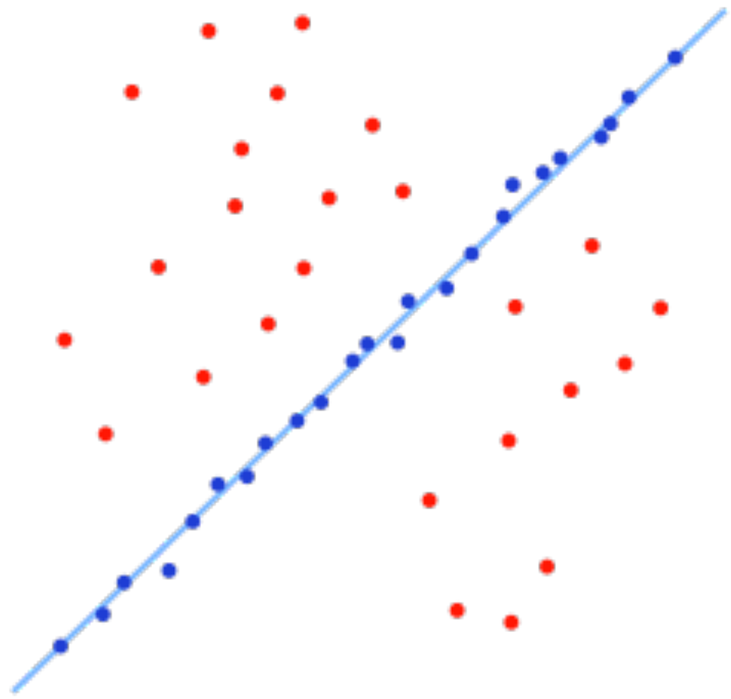


Background part 3: RANSAC

RANSAC

- RANdom Sample And Concensus
- Goal: fit model to collection of noisy data points

```
For i=0 to K:
   Perform random subsampling of datapoints (hypothetical inlier set)
   Fit model Mi to hypothetical inlier set
   For each datapoint d:
      Compute e = error(d, Mi)
      if e < threshold:
          d is in consensus set (it is consistent with model)
   If consensus set for Mi is larger than that for Mbest:
      Mbest = Mi
```



Red data points: outliers
Blue data points: consensus set

Let w = number of inliers / number of data points
= probability of selecting an inlier at random

So:

w^N = probability of selecting all inliers in hypothetical inlier set of size N

$(1-w^N)^K$ = probability that no iteration selects a set of all inliers after K RANSAC iterations

Image credit: Wikipedia

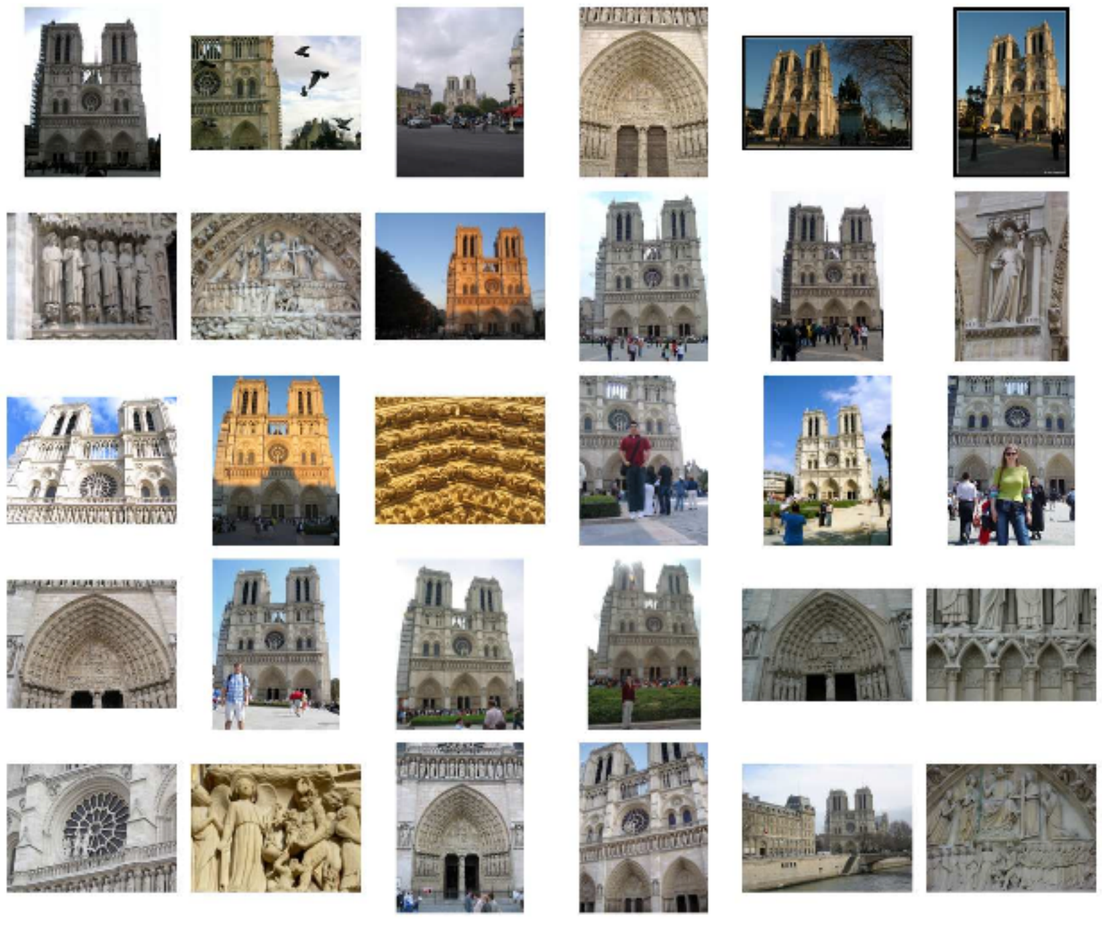
3D reconstruction from photos

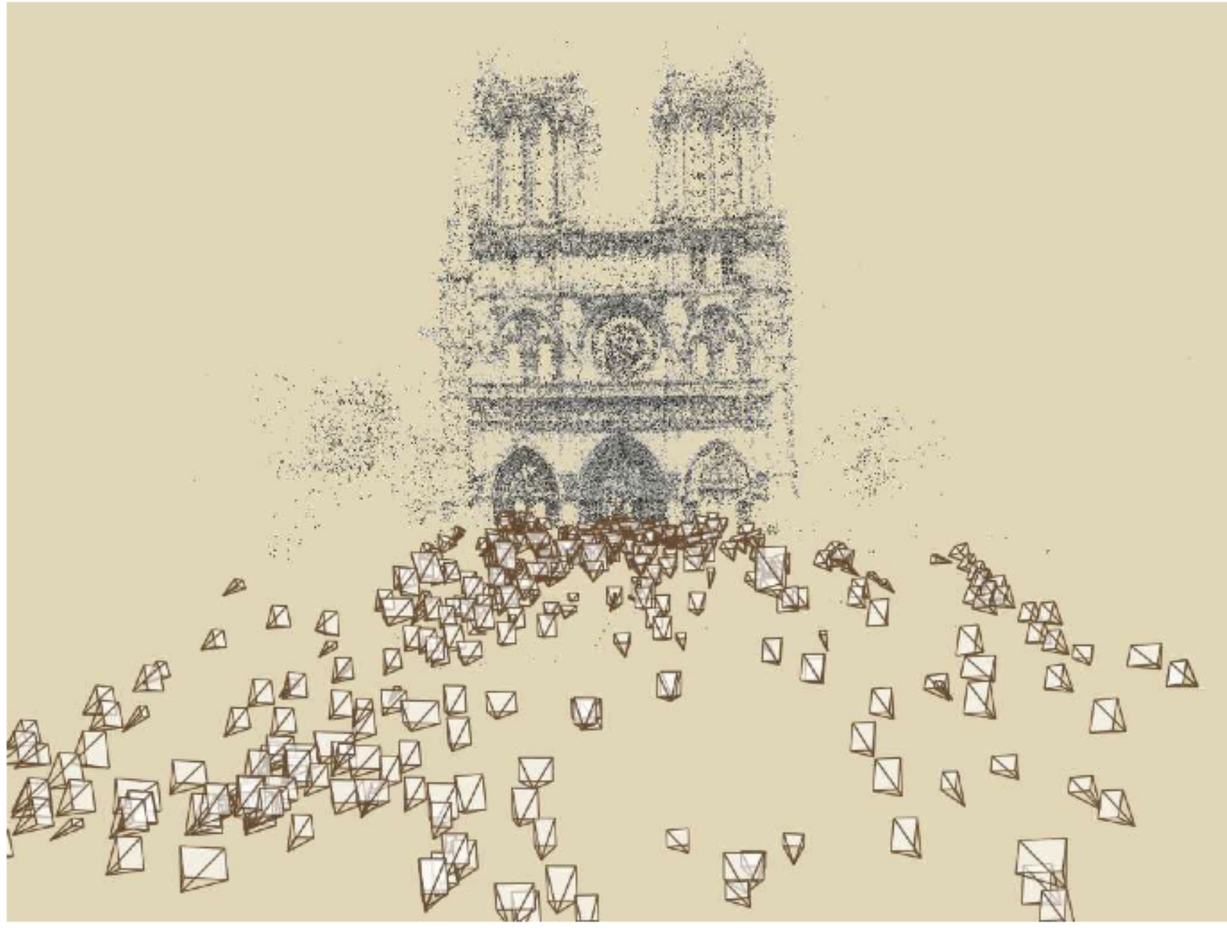
3D reconstruction from photo collections

- A good example of large-scale systems problem
- Efficient solutions involved combination of parallel execution and algorithmic innovation
- Today we will "black box" certain computer-vision techniques to focus on overall algorithm design/systems issues

Reconstructing scenes

- Input:
  - Unstructured collection of photos from same location (e.g., images from Flickr, Facebook)
- Output:
  - Sparse 3D representation of scene (point cloud)
  - Position of camera for each photo





Step 1: find matching images from collection

- Goal: find pairs of "matching" images containing views of the same object
- Step 1: compute feature points for all images (SIFT keypoint descriptors: 128-elements)
  - Generates thousands of keypoints per image
- Step 2: for each pair of images (I, J), determine if a match exists:
  - (A) find potentially matching keypoints (similar descriptors)

```
Compute K-D tree for all keypoints in J
for each keypoint i in I:
   // d1, d2 are distance to first nearest neighbor j1 and second NN j2
   (d1, d2) = perform approximate nearest neighbor (ANN) lookup for i
   if (d1/d2 < threshold)
      i in I and j1 in J are candidates for being the same point on the same surface
```

  - Output of (2-A): pairs of matching keypoints in image I and J
  - (B) verify matching keypoints: attempt to find geometric relationship between the two viewpoints: estimate a fundamental matrix (3x3 matrix, rank 2) for the image pair using RANSAC:
    - Select eight matching keypoints at random, estimate F-matrix
    - If there are not at least 20 inlier keypoints, repeat

Recall: for key point visible at point p_1 in image 1 and p_2 in image 2, p_2 should lie on the epipolar line of p_1: $p_1^T F p_2 = 0$

Geometric verification (example in 2D)

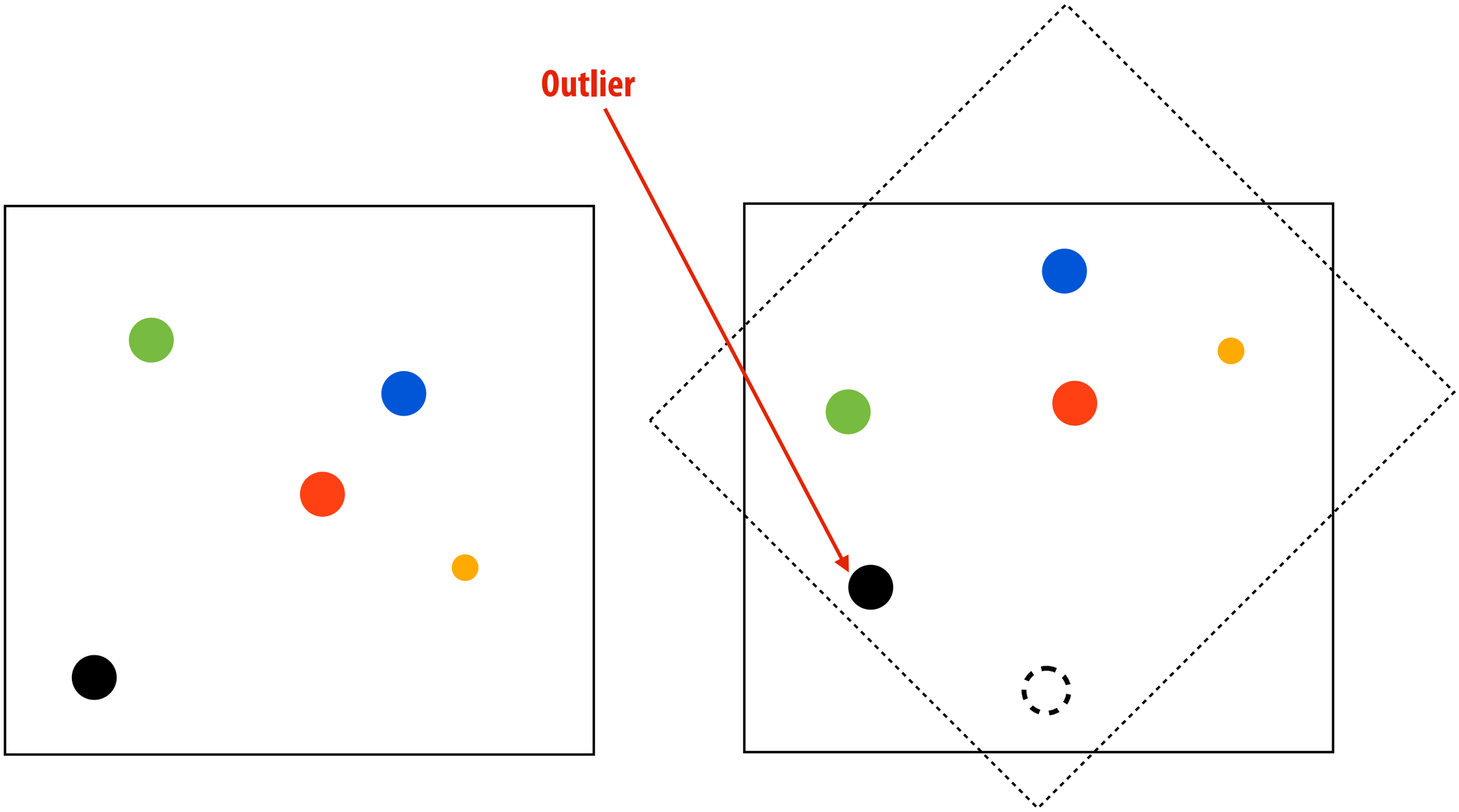
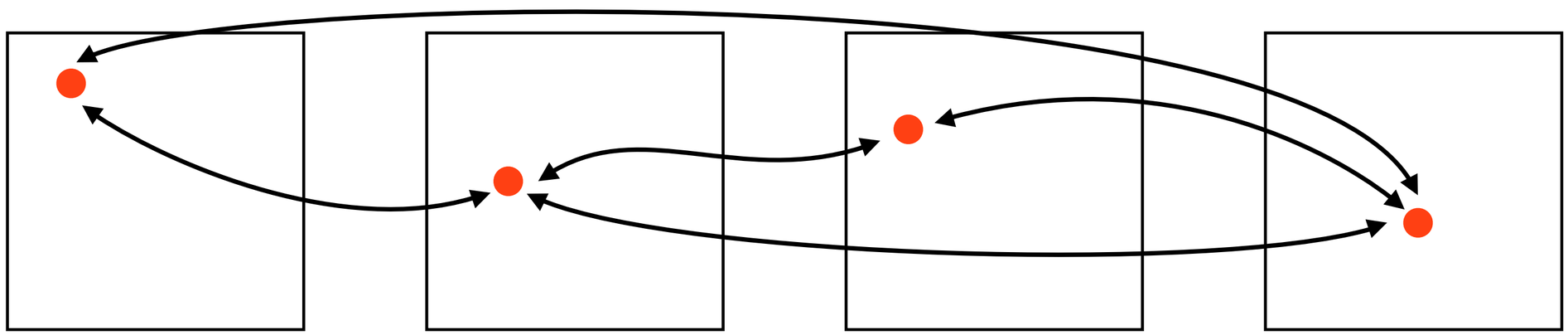


Image 1

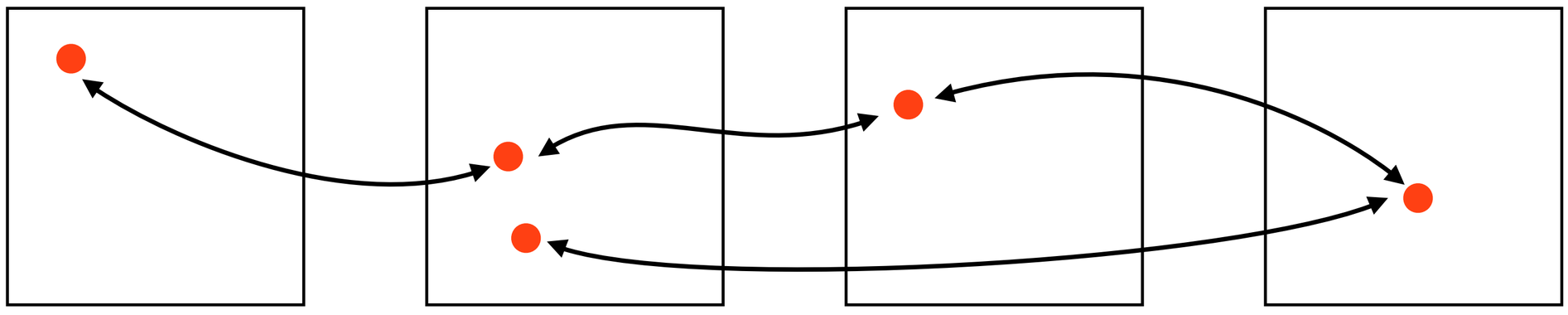
Image 2

Step 2: organize matches into tracks

- Track = connected set of matching keypoints
  - A track corresponds to a single point in the scene
  - Track must contain at least two keypoints
  - All images in a track are different views of that scene point



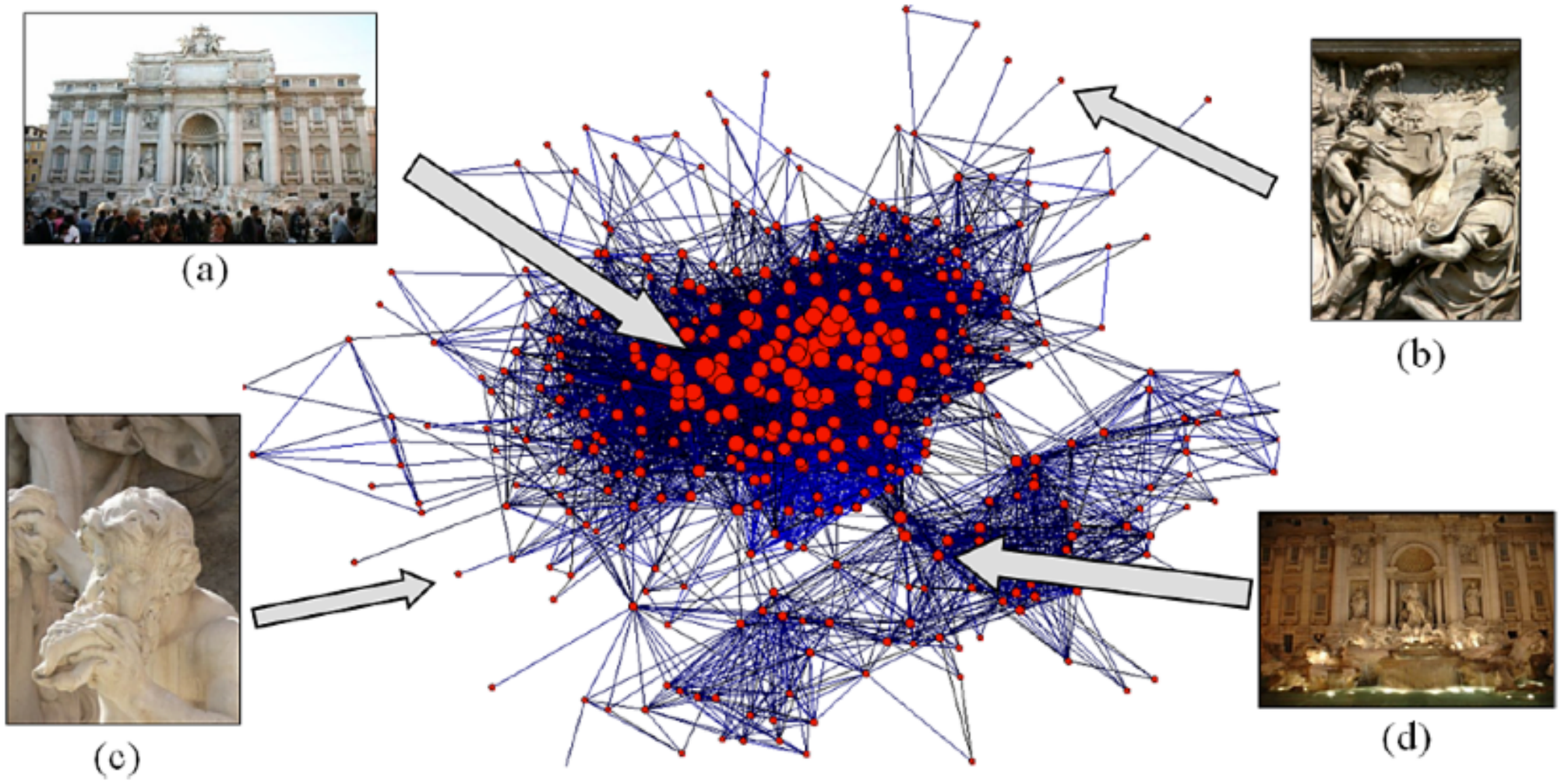
Consistent track: black arrows indicate matching keypoints in difference images



Inconsistent track: contains two keypoints in one image
(clearly, all both keypoints in this image cannot correspond to same scene point)

Image connectivity graph

- Graph nodes = images
- Graph 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

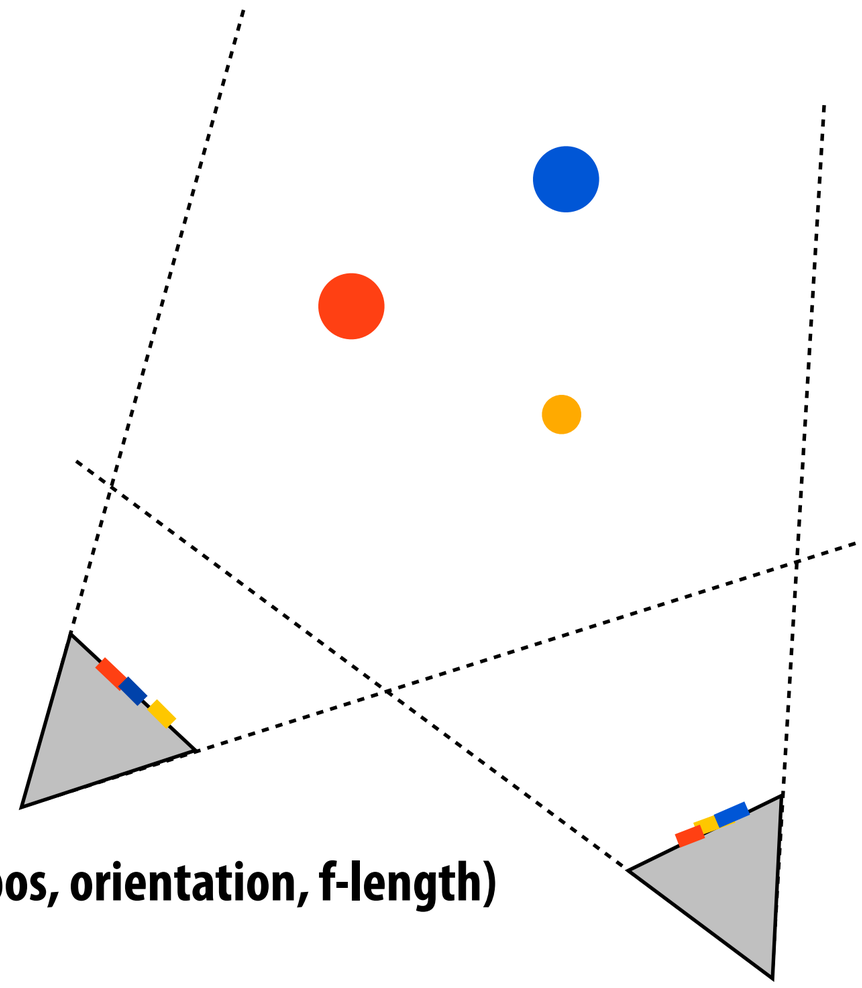
Step 3: structure from motion (SfM)

- 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 algorithm is most confident in (avoid local minima)
- Initialization:
  - Pick 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 number of 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 keypoint 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 into consistent tracks
- Until no new cameras can be estimated:
  - Pick next camera to estimate
  - Refine estimate globally using bundle adjustment

Accelerating match finding

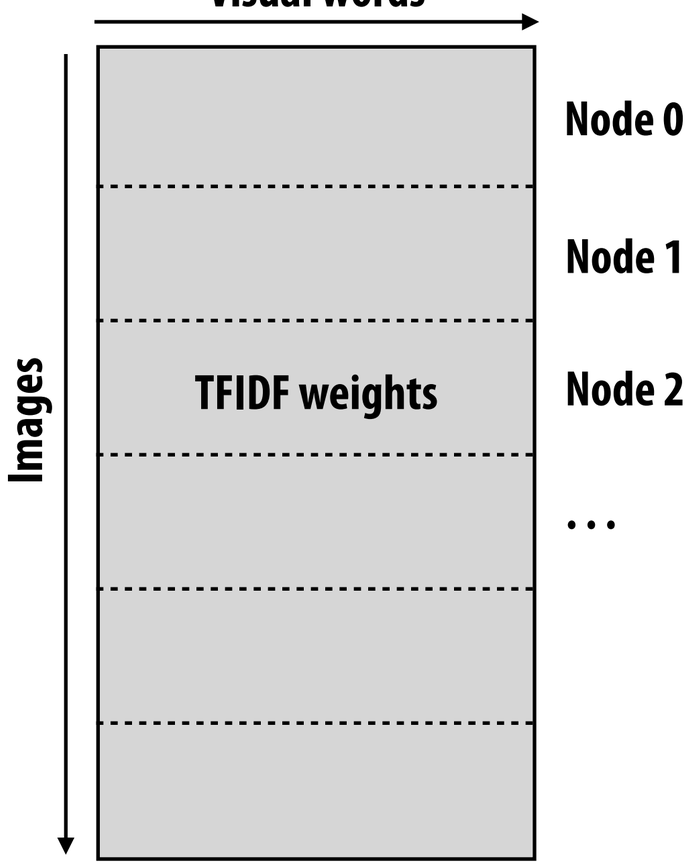
- A naive formulation of match finding is $O(N^2)$: for each image check for match against all other images
  - Large image collections → large N
  - Computing a match is expensive: (recall, requires finding a geometric fit via estimating fundamental matrix): ~ a few matches per core per second
  - N=1,000,000, 10 matches per second per core = 3,100 CPU years

- Must avoid performing expensive check on all possible matches!

- This is a retrieval problem! ("quickly find most likely matches")

Accelerating match finding

- Step 1: use fast retrieval techniques to find candidate matching images
  - e.g., use inverted index with TF-IDF weighting
  - Result: top-k nearest neighbors 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 top-k NN 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)

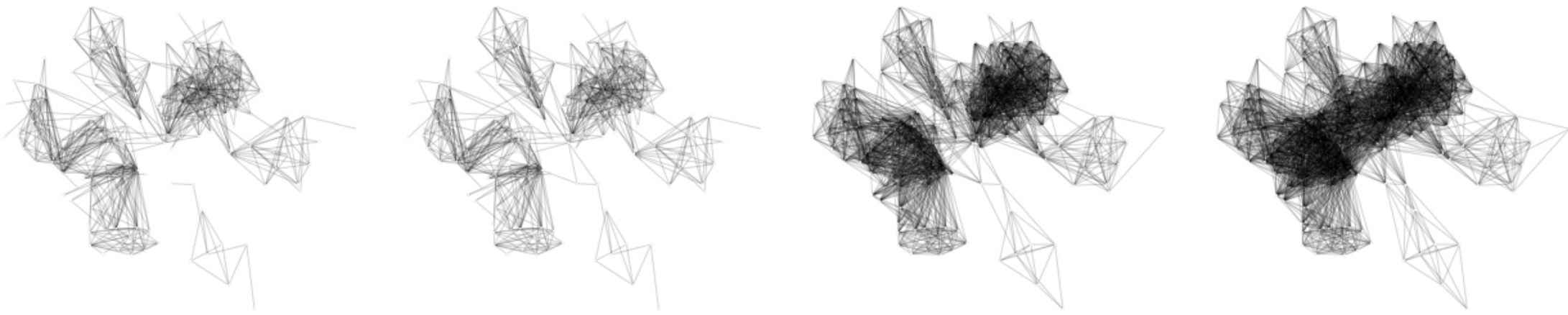
- Want a match graph that is sufficiently dense to enable 3D reconstruction:
  - Want as few connected components in match graph as possible (note: each connected component will be its own 3D scene after reconstruction)
    - Prefer a single, large scene reconstruction, not many "pieces" of scene
  - Want multiple views of the same track (i.e., want multiple images containing the same features to aid robustness of bundle adjustment)

Building a match graph

[Agarwal 2009]

- Step 1: Compute k nearest neighbors using acceleration structure, $k = k_1 + k_2$
- Step 2: Perform geometric verification of top k_1 matches, add graph edge when verification succeeds
- Step 3: Verify next k_2 matches, but only verify image 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
      If I and K are in different components: verify (I, K)
```



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

Putting it all together (distributed implementation)

1. In parallel across all nodes, compute features
2. Compute IDF weights via reduction, broadcast to all nodes
3. Broadcast TFIDF information (weight table) to all nodes
4. Independently compute $K=k_1+k_2$ NN on all nodes
5. For each image i, verify top k_1 candidates (parallelized dynamically via shared work queue across nodes)
6. Compute match graph connected components (sequentially on one node is easiest)
7. For each image i, verify next k_2 candidates if candidate is not in same graph connected component (dynamic parallelization) as i
8. For each image i, verify further matches based on candidates returned from query expansion
   - Repeat for N rounds, or until convergence
9. 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)
10. Compute graph skeletal set (next slide)

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

(a) Match graph

(b) Skeletal set (black nodes are skeletal set images, gray nodes are remaining images)

(c) Reconstruction from skeletal set

(d) Adding addition images with pose estimation

(e) final result post bundle adjustment

Result: 2x to 50x improvement in reconstruction performance

Image credit: Snavely et al. 2008

Systems problems, algorithmic solutions

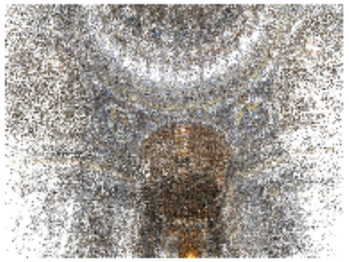
- Desire to work at scale triggered innovation in algorithms
  - Scale imposes new constraints

- 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

Reconstruction from 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

Task: location recognition

- Given a new image, how can we leverage an existing 3D reconstruction to estimate the camera's location and orientation?

Query image

Database image (keypoints shown)

- First-thought solution:
  - For each SIFT feature in query image, finding matching tracks in scene database of all images (recall: tracks correspond to scene features)
    - Possible implementation: ANN lookup using KD-tree built over database
  - Then attempt camera pose estimation for query given the collection of matches

Left image credit: Mark Ordonez (via Flickr), Right image credit: Li et al. 2010

Observation

- Not all scene database 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 (many tourist images of Dubrovnik, Croatia on Flickr contain this feature)

Observation

- Not all scene database 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 (many tourist images of Dubrovnik, Croatia on Flickr contain this feature)

- Idea: use up-front knowledge of likelihood of scene points to appear in images… to accelerate image feature matching

Idea: analyze image database to accelerate matching

- Previously in this lecture: organize database feature points into KD-tree to accelerate search
- Now: leverage co-occurrence and frequency of occurrence
  - We do not desire all matches in the database, only enough matches to estimate camera pose
  - Co-occurrence: it is sufficient to search over a small subset of scene points (since many scene points co-occur in the same images and are similarly useful for pose estimation)
  - Frequency of occurrence: search for the points that are most likely to be in the query image.

K-coverings of scene images

- **Subsample database: compute scene point set that is a K-covering of all images in the database**
  - **K-cover: set of points such that at least K points are present in each image**
  - **Simple greedy algorithm to compute K cover:**

```
S = {}    // set of points in covering
sort all scene points by number of images they appear in
while K-cover not reached by S:
    add point P appearing in largest number of images into S
```

- **Precompute two K-coverings for image database**
  - **P^s: 5-covering, capped to at most 2,000 points**
  - **P^c: 100-covering**

Localization algorithm

- Query = list of feature points
- Database = list of feature points
- Idea: rather than search database for matches to points in query image, search query list for matches database feature points
- Simple algorithm: tests scene points against query image in priority order

```
Compute Kd tree for points in query
Initial prioritization of database points:
  Highest priority points: Ps
  Next highest priority points: Pc
  Remaining points: priority = number of images point is visible in

while additional matching points are required:
   Attempt to match highest priority point against query points
   if match found:
       for each DB image I containing matched point:
           Increase priority of all DB points in I
```

Dynamic reprioritization of DB points based on co-occurrence with matched points.

Recap: how the algorithm works

- Test the most likely to match images from the database first
  - Recall: only need a few matches to estimate 3D camera pose of query image
- Once a match is found, leverage co-occurrence of points in images to predict new matching points
- Desirable system behavior: optimize for the common case!
  - Common images get found very quickly
  - Uncommon images take longer to localize
  - Memory efficient: don't need to store acceleration structure for the entire database of images

Class discussion:

Alternative reconstruction strategy: KinectFusion (Izadi et al.)