Object Recognition and Augmented Reality

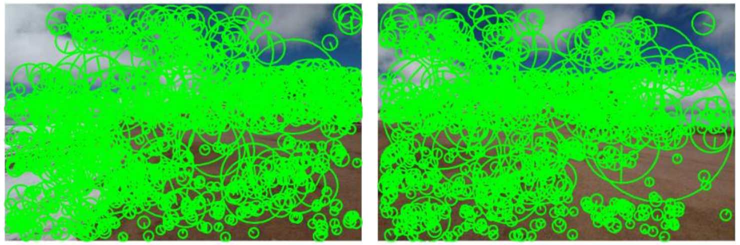
Dali, *Swans Reflecting Elephants*

Computational Photography

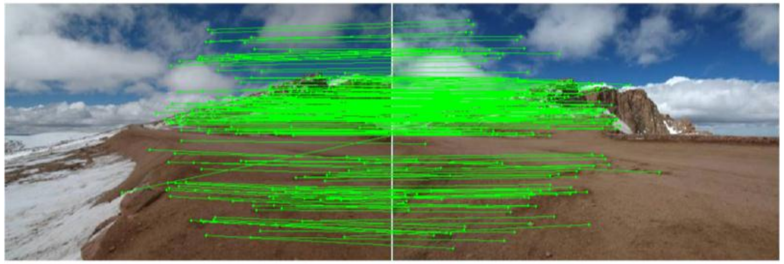
Derek Hoiem, University of Illinois

Last class: Image Stitching

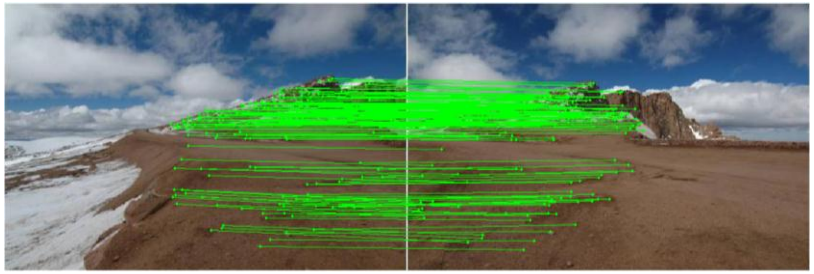
1. Detect keypoints



2. Match keypoints



3. Use RANSAC to estimate homography



4. Project onto a surface and blend

Augmented reality

- Insert and/or interact with object in scene
  - Project by Karen Liu
  - Responsive characters in AR
  - KinectFusion

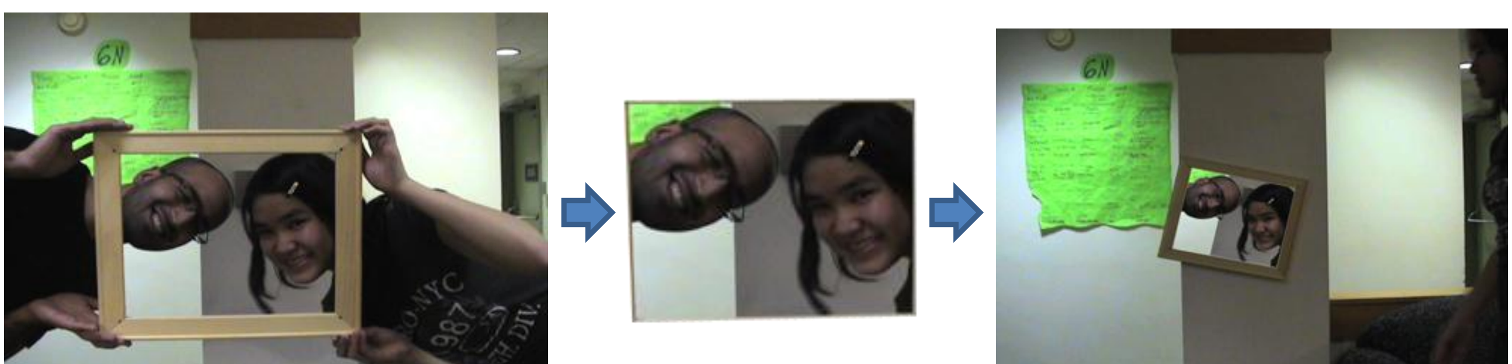
- Overlay information on a display
  - Tagging reality
  - Layar
  - Google goggles
  - T2 video (13:23)

Adding fake objects to real video

Approach

1. Recognize and/or track points that give you a coordinate frame
2. Apply homography (flat texture) or perspective projection (3D model) to put object into scene

Main challenge: dealing with lighting, shadows, occlusion



Information overlay

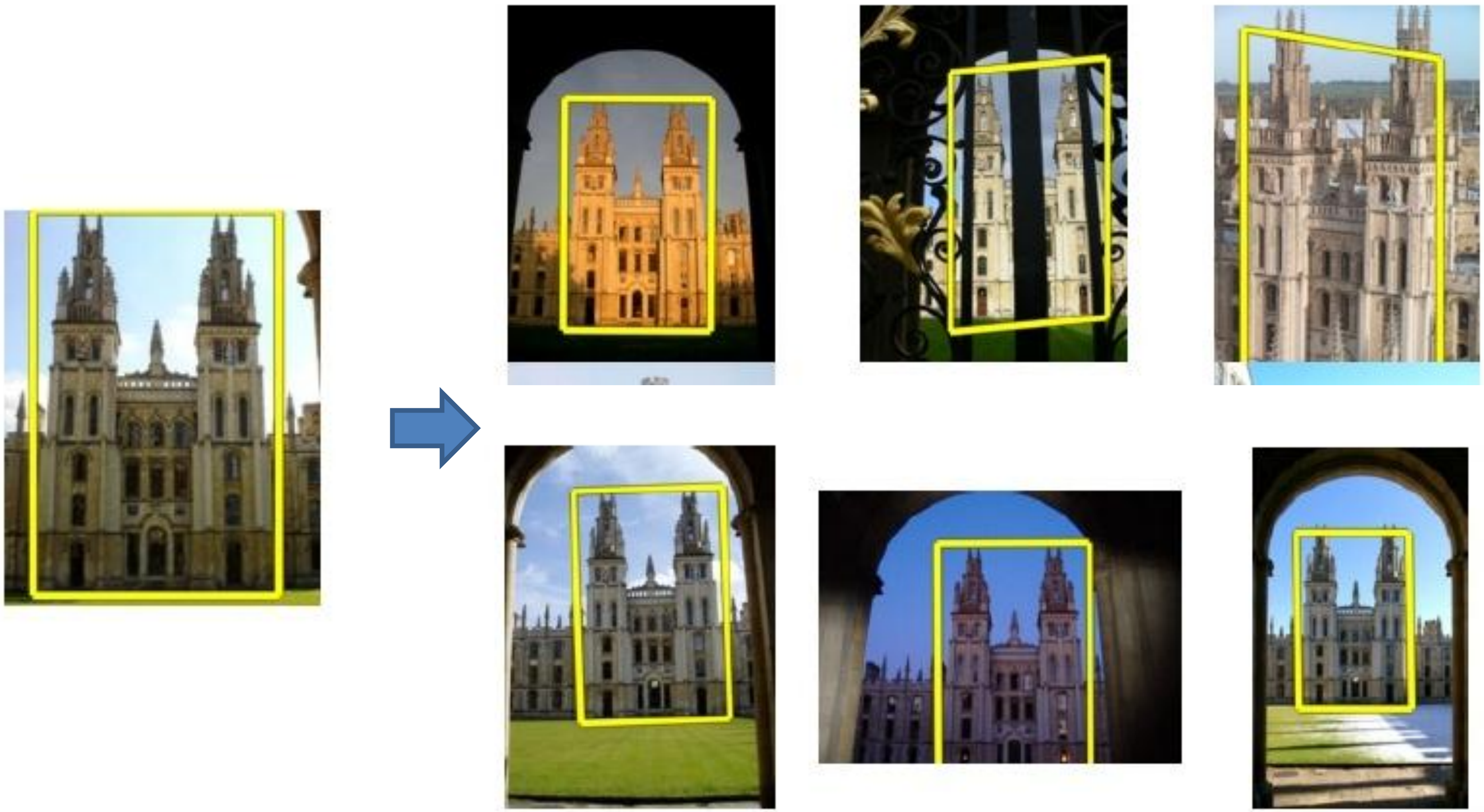
Approach

1. Recognize object that you’ve seen before
2. Retrieve info and overlay

Main challenge: how to match reliably and efficiently?

Today

How to quickly find images in a large database that match a given image region?



Let’s start with interest points

Query

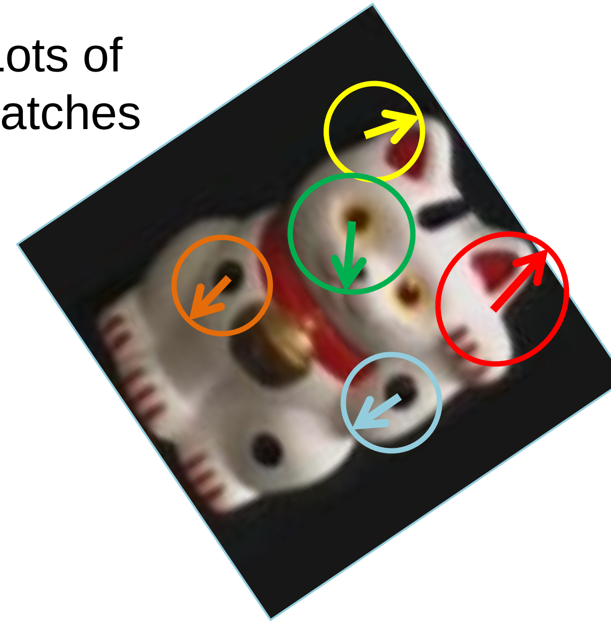
Database

Compute interest points (or keypoints) for every image in the database and the query

Simple idea

See how many keypoints are close to keypoints in each other image

Lots of Matches



Few or No Matches

But this will be really, really slow!

Key idea 1: “Visual Words”

- Cluster the keypoint descriptors

Key idea 1: “Visual Words”

K-means algorithm

1. Randomly select K centers

2. Assign each point to nearest center

3. Compute new center (mean) for each cluster

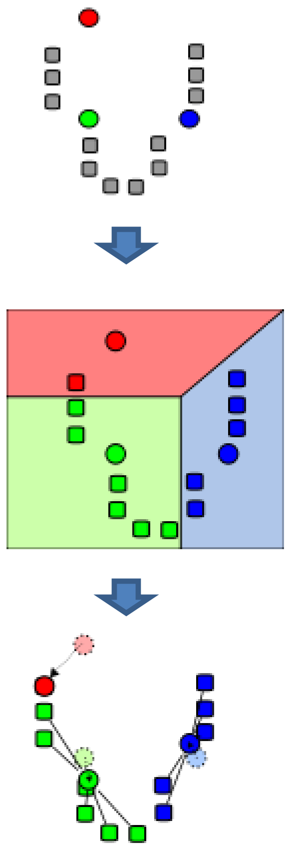


Illustration: http://en.wikipedia.org/wiki/K-means_clustering

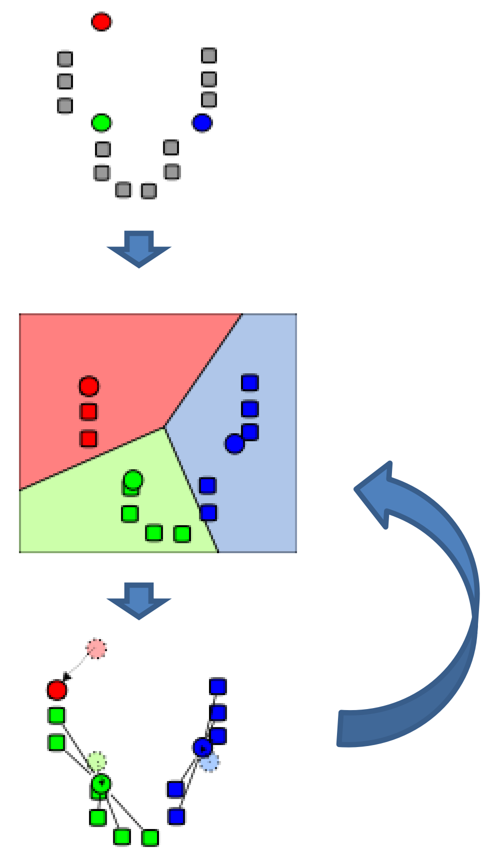
Key idea 1: "Visual Words"

K-means algorithm

1. Randomly select K centers

2. Assign each point to nearest center

3. Compute new center (mean) for each cluster



Back to 2

Illustration: http://en.wikipedia.org/wiki/K-means_clustering

Kmeans: Matlab code

```
function C = kmeans(X, K)

% Initialize cluster centers to be randomly sampled points
[N, d] = size(X);
rp = randperm(N);
C = X(rp(1:K), :);

lastAssignment = zeros(N, 1);
while true

  % Assign each point to nearest cluster center
  bestAssignment = zeros(N, 1);
  mindist = Inf*ones(N, 1);
  for k = 1:K
    for n = 1:N
      dist = sum((X(n, :)-C(k, :)).^2);
      if dist < mindist(n)
        mindist(n) = dist;
        bestAssignment(n) = k;
      end
    end
  end

  % break if assignment is unchanged
  if all(bestAssignment==lastAssignment), break; end;
  lastAssignment = bestAssignmnet;

  % Assign each cluster center to mean of points within it
  for k = 1:K
    C(k, :) = mean(X(bestAssignment==k, :));
  end
end
```

K-means Demo

http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/AppletKM.html

Key idea 1: “Visual Words”

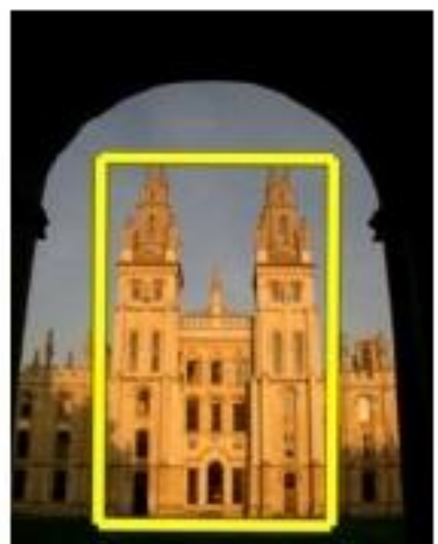
- Cluster the keypoint descriptors
- Assign each descriptor to a cluster number
  - What does this buy us?
  - Each descriptor was 128 dimensional floating point, now is 1 integer (easy to match!)
  - Is there a catch?
    - Need **a lot** of clusters (e.g., 1 million) if we want points in the same cluster to be very similar
    - Points that really are similar might end up in different clusters

Key idea 1: "Visual Words"

- Cluster the keypoint descriptors
- Assign each descriptor to a cluster number
- Represent an image region with a count of these "visual words"

Key idea 1: “Visual Words”

- Cluster the keypoint descriptors
- Assign each descriptor to a cluster number
- Represent an image region with a count of these “visual words”
- An image is a good match if it has a lot of the same visual words as the query region



Naïve matching is still too slow

- Imagine matching 1,000,000 images, each with 1,000 keypoints

Key Idea 2: Inverse document file

- Like a book index: keep a list of all the words (keypoints) and all the pages (images) that contain them.

- Rank database images based on tf-idf measure.

tf-idf: Term Frequency – Inverse Document Frequency

times word appears in document

$$t_i = \frac{n_{id}}{n_d} \log \frac{N}{n_i}$$

documents

documents that contain the word

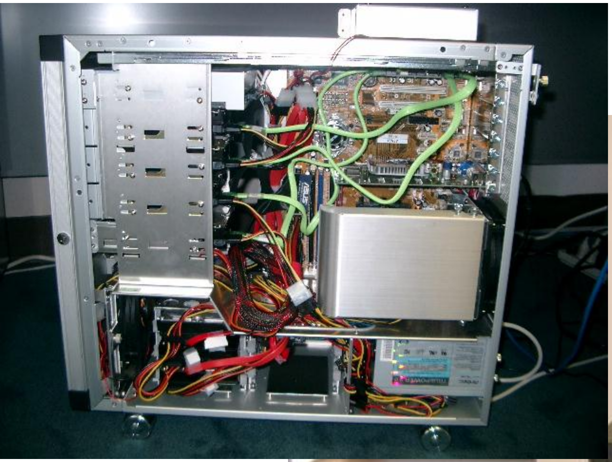
words in document

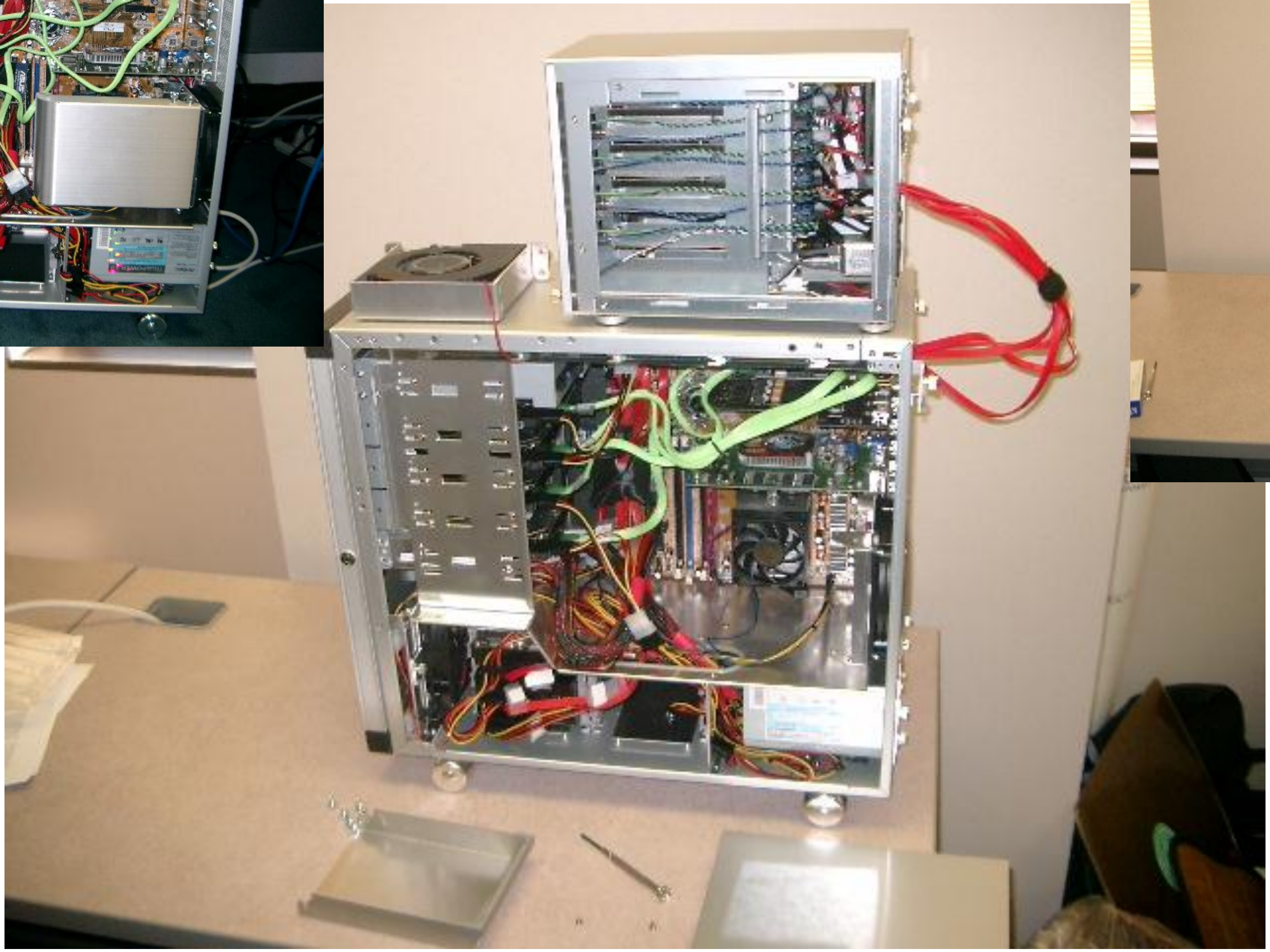
Fast visual search

“Video Google”, Sivic and Zisserman, ICCV 2003

“Scalable Recognition with a Vocabulary Tree”, Nister and Stewenius, CVPR 2006.

110,000,000 Images in 5.8 Seconds

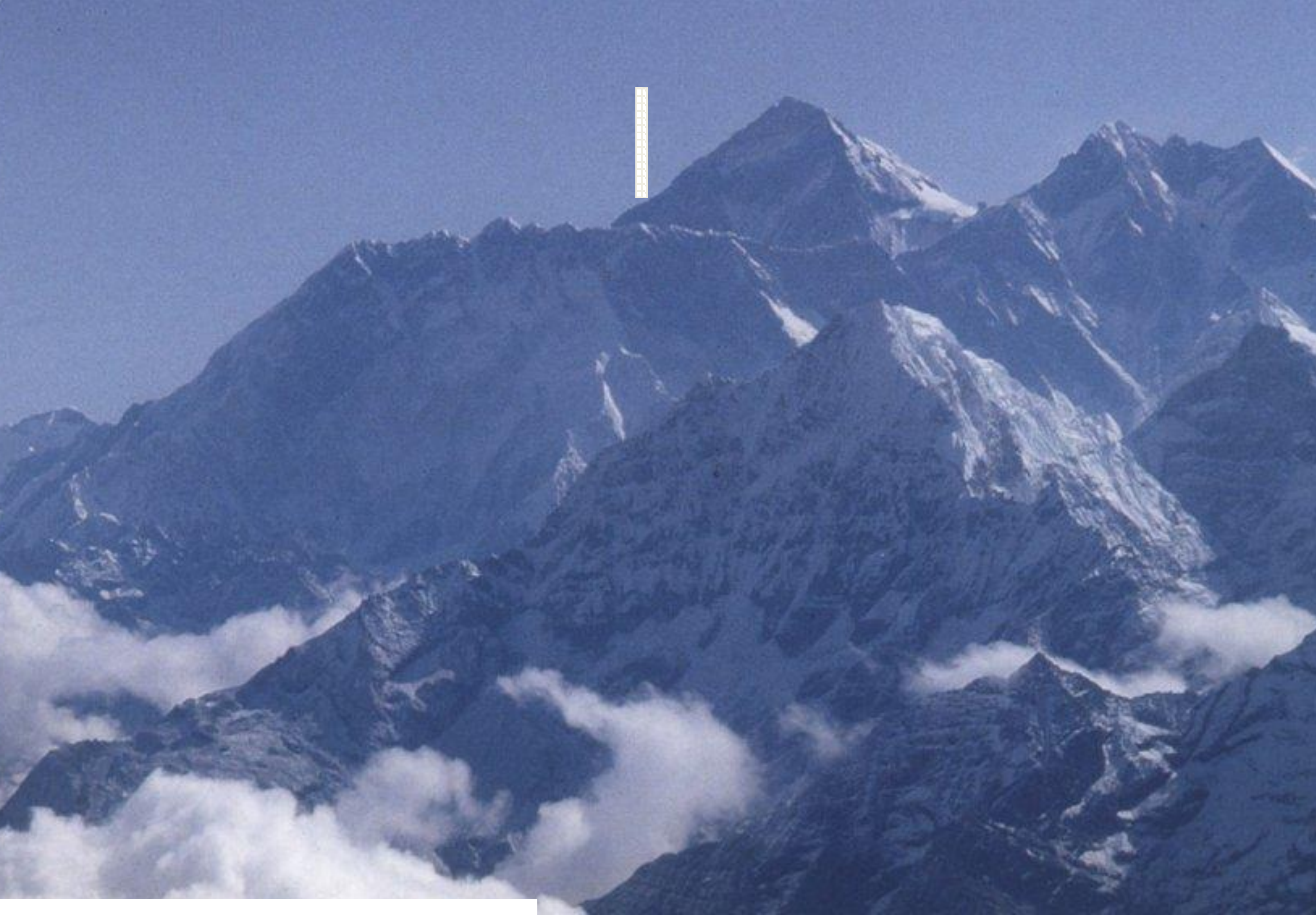




Slide Credit: Nister

Slide Credit: Nister

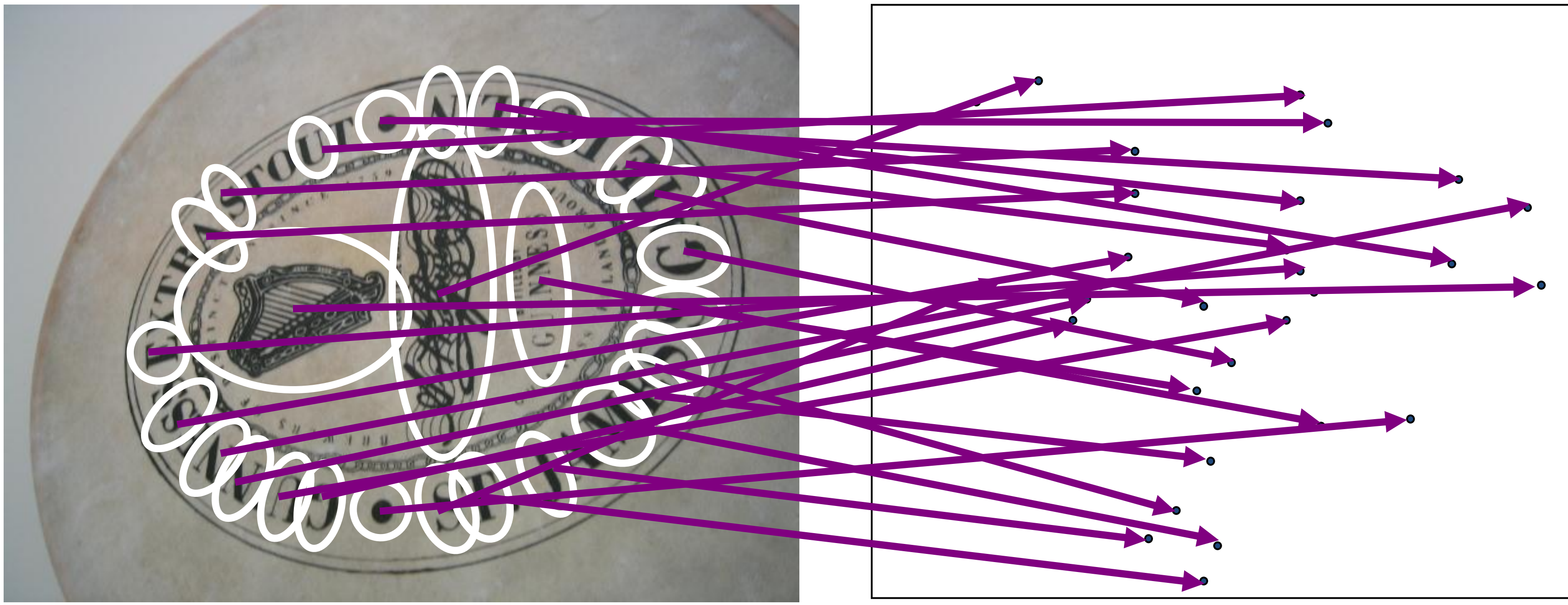
Slide Credit: Nister

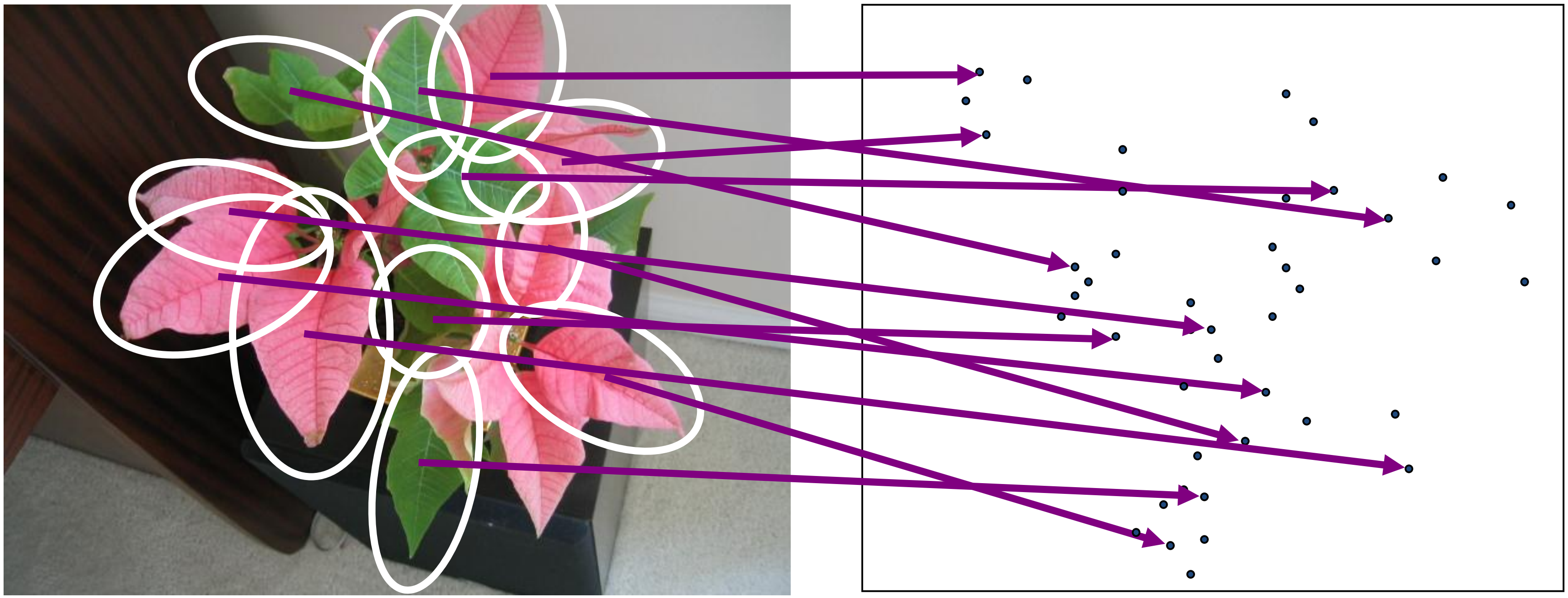


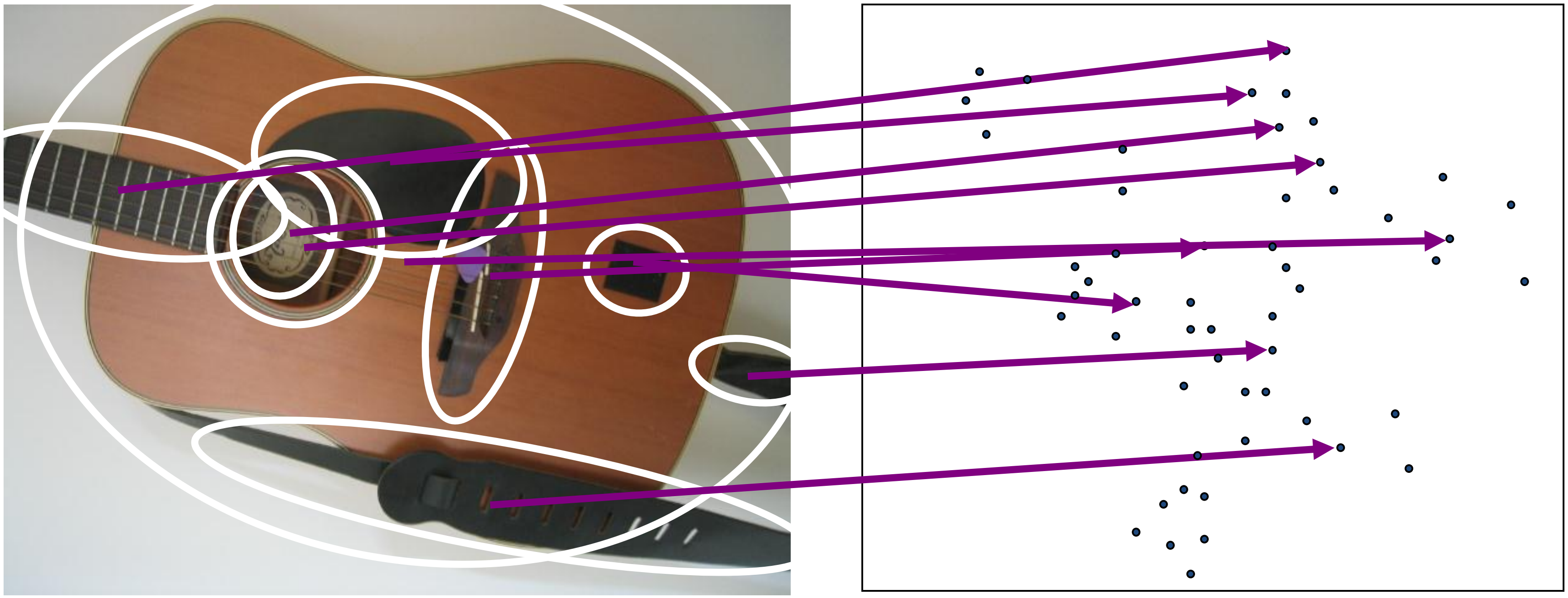
Slide Credit: Nister

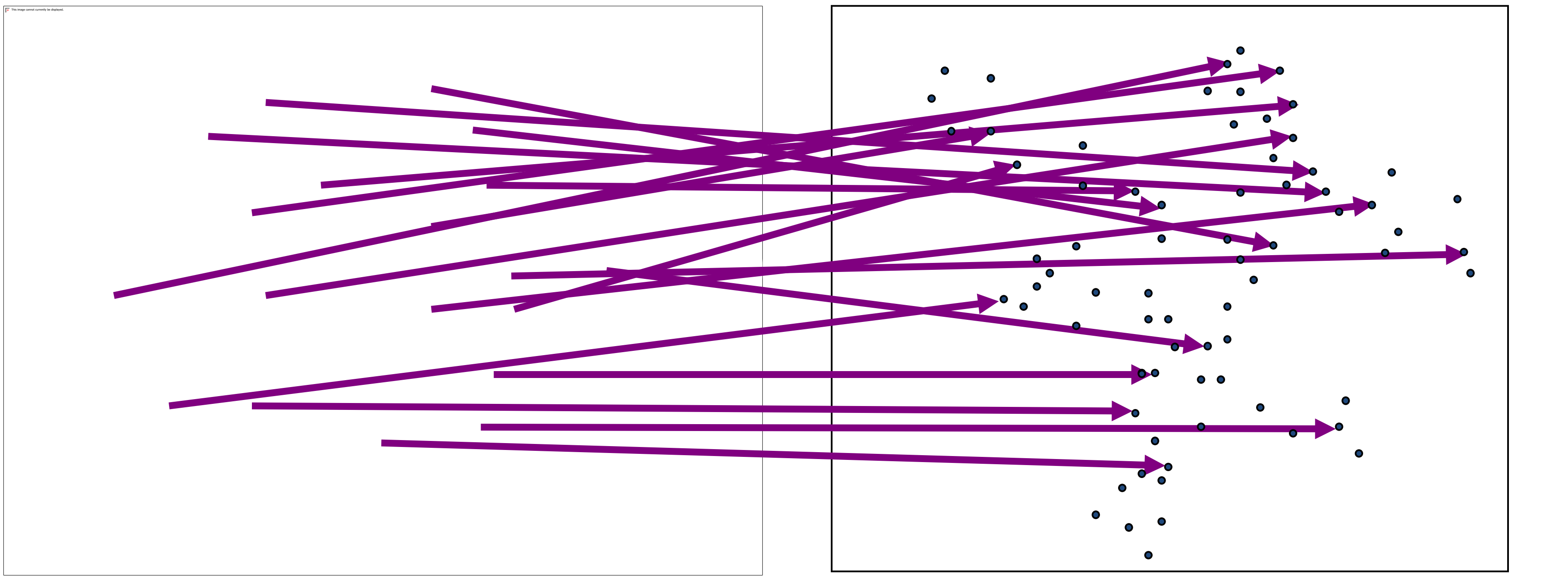
Recognition with K-tree

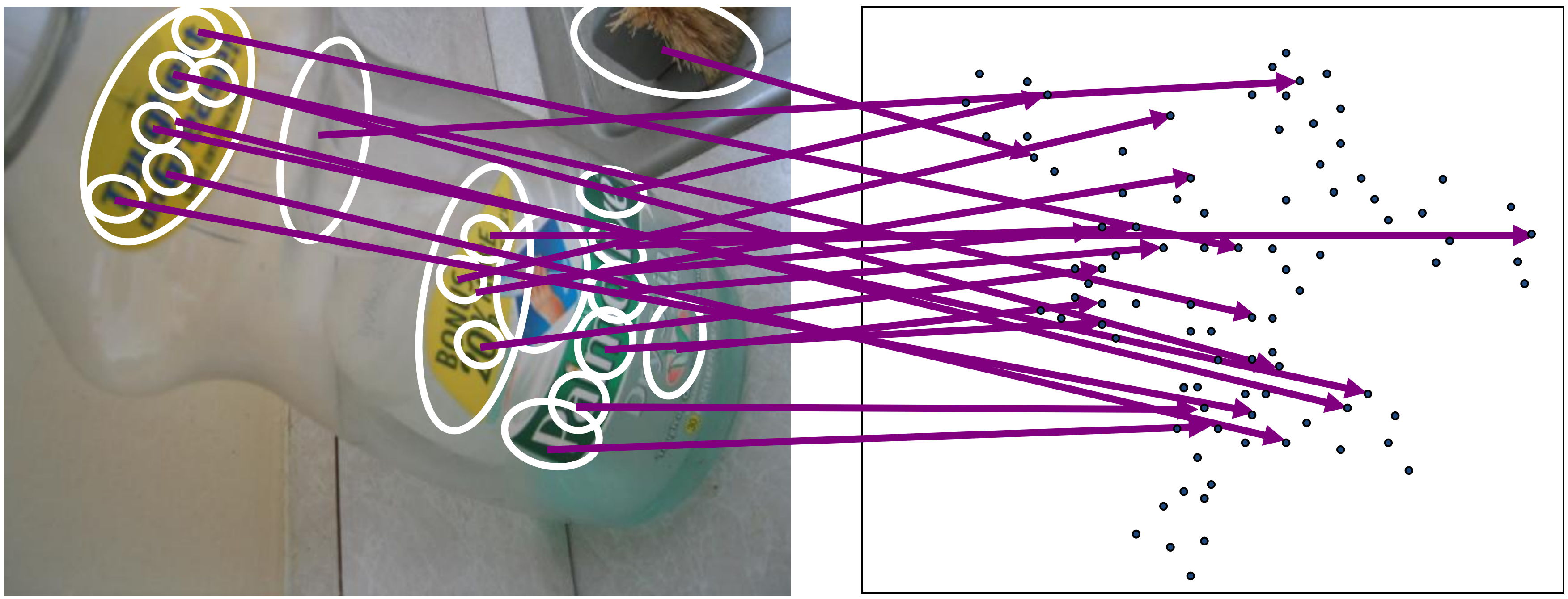
Following slides by David Nister (CVPR 2006)

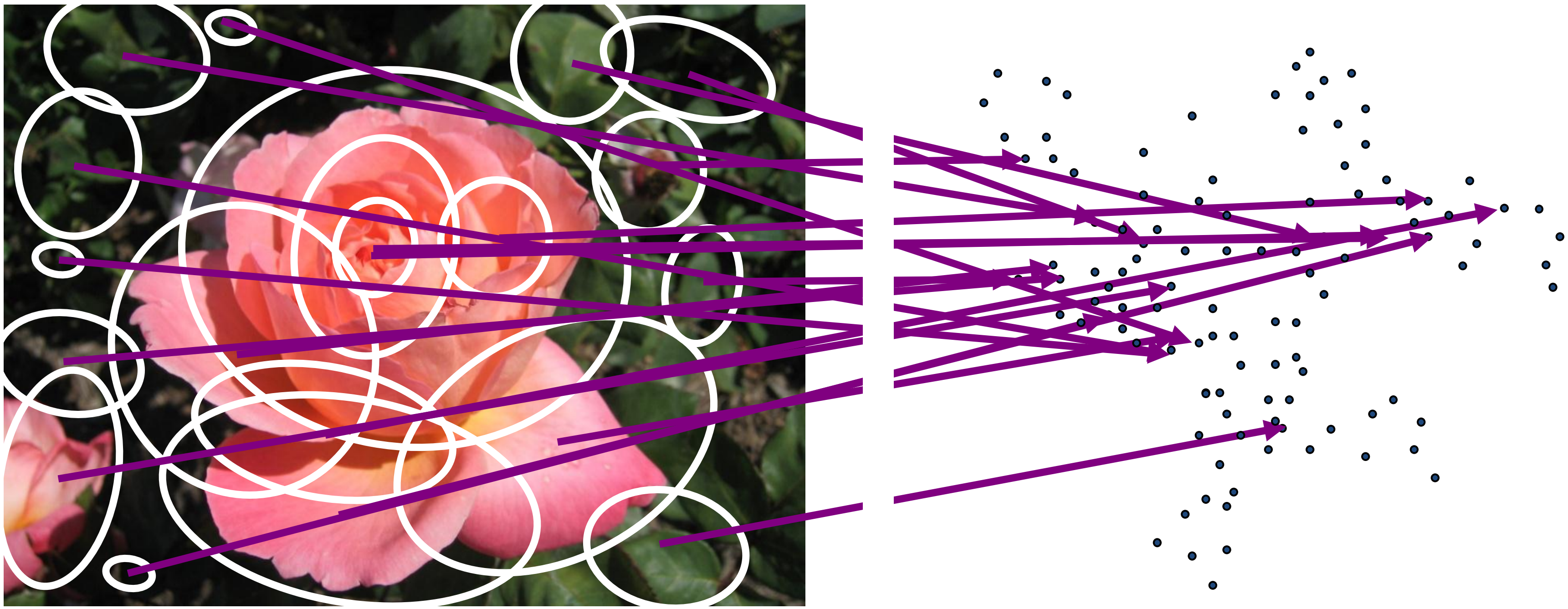

EXTRA STOUT

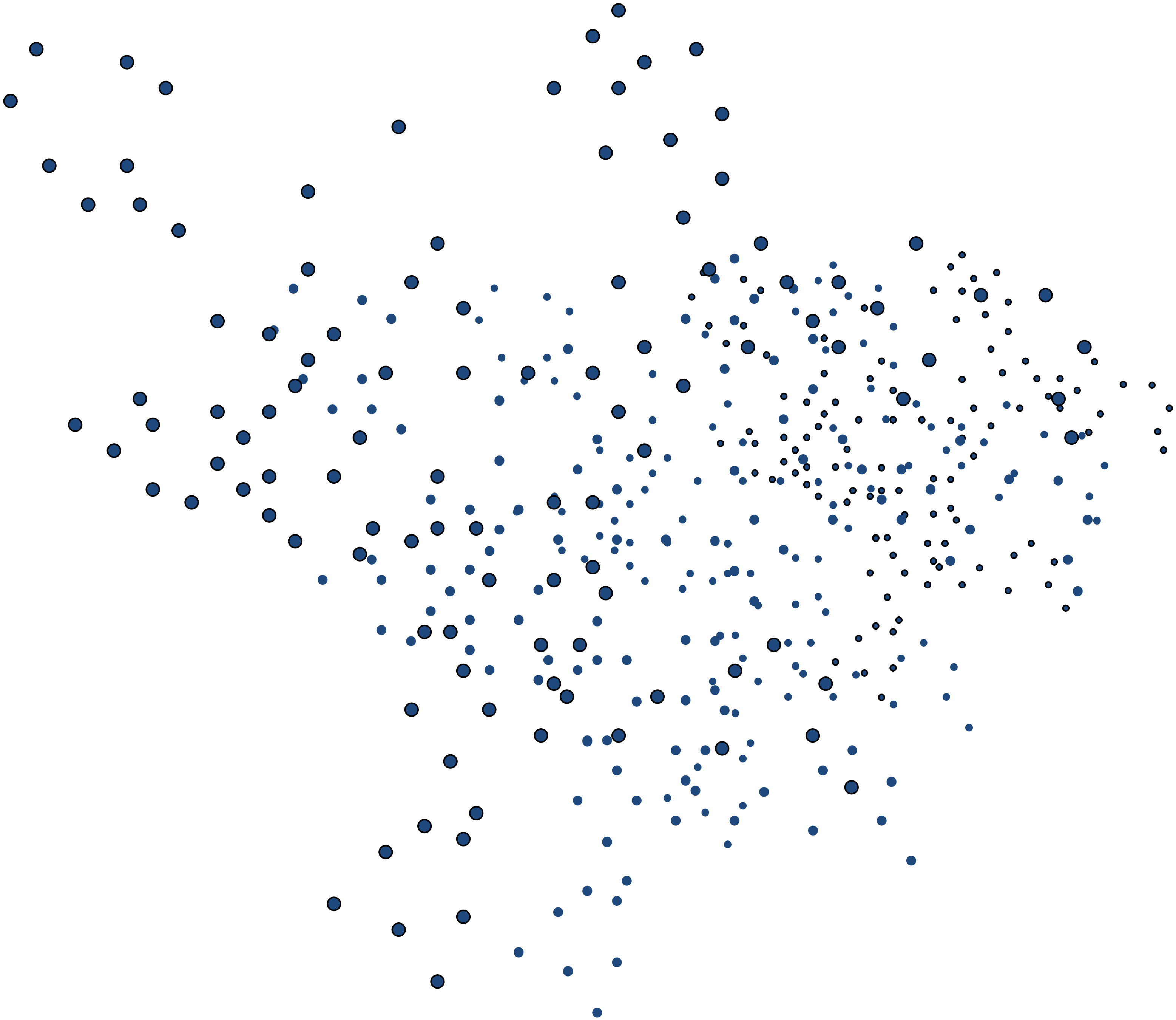


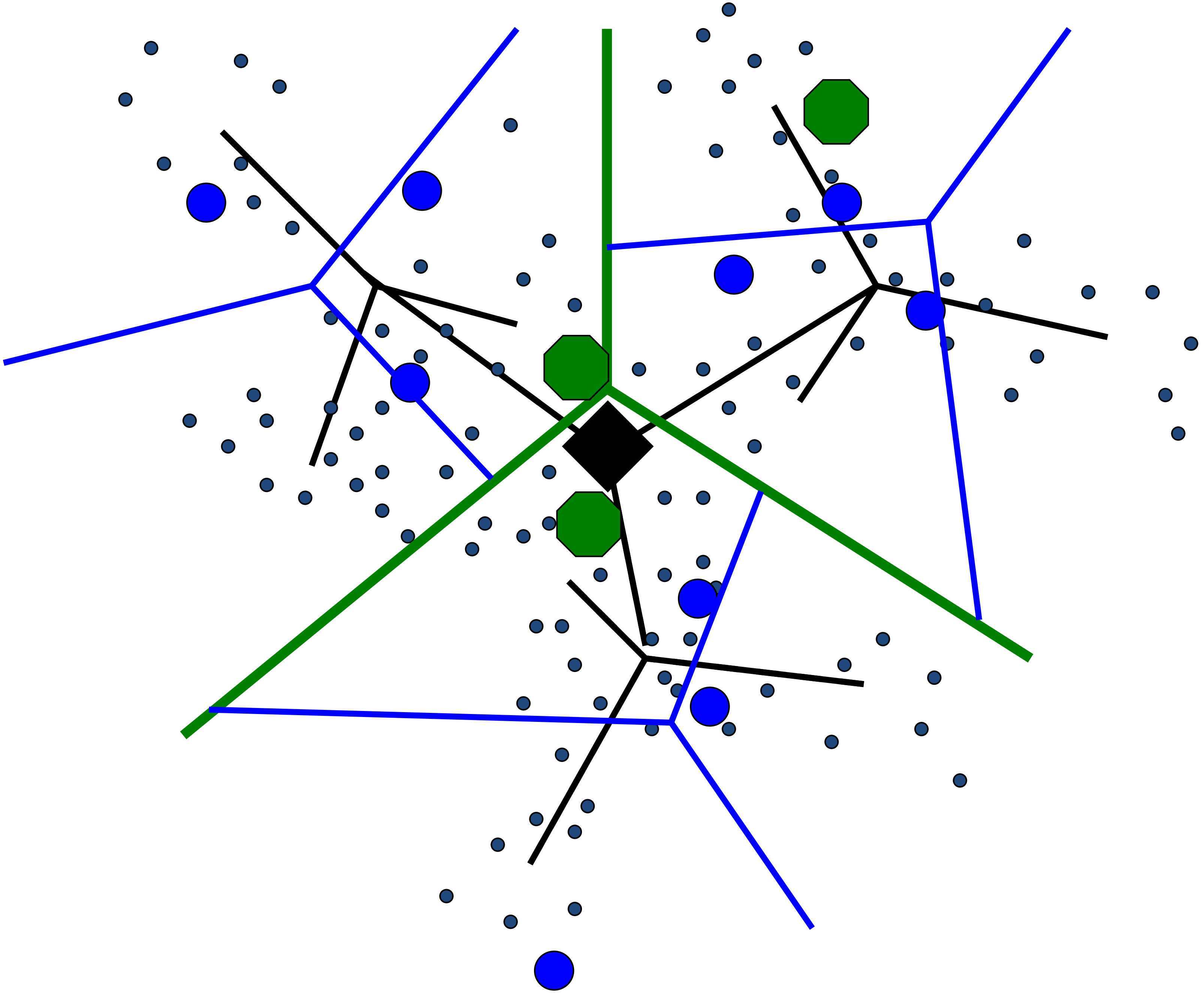


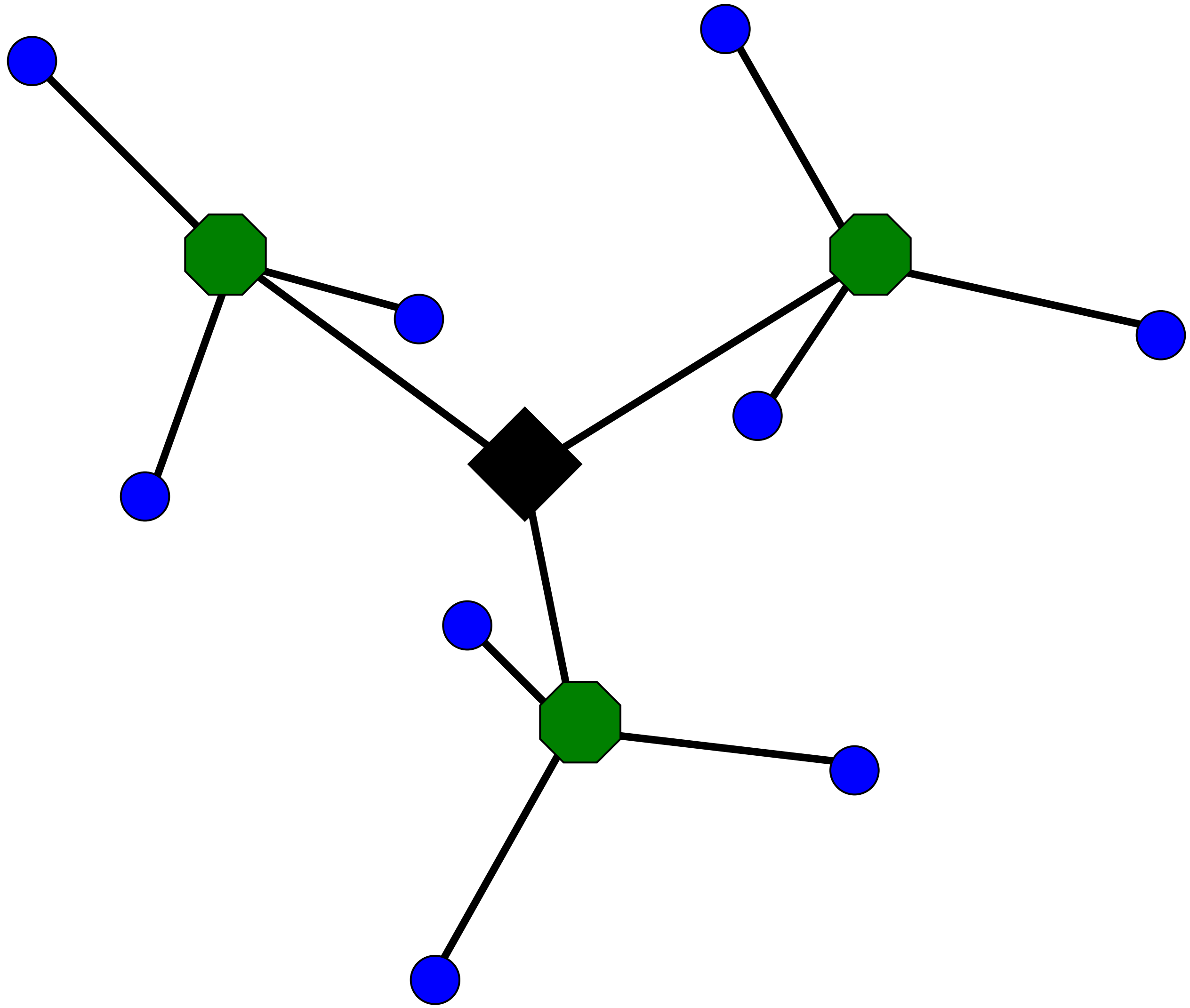


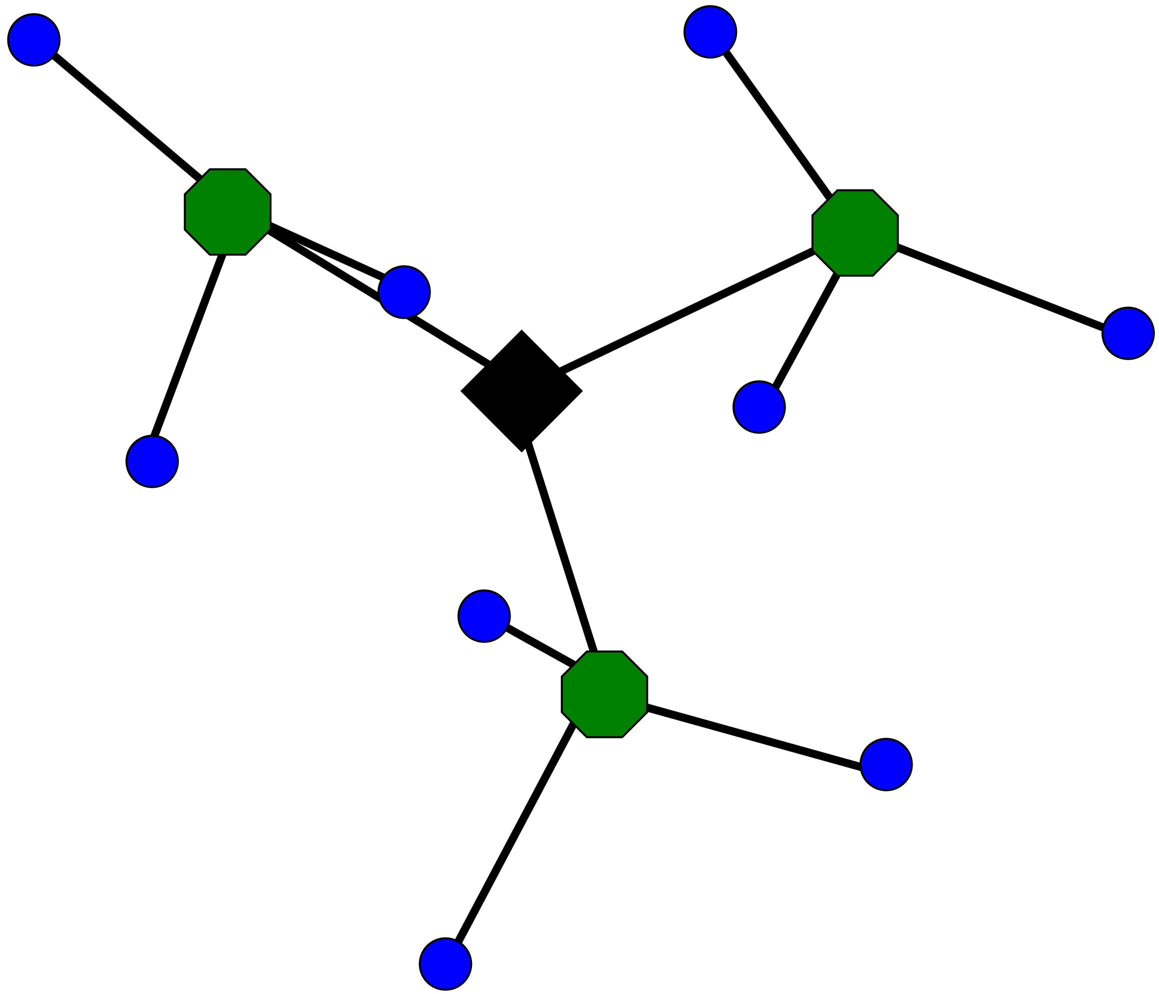


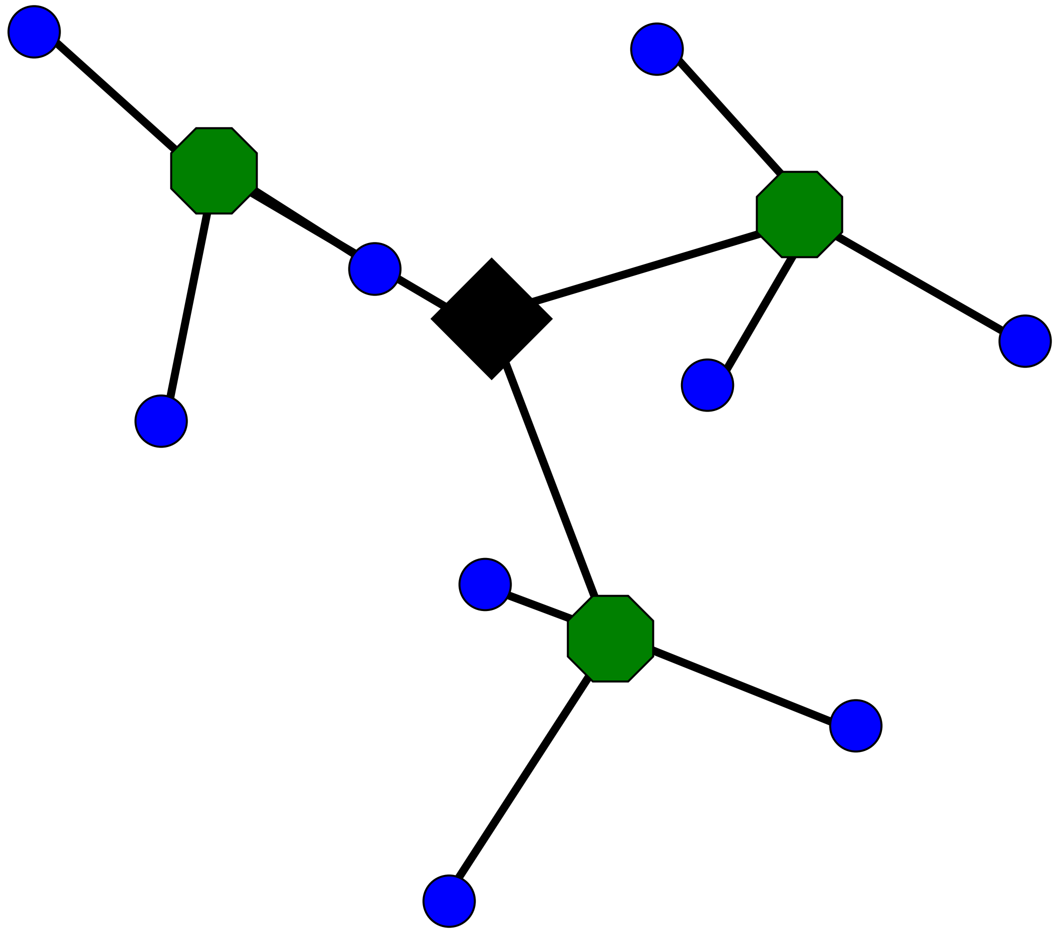


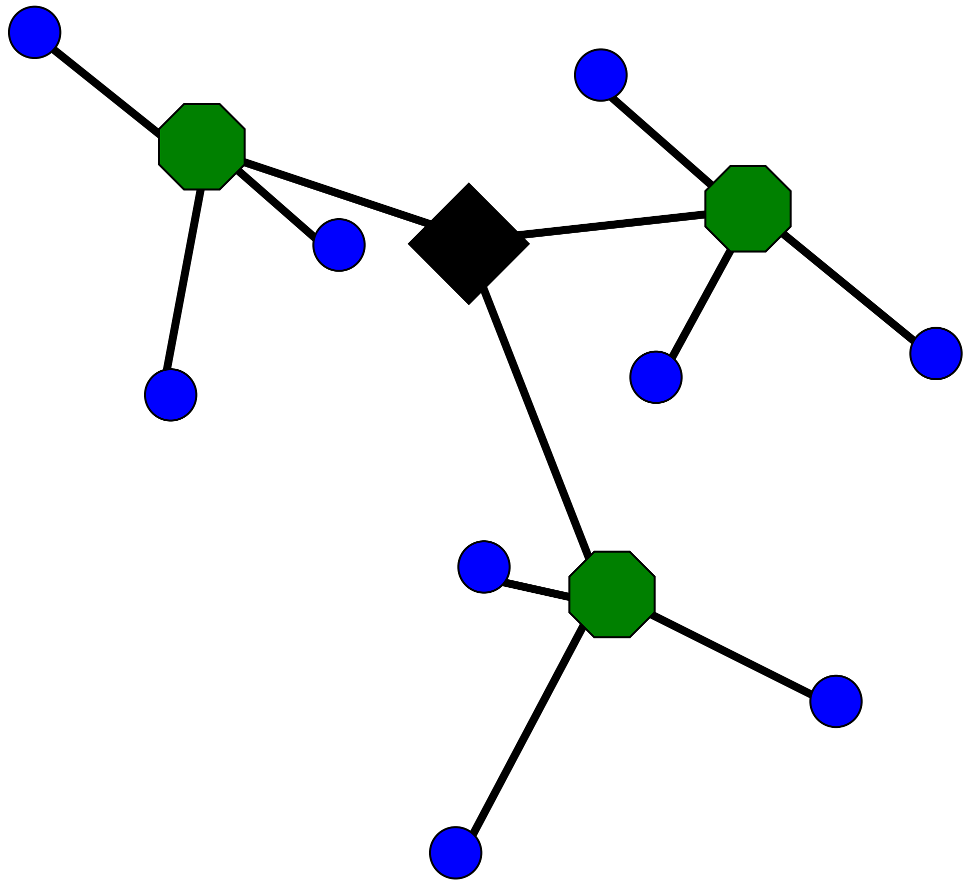


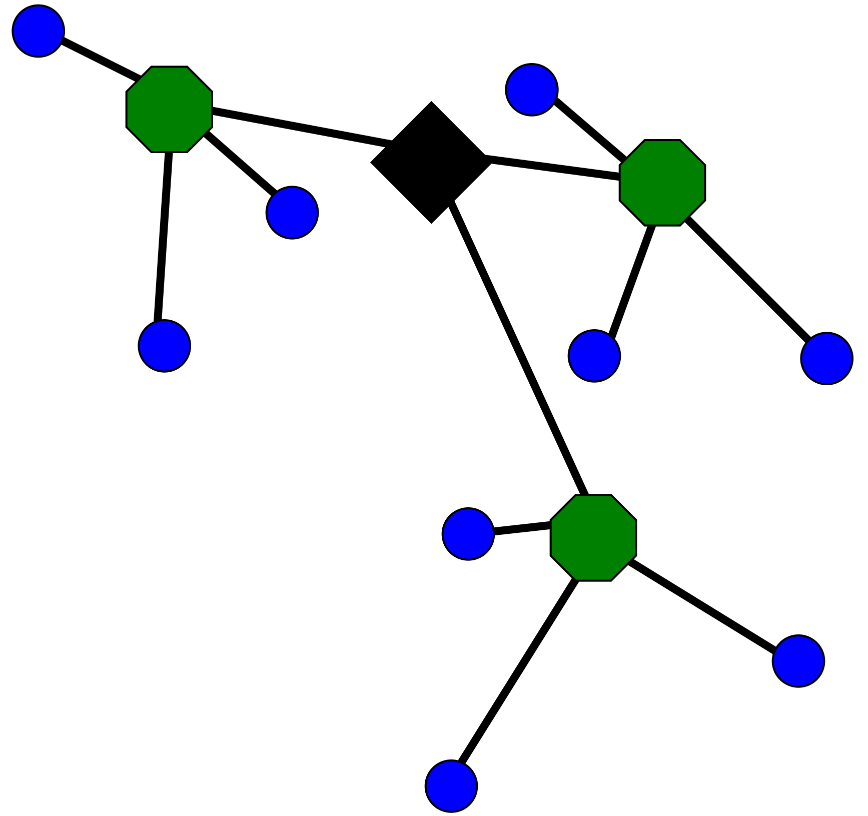


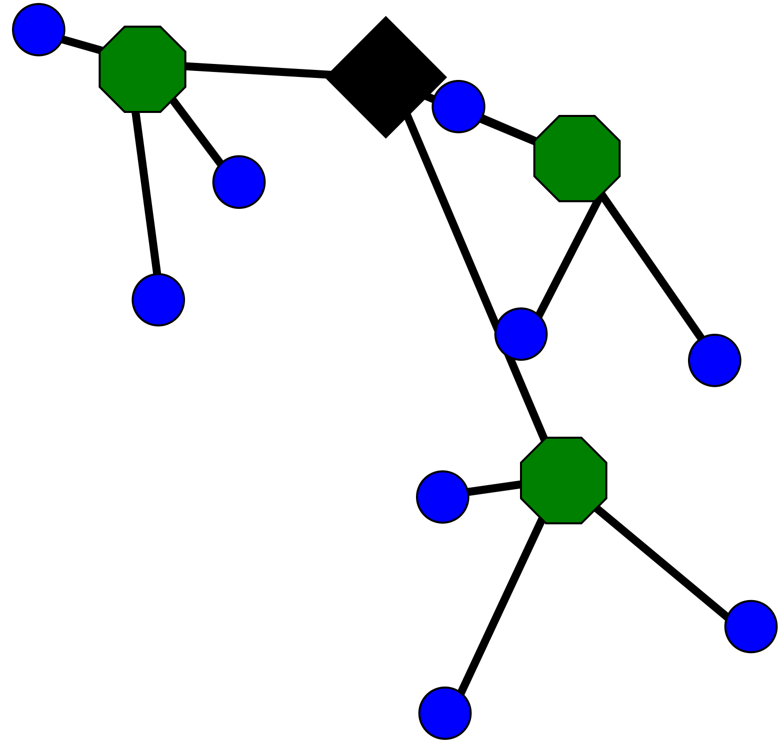


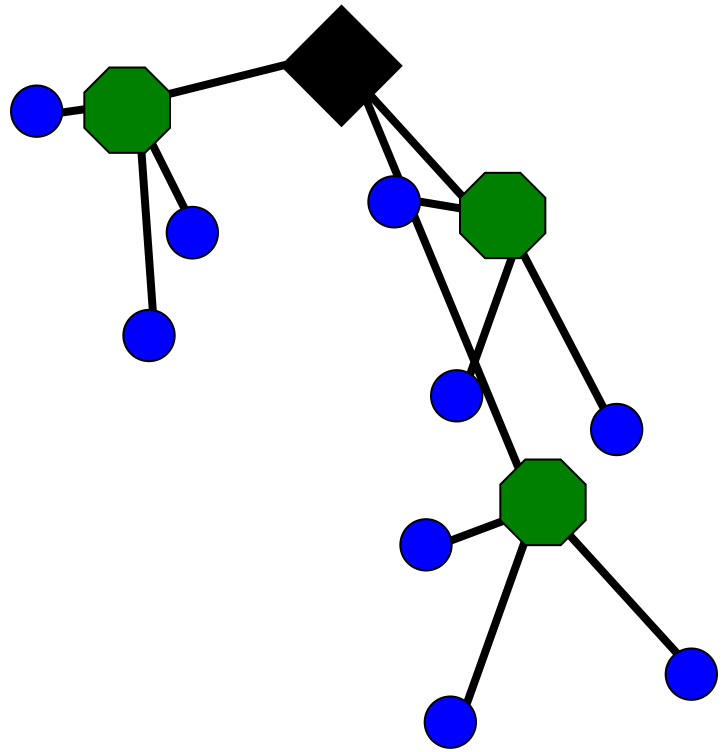


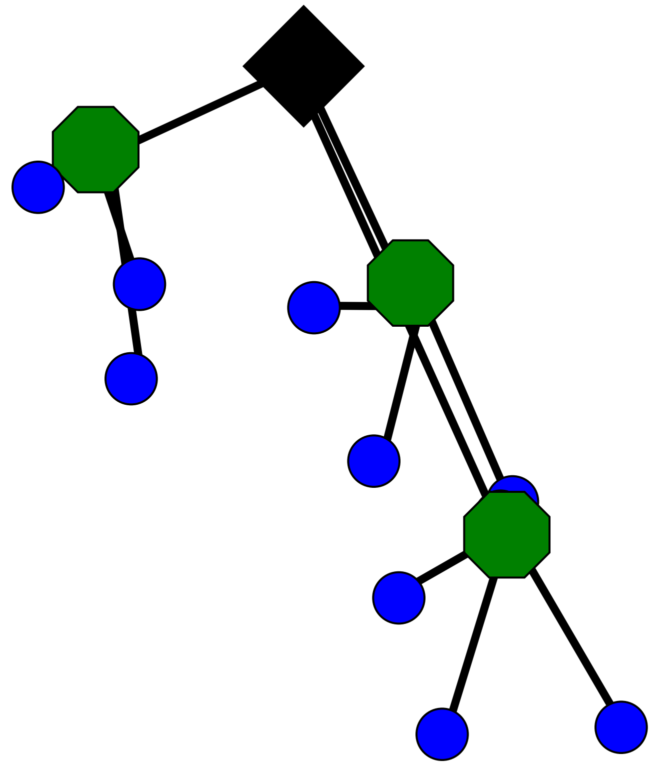


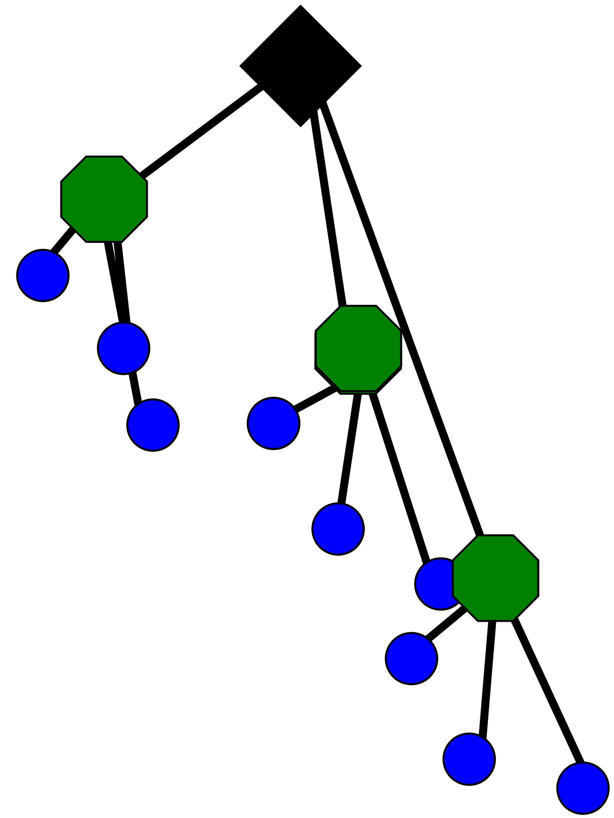


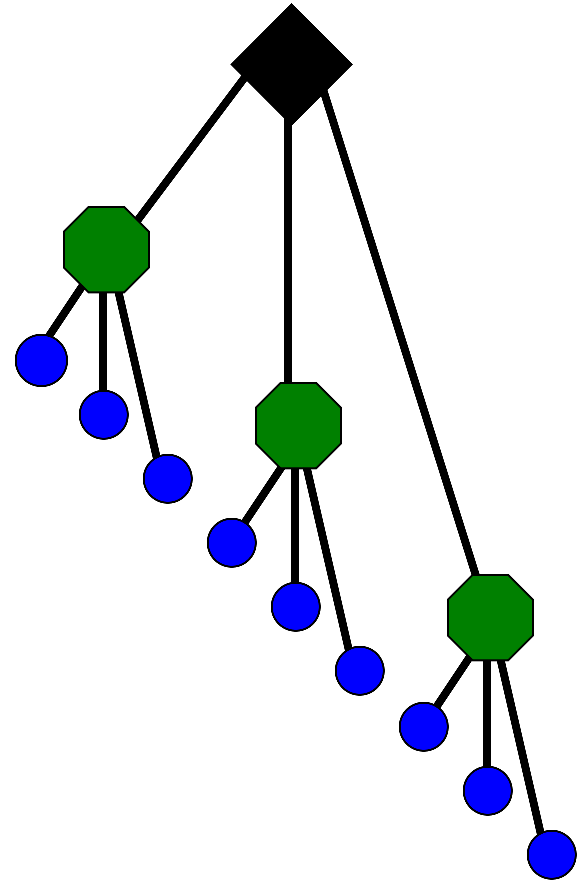


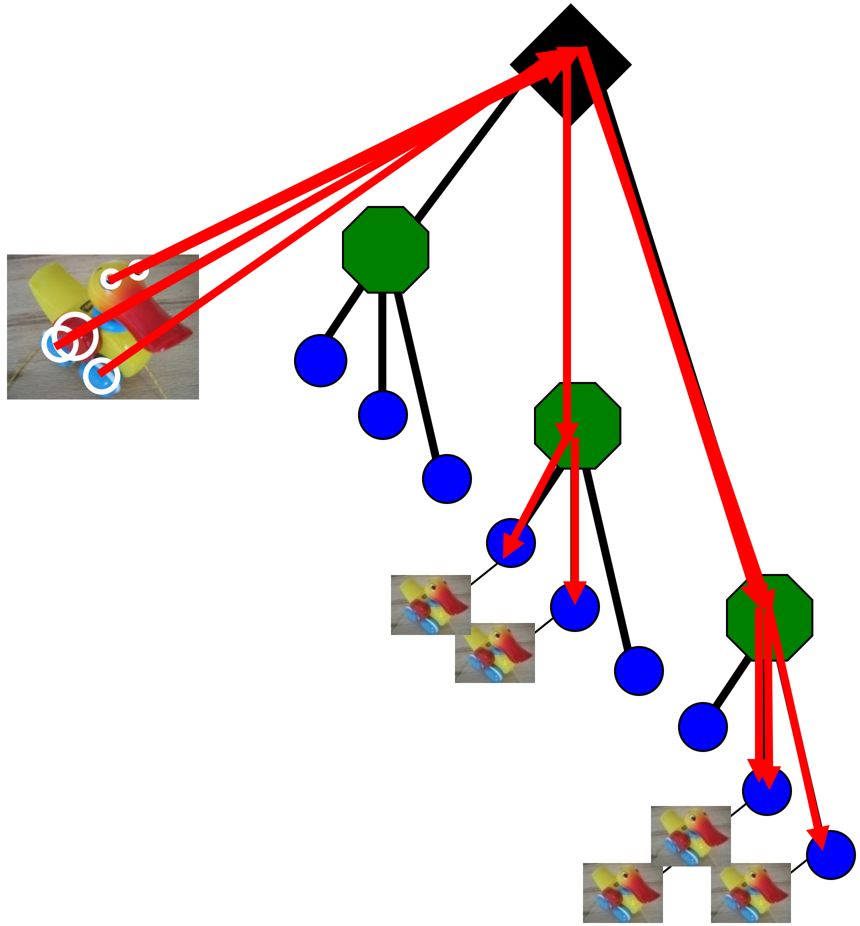


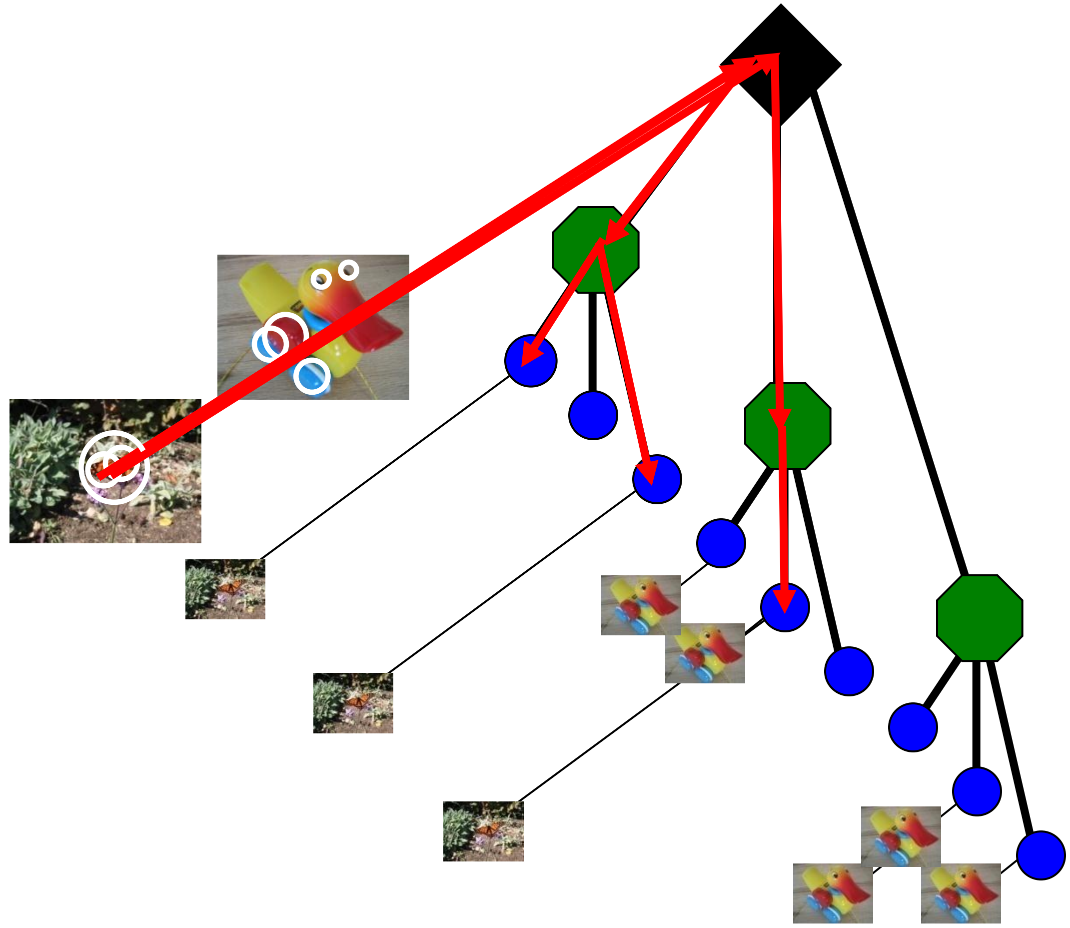


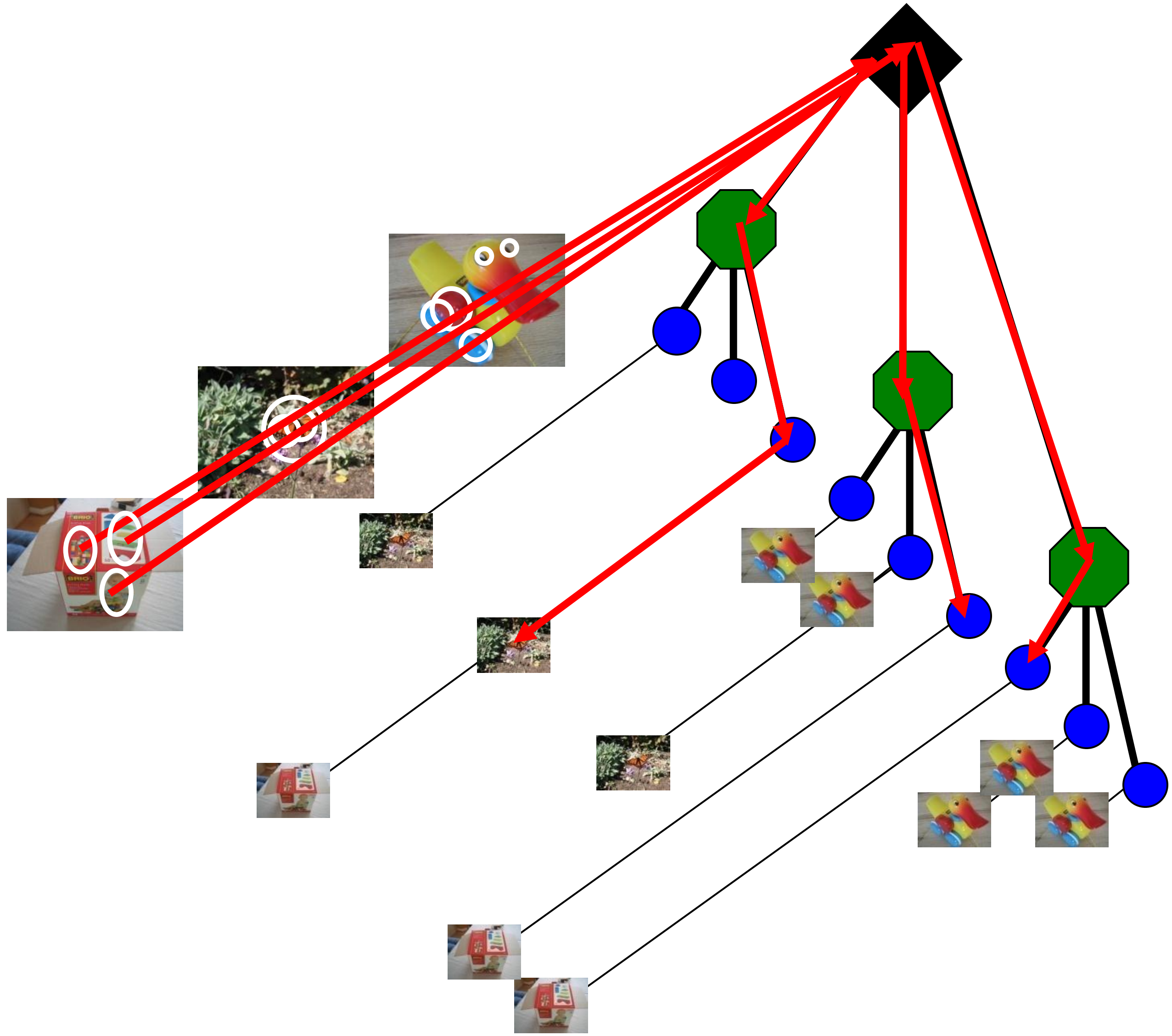


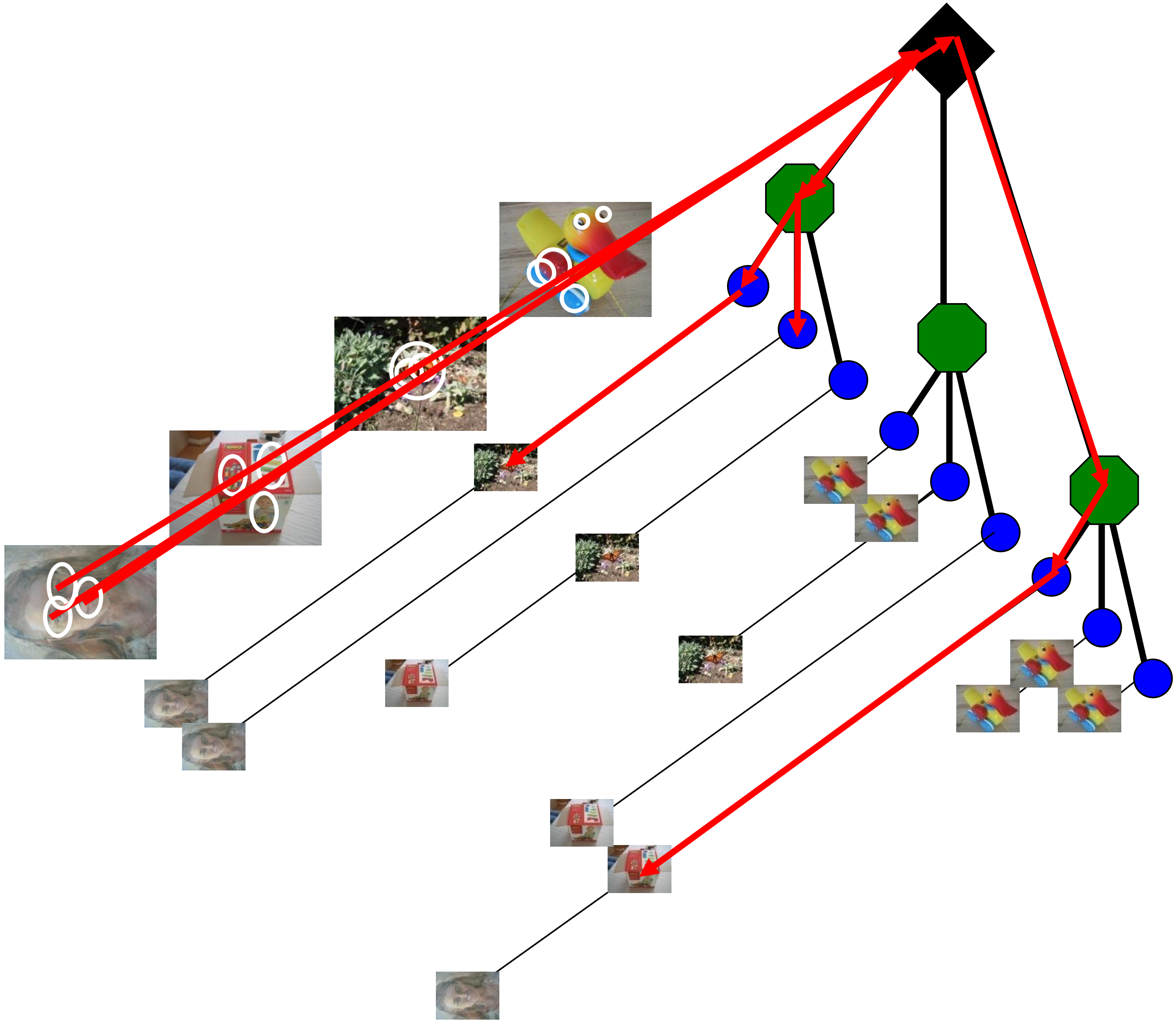


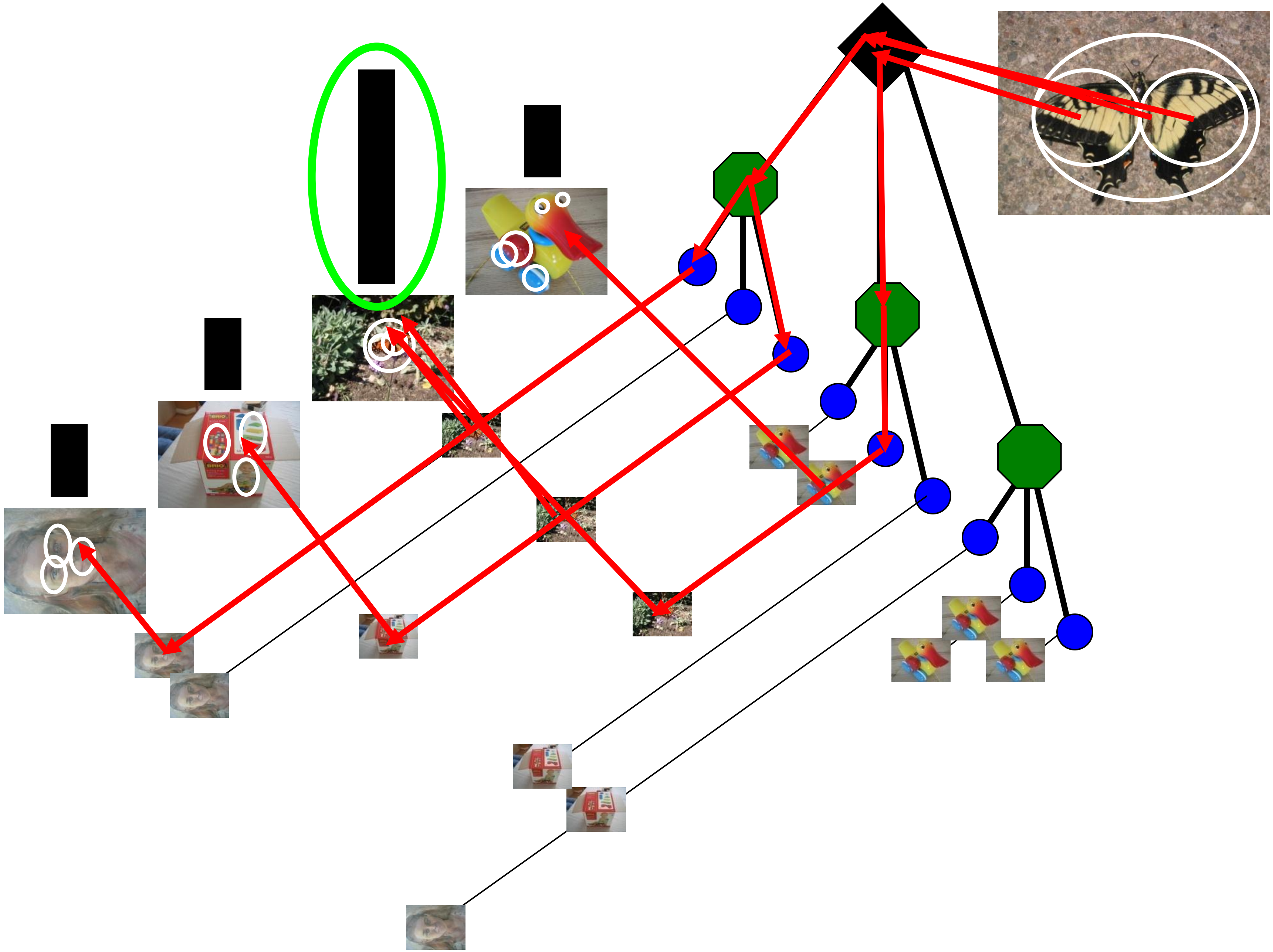




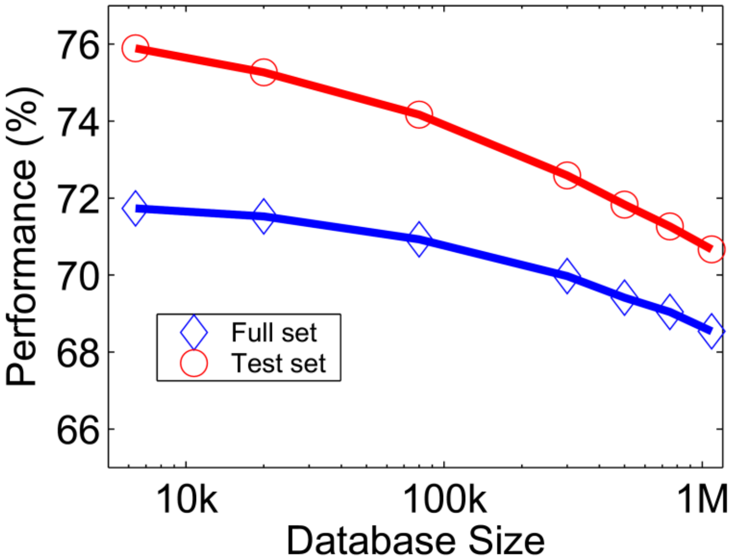








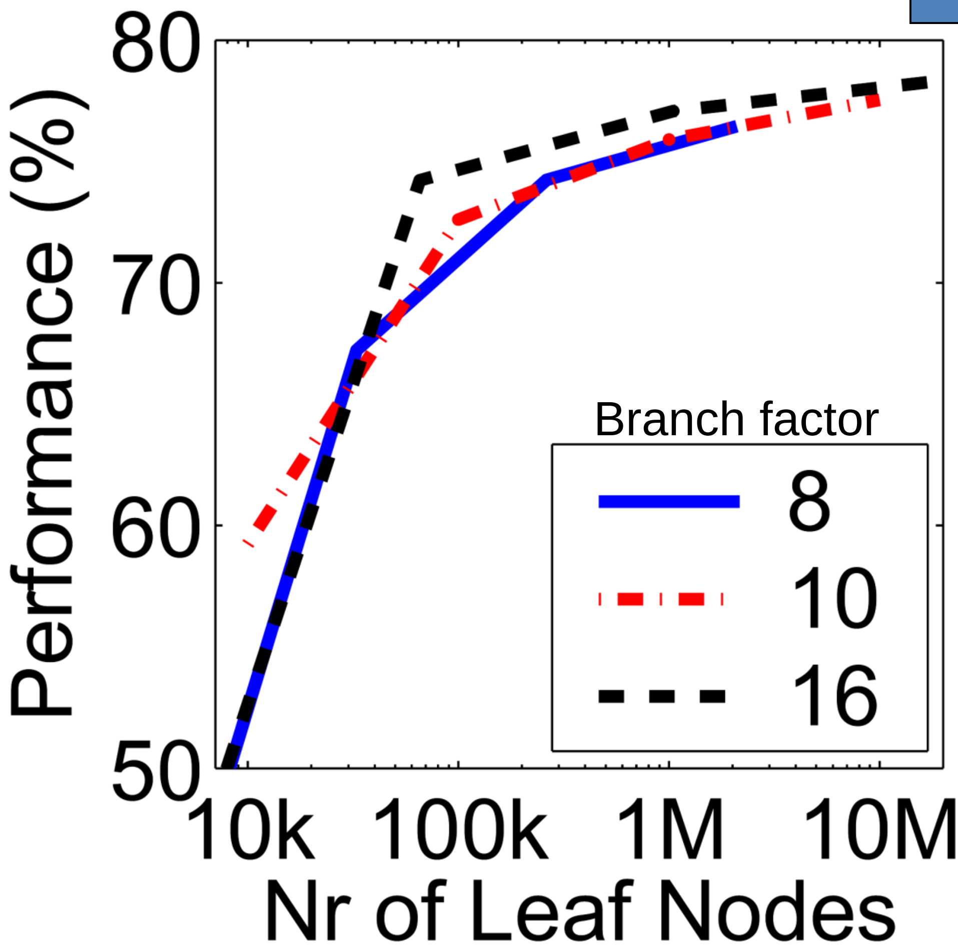
Performance



More words is better

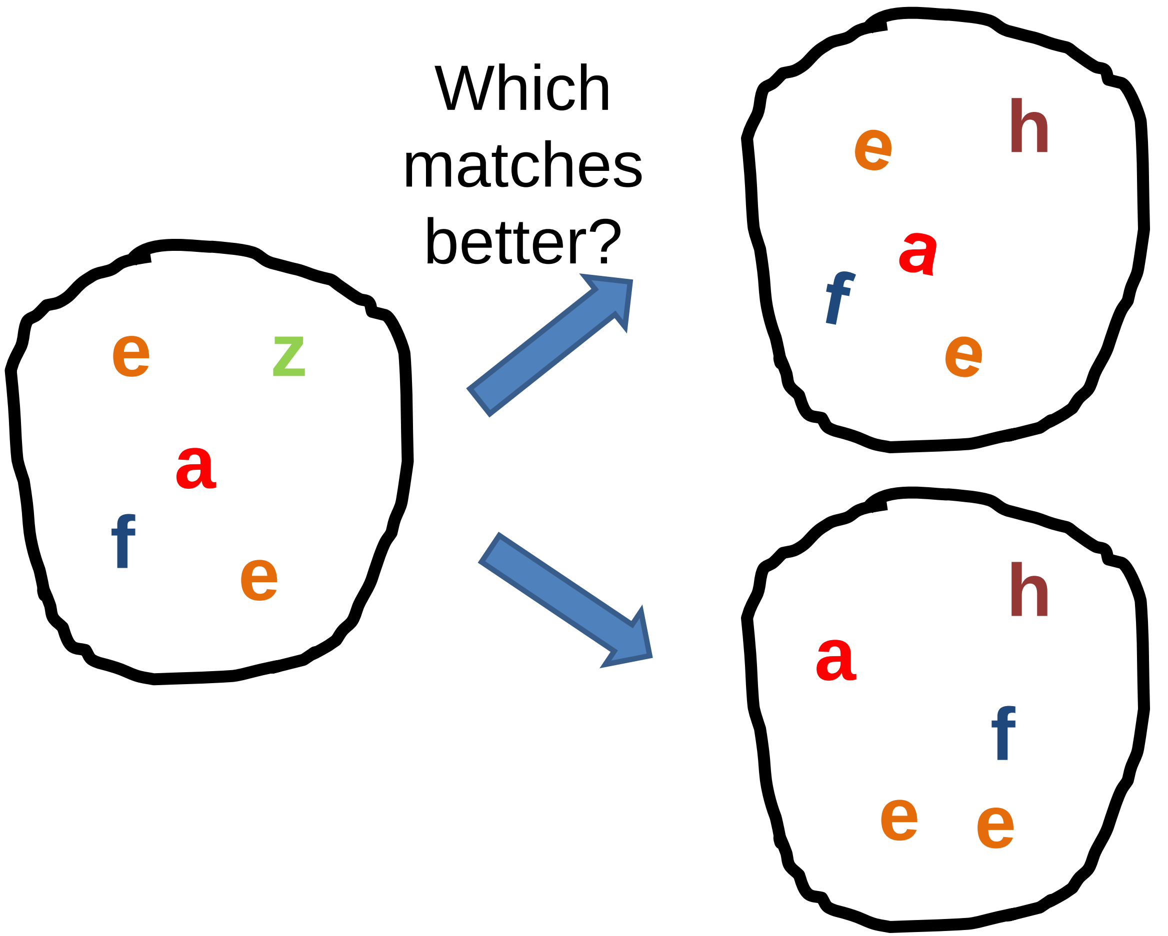
Improves Retrieval

Improves Speed



Can we be more accurate?

So far, we treat each image as containing a "bag of words", with no spatial information



Can we be more accurate?

So far, we treat each image as containing a "bag of words", with no spatial information

Real objects have consistent geometry

Final key idea: geometric verification

- Goal: Given a set of possible keypoint matches, figure out which ones are geometrically consistent

How can we do this?

Final key idea: geometric verification

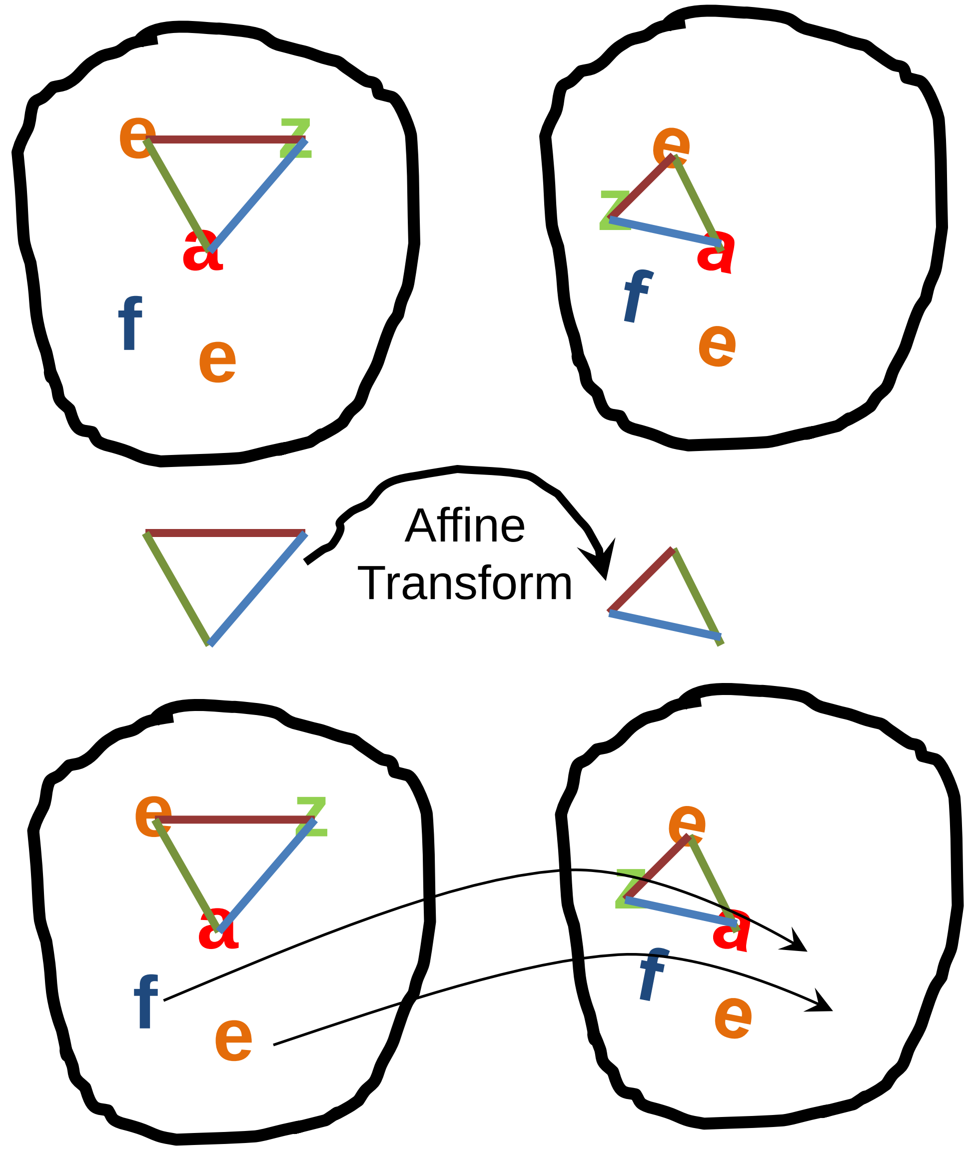
RANSAC for affine transform

Repeat N times:

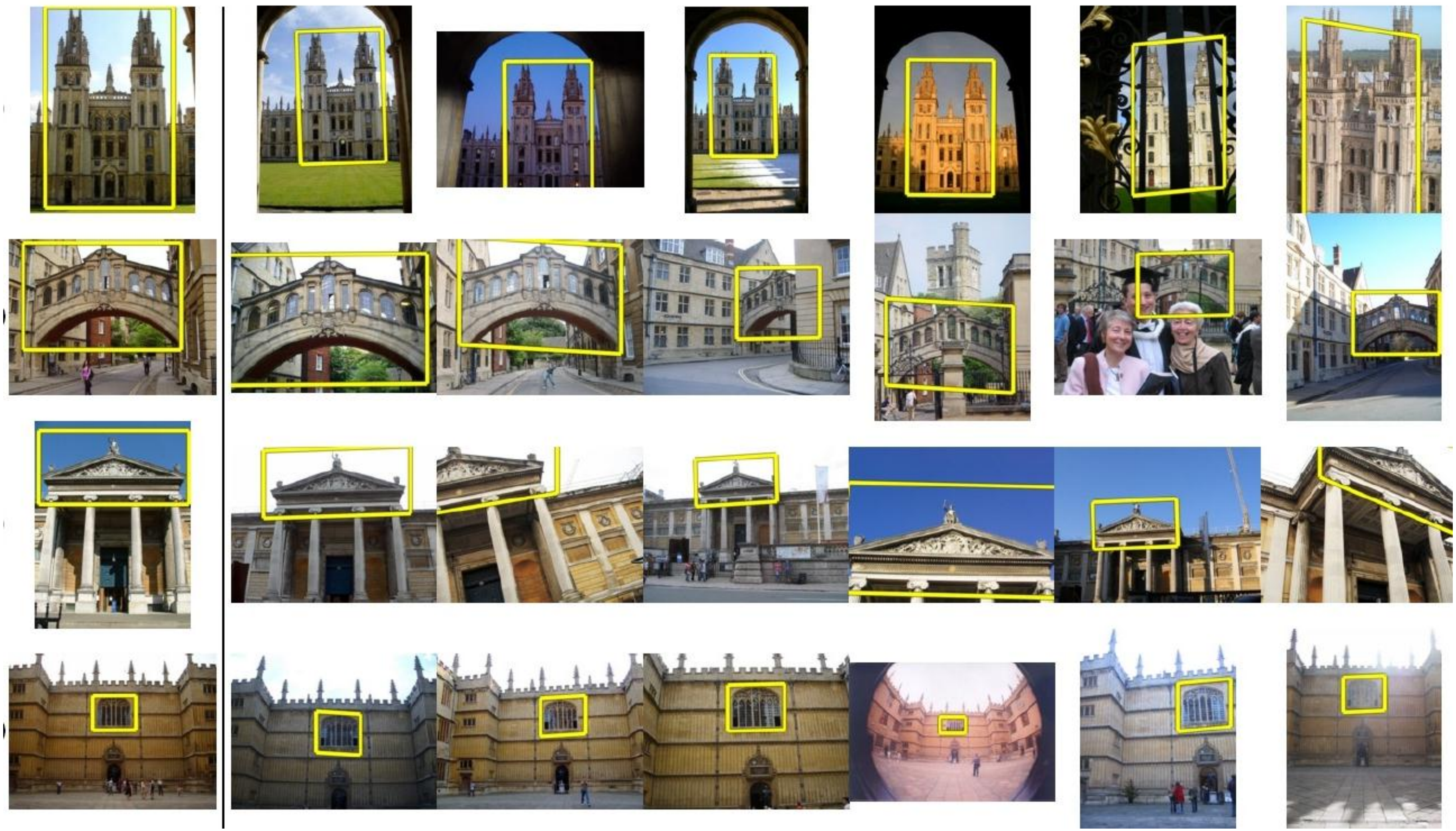
Randomly choose 3 matching pairs

Estimate transformation

Predict remaining points and count “inliers”



Application: Large-Scale Retrieval



Query

Results on 5K (demo available for 100K)

[Philbin CVPR'07]

Application: Image Auto-Annotation

Left: Wikipedia image
Right: closest match from Flickr

[Quack CIVR'08]

Example Applications

Mobile tourist guide

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

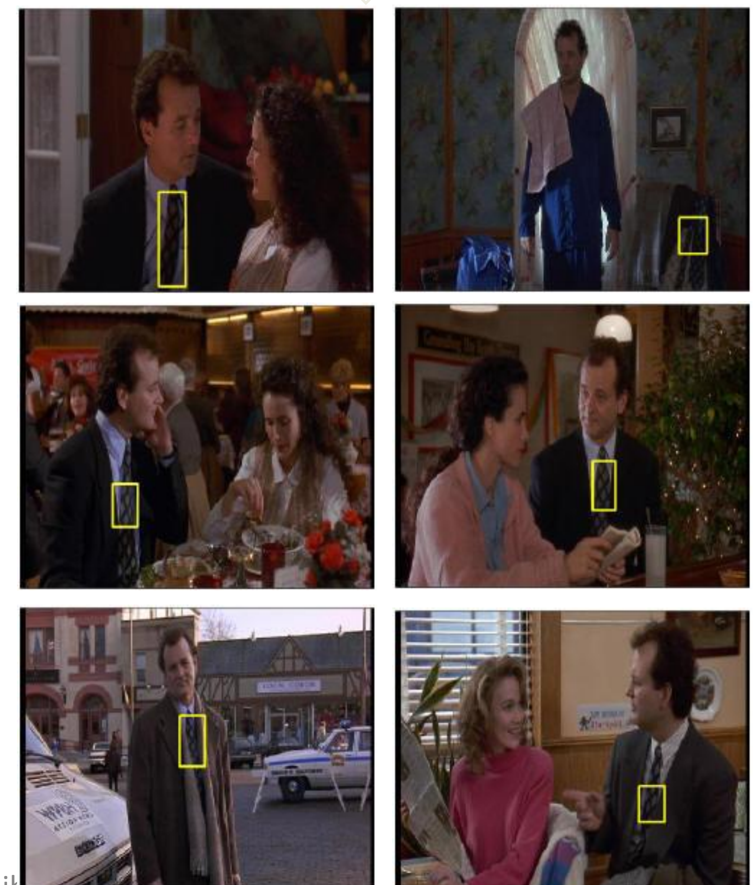
Video Google System

1. Collect all words within query region
2. Inverted file index to find relevant frames
3. Compare word counts
4. Spatial verification

Sivic & Zisserman, ICCV 2003

- Demo online at : http://www.robots.ox.ac.uk/~vgg/research/vgoogle/index.html

Query region



Retrieved frames

Summary: Uses of Interest Points

- Interest points can be detected reliably in different images at the same 3D location
  - DOG interest points are localized in x, y, scale
- SIFT is robust to rotation and small deformation
- Interest points provide correspondence
  - For image stitching
  - For defining coordinate frames for object insertion
  - For object recognition and retrieval

Coming up…

- Now: vote for project 3 favorites
  - Will go over faves on Thurs

- Opportunities of scale: stuff you can do with millions of images
  - Texture synthesis of large regions
  - Recover GPS coordinates
  - Etc.

- Midterm review on next Tuesday