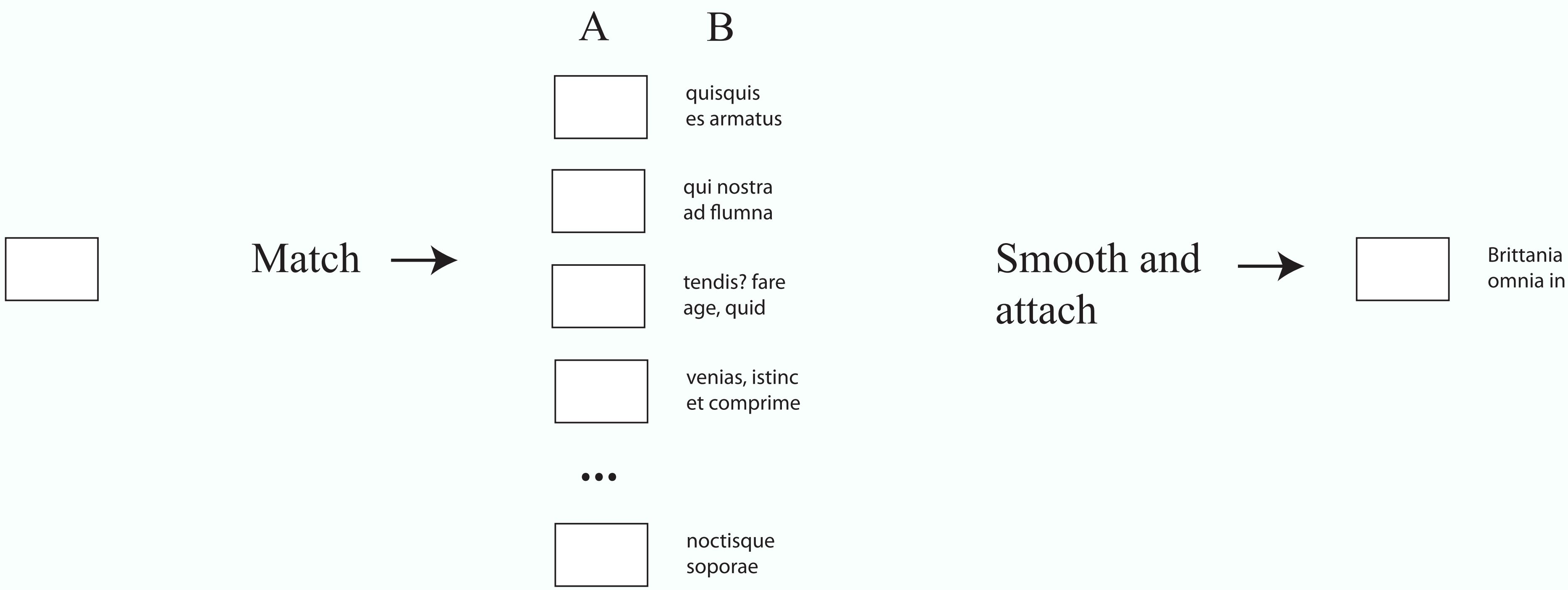
# Visual words and their applications

D.A. Forsyth

# Near Duplicate Image Detection

- Find images similar to example image
- Applications:
  - image query
  - viral marketing
  - space efficiency
  - image classification
  - hole filling
  - geolocation
  - google “goggles”

# Non-parametric regression




Linking A’s and B’s

# A=image, B=body pose

Rosales+Sclaroff, ’00; Shakhnarovich+Darrell, ’03

# A=image with hole, B=image


Efros+Leung, ’99; Hayes+Efros, ’07

# Modern hole filling methods

- Multiple strategies
- Applications
  - remove objects
  - move objects (i.e. cut out, move, fill hole)
  - make picture bigger

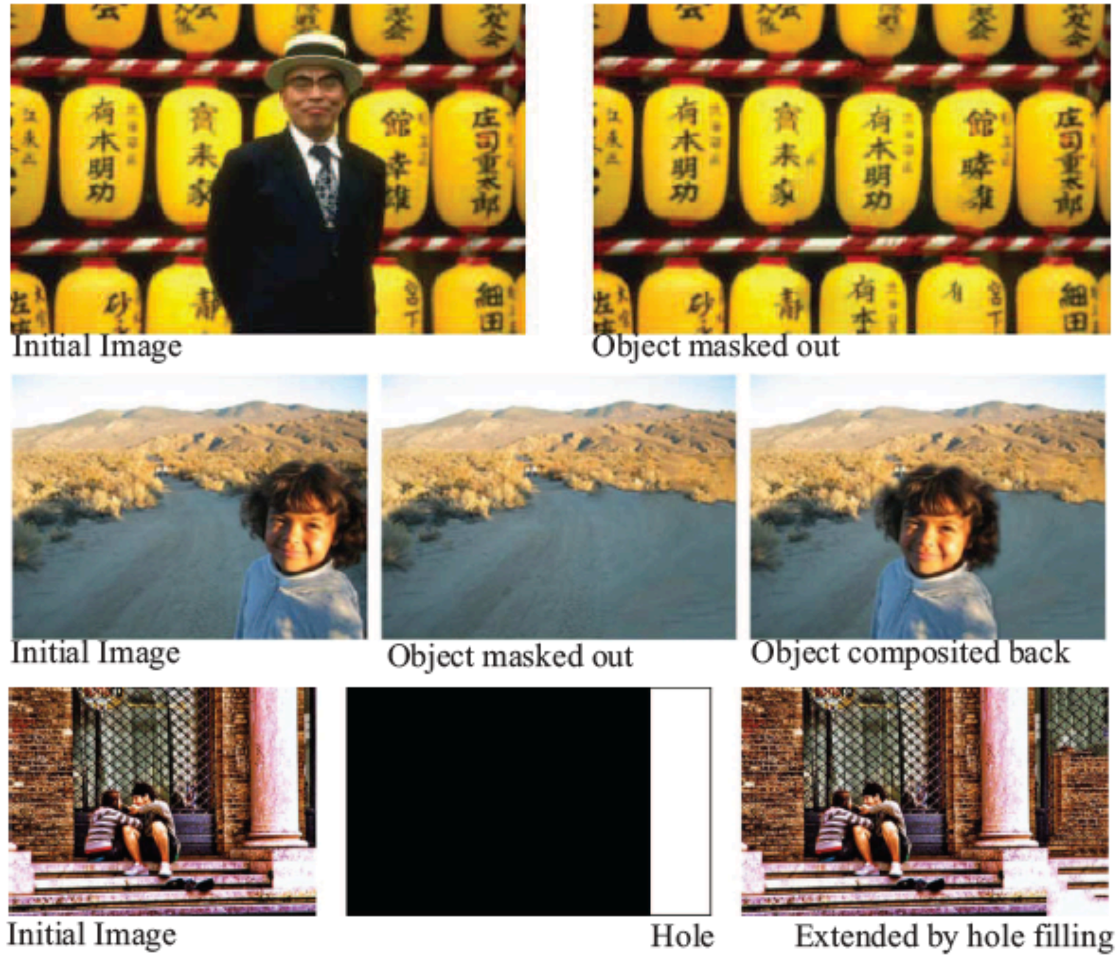



FIGURE 6.15: Modern hole-filling methods get very good results using a combination of texture synthesis, coherence, and smoothing. Notice the complex, long-scale structure in the background texture for the example on the **top** row. The **center** row shows an example where a subject was removed from the image and replaced in a different place. Finally, the **bottom** row shows the use of hole-filling to resize an image. The white block in the center mask image is the "hole" (i.e., unknown pixels whose values are required to resize the image). This block is filled with a plausible texture. *This figure was originally published as Figures 9 and 15 of "A Comprehensive Framework for Image Inpainting," by A. Bugeau, M. Bertalmío, V. Caselles, and G. Sapiro, Proc. IEEE Transactions on Image Processing, 2010 © IEEE, 2010.*

# A=image, B=location

Hayes+Efros, '08

# Accuracy of im2gps

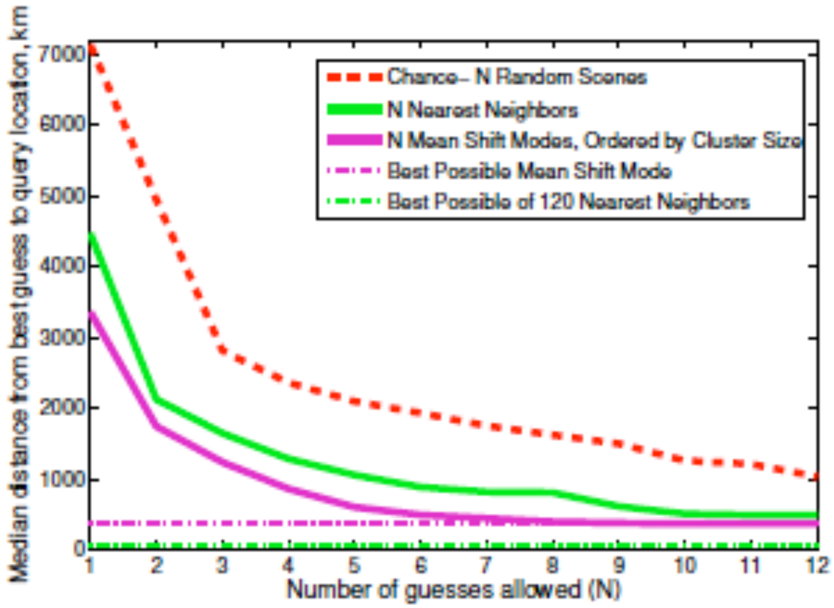
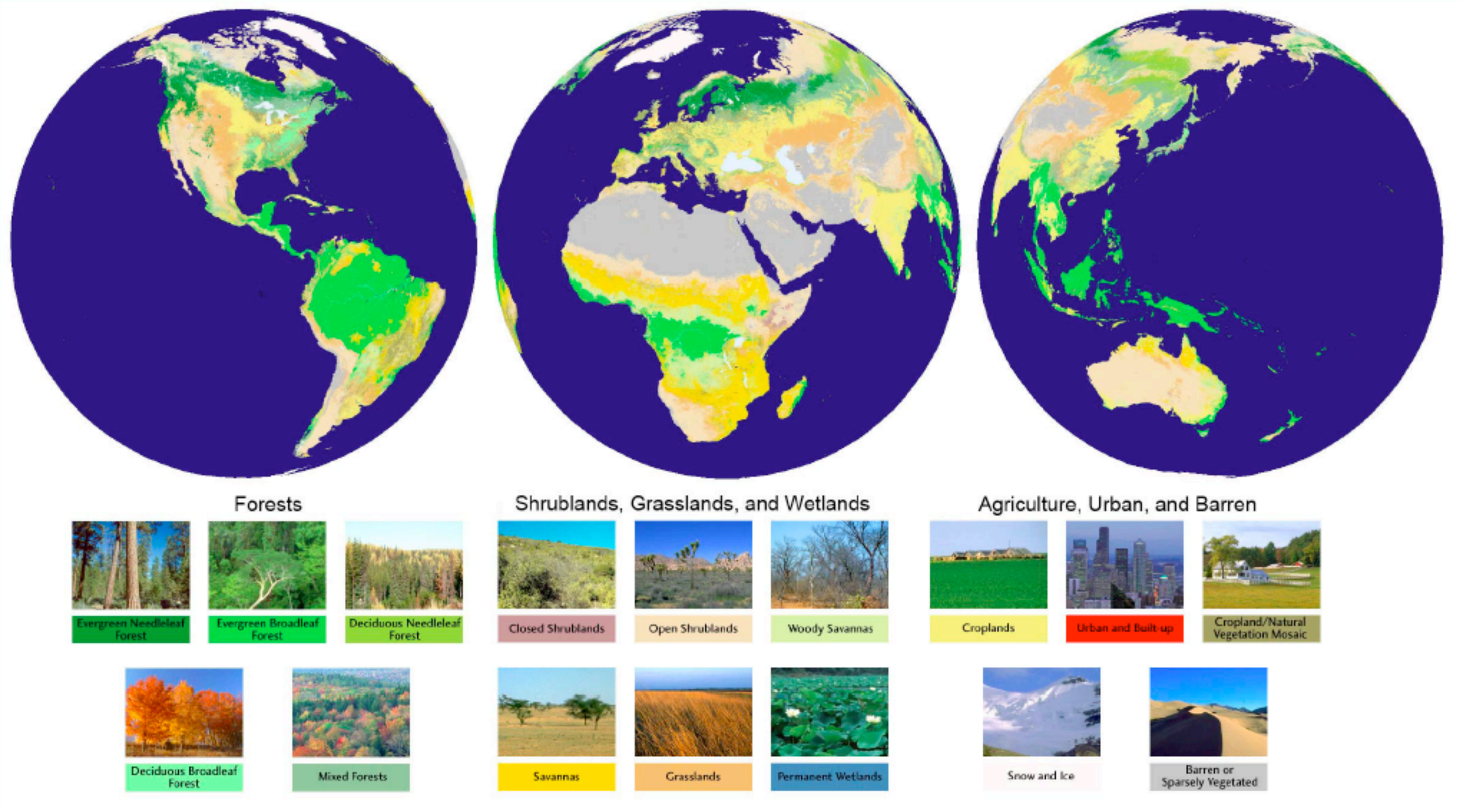


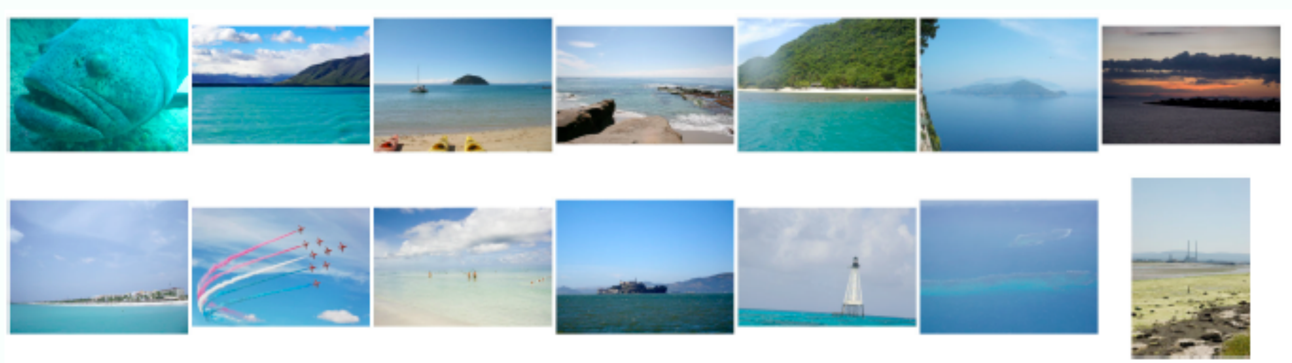

Figure 7. *Geolocation error with multiple guesses.* Median geolocation error for NN and mean-shift modes with increasing numbers of guesses allowed. The error is the distance from an algorithm's best guess to the query's ground truth location. Although the geolocation of a query may be ambiguous among several possibilities, after multiple guesses it is likely that one of the estimates is near the ground truth location.

# Land coverage map



# Image types from predicted GPS

Forest



Water

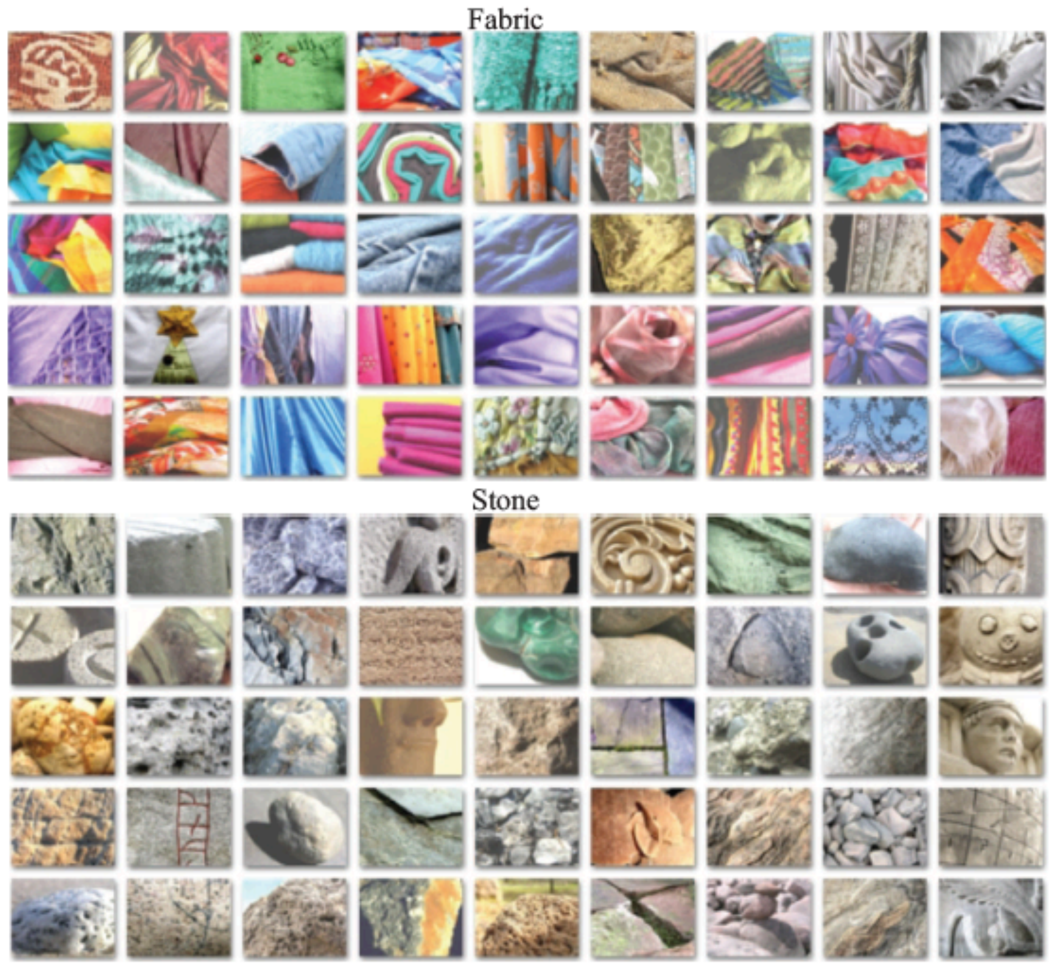
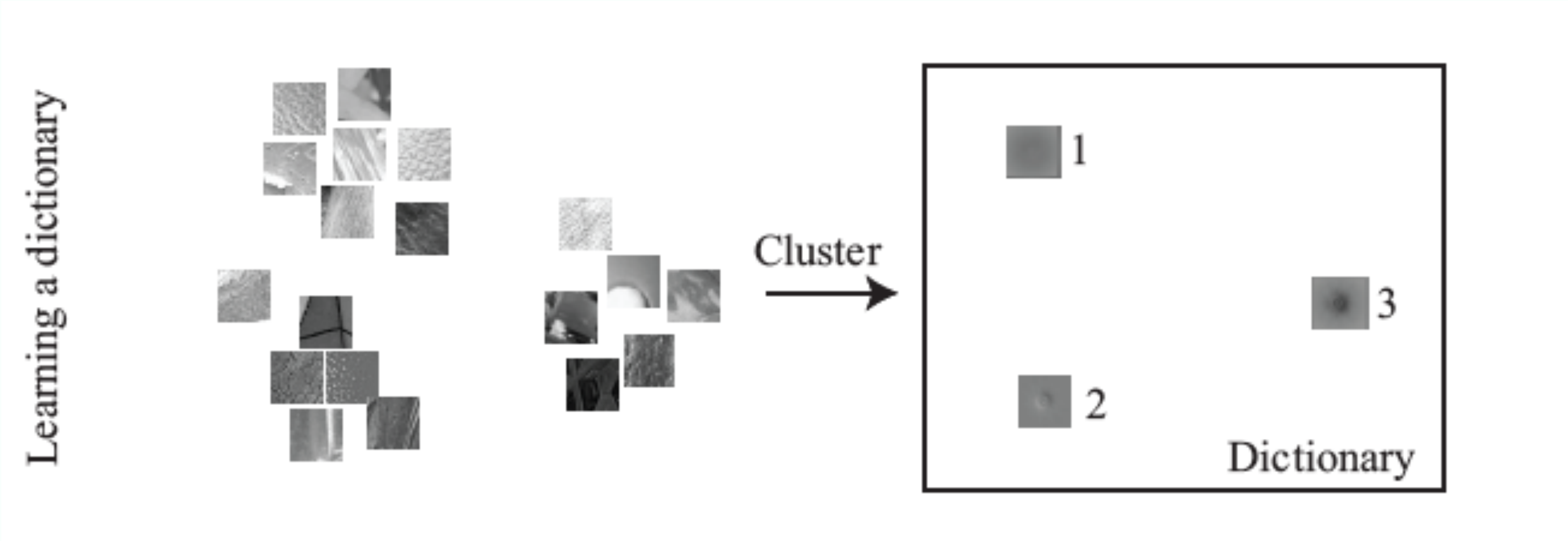



FIGURE 6.2: Typically, different materials display different image textures. These are example images from a collection of 1,000 material images, described in by Sharan *et al.* (2009); there are 100 images in each of the ten categories, including the two categories shown here (fabric and stone). Notice how (a) the textures vary widely, even within a material category; and (b) different materials seem to display quite different textures. *This figure shows elements of a database collected by C. Liu, L. Sharan, E. Adelson, and R. Rosenholtz, and published at* `http://people.csail.mit.edu/lavanya/research_sharan.html`. *Figure by kind permission of the collectors.*

# Making Visual words

- Build a dictionary of image patches by
  - taking lots
  - clustering
- Cluster centers are dictionary



# Pragmatics

- Scales:
  - millions of patches, tens of thousands of cluster centers
    - might need to use more than k-means
- What is the distance?
  - numerous answers
    - sum-of-squared pixel differences
    - sometimes weighted by Gaussian centered on patch
    - other possibilities that emphasize edges
- How do we find the nearest center?
  - small scale: search
  - large scale: approximate nearest neighbor

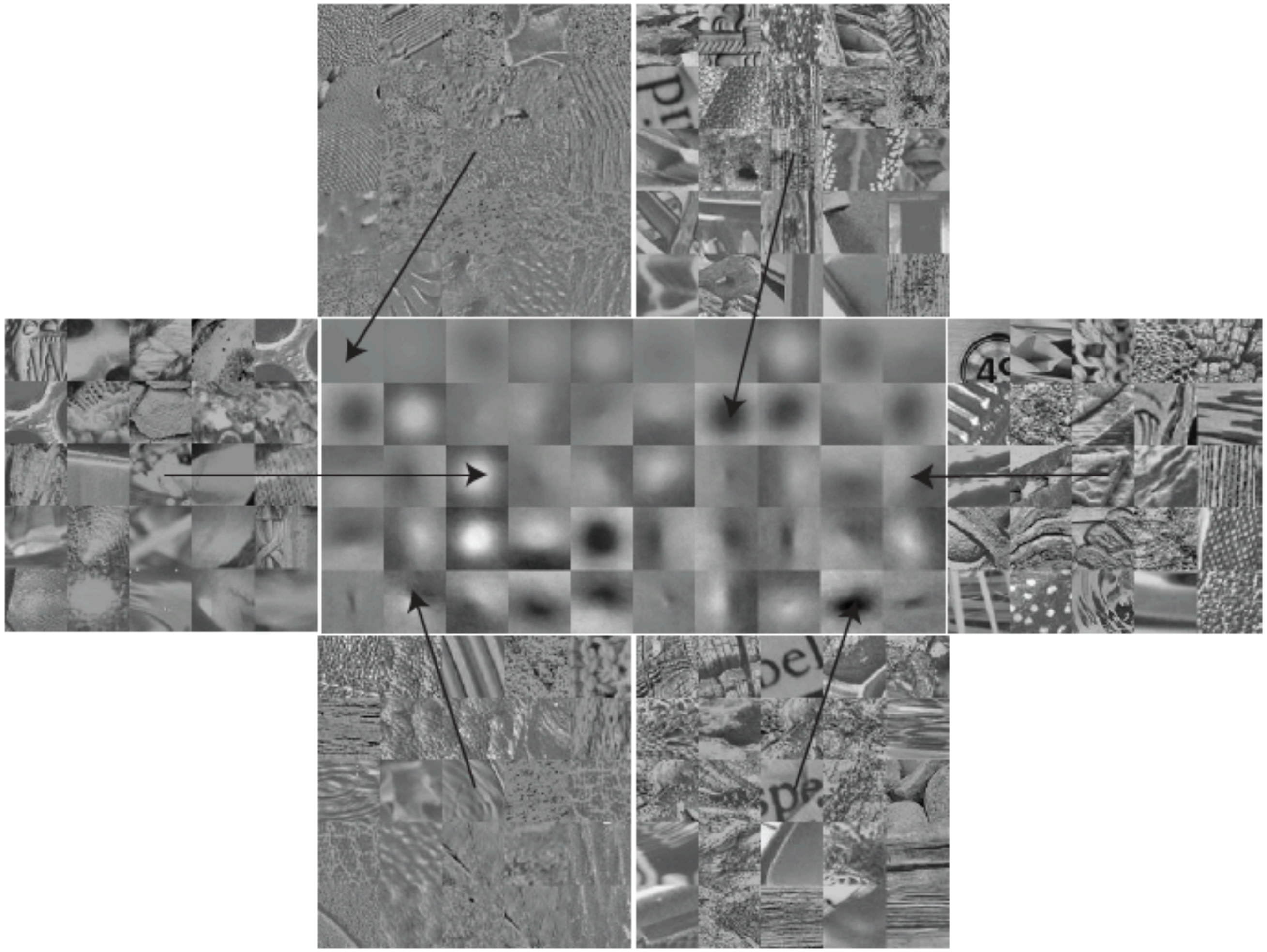



FIGURE 6.10: Pattern elements can also be identified by vector quantizing vectors obtained by reshaping an image window centered on each pixel. Here we show the top 50 pattern elements (or textons), obtained using this strategy from all 1,000 images of the collection of material images described in Figure 6.2. Each subimage here illustrates a cluster center. For some cluster centers, we show the closest 25 image patches. To measure distance, we first subtracted the average image intensity, and we weighted by a Gaussian to ensure that pixels close to the center of the patch were weighted higher than those far from the center. *This figure shows elements of a database collected by C. Liu, L. Sharan, E. Adelson, and R. Rosenholtz, and published at* `http://people.csail.mit.edu/lavanya/research_sharan.html`. *Figure by kind permission of the collectors.*

# Visual words for image denoising

- Strategy:
  - build large dictionary
- To denoise:
  - tile image with overlapping patches
  - for each patch, find dictionary center, place in location
  - at each pixel, take mean (median) value of all patches on that location
  - complex variations possible
  - very effective

FIGURE 6.17: Denoising images artificially corrupted with additive Gaussian noise. **Left:** noisy images. **Right:** restored ones using LSSC. Note that the algorithm reproduces the original brick texture in the house image ($\sigma = 15$) and the hair texture for the man image ($\sigma = 50$), both hardly visible in the noisy images. *Reprinted from "Non-local Sparse Models for Image Restoration," by J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman, Proc. International Conference on Computer Vision, (2009). © 2009, IEEE.*

# Visual words for video google

Search for objects in video by finding matching visual words:
User sketches region; system finds visual words in that region,
then finds matching visual words in the rest of the video.

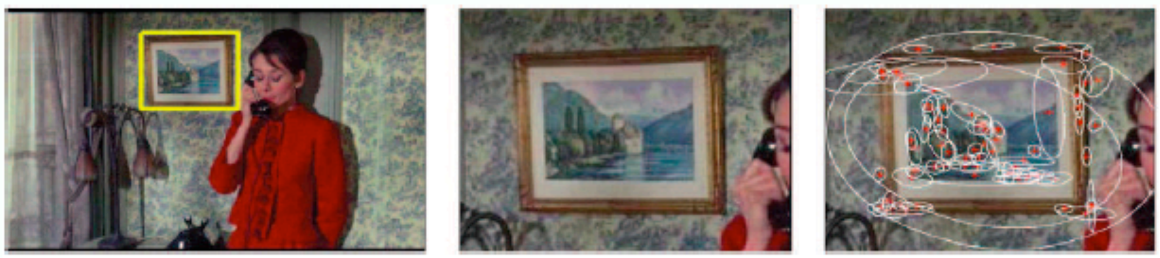


FIGURE 16.5: The original application of visual word representations was to search video sequences for particular patterns. On the **left**, a user has drawn a box around a pattern of interest in a frame of video; the **center** shows a close-up of the box. On the **right**, we see neighborhoods computed from this box. These neighborhoods are ellipses, rather than circles; this means that they are covariant under affine transforms. Equivalently, the neighborhood constructed for an affine transformed patch image will be the affine transform of the neighborhood constructed for the original patch (definition in Section 5.3.2). *This figure was originally published as Figure 11 of J. Sivic and A. Zisserman "Efficient Visual Search for Objects in Videos," Proc. IEEE, Vol. 96, No. 4, April 2008 © IEEE 2008.*

Patches don't have to be rectangular (these are ellipses).
Building an elliptical patch is outside our scope.

# Visual words for video google - II

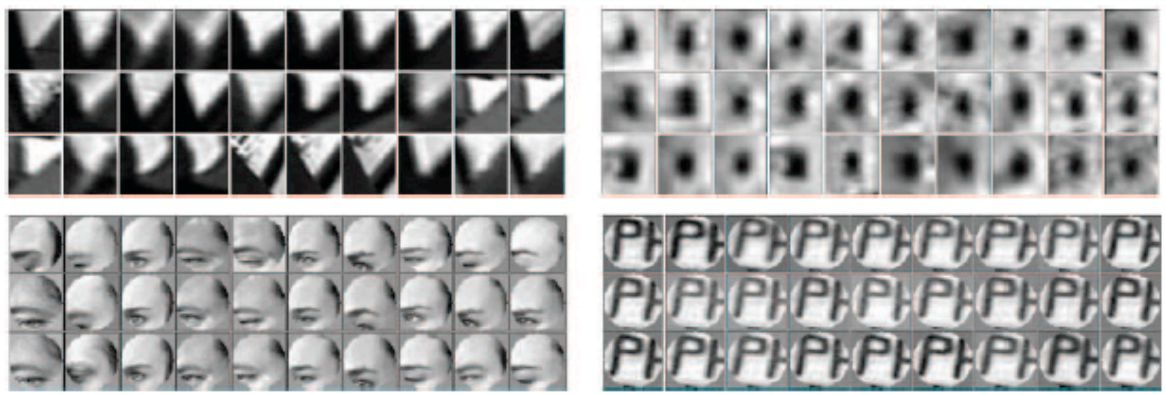



FIGURE 16.6: Visual words are obtained by vector quantizing neighborhoods like those shown in Figure 16.5. This figure shows 30 examples each of instances of four different visual words. Notice that the words represent a moderate-scale local structure in the image (an eye, one and a half letters, and so on). Typical vocabularies are now very large, which means that the instances of each separate word tend to look a lot like one another. *This figure was originally published as Figure 3 of "Efficient Visual Search for Objects in Videos," by J. Sivic and A. Zisserman, Proc. IEEE, Vol. 96, No. 4, April 2008 © IEEE 2008.*

# Visual words for video google - III

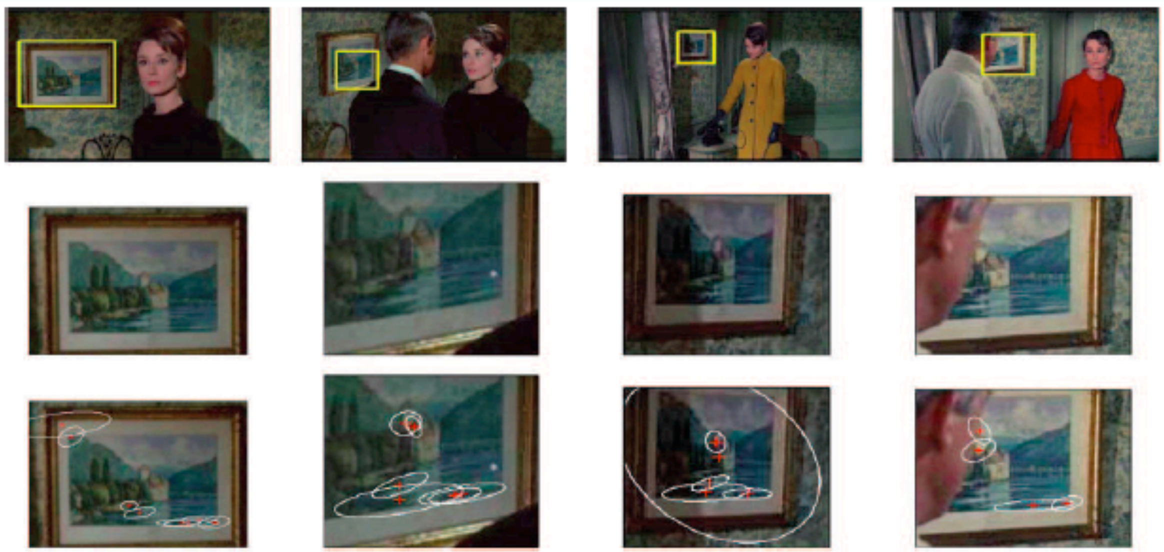


FIGURE 16.7: This figure shows results from the query of Figure 16.5, obtained by looking for image regions that have a set of visual words strongly similar to those found in the query region. The first row shows the whole frame from the video sequence; the second row shows a close-up of the box that is the result (indicated in the first row); and the third row shows the neighborhoods in that box that generated visual words that match those in the query. Notice that some, but not all, of the neighborhoods in the query were matched. *This figure was originally published as Figure 11 of J. Sivic and A. Zisserman "Efficient Visual Search for Objects in Videos," Proc. IEEE, Vol. 96, No. 4, April 2008*

# Scaling k-means

- k-means gets hard for very big datasets and big k
  - nearest neighbor search
- strategy 1:
  - cluster subset of the data, randomly selected
    - uniformly at random (but some images might not contribute tiles)
    - stratified sample (eg r tiles per image)
  - cluster centers shouldn't change much
- strategy 2:
  - hierarchical k-means
    - very good for large k

# Hierarchical k-means

- Clustering
  - Cluster with small k
  - Now divide dataset into k groups (one per cluster center)
    - corresponding to clusters
  - Now cluster each group with k-means,
    - possibly using a different k
    - possibly repeat
- Finding a dictionary entry
  - find entry at top level
    - now in that level, find entry
      - and so on
- Good if you want large number of cluster centers

# Using visual words: Image classification

- Problem: Is this a picture of a tree or not?
  - strategy:
    - form a histogram of visual words
    - feed to classifier (*not* a linear classifier)
    - this is effective
- Impact:
  - most cases are still research topics
  - “adult” image detection is widely applied
    - though not discussed
    - some evidence this is used, with other features

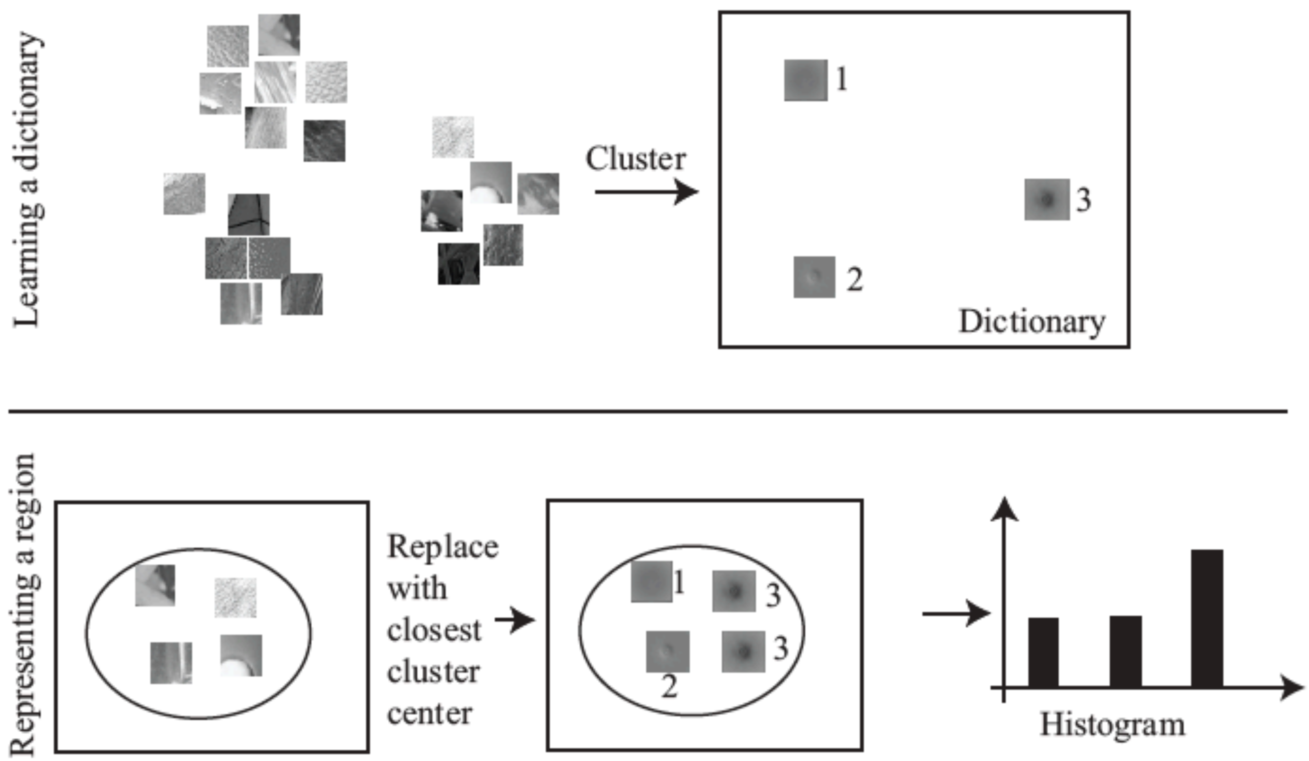



FIGURE 6.8: There are two steps to building a pooled texture representation for a texture in an image domain. First, one builds a dictionary representing the range of possible pattern elements, using a large number of texture patches. This is usually done in advance, using a training data set of some form. Second, one takes the patches inside the domain, vector quantizes them by identifying the number of the closest cluster center, then computes a histogram of the different cluster center numbers that occur within a region. This histogram might appear to contain no spatial information, but this is a misperception. Some frequent elements in the histogram are likely to be textons, but others describe common ways in which textons lie close to one another; this is a rough spatial cue. *This figure shows elements of a database collected by C. Liu, L. Sharan, E. Adelson, and R. Rosenholtz, and published at* `http://people.csail.mit.edu/lavanya/research_sharan.html`. *Figure by kind permission of the collectors.*

# Near Duplicate Image Detection

- Model: Information retrieval
  - find documents using keywords
  - we'll review this area
- Q: what are the keywords?
  - A: visual words
    - which come from k-means clustering

# Information retrieval

- Word frequencies are revealing
  - eg “crimson”, “magenta”, “chrysoprase”, “saffron” ->?
  - eg “pachyderm”, “grey”, “trunk”, “ivory”, “endangered” ->?
  - eg “flour”, “milk”, “eggs”, “sugar”, “vanilla”, “cherries” ->?
- Some words aren’t helpful
  - eg “a”, “an”, “he”, “it”, “she”, “but”, “and” and some others
  - often referred to as “stop words”
- Insight
  - representing a document by word counts works
    - you can weight in various ways

# Information retrieval

- Word by document table
  - entries:
    - drop stopwords
    - 0 if word is not in document, 1 if it is
    - (more later)
  - very very sparse
  - D
- Simplest use
  - as an index (IR people call this an inverted index!)
  - if query is k1, k2, k3 we can
    - get D(k1, :) (which is a list of documents containing k1)
    - intersect with D(k2, :)
    - etc.

# The word-document table

- Problems with D
  - easy
    - all words counted with the same weight
  - hard
    - cannot rank by similarity between documents and query
    - some words are not in documents “by accident”

- Weighting
  - 0-1: (as above)
  - Frequency: replace 1 with count
  - TF-IDF: more weight on words that are common in this document, uncommon in others

# TF-IDF weighting

all documents. Write $N_d$ for the total number of documents and $N_t$ for the number of documents that contain the particular term we are interested in. Then, the inverse document frequency can be estimated as $N_d/(1+N_t)$ (where we add one to avoid dividing by zero). Write $n_t(j)$ for the number of times the term appears in document $j$ and $n_w(j)$ for the total number of words that appear in that document. Then, the tf-idf weight for term $t$ in document $j$ is

$$\left(\frac{n_t(j)}{n_w(j)}\right)/\log\left(\frac{N_d}{(1+N_t)}\right).$$

We divide by the log of the inverse document frequency because we do not want very uncommon words to have excessive weight. Inserting this tf-idf weight into

# Measuring document similarity - I

- Similar documents have similar word counts
  - c is the word count vector; this could be weighted
  - this is a column of D corresponding to the document

is present. This measure might be as simple as a one if the word appears at least once in the document, or might be a count of the number of words. Write $c_1$, $c_2$ for two such vectors; the *cosine similarity* between the documents they represent is

$$\frac{c_1 \cdot c_2}{\| c_1 \| \| c_2 \|}.$$

# Measuring word similarity - I

- Similar words appear in similar documents
  - c is now the vector of documents that contain the word; could be weighted
  - c is now a row of D

is present. This measure might be as simple as a one if the word appears at least once in the document, or might be a count of the number of words. Write $c_1$, $c_2$ for two such vectors; the *cosine similarity* between the documents they represent is

$$\frac{c_1 \cdot c_2}{\| c_1 \| \| c_2 \|}.$$

# “Missing” data

- Word counts of 0 are often misleading
  - eg “elephant”, “trunk”, “ears”, “ivory”
    - should have “pachyderm” (same subject)
    - but might not, because it’s uncommon
  - this will make document similarity estimates unreliable
- Strategy: smooth word counts
  - by projecting word count vectors onto low dimensional basis
  - basis represents “topics”
  - obtain by SVD

# Latent Semantic Analysis - I

- Recall: D is word by document table
- Take an SVD of D to get $$\mathcal{D} = \mathcal{U}\Sigma\mathcal{V}^T$$
  - cols of U are an orthogonal basis for the cols of D
  - cols of V are an orthogonal basis for the rows of D (notice V^T!)
  - Sigma is diagonal; sort the diagonal to get largest at top
  - Notice
    - cols of U span word count vectors (cols of D)
    - cols of U corresponding to big singular values are common types of word count
    - cols of U corresponding to small singular values are uncommon types of word count

# Latent Semantic Analysis - II

- Recall: SVD of D is $\mathcal{D} = \mathcal{U}\Sigma\mathcal{V}^T$
- Strategy for smoothing word counts:
  - take word count vector c
  - expand on some of U's cols corresponding to large singular values
  - yields new, smoothed count vector
    - eg if many "elephant" documents contain "pachyderm", then smoothed "pachyderm" count will be non-zero for all elephant documents.
- Obtain a smoothed word document matrix hat(D) like this

Write $\mathcal{U}_k$ for the matrix consisting of the first $k$ columns of $\mathcal{U}$, $\mathcal{V}_k$ for the matrix consisting of the first $k$ columns of $\mathcal{V}$, $\Sigma_k$ for $\Sigma$ with all but the $k$ largest singular values set to be zero, and write $\hat{\mathcal{D}} = \mathcal{U}_k\Sigma_k\mathcal{V}_k^T$.

# Latent semantic analysis - III

- Want a table of all smoothed document similarities

To compute cosine similarity between an old document and a new document with count vector $\boldsymbol{q}$, we project the new document's count vector onto the columns of $\mathcal{U}_k$ to obtain $\hat{\boldsymbol{q}} = \mathcal{U}_k \mathcal{U}_k^T \boldsymbol{q}$. We can then take the inner product of $\hat{\boldsymbol{q}}$ and $\hat{\boldsymbol{d}}_i$. A complete table of inner products (cosine distances) between documents is given by

$$\hat{\mathcal{D}}^T \hat{\mathcal{D}} = (\mathcal{V}_k \Sigma_k)(\Sigma_k \mathcal{V}^T) = (\Sigma_k \mathcal{V}_k^T)^T (\Sigma_k \mathcal{V}^T),$$

so that we can think of the columns of $\Sigma_k \mathcal{V}^T$ as points in a $k$-dimensional "concept space" that represents the inner products exactly. One could, for example, cluster documents in this space rather than the original count space, and expect a better clustering. Computing the SVD of $\mathcal{D}$ is known as *latent semantic analysis*; using the concept space for indexing is known as *latent semantic indexing*.

# Latent semantic analysis - IV

- Measuring the similarity between words

$\hat{\mathcal{D}}$ is useful in other ways, too. There is a rough tendency of words that have similar meaning to appear near similar words in similar documents, an idea known as *distributional semantics*. This means that cosine similarity between *rows* of $\hat{\mathcal{D}}$ is an estimate of the similarity of meaning of two terms, because it counts the extent to which they co-occur. Furthermore, $\hat{\mathcal{D}}$ can be used as an inverted index. If we use it in this way, we are not guaranteed that every document recovered contains all the words we used in the query; instead, it might contain very similar words. This is usually a good thing. The columns of $\mathcal{U}$ are sometimes called *topics* and can be thought of as model word frequency vectors; the coordinates of a column in the semantic space show the weights with which topics should be mixed to obtain the document.

- Notice you can build hat(D) “on the fly”
  - result: neat trick based on google for finding similarity between two words

# Latent semantic analysis - V

- hat(D) is a “better” word-document table than D
  - a keyword yields a hit to documents that don’t contain it (but should)
  - yields a similarity rank for documents
    - cosine similarity between query and document word vector
  -

# Why visual words are cool - I

- You can do all above with visual words
  - inverted index
  - hat(D)
  - similarity
- Simple near duplicate detection
  - compute visual words for query image
  - keyword query to D
- Smarter near duplicate detection
  - compute visual words for query image
  - keyword query to hat(D)
  - rank by similarity cosines
  - keep top responses

# Why visual words are cool - II

- Visual words depend too strongly on the clustering?
  - affects results
  - easy answer: build three systems and vote (eg by averaging ranking)
    - amazingly effective
- Numerous other systems issues now handled very effectively

# Next up: Image segmentation as clustering