Last time

- Texture synthesis wrap up
- Optical flow: estimating motion in video

Review:
- What can we expect from an Nth order Markov field for texture synthesis (N > 1)?
- What is the aperture problem?
- What can cause flow errors at object boundaries?

Recall: Motion estimation techniques

- Direct methods
  - Directly recover image motion at each pixel from spatio-temporal image brightness variations
  - Dense motion fields, but sensitive to appearance variations
  - Suitable for video and when image motion is small

- Feature-based methods
  - Extract visual features (corners, textured areas) and track them over multiple frames
  - Sparse motion fields, but more robust tracking
  - Suitable when image motion is large (10s of pixels)

Motion magnification

original

magnified
Motion magnification

http://people.csail.mit.edu/mrub/vidmag/

So far: features and filters

Transforming images; gradients, textures, edges, flow

Now: Fitting

• Want to associate a model with observed features

Fitting: Main idea

• Choose a parametric model to represent a set of features
• Membership criterion is not local
  • Can’t tell whether a point belongs to a given model just by looking at that point
• Three main questions:
  • What model represents this set of features best?
  • Which of several model instances gets which feature?
  • How many model instances are there?
• Computational complexity is important
  • It is infeasible to examine every possible set of parameters and every possible combination of features

Case study: Line fitting

• Why fit lines?
  Many objects characterized by presence of straight lines

• Wait, why aren’t we done just by running edge detection?

Difficulty of line fitting

• Extra edge points (clutter), multiple models:
  • which points go with which line, if any?
• Only some parts of each line detected, and some parts are missing:
  • how to find a line that bridges missing evidence?
• Noise in measured edge points, orientations:
  • how to detect true underlying parameters?
Voting
- It’s not feasible to check all combinations of features by fitting a model to each possible subset.
- Voting is a general technique where we let the features vote for all models that are compatible with it.
  - Cycle through features, cast votes for model parameters.
  - Look for model parameters that receive a lot of votes.
- Noise & clutter features will cast votes too, but typically their votes should be inconsistent with the majority of “good” features.

Fitting lines: Hough transform
- Given points that belong to a line, what is the line?
- How many lines are there?
- Which points belong to which lines?
- Hough Transform is a voting technique that can be used to answer all of these questions.
  Main idea:
  1. Record vote for each possible line on which each edge point lies.
  2. Look for lines that get many votes.

Finding lines in an image: Hough space
Connection between image (x,y) and Hough (m,b) spaces
- A line in the image corresponds to a point in Hough space
- To go from image space to Hough space:
  - given a set of points (x,y), find all (m,b) such that y = mx + b

Finding lines in an image: Hough algorithm
How can we use this to find the most likely parameters (m,b) for the most prominent line in the image space?
- Let each edge point in image space vote for a set of possible parameters in Hough space
- Accumulate votes in discrete set of bins; parameters with the most votes indicate line in image space.
Polar representation for lines

Issues with usual (m,b) parameter space: can take on infinite values, undefined for vertical lines.

$d$: perpendicular distance from line to origin

$\theta$: angle the perpendicular makes with the x-axis

$x \cos \theta - y \sin \theta = d$

Point in image space $\rightarrow$ sinusoid segment in Hough space

Hough transform algorithm

Using the polar parameterization:

$x \cos \theta - y \sin \theta = d$

Basic Hough transform algorithm

1. Initialize $H[d, \theta] = 0$
2. for each edge point $(x, y)$ in the image
   for $\theta = \theta_{\text{min}}$ to $\theta_{\text{max}}$ // some quantization
   $H[d, \theta] += 1$
3. Find the value(s) of $(d, \theta)$ where $H[d, \theta]$ is maximum
4. The detected line in the image is given by $d = x \cos \theta - y \sin \theta$

Time complexity (in terms of number of votes per pt)?

Example: What was the shape?

Square:

Example: Hough transform for straight lines

Which line generated this peak?
Showing longest segments found

Impact of noise on Hough

Here, everything appears to be "noise", or random edge points, but we still see peaks in the vote space.

Extensions

Extension 1: Use the image gradient
1. same
2. for each edge point (x,y) in the image
   compute unique (d, θ) based on image gradient at (x,y)
   \( H[d, θ] += 1 \)
3. same
4. same
   (Reduces degrees of freedom)

Extension 2
• give more votes for stronger edges (use magnitude of gradient)

Extension 3
• change the sampling of (d, θ) to give more/less resolution

Extension 4
• The same procedure can be used with circles, squares, or any other shape...

Hough transform for circles

• Circle: center (a,b) and radius r
  \((x - a)^2 + (y - b)^2 = r^2\)
• For a fixed radius r, unknown gradient direction

Source: Steve Seitz

Impact of noise on Hough

What difficulty does this present for an implementation?
Hough transform for circles

- Circle: center \((a,b)\) and radius \(r\)
  \[(x - a)^2 + (y - b)^2 = r^2\]
- For a fixed radius \(r\), unknown gradient direction

\[
\begin{align*}
\text{Hough space} & \\
\text{Image space} & \\
\end{align*}
\]

Intersection: most votes for center occur here.

Hough transform for circles

- Circle: center \((a,b)\) and radius \(r\)
  \[(x - a)^2 + (y - b)^2 = r^2\]
- For an unknown radius \(r\), unknown gradient direction

\[
\begin{align*}
\text{Hough space} & \\
\text{Image space} & \\
\end{align*}
\]

Hough transform for circles

- Circle: center \((a,b)\) and radius \(r\)
  \[(x - a)^2 + (y - b)^2 = r^2\]
- For an unknown radius \(r\), unknown gradient direction

\[
\begin{align*}
\text{Hough space} & \\
\text{Image space} & \\
\end{align*}
\]

Example: detecting circles with Hough

For every edge pixel \((x,y)\):
For each possible radius value \(r\):
  For each possible gradient direction \(\theta\):
    // or use estimated gradient at \((x,y)\)
    \[
    \begin{align*}
    a &= x + r \cos(\theta) \text{ // column} \\
    b &= y - r \sin(\theta) \text{ // row} \\
    H[a,b,r] &= 1
    \end{align*}
    \]
end
end

Time complexity per edge pixel?

- Check out online demo: [http://www.markschulze.net/java/hough/](http://www.markschulze.net/java/hough/)

Note: a different Hough transform (with separate accumulators) was used for each circle radius (quarters vs. penny).
Example: detecting circles with Hough

Example: iris detection

Voting: practical tips

Hough transform: pros and cons

Generalized Hough Transform

Pros
- All points are processed independently, so can cope with occlusion, gaps
- Some robustness to noise: noise points unlikely to contribute consistently to any single bin
- Can detect multiple instances of a model in a single pass

Cons
- Complexity of search time increases exponentially with the number of model parameters
- Non-target shapes can produce spurious peaks in parameter space
- Quantization: can be tricky to pick a good grid size

Pro: Original Edges

Voting: Quarter


Voting: practical tips

- Minimize irrelevant tokens first
- Choose a good grid / discretization
- Vote for neighbors, also (smoothing in accumulator array)
- Use direction of edge to reduce parameters by 1
- To read back which points voted for “winning” peaks, keep tags on the votes.

Generalized Hough Transform

What if we want to detect arbitrary shapes?

Intuition:

Now suppose those colors encode gradient directions…
Generalized Hough Transform

- Define a model shape by its boundary points and a reference point.

**Offline procedure:**
At each boundary point, compute displacement vector: \( \mathbf{r} = \mathbf{a} - \mathbf{p} \).

Store these vectors in a table indexed by gradient orientation \( \theta \).

**Detection procedure:**
- For each edge point:
  - Use its gradient orientation \( \theta \) to index into stored table
  - Use retrieved \( \mathbf{r} \) vectors to vote for reference point

Assuming translation is the only transformation here, i.e., orientation and scale are fixed.

Generalized Hough for object detection

- Instead of indexing displacements by gradient orientation, index by matched local patterns.

**Image:**
- Training image:
  - "Visual codeword" with displacement vectors

**Image:**
- Test image:
  - "Visual codeword" with displacement vectors

Example: Results on Cows

**Image:**
- Original image

**Image:**
- Interest points
**Example: Results on Cows**

Matched patches

**Example: Results on Cows**

Votes

**Example: Results on Cows**

1st hypothesis

**Example: Results on Cows**

2nd hypothesis

**Example: Results on Cows**

3rd hypothesis

**Summary**

- **Fitting** problems require finding any supporting evidence for a model, even within clutter and missing features.
  - associate features with an explicit model

- **Voting** approaches, such as the Hough transform, find likely model parameters without searching all combinations of features.
  - Hough transform approach for lines, circles, …, arbitrary shapes defined by a set of boundary points, recognition from patches.