**CSci 4968 and 6270**
**Computational Vision,**
**Fall Semester, 2011-2012**
**Lectures 4 & 5, Feature Extraction**

# 1 Introduction

## Overview

- Edge extraction
  - Smoothing
  - 1st and 2nd order differentiation
  - After differentiation: non-maximum suppression, thresholding, sub-pixel localization
  - Linking into edgel chains
- Line extraction
  - Line representations
  - Line distances and line fitting
  - Hough transforms

# 2 Edge Detection

## Overview of the Steps of Edge Detection

1. Smoothing
2. Differentiation
3. Thinning / non-maximum suppression
4. Subpixel localization
5. Linking
6. Thresholding, perhaps adaptively.

## Smoothing for Edge Detection

Choice is the Gaussian, for reasons we have already discussed

- Istotropic and separable
- Fast implementations
- Good spatial and frequency properties

## Choosing the Smoothing Scale: Scale Space

- Smaller values of $\sigma$ lead to more detailed edges and better edge localization.
- Larger values of $\sigma$ lead to fewer and perhaps more significant edges
- Different scales are needed when images have different sizes or magnifications
- We can combine these by working with multiple scales, treating $\sigma$ as an additional variable, producing what is called *scale space*.
  - We will look at scale space in much more detail in future lectures.
- For the purposes of these notes, we will work with the results of smoothing with a single value of $\sigma$.

## Differential Operations

- Two common choices
  - 1st derivative based on the directional derivatives, the gradient, and finding peak.
  - 2nd derivative based on the Laplacian and finding "zero-crossings".
- While our emphasis will be on first derivative operations, during lecture we will take a brief look at the formal structure of the Laplacian and why it is used. In particular, we will discuss the equation

  $$\nabla^2(I * G)$$

  where

  $$\nabla^2(f) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

  We will return to the use of the Laplacian during "interest point" extraction.

## Computing the Gradient

- Let $I_s$ be the smoothed image.
- Compute $I_x = \partial I_s/\partial x$ and $I_y = \partial I_s/\partial y$ using discrete differentiation.
- Then we can compute the gradient direction and magnitude as

  $$I_m(x,y) = \|I_x(x,y)^2 + I_y(x,y)^2\|^{1/2}$$
  $$I_\theta(x,y) = \text{atan2}(I_x(x,y), I_y(x,y))$$

- The direction is "normal" to the direction of the edge contour and points in the direction of most rapid intensity increase.

### Other Operators

- Other partial derivative operators have been used. For example, the Sobel and Prewitt $\partial f/\partial x$ kernels are

$$\begin{pmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{pmatrix}$$

  respectively.

- Notice the implicit smoothing in these operators.

### Additional Steps are Needed Following Smoothing and Gradient Computation

- After gradient computation, we still just have an image
- Pure thresholding leaves us with pixel locations of strong gradients, not edges (edge elements).
- We want to "extract" the edges, both individually and in contours.
- Decisions about what is a true edge and what is not may be made in groups of edgels.

### Non-Maximum Suppression

- For each pixel, suppress it as a "non-edge" if a neighboring pixel has a stronger gradient.
- Here the neighbors should only be neighboring pixels in the gradient direction.
  - Discretization effects must be considered here, since the determination of who is a "neighbor" may not be symmetric.
- Candidate edgels are the pixels that are left after non-maximum suppression.
  - This can include locations with very low gradient magnitudes!

### Sub-Pixel Localization of Edges

Still only know pixel locations of peaks, so...

- At each surviving peak, fit a parabola to the gradient magnitudes along the gradient direction.
- Offset the edge location along the normal based on the location of the peak.
- Yields the subpixel edge location
- While fitting a parabola sounds like an expensive computation, we will see in class that it reduces to straightforward arithmetic.

### Edge Linking

Group edgels into chains:

- Tangent direction is 90° rotated from the gradient direction.
- Record "next" and "previous" edgels at each edge (think of a doubly-linked list) — the edges "ahead" and "behind" the current one.
- Resolving ambiguities:
  - Find candidates for linking based on consistency of position **and** orientation.
  - If there are two candidates ahead (or behind) and they are consistent with each other, choose the four-connected neighbor first
  - If they aren't consistent, then this is a branching location. We can either
    - Terminate the chain or
    - Attempt to find the stronger link and continue it.
- Result is a series of edgel chains.

### Edge Thresholding and Hysteresis

We still haven't decided which points are really "edgels", so we need to apply a threshold:

- Could use a single global threshold, perhaps computed from the image gradient statistics.
  - For example, assume the edgels occupy no more than a certain percentage of the image and use the histogram to compute a threshold.
- Could compute different thresholds in different (but overlapping) regions, perhaps interpolating between them to generate (potentially) different thresholds at each pixel.
- Double thresholding — also called "hysteresis" thresholding:
  - Two thresholds: $\theta_1 < \theta_2$.
  - All edgels with gradient magnitude below $\theta_1$ are eliminated
  - Those above $\theta_2$ are kept.
  - Those in between must be connected by a chain of edgels with strengths above $\theta_1$ to an edgel with strength above $\theta_2$ in order to survive.

### Examples

Using Matlab's edge function, we will examine edge detection results on a number of images.

### Summary: How Well Does it Work?

- When thinking about how we want a vision system to work, we think "boundaries" instead of "edges".
- Therefore, we want "edges" where there are no significant intensities changes, and we want to ignore many prominent edges.
- Some of this can be seen in the following examples from Martin, Fowlkes, Malik, IEEE T-PAMI 2004:

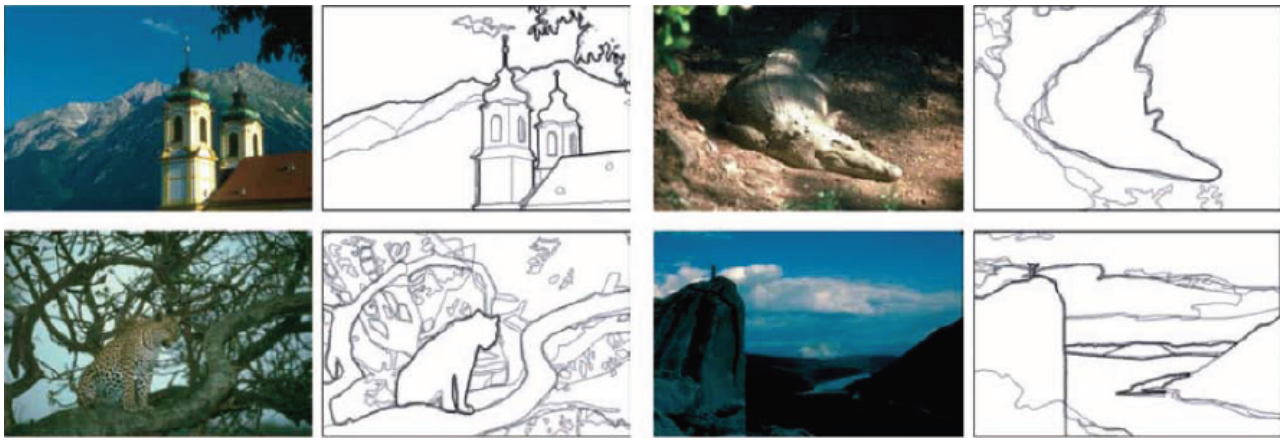

Figure 1: How well does edge detection work?

# 3 Edgels to Lines and Curves

## Overview

- "Edge detection", so far, has produced only edge values at particular pixels, located to subpixel accuracy, and perhaps formed into linked lists.
- This isn't "perceptual"
- We are interested in more complete geometric entities: lines, curves, etc.
- Our focus in the rest of these notes is line extraction. Along the way we'll study a bit of mathematics.

## Approximating Edgel Chains with Lines

Simple technique to split chain:

1. Form line between chain end points
2. Find point on contour that is furthest from the line
3. If this distance is below a threshold, stop
4. Otherwise, split the contour into two chains at this farthest point, and continue recursively.

Produces what is called a "poly-line".

## Line Representations

Many different representations, starting with the simple version most of us learned first.

- *Slope-intercept:* Parameters are $m$ and $b$,

  $$y = mx + b.$$

  This has two problems for our use. What are they?

- *Point / normal*: Parameters are normal $\boldsymbol{\eta}$ and point on line $\mathbf{p}_0$. Point $\mathbf{p} = (x, y)^\top$ is on the line iff

  $$\boldsymbol{\eta} \cdot (\mathbf{p} - \mathbf{p}_0) = 0.$$

  This appears to have four degrees of freedom, but really there are only two.

  – First example of an "over-parameterized" equation — more parameters than geometric degrees of freedom.

- *Implicit form:* Given parameter vector $\mathbf{a} = (a, b, c)^\top$, $\mathbf{p} = (x, y)^\top$ is on the line if

  $$ax + by + c = 0.$$

  By imposing the constraint $a^2 + b^2 = 1$, we end up with only two degrees of freedom.

- *Closest point form:* Find the point, $(\rho, \theta)$, in polar coordinates of the point on the line that is closest to the origin. This is really a special case of the point/normal form:

- It is pretty easy to see that the vector $(\cos\theta, \sin\theta)^\top$ is normal to (perpendicular to) the direction of the line. Hence, when the line goes through the origin – and therefore $\rho = 0$ — the value of $\theta$ can be chosen to enforce normality.
- We can plug $\boldsymbol{\eta} = (\cos\theta, \sin\theta)^\top$ and $\mathbf{p}_0 = (\rho\cos\theta, \rho\sin\theta)^\top$ into the above point/normal formulation. After some algebraic manipulation we can show that a point $\mathbf{p} = (x, y)^\top$ is on the line if and only if

$$x\cos\theta + y\sin\theta - \rho = 0.$$

- This clarifies the relationship between the point/normal form and the implicit form, since $a = \cos\theta$, $b = \sin\theta$, $c = -\rho$.

- *Parametric form:* Given $x_0$, $x_1$, $y_0$, $y_1$, we can generate points on the line as

$$x(t) = x_1 t + x_0$$
$$y(t) = y_1 t + y_0$$

Once again this only has two real degrees of freedom.

### Line Representation Issues

- Need to represent all possible lines and introduce no bias in the representation
- Some representations are better for generating points, while others are better for determining when a point is on a line and for determining the point-to-line distance.
- Some parameterizations have more parameters than degrees of freedom. We need to handle this during any estimation process.

### Calculations Needed on Lines

- Is a point on a line?
  - This is immediate with all forms except the parametric.
- Generating points on a line (as in graphics)
  - Easy using the parametric form; quite hard using the others
- We have two more calculations — the most important for our purposes:
  - What is the distance of a point to a line?
  - Given a set of points, what is the best-fitting line?

  Both depend on what we mean by "distance".
- We will define two different distances and then look at several solutions.

### Distance Measures

A data point will be written as $(x_i, y_i)^\top$,

- For the slope/intercept form, the distance is generally measured as the difference in $y$:
$$d_i = |y_i - mx_i - b|$$
- This is appropriate for "regression" problems where the $x$ value is given and fixed — the "independent variable" — and the $y$ value is measured — the "dependent variable".
- For most other forms the distance is the distance between $(x_i, y_i)^\top$ and the closest point on the line in a Euclidean distance sense.
- The easist way to envision this is geometrically with the closest-point formulation.
- With a little bit of work, combining geometric and algebraic arguments, this leads to the simple solution that
$$d_i = |x_i \cos\theta + y_i \sin\theta - \rho|.$$
- For the implicit form, we need to solve for a different $t$ for each data point to find the closest point. This leads back to the implicit and point/normal formulations, and therefore the implicit formulation is not used directly in line fitting.

### Least-Squares Line Fitting

It is important to be clear about our goals:

- Given is a set of $N$ point locations $\{(x_i, y_i)\}, i = 1, \ldots, N$. In other words these locations are the data and are "given" to use (for the purposes of the line fitting problem).
- The *unknowns* are the parameters of the line: $m$ and $b$; $a$, $b$ and $c$; or $\rho$ and $\theta$.
- For a given data set, we are to find the parameters that minimize the sum of the squared distances:
$$\sum_{i=1}^{N} d_i^2.$$
- This sum is our "objective function". There are other potential forms of the objective function.
- Formulating and then solving objective functions (through minimization) is a central theme to the computations of computer vision.

### Least-Squares Regression

For simplicity, we'll start with the slope/intercept form

- Objective function to minimize

$$E(m, b) = \sum_i (y_i - x_i m - b)^2$$

- We calculate the partial derivatives

$$\frac{\partial E}{\partial m} \qquad \text{and} \qquad \frac{\partial E}{\partial b}$$

- We then set each to 0 and solve for $m$ and $b$.
- This leads to the linear equation

$$\begin{pmatrix} \sum_i x_i^2 & \sum_i x_i \\ \sum_i x_i & N \end{pmatrix} \begin{pmatrix} m \\ b \end{pmatrix} = \begin{pmatrix} \sum_i x_i y_i \\ \sum_i y_i \end{pmatrix}$$

  which we then solve for $m$ and $b$.

### Orthogonal Least-Squares Regression

Now we turn to the more useful form, at least for fitting lines to edgel point locations.

- We'll form the objective function in terms of the polar coordinates of the closest point.

$$E(\rho, \theta) = \sum_i (x_i \cos\theta + y_i \sin\theta - \rho)^2.$$

  We want to minimize this objective function with respect to $\rho$ and $\theta$.

- Start by taking the derivative with respect to $\rho$:

$$\frac{\partial E}{\partial \rho} = -2 \sum_i (x_i \cos\theta + y_i \sin\theta - \rho)$$

- Setting this equal to 0 and solving shows that

$$\rho = \mu_x \cos\theta + \mu_y \sin\theta$$

  where

$$\mu_x = \sum_i x_i / N \qquad \text{and} \qquad \mu_y = \sum_i y_i / N$$

  form the "center of mass" of the data points.

- Substituting this expression for $\rho$ into our original objective function and making the substitution $u_i = x_i - \mu_x$ and $v_i = y_i - \mu_y$, the new objective function becomes

$$E(\theta) = \sum_i (u_i \cos\theta + v_i \sin\theta)^2.$$

- We will solve this for $\theta$ and then substitute the resulting value of $\theta$ into the expression for $\rho$ to obtain our final answer.
- Taking the derivative with respect to $\theta$, setting the result to 0, applying some well-known trigonometric formulas results in

$$\tan 2\theta = \frac{2\sum_i u_i v_i}{\sum_i (u_i^2 - v_i^2)}.$$

- Letting $\phi = 2\theta$, we have

$$\phi = \tan^{-1} \frac{2\sum_i u_i v_i}{\sum_i (u_i^2 - v_i^2)},$$

  which is really two solutions $\phi$ and $\phi + \pi$.
- Finally, we get two solutions for $\theta$

$$\theta = \phi/2 \qquad \text{and} \qquad \theta = \phi/2 + \pi/2.$$

- These correspond to the normal direction (minimum) and tangent direction (maximum) of the line. To choose between them, we must either apply the second derivative test, or simply substitute the two angles into the objective function in turn.

### Orthogonal Least-Squares Regression, Solution #2

A second solution can be obtained using the method of Lagrange multipliers, a method I certainly don't expect you to already know, but one that appears in a number of forms in computer vision problems

- Start using $a$ and $b$ instead of $\cos\theta$ and $\sin\theta$ in the above expression:

$$E(a, b) = \sum_i (u_i a + v_i b)^2.$$

- The immediate problem with this is that $a = b = 0$ produces $E = 0$, but we can't have $a = b = 0$.
- We address this by imposing the constraint that $a^2 + b^2 = 1$, which means that $(a, b)^\top$ is a unit vector.
- This is incorporated into a new objective function using a Lagrange multiplier, $\lambda$ to produce

$$F(a, b, \lambda) = \sum_i (u_i a + v_i b)^2 - \lambda(a^2 + b^2 - 1)$$

- Computing partial derivatives with respect to $a$, $b$ and $\lambda$ yields

$$\begin{aligned} \frac{\partial F}{\partial a} &= 2\textstyle\sum_i (u_i^2 a + u_i v_i b) - 2\lambda a \\ \frac{\partial F}{\partial b} &= 2\textstyle\sum_i (u_i v_i a + v_i^2 b) - 2\lambda b \\ \frac{\partial F}{\partial \lambda} &= a^2 + b^2 - 1 \end{aligned}$$

- Setting all three to 0 and rearranging yields (in matrix form)

$$\begin{pmatrix} \sum_i u_i^2 & \sum_i u_i v_i \\ \sum_i u_i v_i & \sum_i v_i^2 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \lambda \begin{pmatrix} a \\ b \end{pmatrix}$$

  and $a^2 + b^2 = 1$

- Stated another way, the line normal vector $(a, b)^\top$ is a unit eigenvector of the $2 \times 2$ matrix

$$\mathbf{M} = \begin{pmatrix} \sum_i u_i^2 & \sum_i u_i v_i \\ \sum_i u_i v_i & \sum_i v_i^2 \end{pmatrix}$$

- Just like the our previous solution where we directly estimated $\theta$, there are two solutions, producing vectors orthogonal to each other. The solution we want corresponds to the *smaller* eigenvalue of the "moment matrix" $\mathbf{M}$.

- We can solve for $\rho$ in the same manner as the previous solution. Recalling that $c = -\rho$, we have therefore solved the minimization problem for the implicit form of the line.

- This new solution generalizes much more easily and cleanly to higher dimensions.

### Back to Using a Set of Lines to Approximate a Contour

Recall that we are working with an edgel chain:

- We now know how to compute the point to line distances in finding the furthest point from the line:
  - Perhaps you already knew all of this, but now you are sure and you know why.
- We can apply least-squares estimation to obtain better line approximations.

### We Need To Do Better Than Approximating a Single Edgel Chain

- Looking at some simple images, we will see that just linking edgels into chains and then approximating the chains with lines does not produce the desired results.
  - Contours are broken and sometimes take surprising turns!
- Approaches to solving this:
  - Perceptual grouping, a big topic
  - Hough transformations
  - Robust estimation
- We will focus here on the Hough transform.

## Introduction to Hough Transforms

Our first example of the concept of "voting" in computer vision

- Each point location $(x, y)$ (each edge location) will vote for the set of lines that it could reasonably belong to.
- Votes are accumulated from all edge points:
  - This could be across an entire image or it could be in a region of the image.
- Parameters of lines with a large number of votes are likely to correspond to true lines in the image.

## Line Parameterization for the Hough Transform

Consider a point $(x, y)^\top$, and think about what lines it might belong to:

- Thinking of the "closest point" formulation, this is all lines $(\rho, \theta)$ such that
$$x \cos\theta + y \sin\theta - \rho = 0.$$
- Using this representation, $\rho$ is the distance form the origin of our coordinate system to the line.
- Now, for each possible value of $\theta$, there is one possible value of $\rho$:
$$\rho = x \cos\theta + y \sin\theta.$$
- The range of parameterizations we will follow assigns;
  - $\theta$ to be in the range $[-\pi/2, \pi/2)$ (or $[-90°, 90°)$),
  - implying that $\rho$ becomes a signed distance.

## Hough Transformation Voting

Form a 2d accumulator array

- The column (horizontal) axis will correspond to $\theta$ in the interval $[-\pi/2, \pi/2)$:
  - If we allow increments of $\Delta\theta$ (a parameter to the algorithm such as $2\pi/180$ radians), then we have
  $$k_\theta = \lceil \pi/\Delta\theta \rceil$$
  columns.
- The row (vertical) axis will correspond to $\rho$:
  - Move the coordinate system to the center of the image and normalize the coordinate system so that the domain is $[-1, 1] \times [-1, 1]$.
  - Then, the values of $\rho$ must cover the interval $[-\sqrt{2}, \sqrt{2}]$.
  - Using increments of $\Delta\rho$ (a parameter to the algorithm), we end up with
  $$k_\rho = \lceil 2\sqrt{2}/\Delta\rho \rceil$$
  rows.
- Overall, the accumulator array has $k_\theta k_\rho$ entries.

### Hough Transformation Voting Procedure

Let $A$ be the accumulator array:

1. For each edgel point location $(x_i, y_i)$
   (a) For each $j$ in $[0, \ldots, k_\theta)$,
      i. $\theta_j = j\Delta\theta$
      ii. $c = -(x_i \cos\theta_j + y_i \sin\theta_j)$.
      iii. $m = (c + \sqrt{2})/\Delta c$
      iv. $A(m, j) + +$.
2. Analyze the accumulator array to find peaks. These determine the lines in the image.

### Hough Examples

We will study the behavior of the Hough transform using the following Matlab functions

- `checkerboard`
- `edge`
- `hough`
- `houghpeaks`
- `houghlines`

### Details of and Variations on the Hough Transformation

- Ideas for handling discretization effects include smoothing the Hough array prior to peak extraction or having each point vote for multiple $\rho$ values for each $\theta_j$.
- In order to accurately estimate the parameters of a line, compute a final line estimate using a least-squares fit to the points that contribute to a peak.
- Use edgel gradient direction to narrow the voting domain for each point. This reduces the random noise in the bin votes.
- Could attempt hierarchical voting, where we start with large "bins" (large values of $\Delta\theta$ and $\Delta\rho$) and then "focus" the search around peaks.
- Implicit representation of the Hough array may be used where it is potentially quite sparse.

# 4 Summary and Review

## Techniques

- Edge detection involves smoothing and differentiation, followed by a number of additional operations: peak detection and non-maximum suppression, subpixel localization, linking, and (potentially adaptive) thresholding.
- Edge elements may be linked to into chains.
- Chains may be approximately by polylines and least-squares fitting lines.
- Hough transformations produce estimates of more global (image-wide) lines without the use of the linked chains.

## Conceptual / Mathematical View

- "Views" of an image
  - Function
  - Surface
  - Matrix / array
  - Set
- Mathematics:
  - Geometric entities (lines) may be represented algebraically in a number of ways, each with its advantages and disadvantages
  - Be aware of the difference between the number of parameters in an algebraic representation and the degrees of freedom of the represented geometric entity.
  - Constrained optimization problems may often be solved using Lagrange multipliers.
  - Least-squares estimation (regression and moment analysis) shows up in a wide variety of contexts.
  - Linear algebra techniques can make many of these problems/solutions more compactly described and therefore clearer and more effectively manipulated.
- Voting, sometimes in the form of a Hough transform, can be used to find points / values that "agree" with each other despite substantial clutter in images.

### Project Ideas

The Szeliski text includes a discussion of finding vanishing points and rectangles in images. We will soon be discussing vanishing points in the context of camera modeling and camera calibration. The project idea is to study vanishing point and rectangle estimate methods, choose one, implement it, and evaluate it carefully.