

Lecture 5: Feature Detection

CAP 5415: Computer Vision
Fall 2010

In the text

- Today's material roughly matches Chapter 4 of the text

Feature Detection

- Images are big
- Often not a lot going on at many pixels
- What if we only wanted to find a few salient points to look at?



(Image from Szeliski Text)

Feature Detection

- In addition, many important algorithms involve some kind of matching
- We want to find good points for matching



(Image from Szeliski Text)

Today's Lecture

- Feature Point Detection
 - We will use the task of finding matching points to motivate how points are selected
- Edge Detection

Finding Good Features

- Remember, our underlying goal is to find points that we can match easily



(Image from Szeliski Text)

Finding Good Features

- Would this be a good point to match?



(Image from Szeliski Text)

Finding Good Features

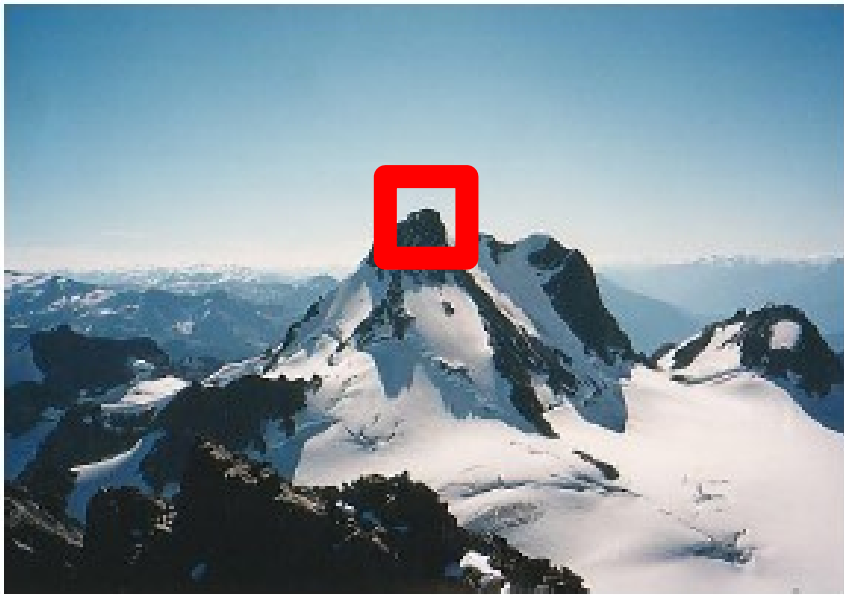
- This might be better, but what is the problem?



(Image from Szeliski Text)

Finding Good Features

- How about this point?



(Image from Szeliski Text)

Finding Good Features

- To formulate this mathematically, let's establish our criterion for matching patches
- We'll use the Sum-of-Squared Differences

$$E_{\text{WSSD}}(\mathbf{u}) = \sum_i w(\mathbf{x}_i) [I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)]^2$$

- If the SSD is low, the patches match well

But wait!

- We don't want to match patches, we want to find patches that will be good to match.
- So, we'll look at something related:
 - How well can the patch be matched to its neighbors?
- We'll compare a patch against patches in the same neighborhood

Finding Good Features

- How about this point?



Won't match well

Will match okay along line

Will match well in all directions

Mathematically

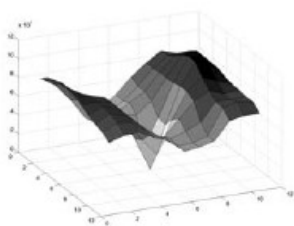
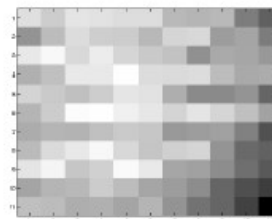
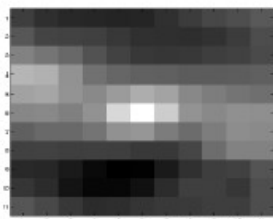
- We will compute the *auto-correlation surface*

$$E_{AC}(\Delta \mathbf{u}) = \sum_i w(\mathbf{x}_i) [I_0(\mathbf{x}_i + \Delta \mathbf{u}) - I_0(\mathbf{x}_i)]^2$$

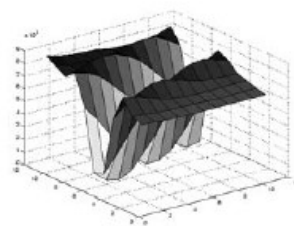
Example from Szeliski Text



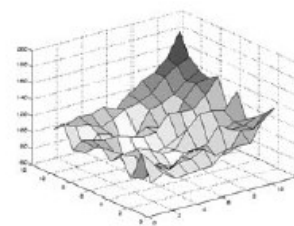
(a)



(b)



(c)



(d)

Doing this at every pixel will be slow!

- So, we will need to approximate it.
- Use a Taylor series

$$I_0(\mathbf{x}_i + \Delta \mathbf{u}) \approx I_0(\mathbf{x}_i) + \nabla I_0(\mathbf{x}_i) \cdot \Delta \mathbf{u}$$

Now the auto-correlation surface becomes

$$\begin{aligned} E_{\text{AC}}(\Delta \mathbf{u}) &= \sum_i w(\mathbf{x}_i) [I_0(\mathbf{x}_i + \Delta \mathbf{u}) - I_0(\mathbf{x}_i)]^2 \\ &\approx \sum_i w(\mathbf{x}_i) [I_0(\mathbf{x}_i) + \nabla I_0(\mathbf{x}_i) \cdot \Delta \mathbf{u} - I_0(\mathbf{x}_i)]^2 \\ &= \sum_i w(\mathbf{x}_i) [\nabla I_0(\mathbf{x}_i) \cdot \Delta \mathbf{u}]^2 \\ &= \Delta \mathbf{u}^T \mathbf{A} \Delta \mathbf{u}, \end{aligned}$$


$$\nabla I_0(\mathbf{x}_i) = \left(\frac{\partial I_0}{\partial x}, \frac{\partial I_0}{\partial y} \right)(\mathbf{x}_i) \quad \mathbf{A} = w * \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

What is this?

$$\nabla I_0(\mathbf{x}_i) = \left(\frac{\partial I_0}{\partial x}, \frac{\partial I_0}{\partial y} \right)(\mathbf{x}_i)$$

- This is the image gradient

This is a blurring that acts as the summation over the window

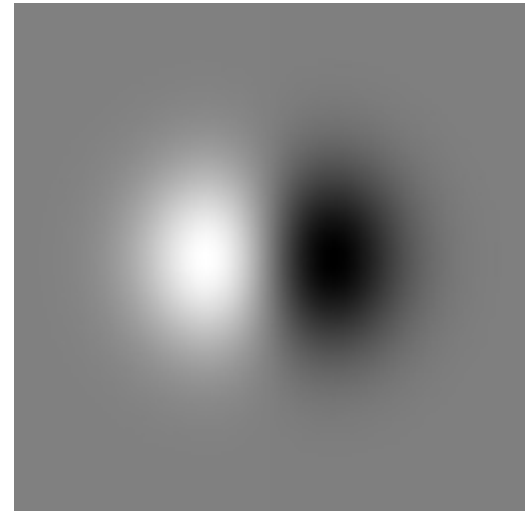

$$\mathbf{A} = w * \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

- This *autocorrelation matrix* depends on image derivatives also

How do we compute image gradients?

- Most commonly used method is the Derivative of Gaussian filter
 - (Remember Lecture 1?)

$$\frac{\partial K(i, j)}{\partial i} = \frac{-i}{\sigma^2 Z} \exp \left(-\frac{i^2 + j^2}{2\sigma^2} \right)$$



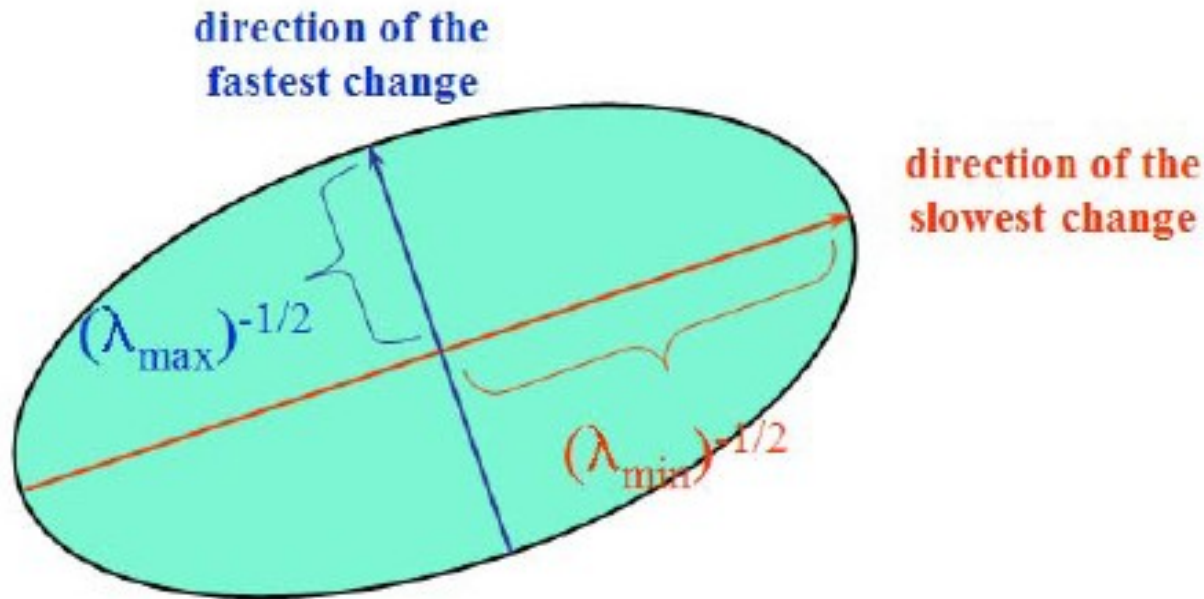
Now, we need a number to tell us if this is a good point

- “Harris Corner Detector”

$$\det(\mathbf{A}) - \alpha \operatorname{trace}(\mathbf{A})^2 = \lambda_0 \lambda_1 - \alpha (\lambda_0 + \lambda_1)^2$$

- Shi-Tomasi
 - Minimum Eigenvalue

Why the Minimum Eigenvalue?



(Figure from Szeliski Text)

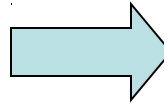
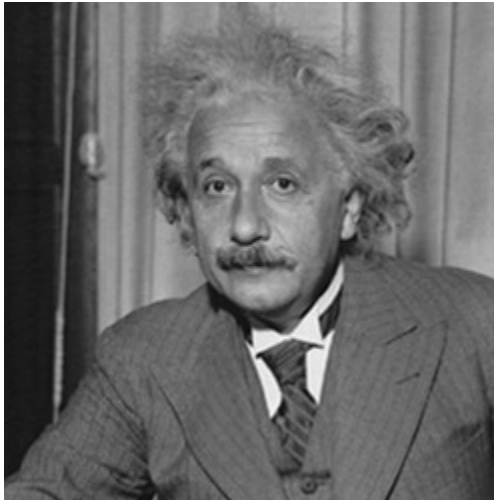
Steps

- 1) Compute horizontal and vertical derivatives of the image (Convolve with DoG filters)
- 2) Compute three images: I_x^2 , I_y^2 , $I_x I_y$
- 3) Convolve these images with a Gaussian
- 4) Computer numerical measure
- 5) Find points above the threshold
- 6) Non-maximal Suppression (Will talk about this in a minute)

Note

- These feature detectors are not just used where matching is important
- They are also

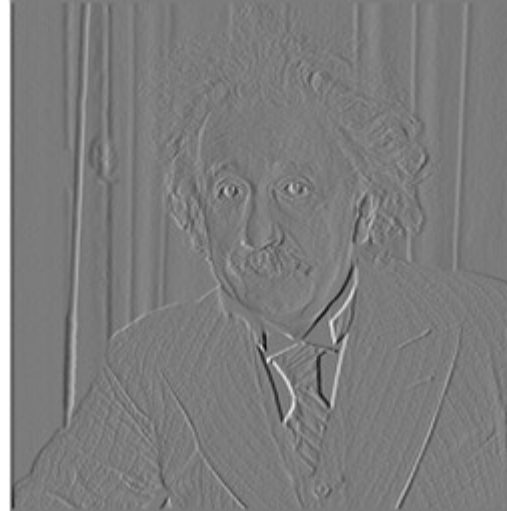
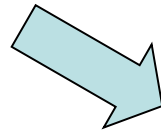
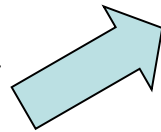
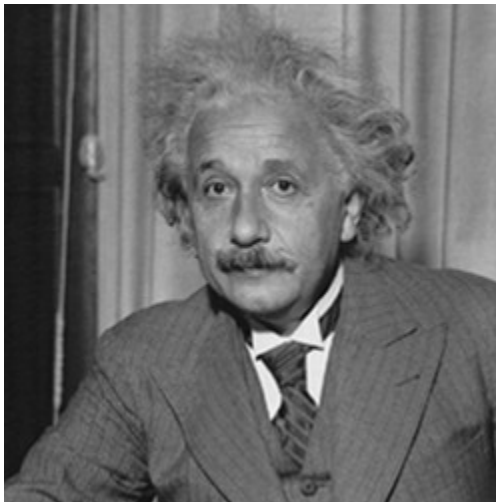
Edge Detection



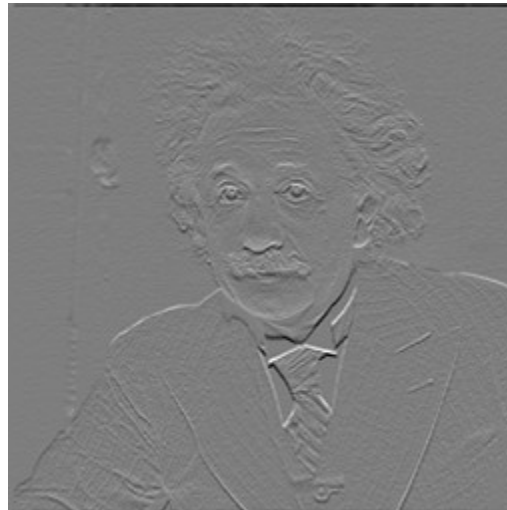
“Nonetheless, experience building vision systems suggests that very often, interesting things are happening in an image at an edge and it is worth knowing where the edges are.” – Forsyth and Ponce in “Computer Vision- A Modern Approach

Simple Strategy

- First compute gradients



f_x

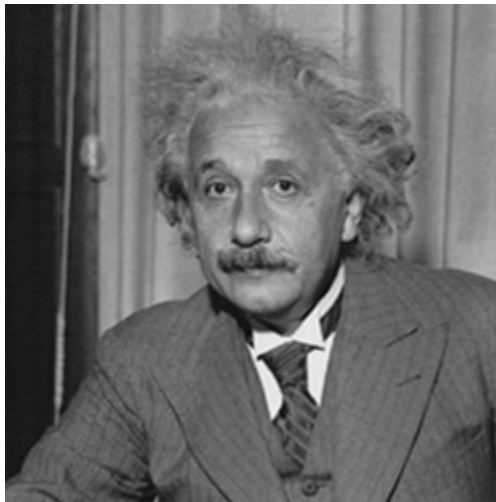


f_y

Simplest Strategy

- Use these gradients to calculate the magnitude of the image gradient

$$M(x, y) = \sqrt{f_x^2(x, y) + f_y^2(x, y)}$$

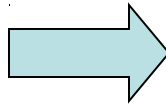


Simplest Strategy

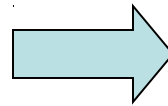
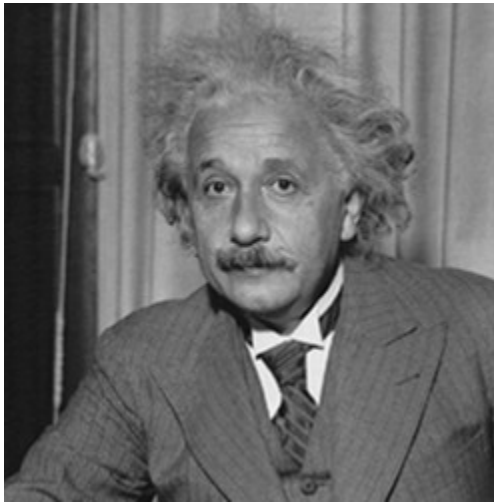
- Calculate the magnitude of the image gradient

$$M(x, y) = \sqrt{f_x^2(x, y) + f_y^2(x, y)}$$

- Compare against a threshold



Problems

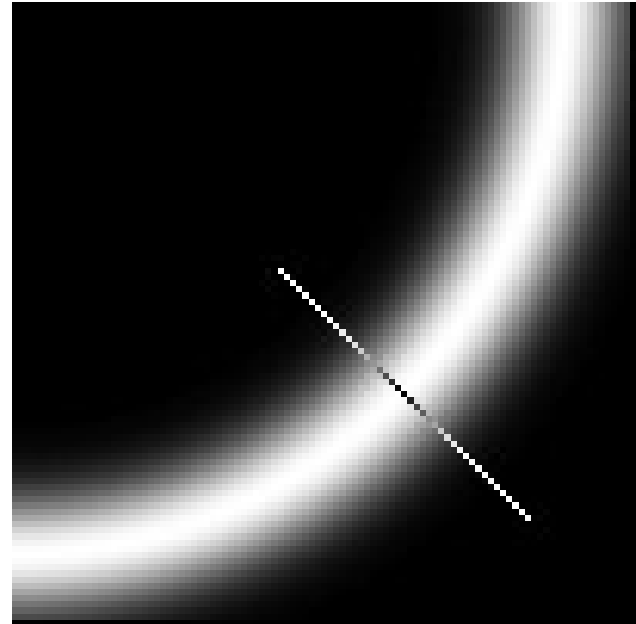
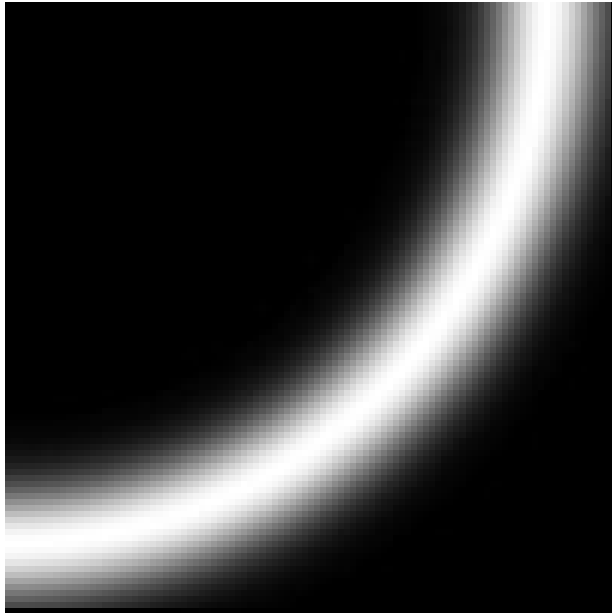


- We get thick edges
- Redundant, especially if we going to be searching in places where edges are found

Solution

- Identify the local maximums
- Called “non-maximal suppression”

Basic Idea

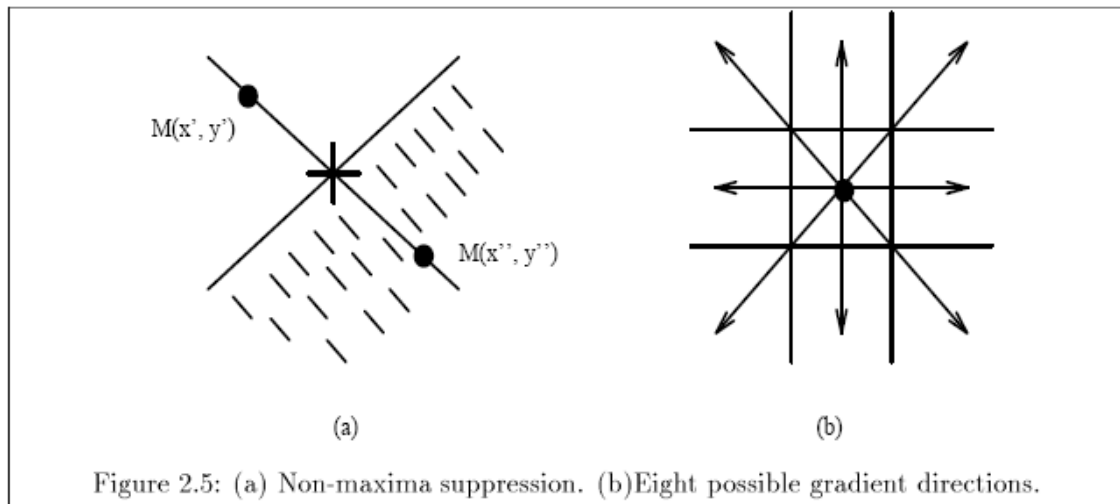


- The gradient will be perpendicular to the edge
- Search along the gradient for the maximum point

(From Slides by Forsyth)

Implementation

- Quantize the gradient orientation to a fixed number of orientations
- Search along those lines in a window around points above a threshold



(From Course Notes by Shah)

Comparison



Gradient Thresholding



With non-maximal suppression

Avoiding Non-Maximal Suppression

- Can we formulate the problem so we can eliminate non-maximal suppression?
- Remember: we want to find the maxima and minima of the gradient
- Look for places where 2nd derivative is zero
- Called zero-crossings (you may not see the 2nd derivative actually be zero)

Finding Zero-Crossings

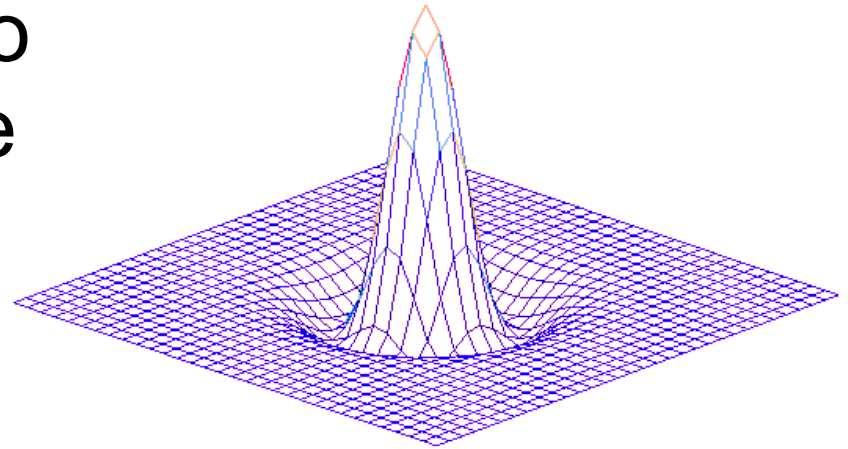
- The second derivative is directional, but we can use the *Laplacian* which is rotationally invariant

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

- The image is usually smoothed first to eliminate noise
- Computing Laplacian after filtering with a Gaussian is equivalent to convolving with a Laplacian of a Gaussian

The Laplacian of Gaussian

- Sometimes called a center-surround filter
- Response is similar to the response of some neurons in the visual cortex

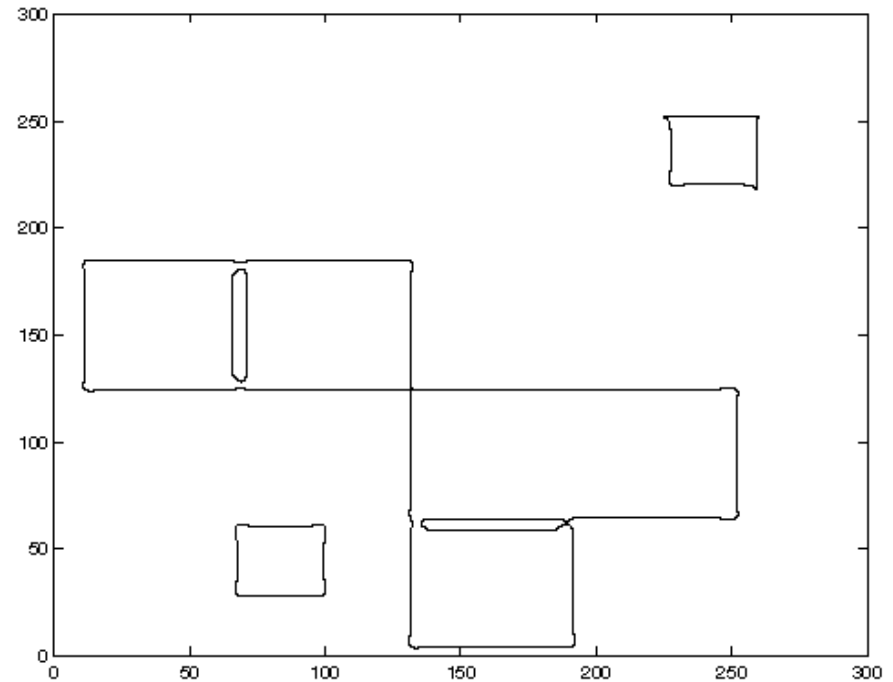


Implementation

- Filter with Laplacian of Gaussian
- Look for patterns like $\{+, 0, -\}$ or $\{+, -\}$
- Also known as the Marr-Hildreth Edge Detector

Disadvantages

- Behaves poorly at corners



(From Slides by Forsyth)

Noise and Edge Detection

- Noise is a bad thing for edge-detection
 - Usually assume that noise is white Gaussian noise (not likely in reality!)
- Introduces many spurious edges
- Low-pass filtering is a simple way of reducing the noise
 - For the Laplacian of Gaussian Method, it is integrated into the edge detection

Why does filtering with a Gaussian reduce noise?

- Forsyth and Ponce's explanation:

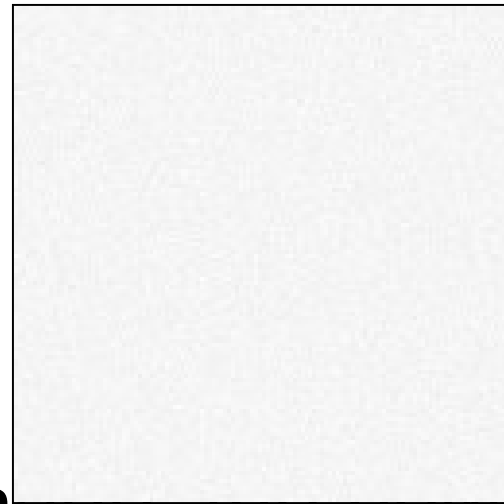
“Secondly, pixels will have a greater tendency to look like neighbouring pixels — if we take stationary additive Gaussian noise, and smooth it, the pixel values of the resulting signal are no longer independent. In some sense, this is what smoothing was about — recall we introduced smoothing as a method to predict a pixel's value from the values of its neighbours. However, if pixels tend to look like their neighbours, then derivatives must be smaller (because they measure the tendency of pixels to look different from their neighbours).”

– from Computer Vision – A Modern Approach

How I like to see it

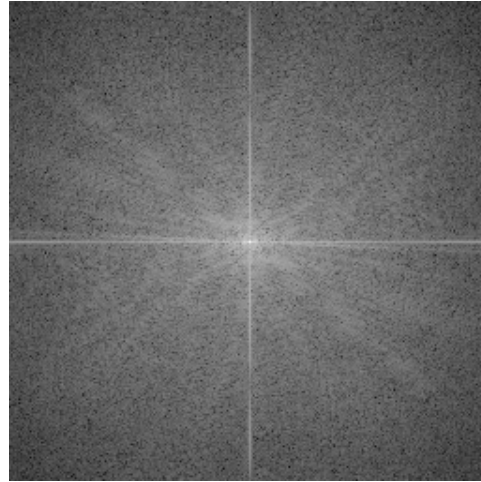
- If F is the DFT of an image, then $|F|^2$ is the power at each frequency

```
for i=1:10000
    cimg=randn(128);
    cum=cum+abs(fft2(cimg)).^2;
end
```



- In the average power image, the power is equal at every frequency

Remember: Energy in Images is concentrated in low-frequencies

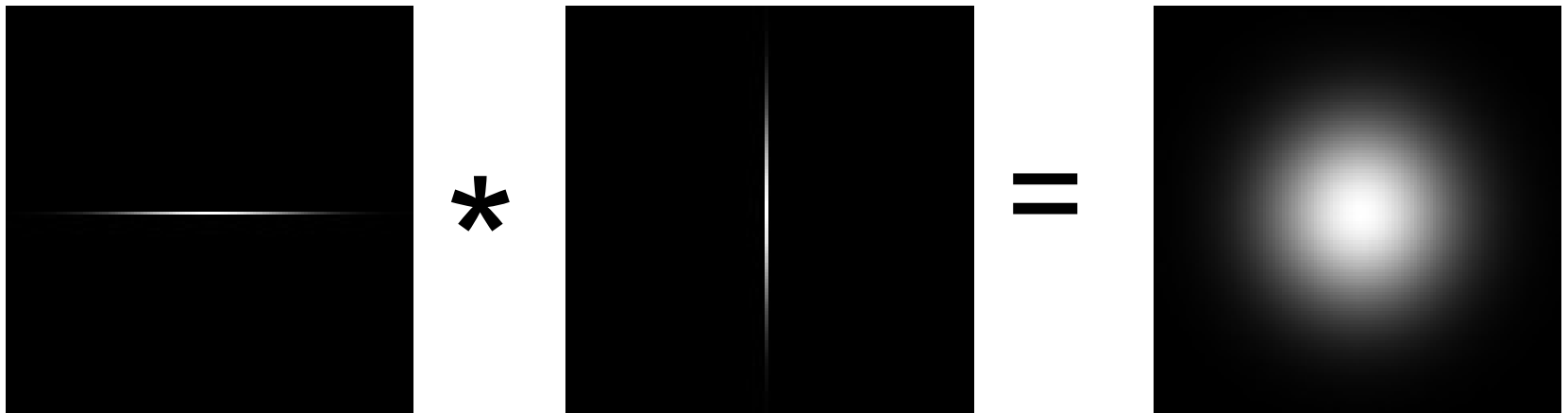


Log-Spectrum of the Einstein Image

- When you low-pass filter, you're retaining the relatively important low-frequencies and eliminating the noise in the high frequencies, where you don't expect much image energy

Smoothing Fast

- The Gaussian Filter is a separable filter



- The 2D filter can be expressed as a series of 2 1D convolutions
- For a 9x9 filter, a separable filter requires 18 computations versus 81 for the 2D filter

Further Improvements to the Gradient Edge-Detector

- Fundamental Steps in Gradient-Based Edge Detection that we have talked about:
 - Smooth to reduce noise
 - Compute Gradients
 - Do Non-maximal suppression
- There is one more heuristic that we can advantage of.
- Edges tend to be continuous

Hysteresis Thresholding

- Edges tend to be continuous
- Still threshold the gradient
- Use a lower threshold if a neighboring point is an edge
- The “Canny Edge Detector” uses all of these heuristics

Is there better information than the gradient?

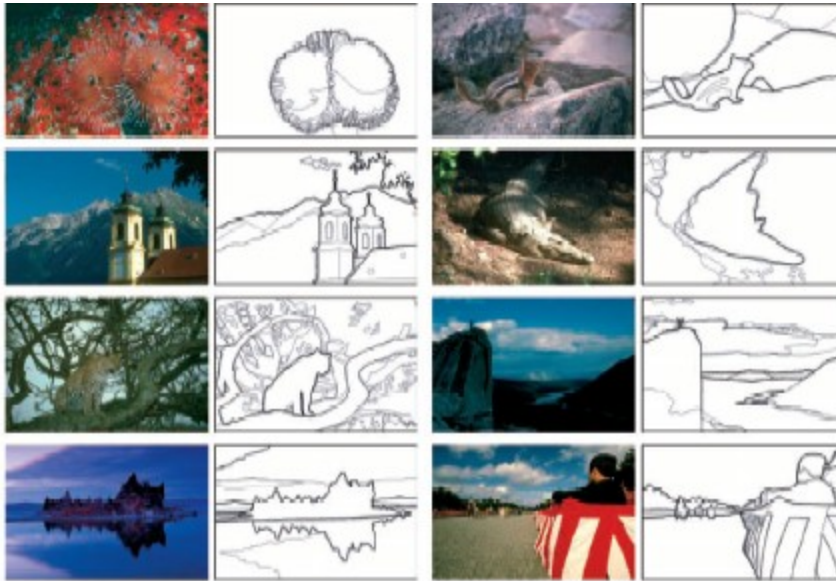
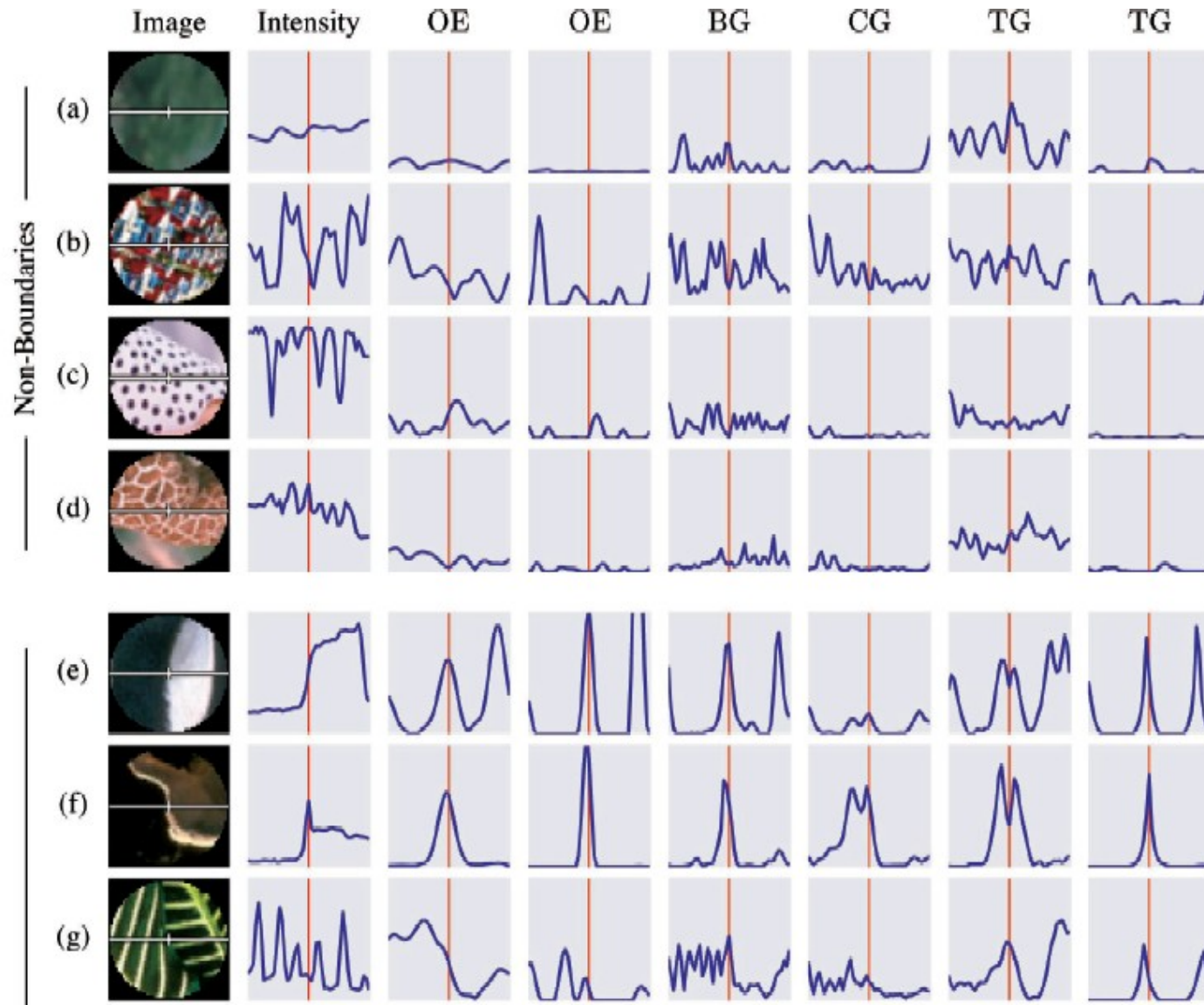


Fig. 1. Example images and human-marked segment boundaries. Each image shows multiple (4-8) human segmentations. The pixels are darker where more humans marked a boundary. Details of how this ground-truth data was collected are discussed in Section 3.

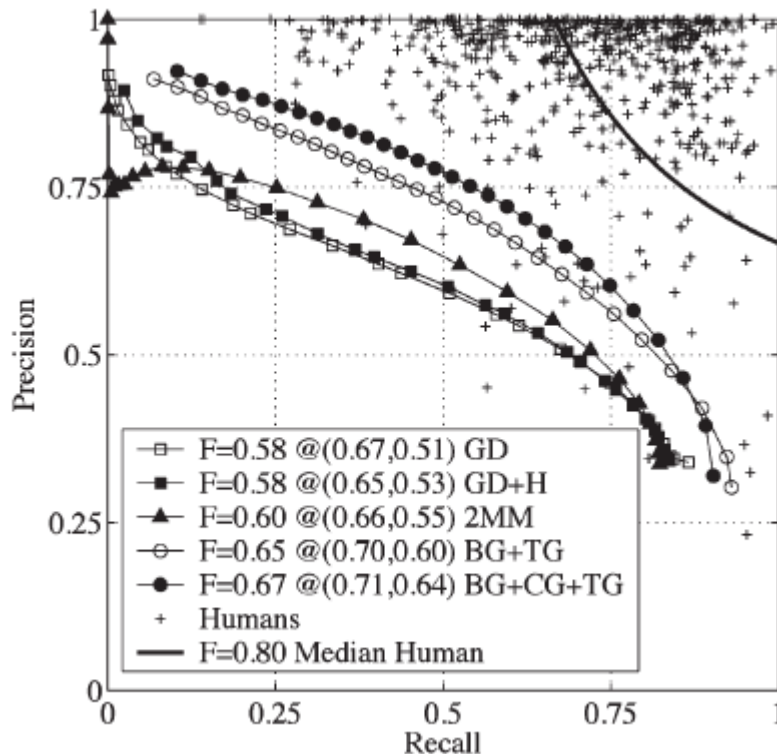
- Martin, Fowlkes, and Malik gathered a set of human segmentations

From Martin, Fowlkes, and Malik (2004)

Capturing Texture



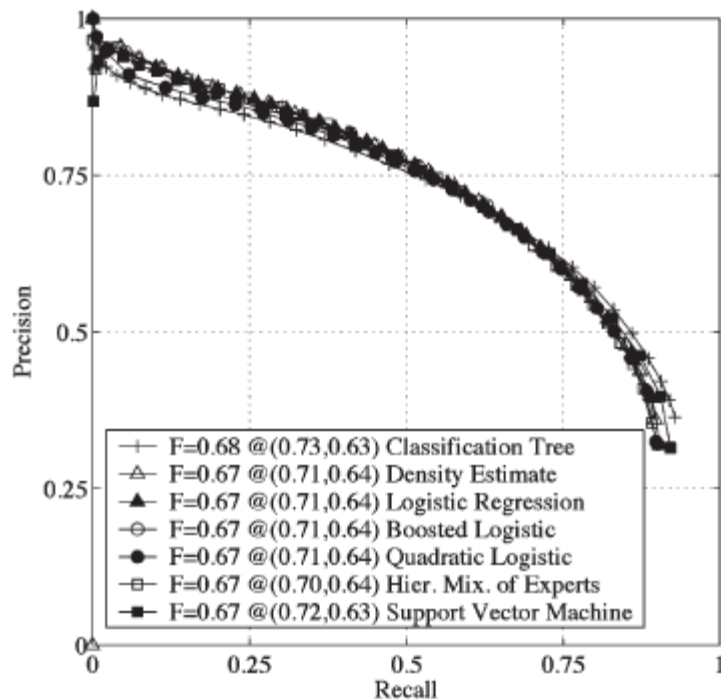
Texture and Color Help



- This is a precision-recall curve
- The highest curve has the best performance
- The best performance comes from using brightness (Image gradient), texture and color

Interesting Side-note

- The edge-detector was a classifier
- The type of classifier didn't matter for this task



- Next Time:
 - We'll start talking about learning to classify things like edges