CMP717
Image Processing

Linear filtering, Edge Detection, Boundary Detection, Segmentation

Erkut Erdem
Hacettepe University
Computer Vision Lab (HUCVL)

Today

- Linear Filtering
  - Gauss filter, Linear diffusion
- Edge Detection
  - Derivative filters, Laplacian of Gaussian, Canny edge detector
- Boundary Detection
- Segmentation
  - K-means clustering, Graph-theoretic segmentation

Today

- Linear Filtering
  - Gauss filter, Linear diffusion
- Edge Detection
  - Derivative filters, Laplacian of Gaussian, Canny edge detector
- Boundary Detection
- Segmentation
  - K-means clustering, Graph-theoretic segmentation

Filtering

- The name “filter” is borrowed from frequency domain processing
- Accept or reject certain frequency components
- <u>Fourier (1807):</u>
  Periodic functions could be represented as a weighted sum of sines and cosines

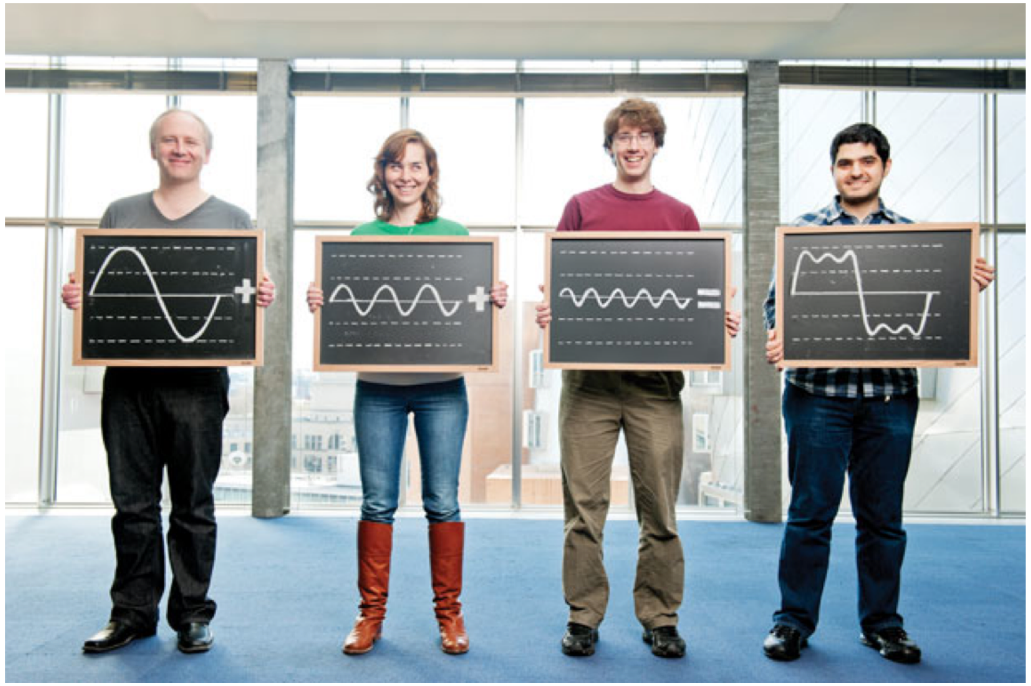
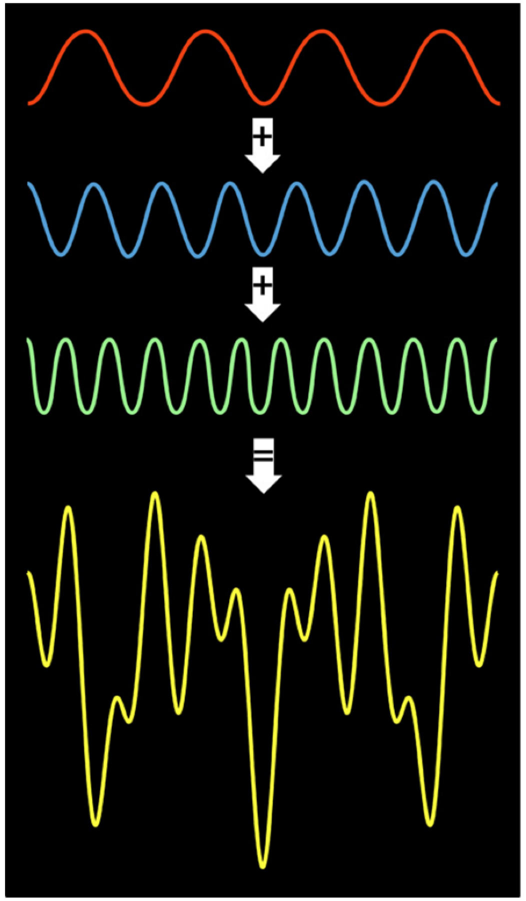


Image courtesy of Technology Review

Signals

- A signal is composed of low and high frequency components



low frequency components: smooth /
piecewise smooth

Neighboring pixels have similar brightness values

You're within a region

high frequency components: oscillatory

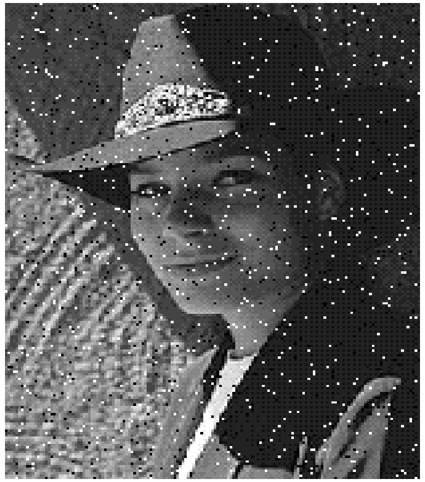
Neighboring pixels have different brightness values

You're either at the edges or noise points

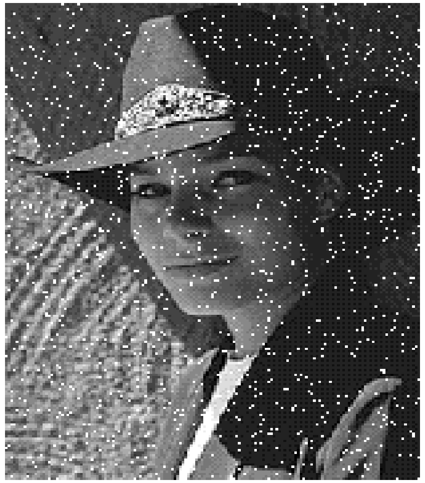
Common Noise Types

- Salt and pepper noise: random occurrences of black and white pixels
- Impulse noise: random occurrences of white pixels
- Gaussian noise: variations in intensity drawn from a Gaussian normal distribution

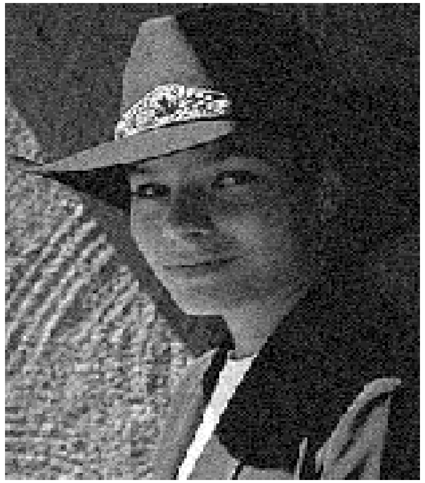
Original



Salt and pepper noise



Impulse noise



Gaussian noise

Slide credit: S. Seitz

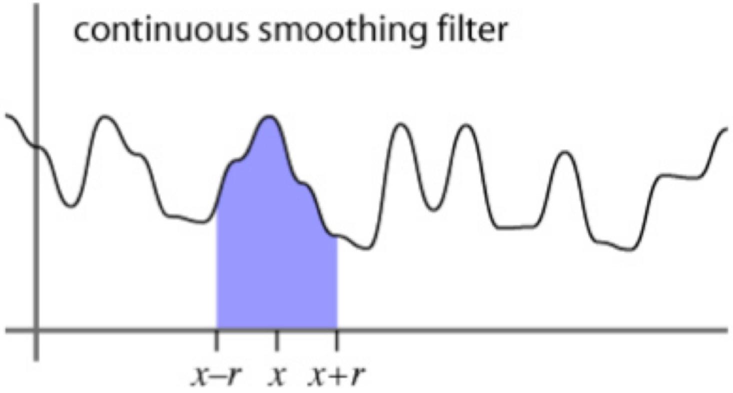
Image Filtering

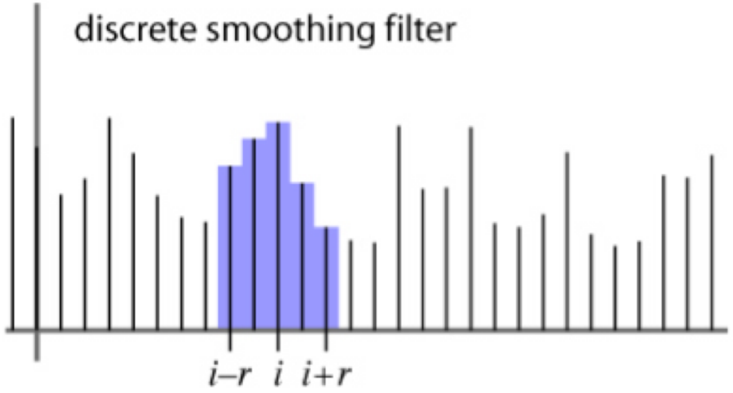
- Idea: Use the information coming from the neighboring pixels for processing
- Design a transformation function of the local neighborhood at each pixel in the image
  - Function specified by a “filter” or mask saying how to combine values from neighbors.
- Various uses of filtering:
  - Enhance an image (denoise, resize, etc)
  - Extract information (texture, edges, etc)
  - Detect patterns (template matching)

Adapted from: K. Grauman

Filtering

- Processing done on a function
  - can be executed in continuous form (e.g. analog circuit)
  - but can also be executed using sampled representation
- Simple example: smoothing by averaging
- Can be modeled mathematically by <u>convolution</u>





Slide credit: S. Marschner

Discrete convolution

- Simple averaging:

$$b_{\mathrm{smooth}}[i] = \frac{1}{2r+1} \sum_{j=i-r}^{i+r} b[j]$$

– every sample gets the same weight

- Convolution: same idea but with <u>weighted</u> average

$$(a \star b)[i] = \sum_j a[j]b[i-j]$$

– each sample gets its own weight (normally zero far away)

- This is all convolution is: it is a <u>moving weighted average</u>

Slide credit: S. Marschner

Filters

- Sequence of weights $a[j]$ is called a *filter*
- Filter is nonzero over its *region of support*

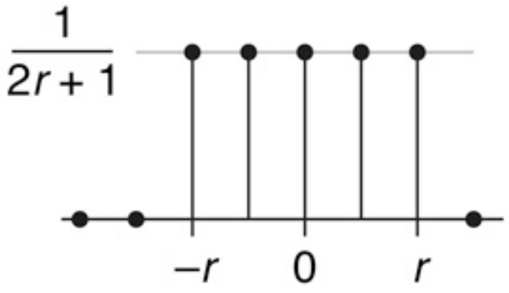
– usually centered on zero: support radius *r*

- Filter is *normalized* so that it sums to 1.0

– this makes for a weighted average, not just any old weighted sum

- Most filters are symmetric about 0

– since for images we usually want to treat left and right the same

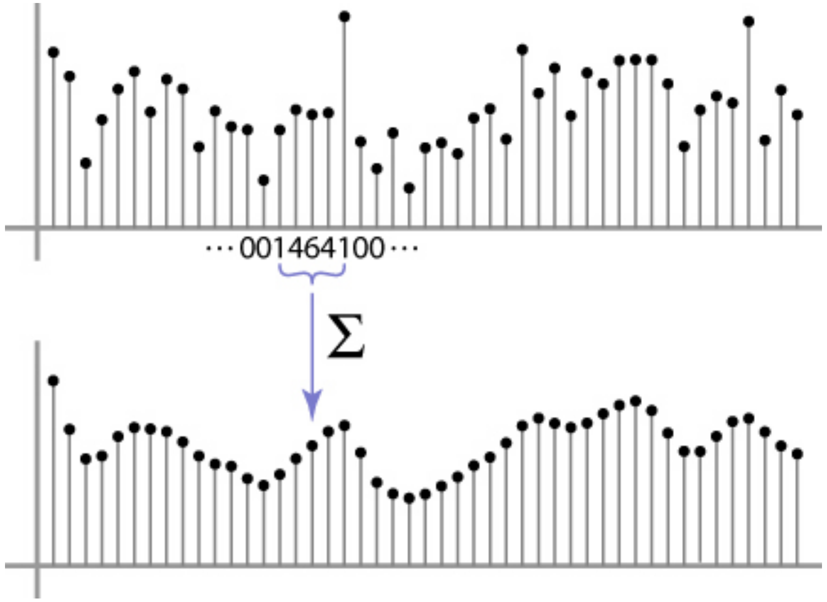


a box filter

Slide credit: S. Marschner

Convolution and filtering

- Convolution applies with any sequence of weights
- Example: bell curve (gaussian-like) [..., 1, 4, 6, 4, 1, ...]/16



Slide credit: S. Marschner

Discrete filtering in 2D

- Same equation, one more index

$$(a \star b)[i, j] = \sum_{i', j'} a[i', j']b[i - i', j - j']$$

– now the filter is a rectangle you slide around over a grid of numbers

- Usefulness of associativity

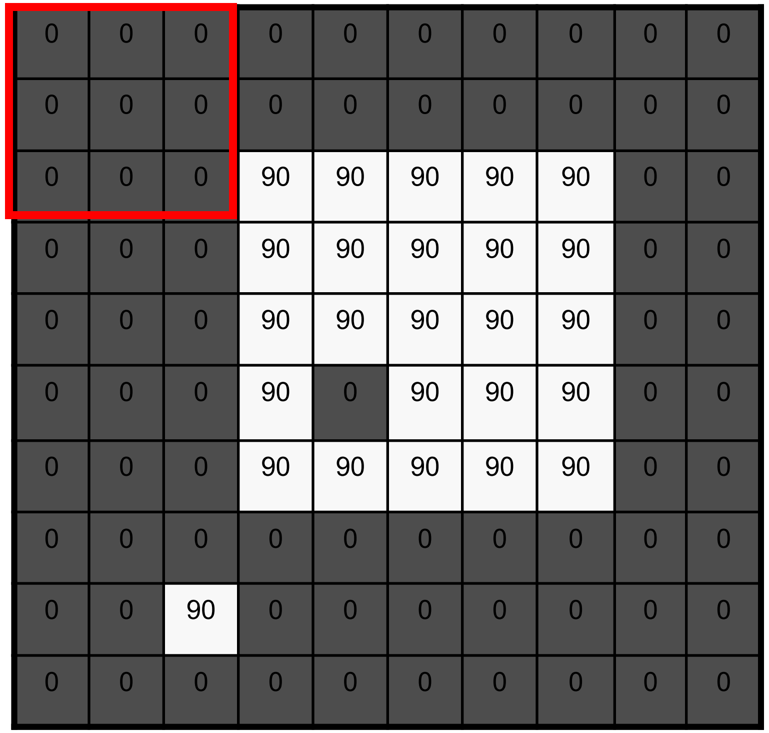
– often apply several filters one after another: (((a * b_1) * b_2) * b_3)

– this is equivalent to applying one filter: a * (b_1 * b_2 * b_3)

Slide credit: S. Marschner

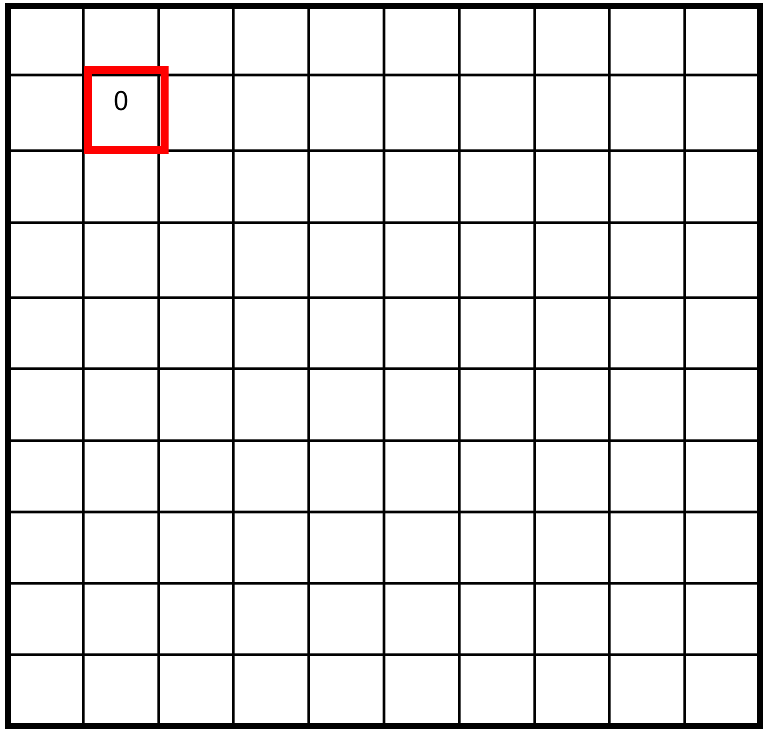
Moving Average In 2D

$F[x, y]$



0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

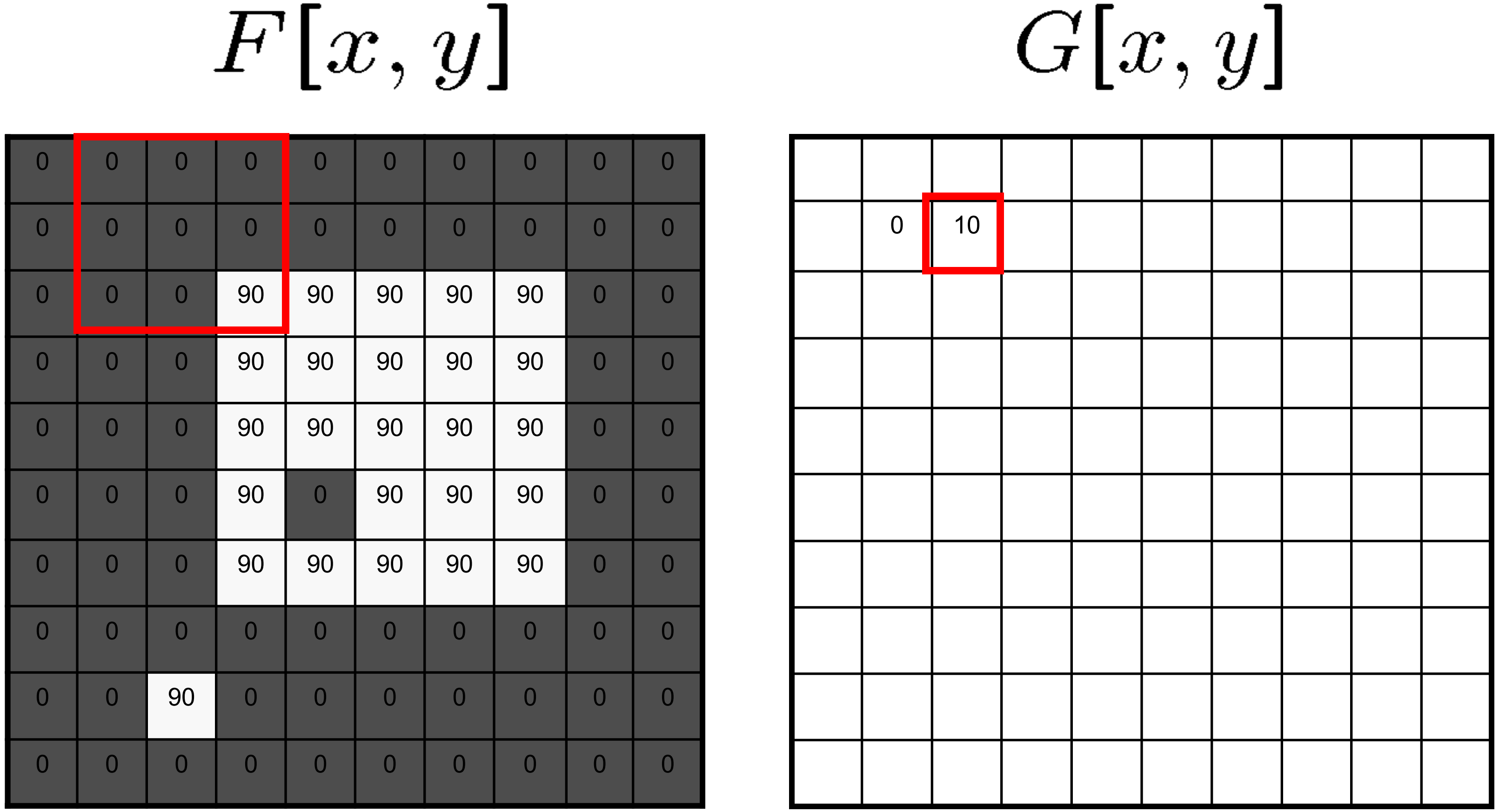
$G[x, y]$



	0								

Slide credit: S. Seitz

Moving Average In 2D



$F[x, y]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$G[x, y]$

	0	10							

Slide credit: S. Seitz

Moving Average In 2D

$$F[x, y]$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

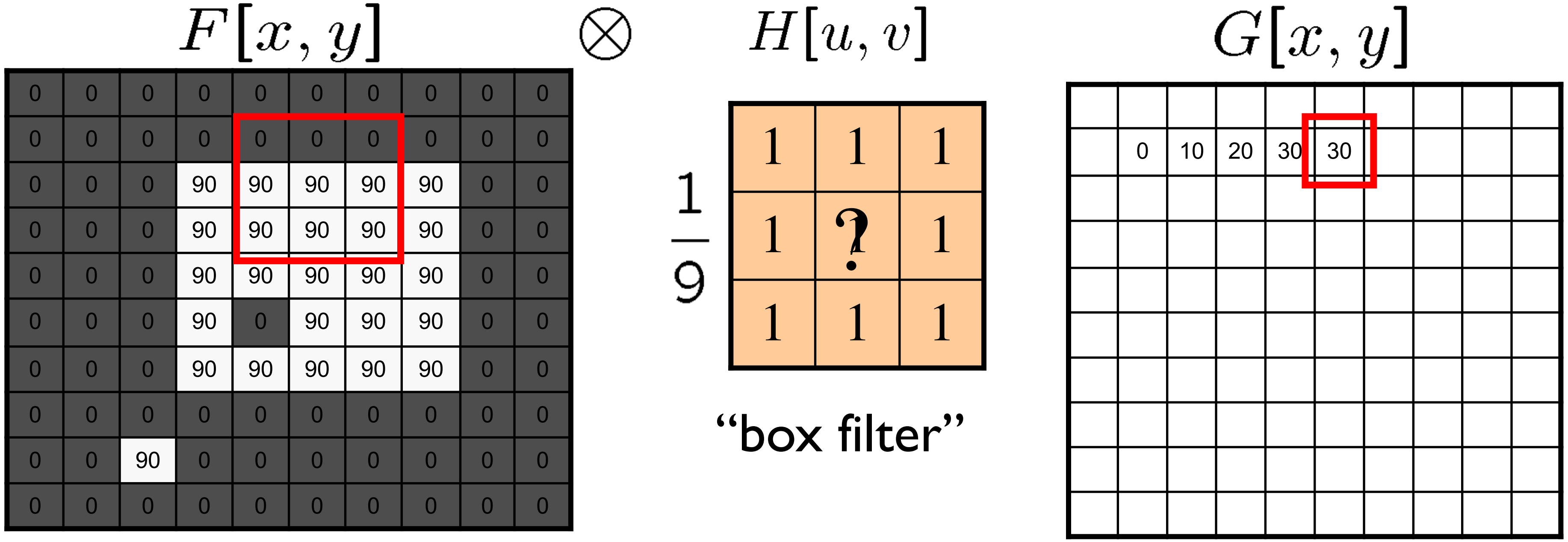
$$G[x, y]$$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

Slide credit: S. Seitz

Averaging filter

- What values belong in the kernel *H* for the moving average example?



$F[x, y]$ $\otimes$ $H[u, v]$ $G[x, y]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & ? & 1\\ 1 & 1 & 1\end{bmatrix}$$

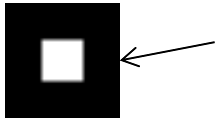
"box filter"

	0	10	20	30	30				

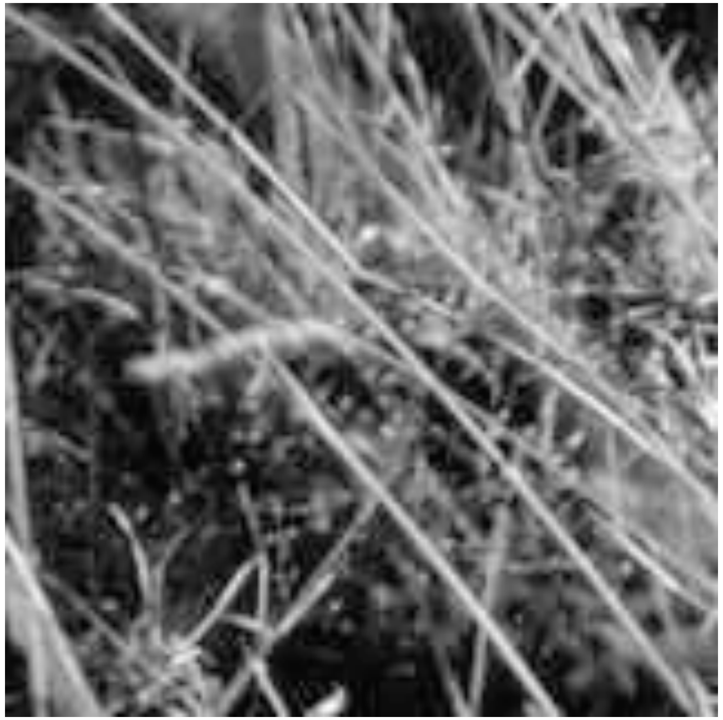
$$G = H \otimes F$$

Slide credit: K. Grauman

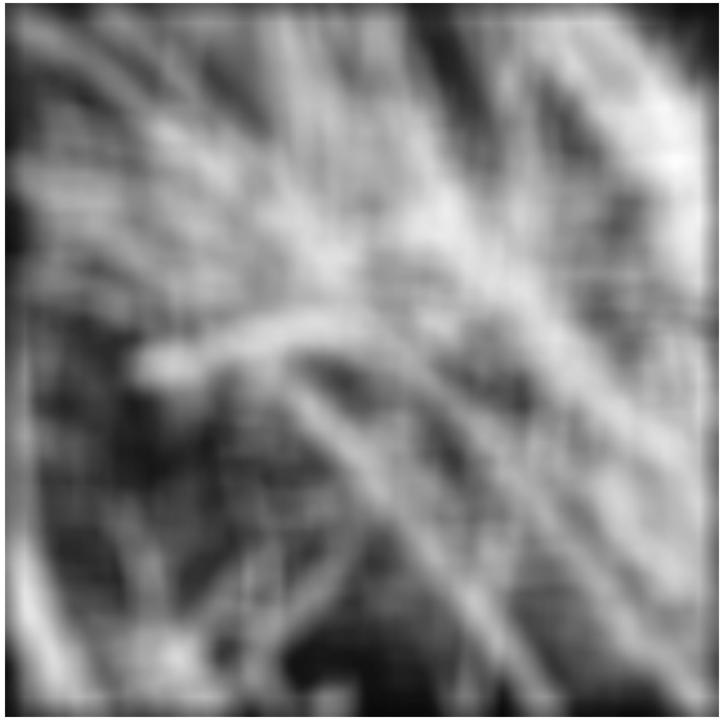
Smoothing by averaging



depicts box filter:
white = high value, black = low value



original

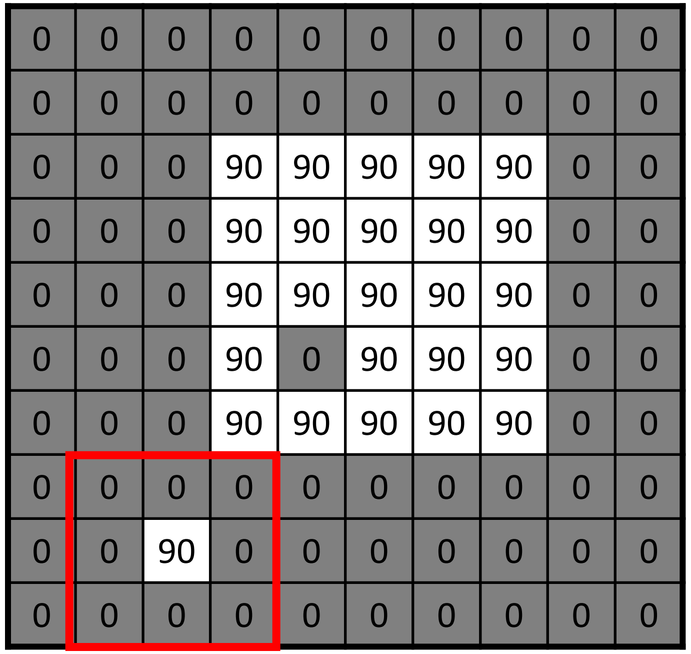


filtered

Slide credit: K. Grauman

Gaussian filter

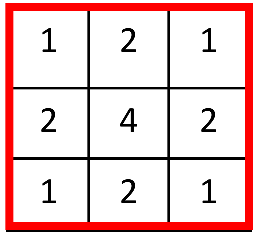
- What if we want nearest neighboring pixels to have the most influence on the output?



0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$F[x, y]$

$\frac{1}{16}$

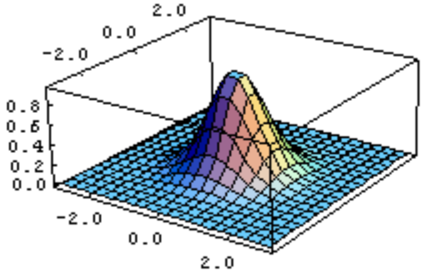


1	2	1
2	4	2
1	2	1

$H[u, v]$

This kernel is an approximation of a 2d Gaussian function:

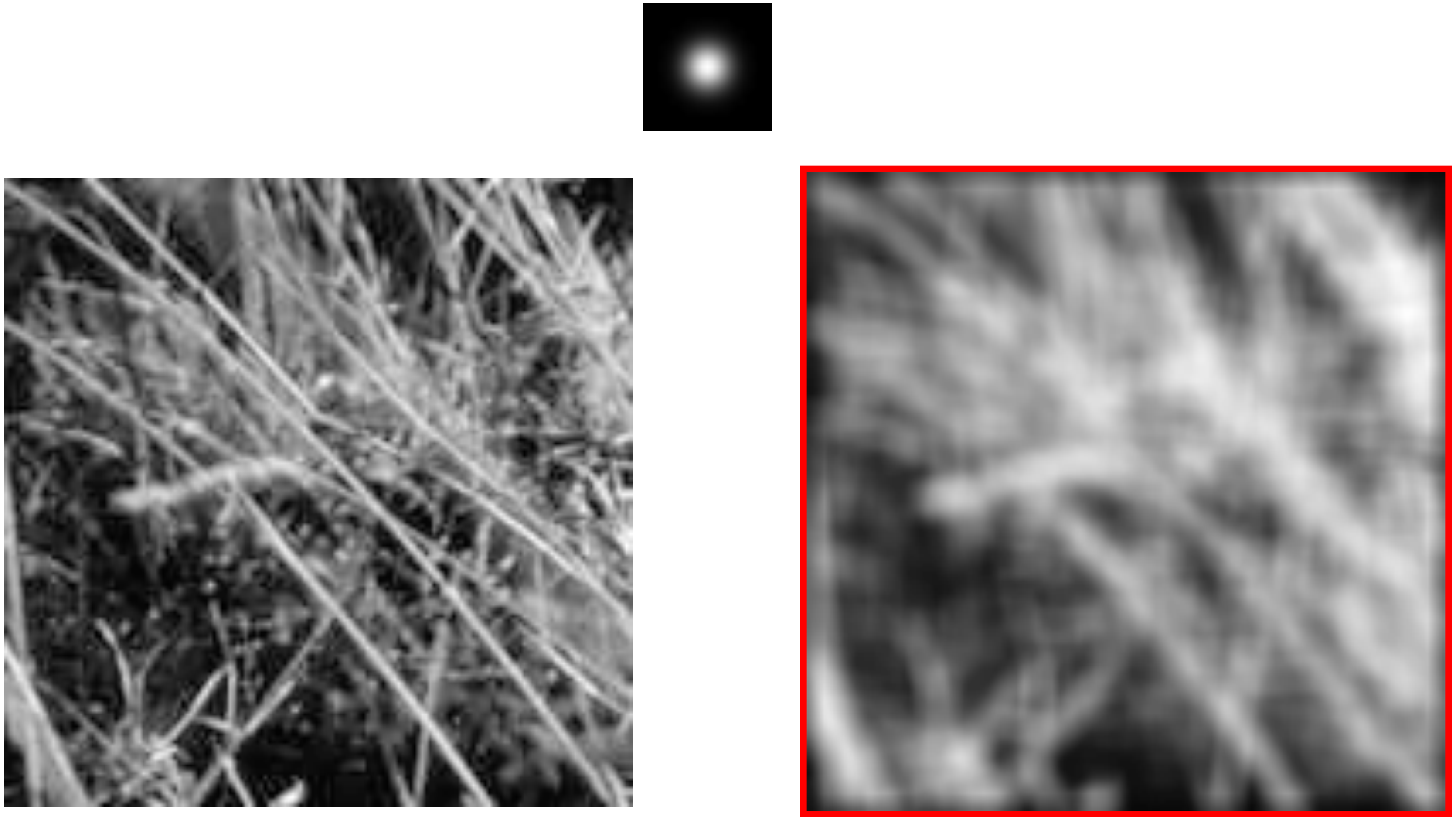
$$h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$



- Removes high-frequency components from the image (“low-pass filter”).

Slide credit: S. Seitz

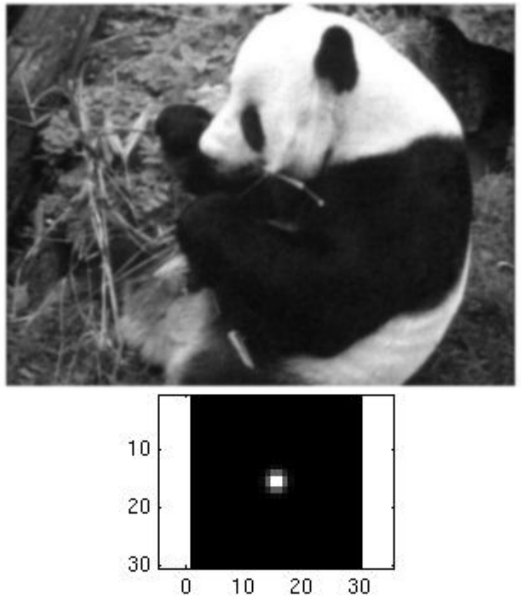
Smoothing with a Gaussian

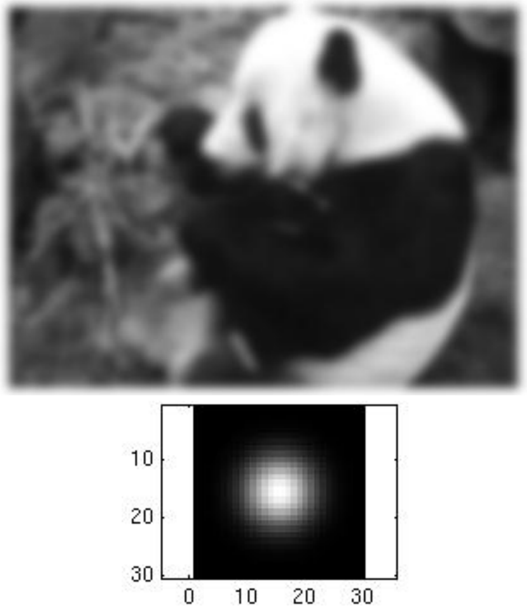


Slide credit: K. Grauman

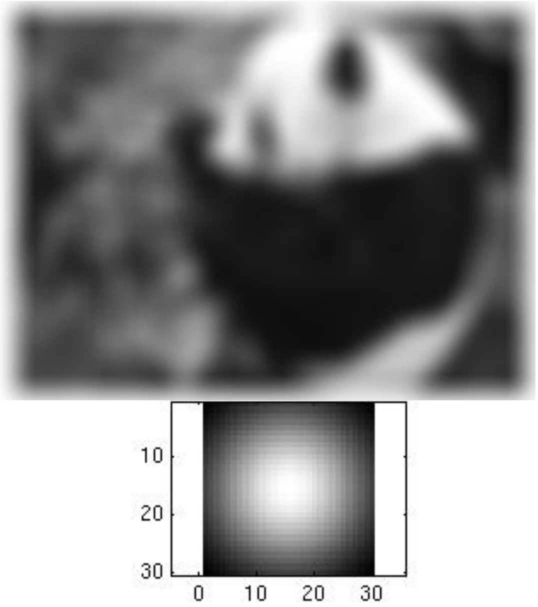
Smoothing with a Gaussian

Parameter σ is the “scale” / “width” / “spread” of the Gaussian kernel, and controls the amount of smoothing.





. . .



```
for sigma=1:3:10
   h = fspecial('gaussian‘, fsize, sigma);
   out = imfilter(im, h);
   imshow(out);
   pause;
end
```

Slide credit: K. Grauman

Linear Diffusion

- Let *f* (*x*) denote a grayscale (noisy) input image and *u*(*x*, *t*) be initialized with $u(x,0) = u^0(x) = f(x)$.
- The linear diffusion process can be defined by the equation:

$$\frac{\partial u}{\partial t} = \nabla \cdot (\nabla u) = \nabla^2 u$$

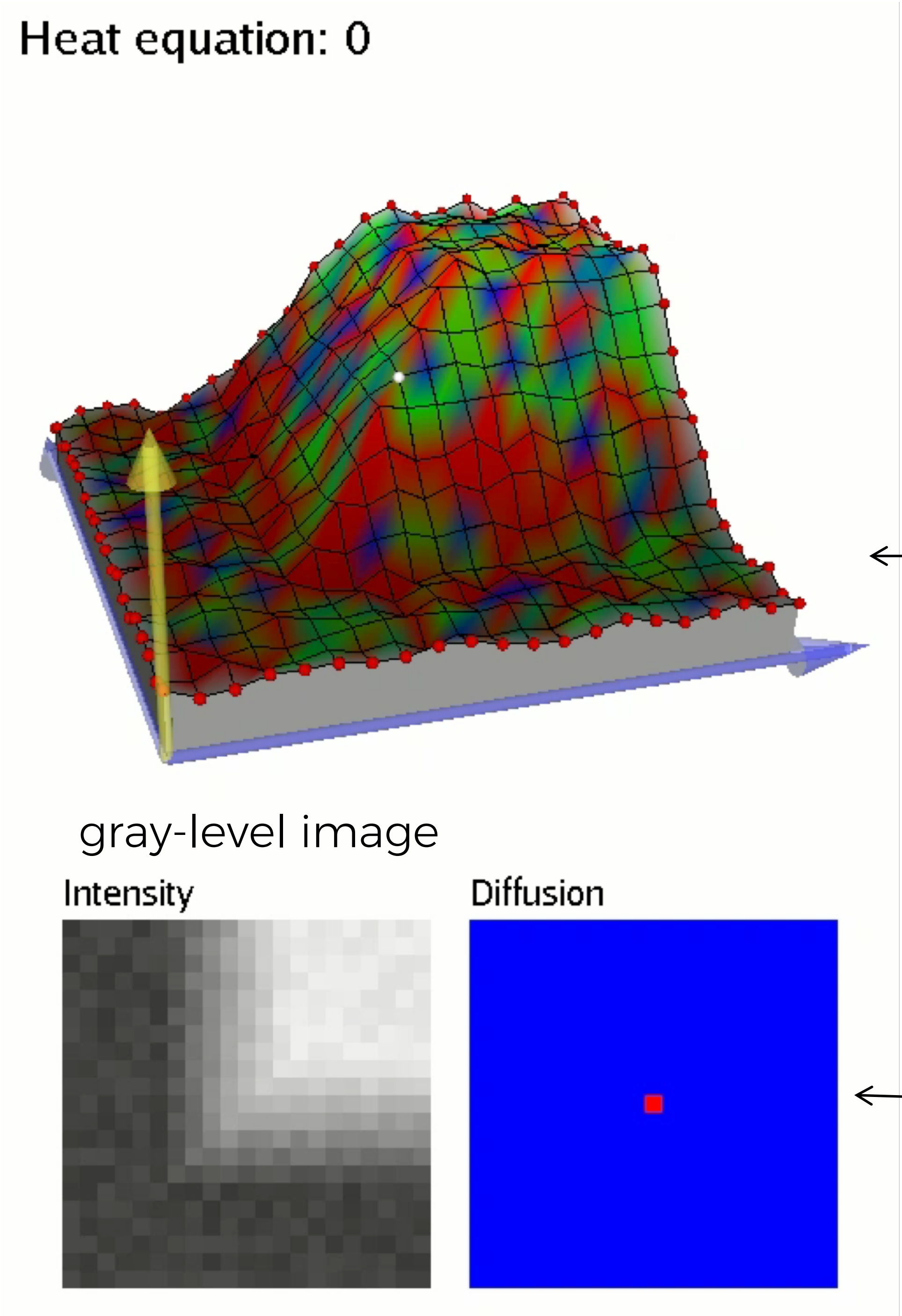
where ∇· denotes the divergence operator. Thus,

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

Linear Diffusion (cont'd.)

- Diffusion process as an evolution process.
- Artificial time variable t denotes the *diffusion time*
- Input image is smoothed at a constant rate in all directions.
  - $u^0(x)$: initial image,
  - $u(x, t)$: the evolving images under the governed equation representing the successively smoothed versions of the initial input image $f(x)$.
- Diffusion process creates a *scale space* representation of the given image f, with $t > 0$ being the scale.

Linear Diffusion (cont'd.)



$$\frac{\partial u}{\partial t} = \nabla \cdot (\nabla u) = \nabla^2 u$$

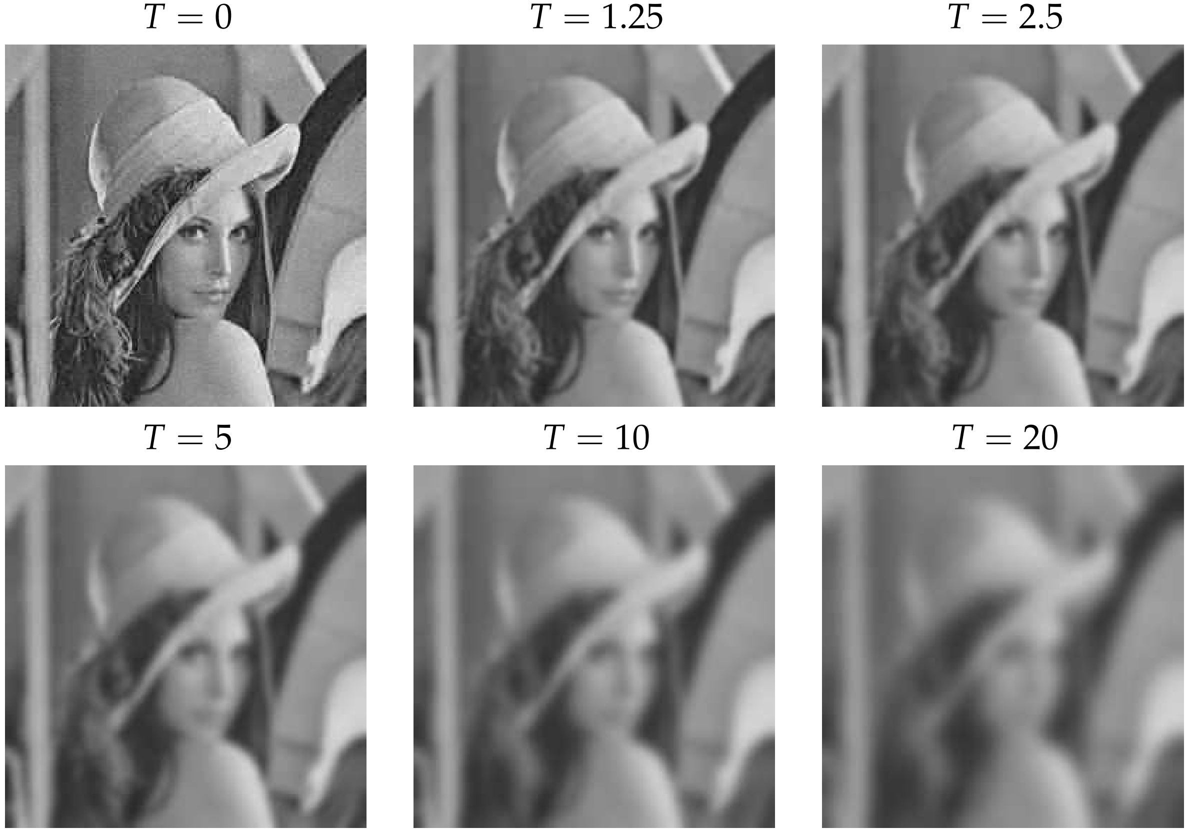
red: active areas
blue: inactive area

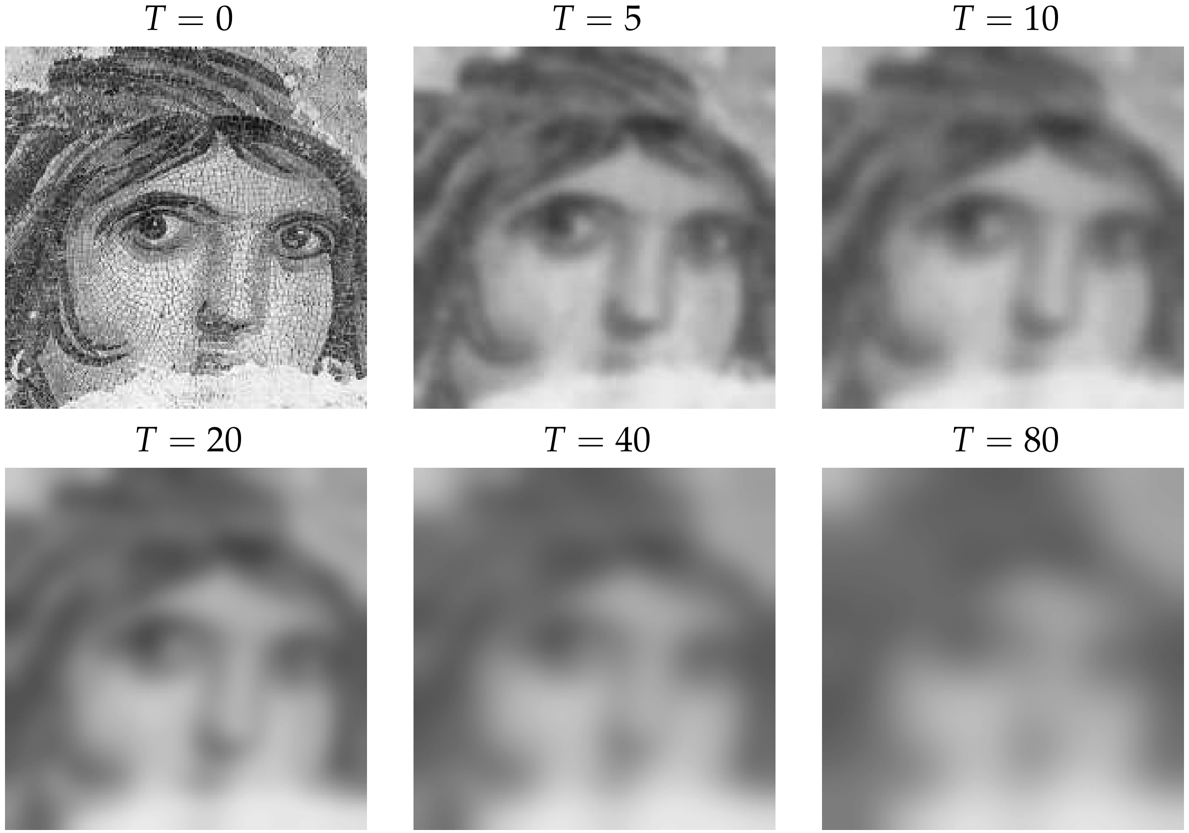
influence of the central pixel on the other pixels (red: high, blue: low)

Credit: S. Paris

Linear Diffusion (cont'd.)

- As we move to coarser scales,
  - Evolving images become more and more simplified
  - Diffusion process removes the image structures at finer scales.





Linear Diffusion and Gaussian Filtering

- The solution of the linear diffusion can be explicitly estimated as:

$$u(x,T) = \left(G_{\sqrt{2T}} * f\right)(x)$$

with

$$G_\sigma(x) = \frac{1}{2\pi\sigma^2} exp\left(-\frac{|x|^2}{2\sigma^2}\right)$$

- Solution of the linear diffusion equation is equivalent to a proper convolution of the input image with the Gaussian kernel $G_\sigma(x)$ with standard deviation $\sigma = \sqrt{2T}$
- The higher the value of *T*, the higher the value of σ, and the more smooth the image becomes.

Numerical Implementation

- Solving the linear diffusion equation requires discretization in both spatial and time coordinates.
- Central differences for the spatial derivatives:

$$\frac{d^2 u_{i,j}}{dx^2} \approx \frac{u_{i+h_x,j} - 2u_{i,j} + u_{i-h_x,j}}{h_x^2}$$

$$\frac{d^2 u_{i,j}}{dy^2} \approx \frac{u_{i,j+h_y} - 2u_{i,j} + u_{i,j-h_y}}{h_y^2}$$

where $u_{i,j}$ denotes the gray value or the brightness of the evolving image at pixel location (*i*, *j*).

- We take $h_x = h_y = 1$ for a regular grid.

Numerical Implementation (cont'd.)

- Original model:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

- Space discrete version:

$$\frac{du_{i,j}}{dt} = u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}$$

- Space-time discrete version:

$$\frac{u_{i,j}^{k+1} - u_{i,j}^{k}}{\Delta t} = u_{i+1,j}^{k} + u_{i-1,j}^{k} + u_{i,j+1}^{k} + u_{i,j-1}^{k} - 4u_{i,j}^{k}$$

homogeneous Neumann boundary condition along the image boundary

Δt ≤ 0.25 is required for numerical stability

Tikhonov energy functional

$$E(u) = \int\limits_{\Omega} \Big(\underbrace{(u - f)^2}_{\text{data fidelity term}} + \underbrace{\alpha |\nabla u|^2}_{\text{regularization term}} \Big)\, dx$$

- $\Omega \subset \mathrm{R}^2$ is connected, bounded, open subset representing the image domain,
- f is an image defined on Ω,
- u is the smooth approximation of f ,
- $\alpha > 0$ is the scale parameter.

Variational Regularization and Diffusion Equations

- A strong relation between variational regularization methods and diffusion equations.
- The minimizing function *u* of the Tikhonov energy functional formally satisfies the Euler-Lagrange equation:

$$(u - f) - \alpha \nabla^2 u = 0$$

with the Neumann boundary condition $\left.\frac{\partial u}{\partial n}\right|_{\partial\Omega} = 0$

- can be rewritten as:

$$\frac{u - u^0}{\alpha} = \nabla^2 u \qquad \text{with} \qquad u^0 = f$$

implicit time discretization of the linear diffusion equation with a single time step ($T = \alpha$)

Today

- Linear Filtering
  - Gauss filter, Linear diffusion
- Edge Detection
  - Derivative filters, Laplacian of Gaussian, Canny edge detector
- Boundary Detection
- Segmentation
  - K-means clustering, Graph-theoretic segmentation

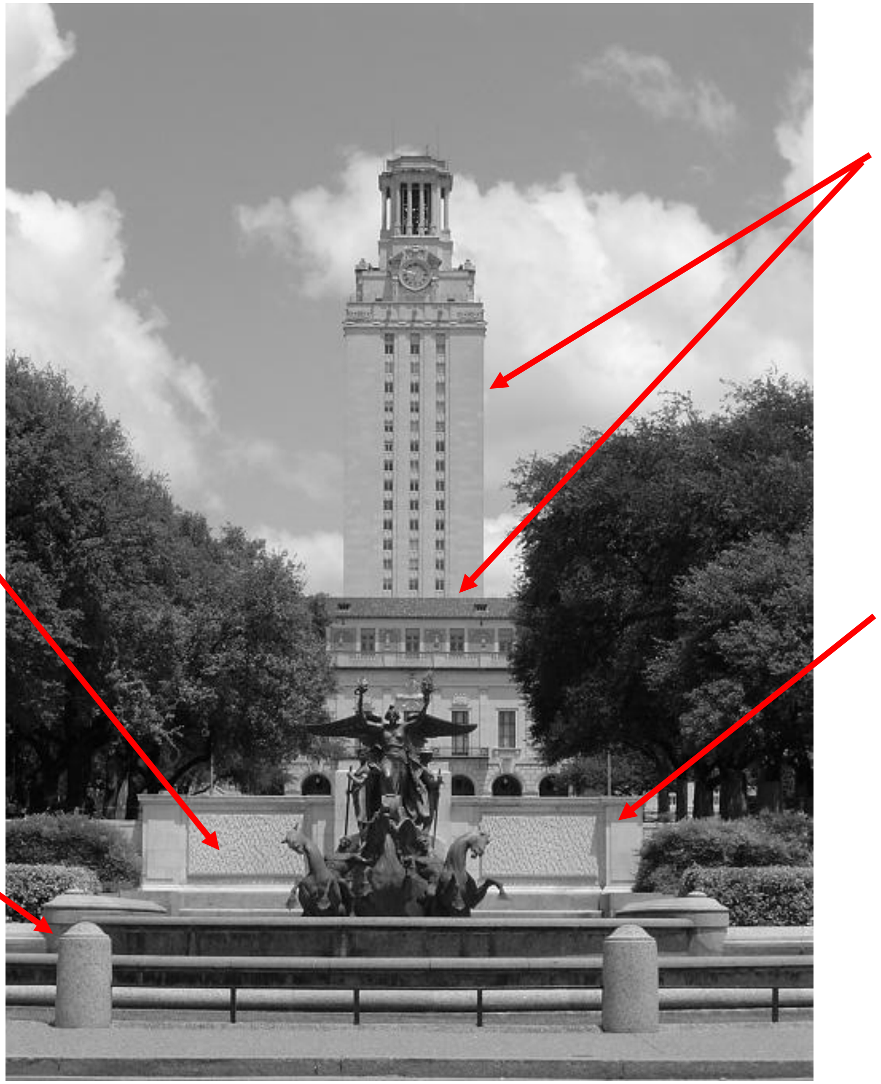
Edge detection

- **Goal:** Identify sudden changes (discontinuities) in an image
  - Intuitively, most semantic and shape information from the image can be encoded in the edges
  - More compact than pixels

- **Ideal:** artist’s line drawing (but artist is also using object-level knowledge)

Slide credit: D. Lowe

What causes an edge?



Reflectance change: appearance information, texture

Change in surface orientation: shape

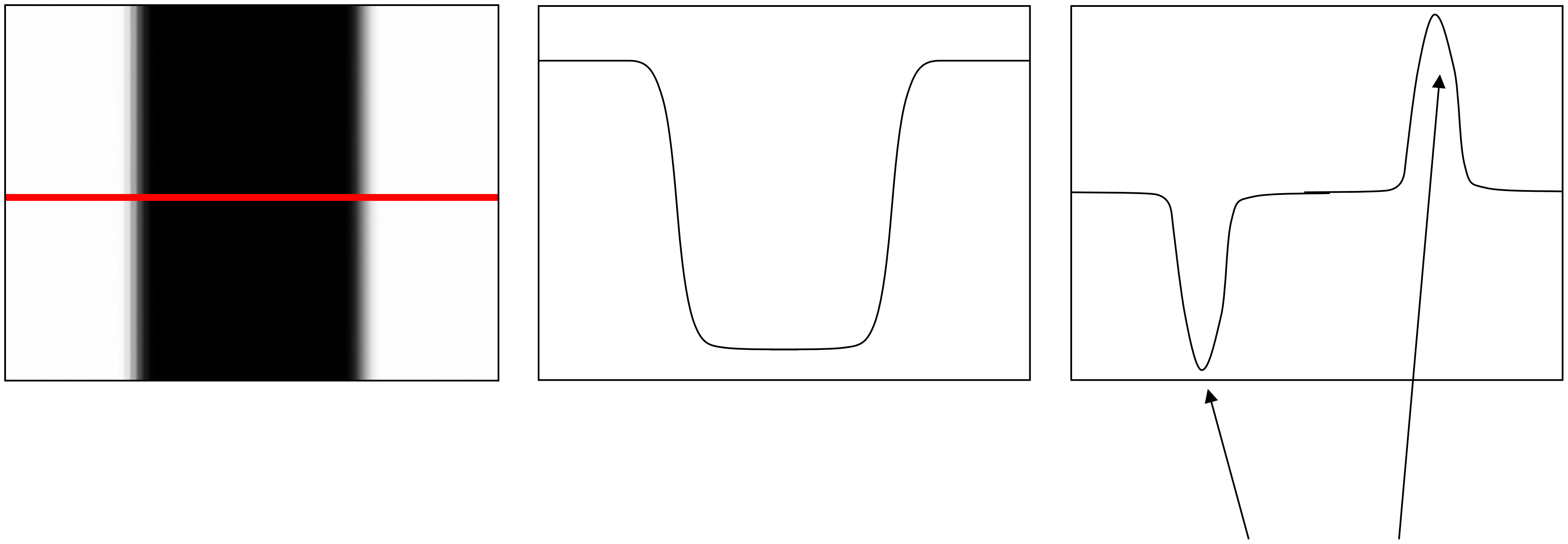
Depth discontinuity: object boundary

Cast shadows

Slide credit: K. Grauman

Characterizing edges

- An edge is a place of rapid change in the image intensity function



image

intensity function
(along horizontal scanline)

first derivative

edges correspond to
extrema of derivative

Slide credit: K. Grauman

Derivatives with convolution

For 2D function f(x,y), the partial derivative is:

$$\frac{\partial f(x,y)}{\partial x} = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon, y) - f(x,y)}{\varepsilon}$$

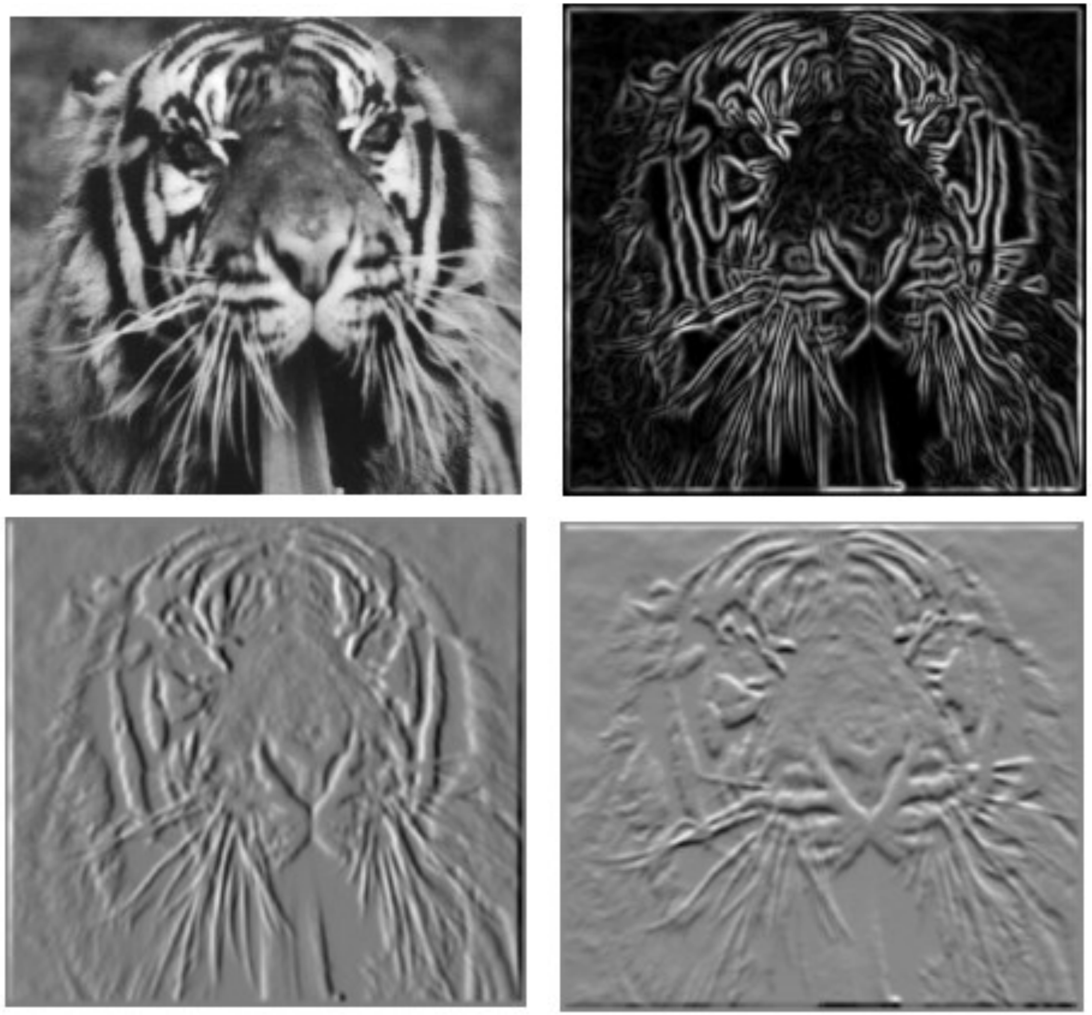
For discrete data, we can approximate using finite differences:

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1, y) - f(x,y)}{1}$$

Slide credit: K. Grauman

Partial derivatives of an image

$$\frac{\partial f(x,y)}{\partial x}$$



$$\frac{\partial f(x,y)}{\partial y}$$

$$\begin{bmatrix} -1 & 1 \end{bmatrix}$$

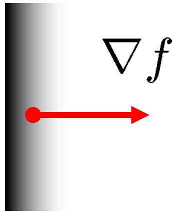
$$\begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

Which shows changes with respect to x?

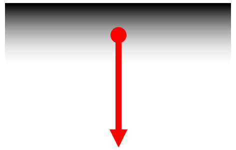
Slide credit: K. Grauman

Image gradient

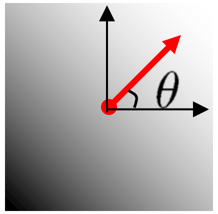
- The gradient of an image: $$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$



$$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$$



$$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$$



$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

The gradient points in the direction of most rapid increase in intensity

- How does this direction relate to the direction of the edge?

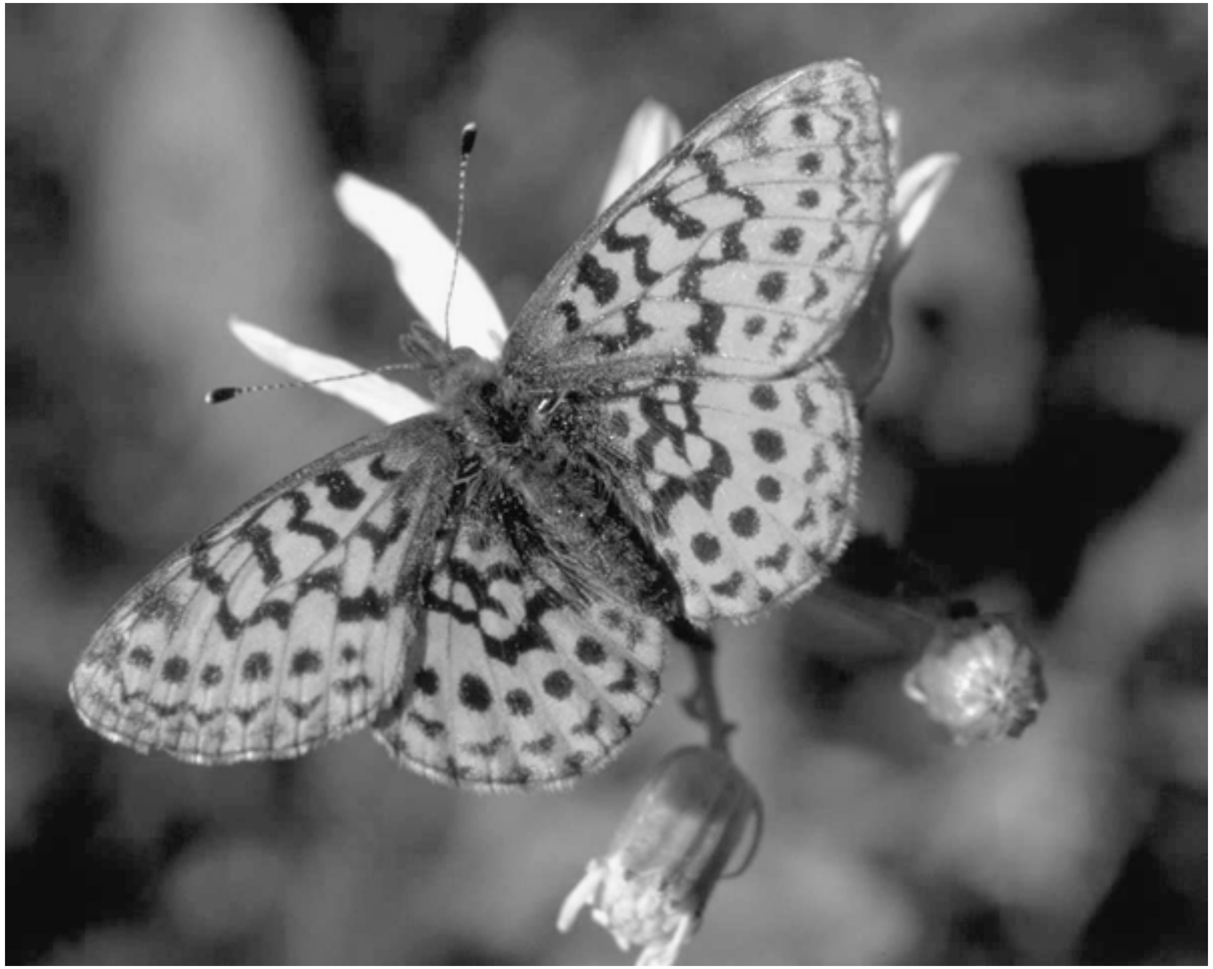
The gradient direction is given by $$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

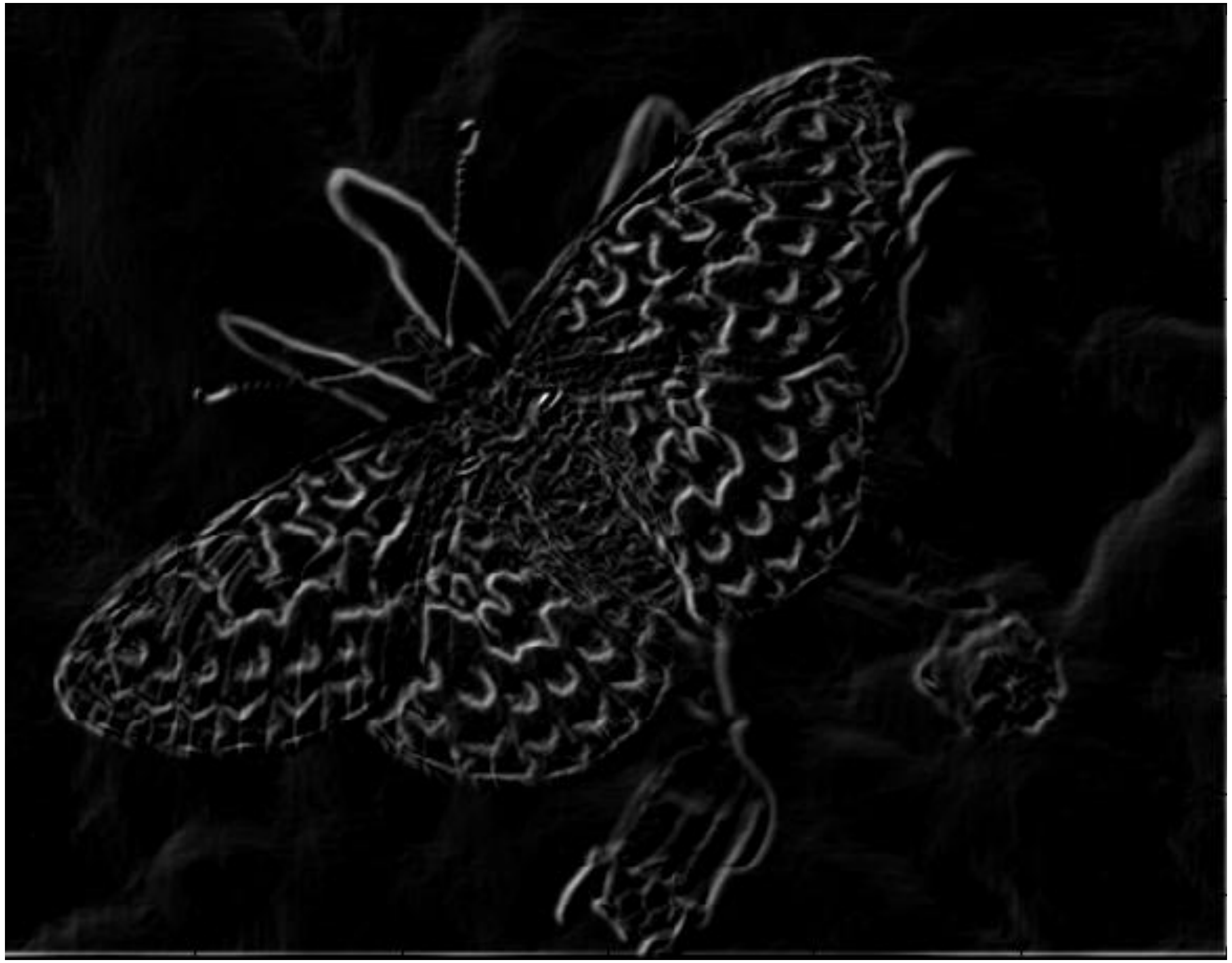
The edge strength is given by the gradient magnitude

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

Slide credit: S. Seitz

Original Image vs. Gradient magnitude image

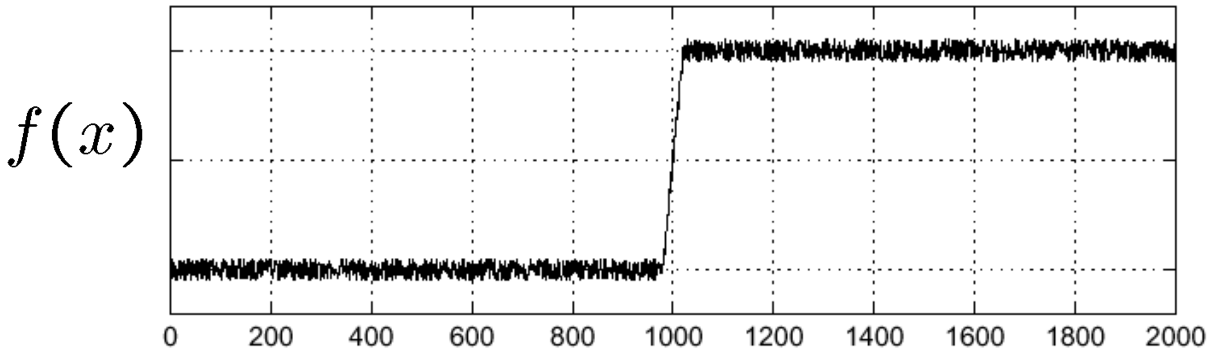




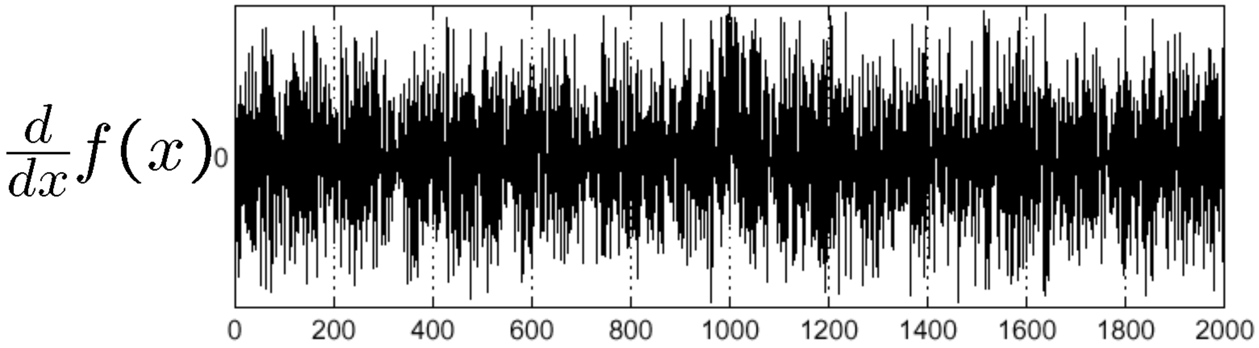
Slide credit: K. Grauman

Effects of noise

- Consider a single row or column of the image
  - Plotting intensity as a function of position gives a signal



Where is the edge?

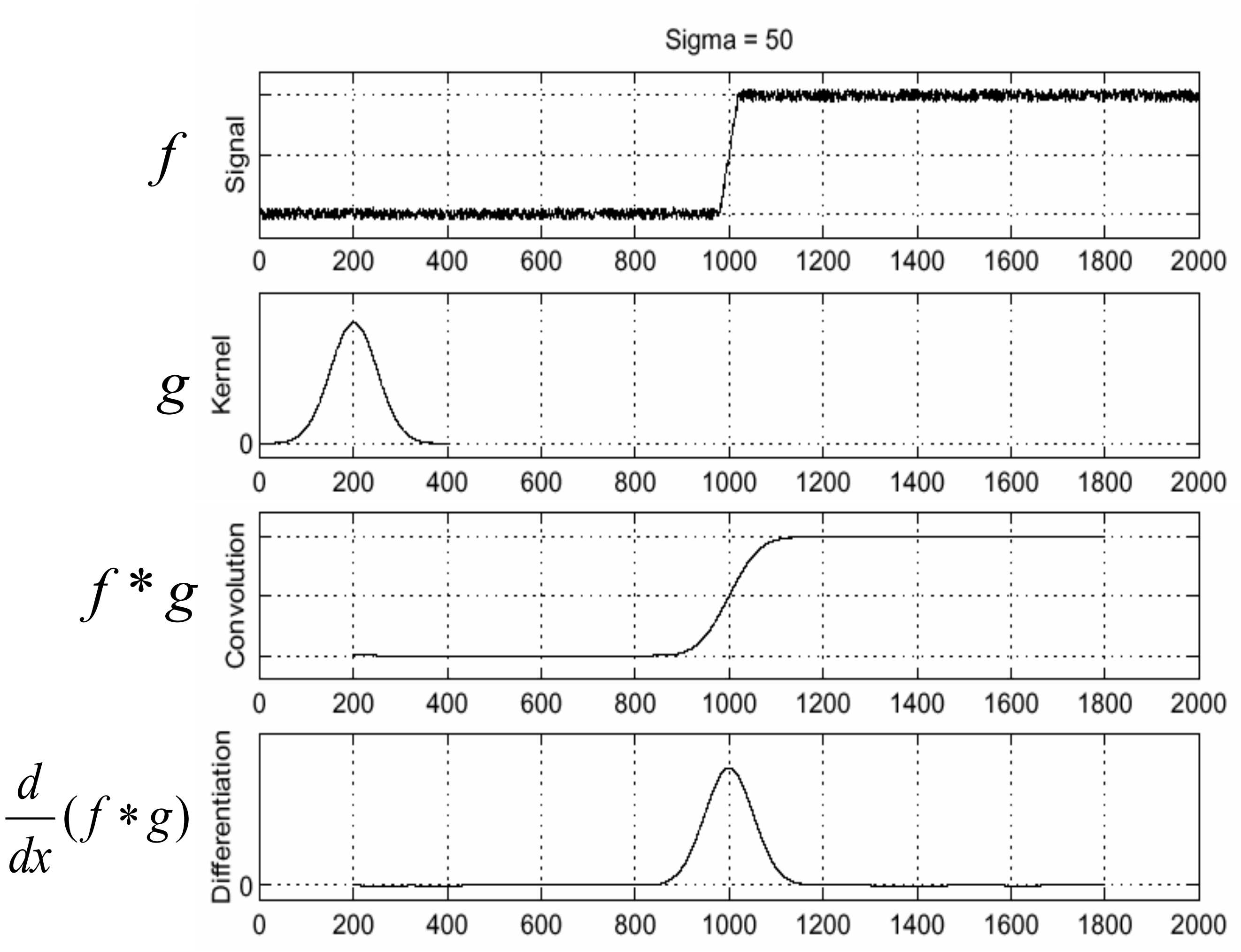


Difference filters respond strongly to noise.

What can we do about it?

Slide credit: S. Seitz

Solution: smooth first



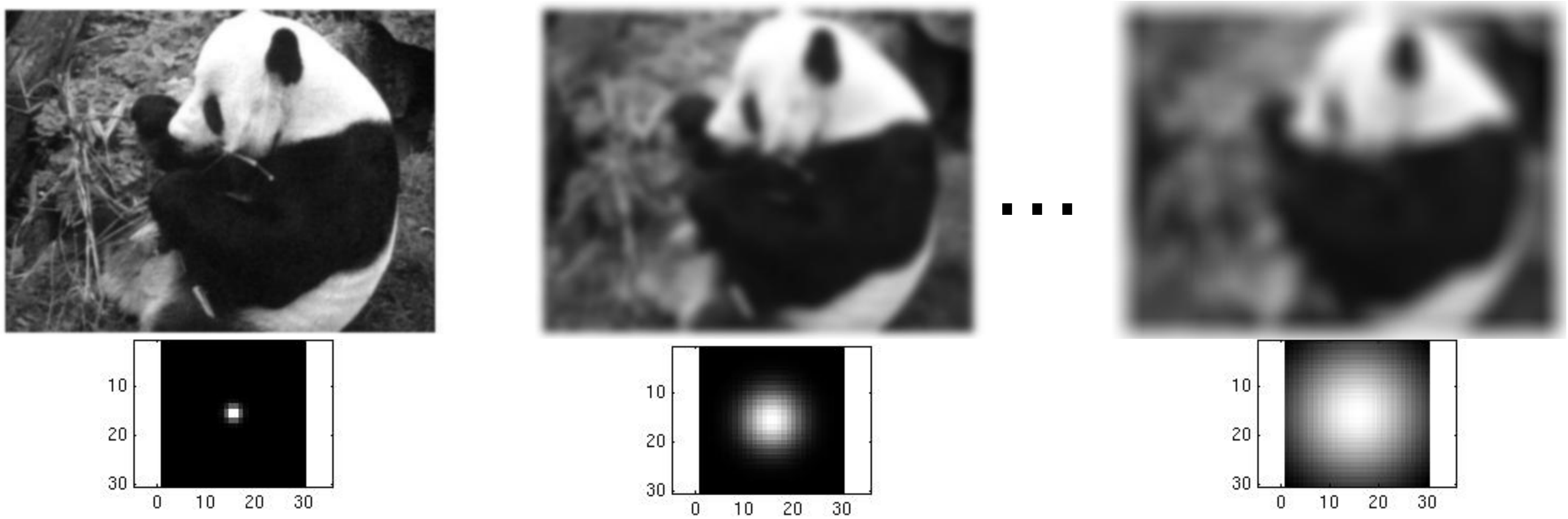
- To find edges, look for peaks in

$$\frac{d}{dx}(f * g)$$

Slide credit: S. Seitz

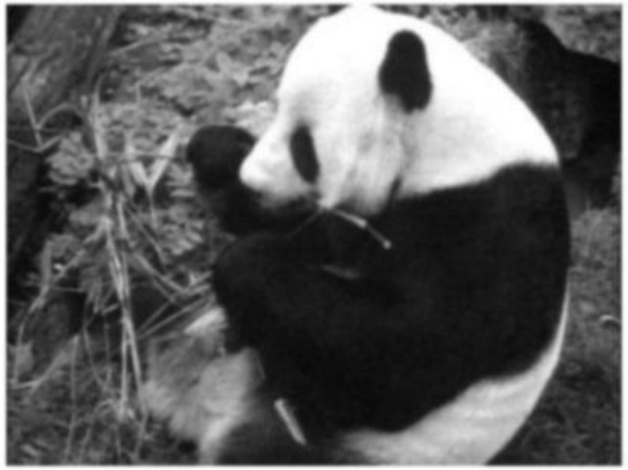
Smoothing with a Gaussian

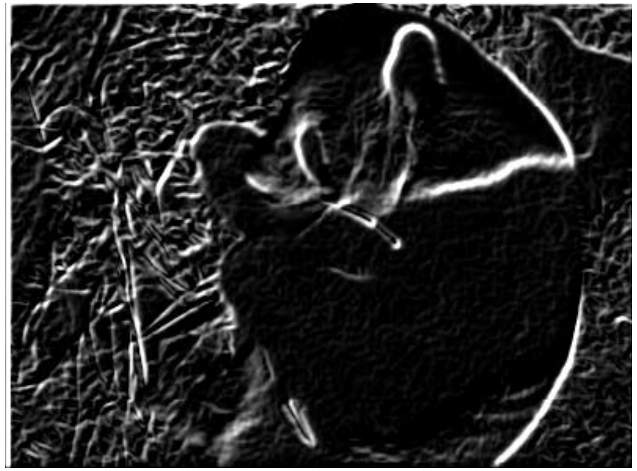
Recall: parameter σ is the “scale” / “width” / “spread” of the Gaussian kernel, and controls the amount of smoothing.



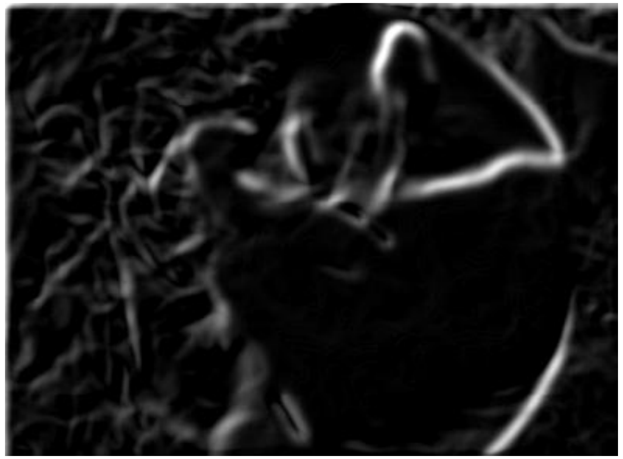
Slide credit: K. Grauman

Effect of σ on derivatives





σ = 1 pixel



σ = 3 pixels

The apparent structures differ depending on Gaussian’s scale parameter.

Larger values: larger scale edges detected
Smaller values: finer features detected

Slide credit: K. Grauman

Smoothing and Edge Detection

- While eliminating noise via smoothing, we also lose some of the (important) image details.
  - Fine details
  - Image edges
  - etc.
- What can we do to preserve such details?
  - Use edge information during denoising!
  - This requires a definition for image edges.

Chicken-and-egg dilemma!

- Edge preserving image smoothing (Next week’s topic!)

Derivative theorem of convolution

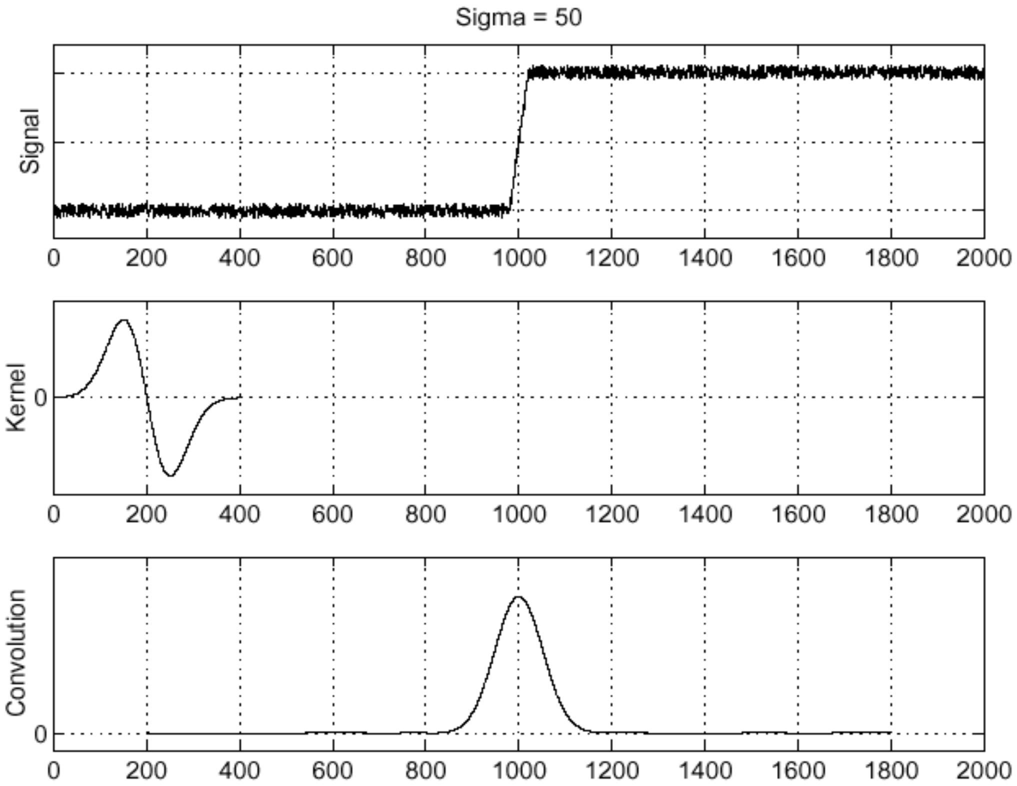
- Differentiation is convolution, and convolution is associative:
- This saves us one operation:

$$\frac{d}{dx}(f * g) = f * \frac{d}{dx} g$$

f

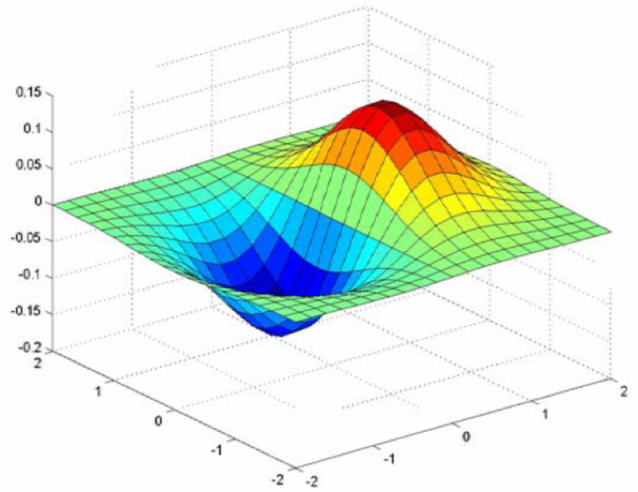
$\frac{d}{dx} g$

$f * \frac{d}{dx} g$

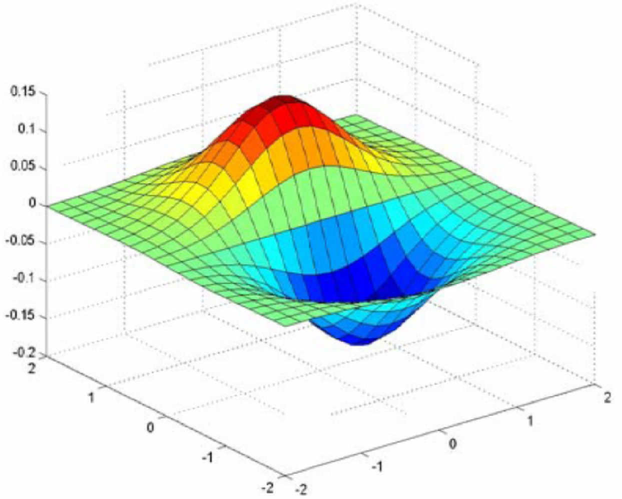


Slide credit: S. Seitz

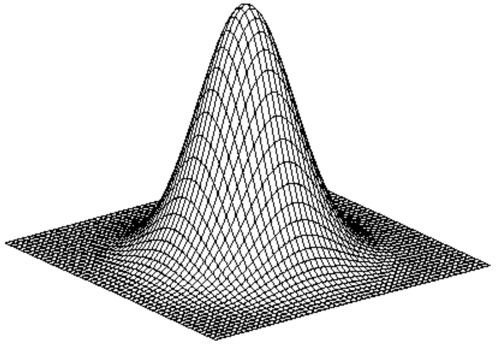
Derivative of Gaussian filter



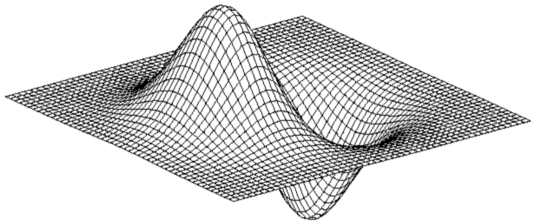
x-direction



y-direction



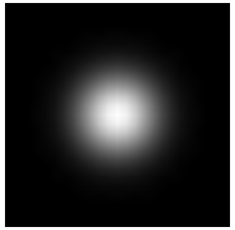
* [1 -1] =



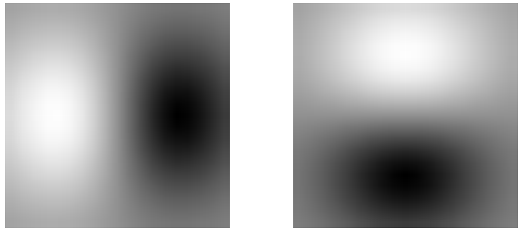
Slide credit: S. Lazebnik

Smoothing vs. derivative filters

- Smoothing filters
  - Gaussian: remove “high-frequency” components; “low-pass” filter
  - Can the values of a smoothing filter be negative?
  - What should the values sum to?
    - **One**: constant regions are not affected by the filter



- Derivative filters
  - Derivatives of Gaussian
  - Can the values of a derivative filter be negative?
  - What should the values sum to?
    - **Zero**: no response in constant regions
  - High absolute value at points of high contrast



Slide credit: S. Lazebnik

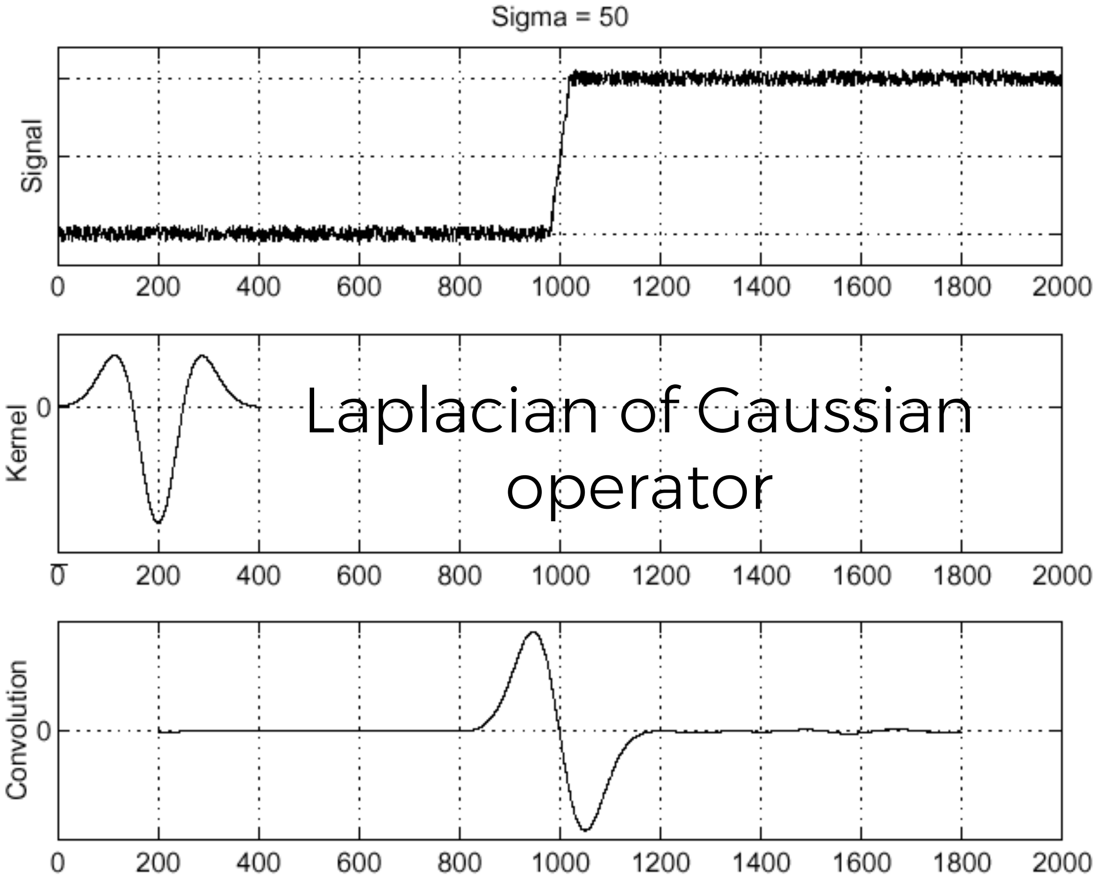
Laplacian of Gaussian

Consider $\frac{\partial^2}{\partial x^2}(h \star f)$

f

$\frac{\partial^2}{\partial x^2}h$

$(\frac{\partial^2}{\partial x^2}h) \star f$

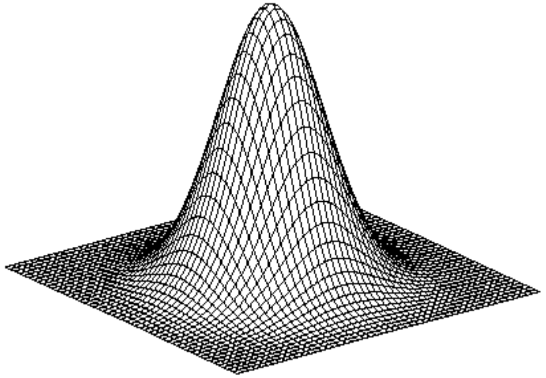


Where is the edge?

Zero-crossings of bottom graph

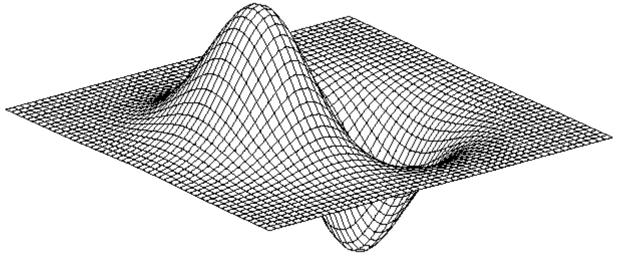
Slide credit: K. Grauman

2D edge detection filters



Gaussian

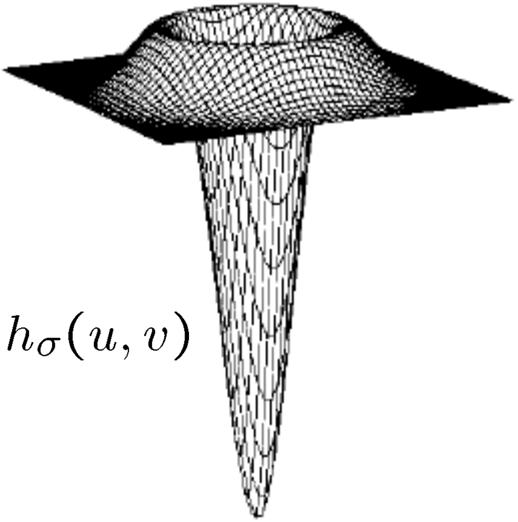
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



derivative of Gaussian

$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

Laplacian of Gaussian

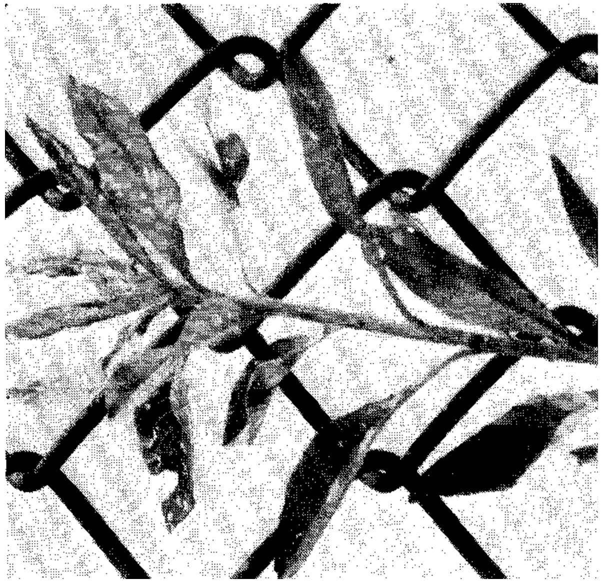


$$\nabla^2 h_\sigma(u,v)$$

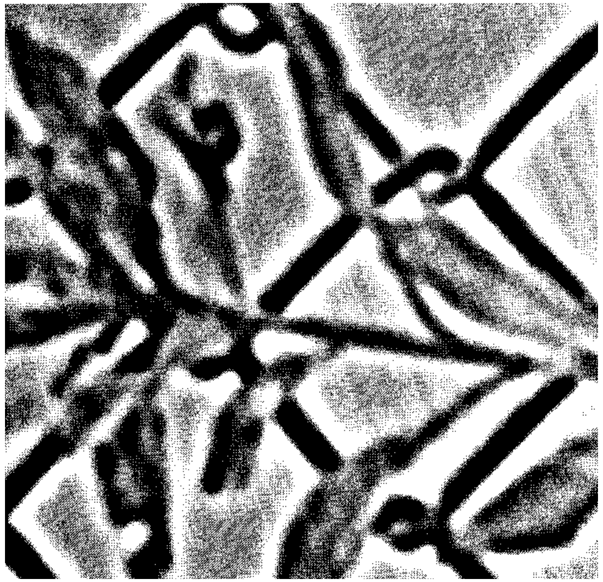
- The Laplacian operator: $\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$

Slide credit: K. Grauman

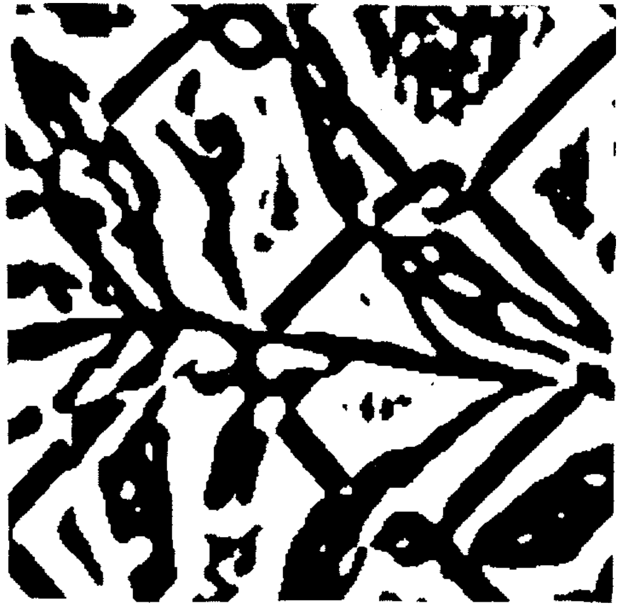
Laplacian of Gaussian



original image



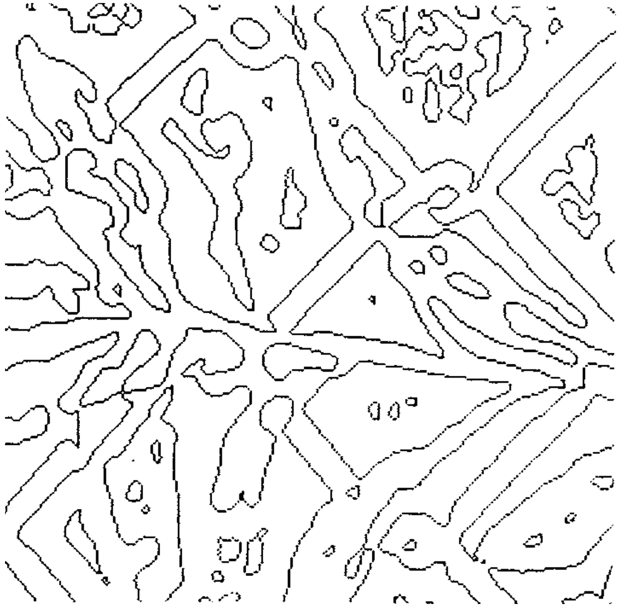
convolution with $\nabla^2 h_\sigma(u,v)$



convolution with $\nabla^2 h_\sigma(u,v)$

pos. values – white

neg. values – black



zero-crossings

Source: D. Marr and E. Hildreth (1980)

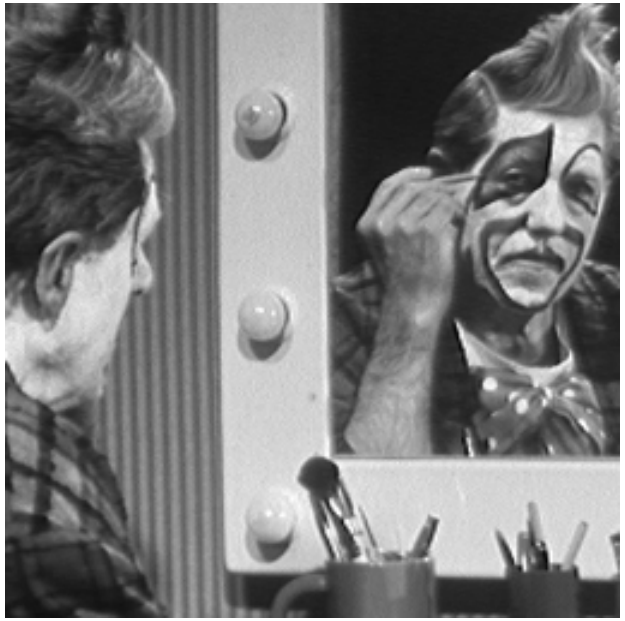
Canny edge detector

1. Filter image with derivative of Gaussian
2. Find magnitude and orientation of gradient
3. **Non-maximum suppression**:
   - Thin wide "ridges" down to single pixel width
4. **Linking and thresholding** (**hysteresis**):
   - Define two thresholds: low and high
   - Use the high threshold to start edge curves and the low threshold to continue them

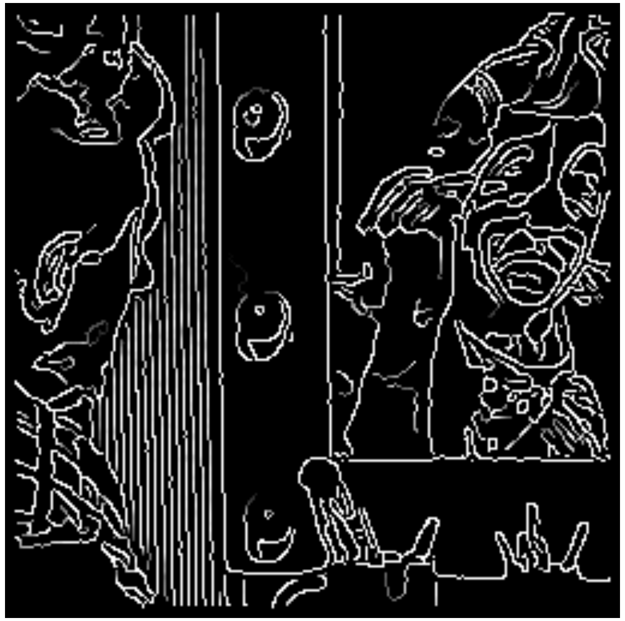
- MATLAB: `edge(image, ‘canny’);`

Slide credit: D. Lowe, L. Fei-Fei

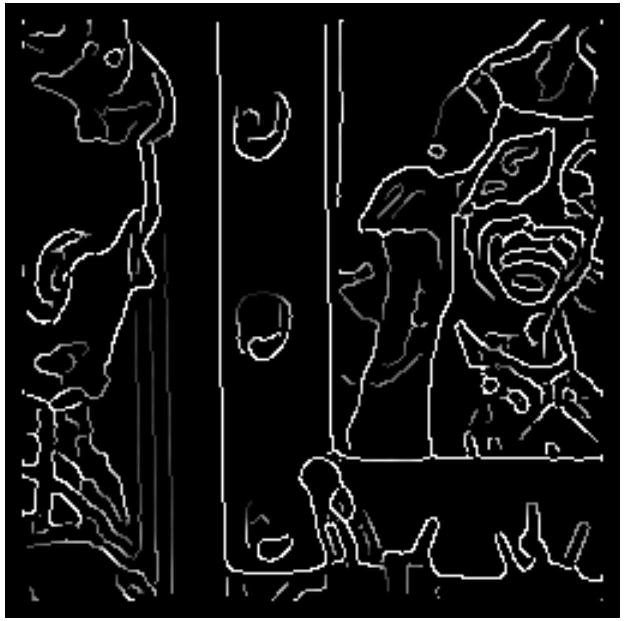
Effect of σ (Gaussian kernel spread/size)



original



Canny with $\sigma = 1$



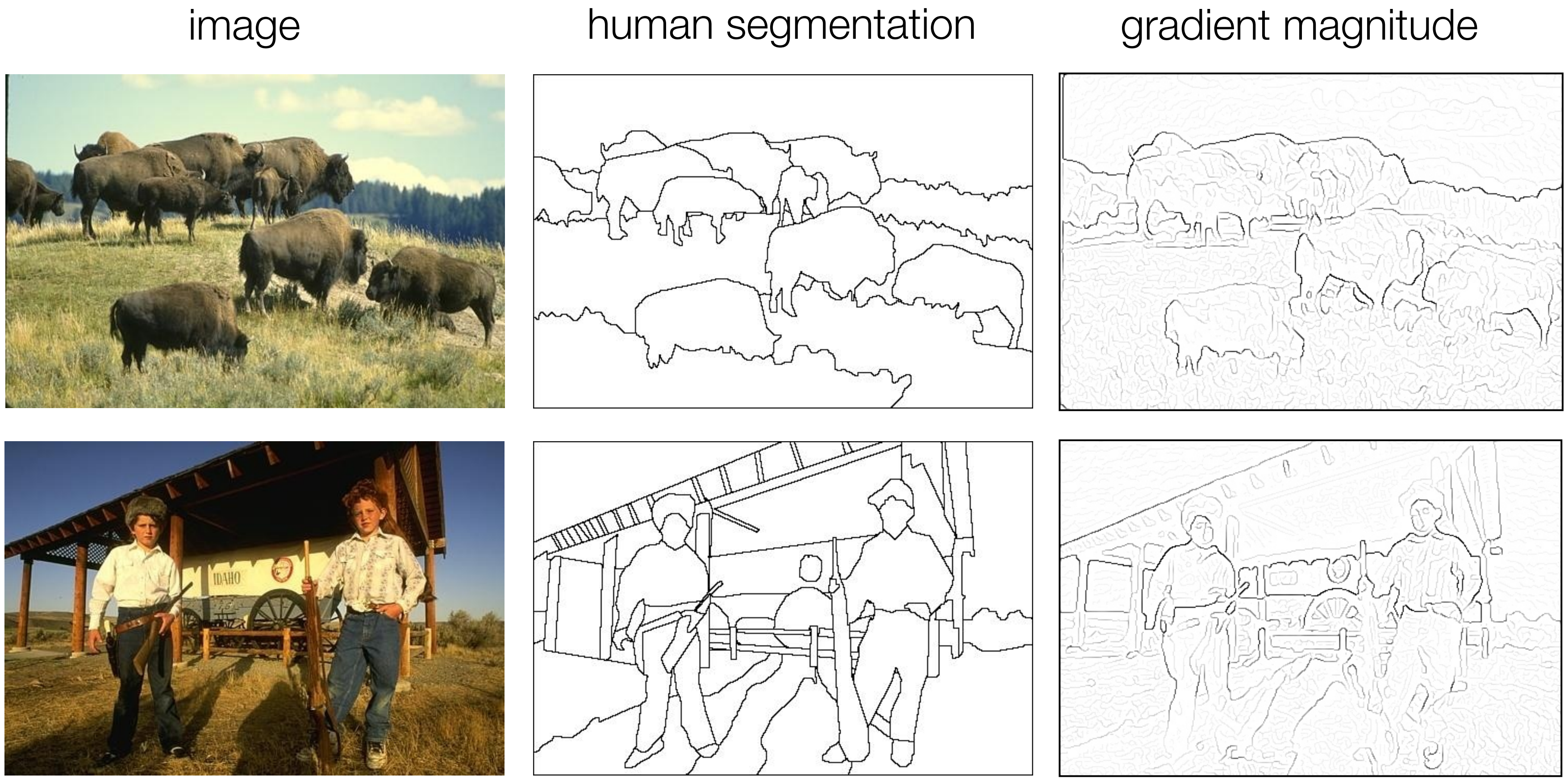
Canny with $\sigma = 2$

The choice of σ depends on desired behavior

- large σ detects large scale edges
- small σ detects fine features

Slide credit: S. Seitz

Edge detection is just the beginning…



image

human segmentation

gradient magnitude

- Berkeley segmentation database:
  http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/

Source: S. Lazebnik

Berkeley Segmentation Data Set - Protocol

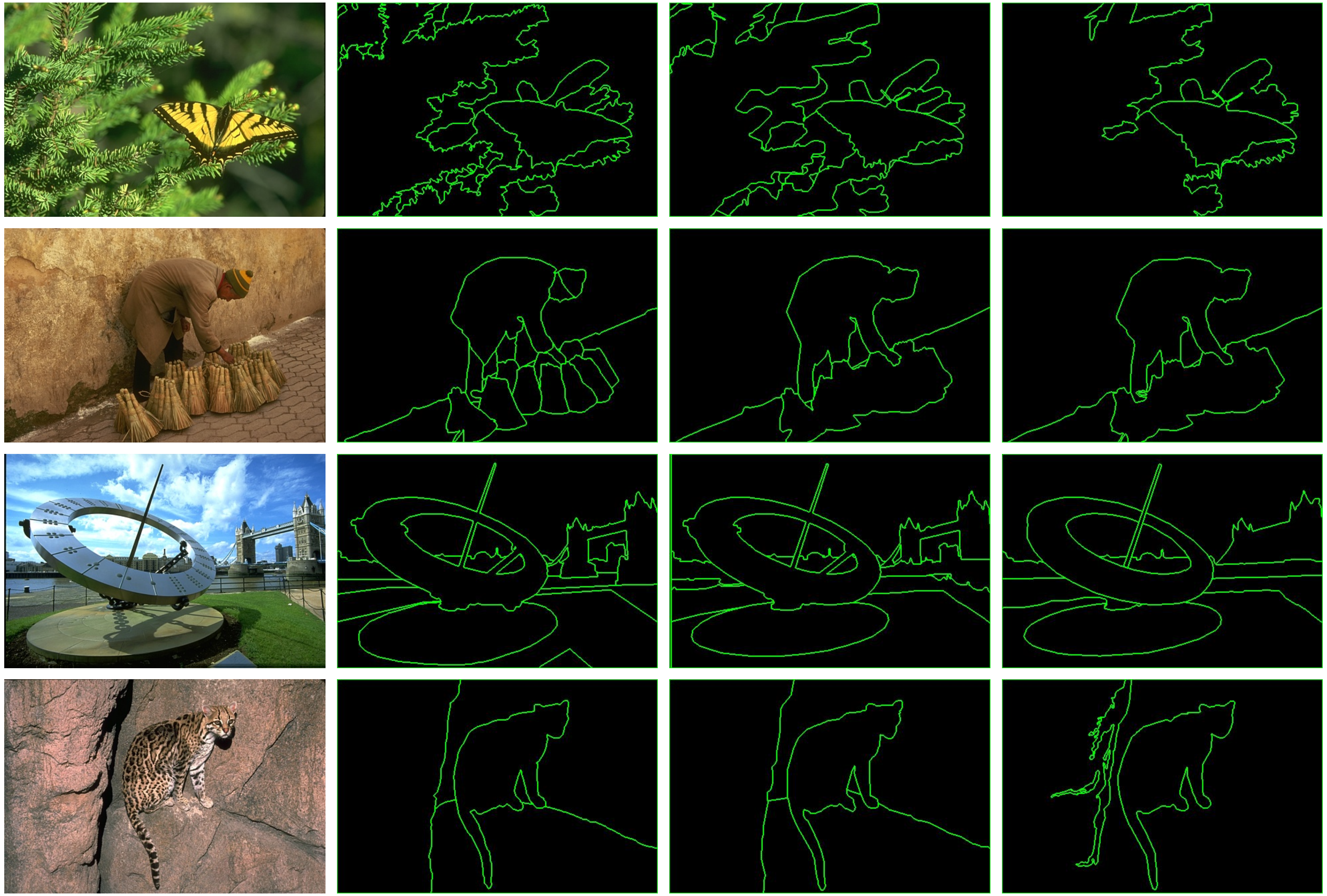
You will be presented a photographic image. Divide the image into some number of segments, where the segments represent "things" or "parts of things" in the scene. The number of segments is up to you, as it depends on the image. Something between 2 and 30 is likely to be appropriate. It is important that all of the segments have approximately equal importance.

- Custom segmentation tool
- Subjects obtained from work-study program (UC Berkeley undergraduates)

Slide credit: J. Hays

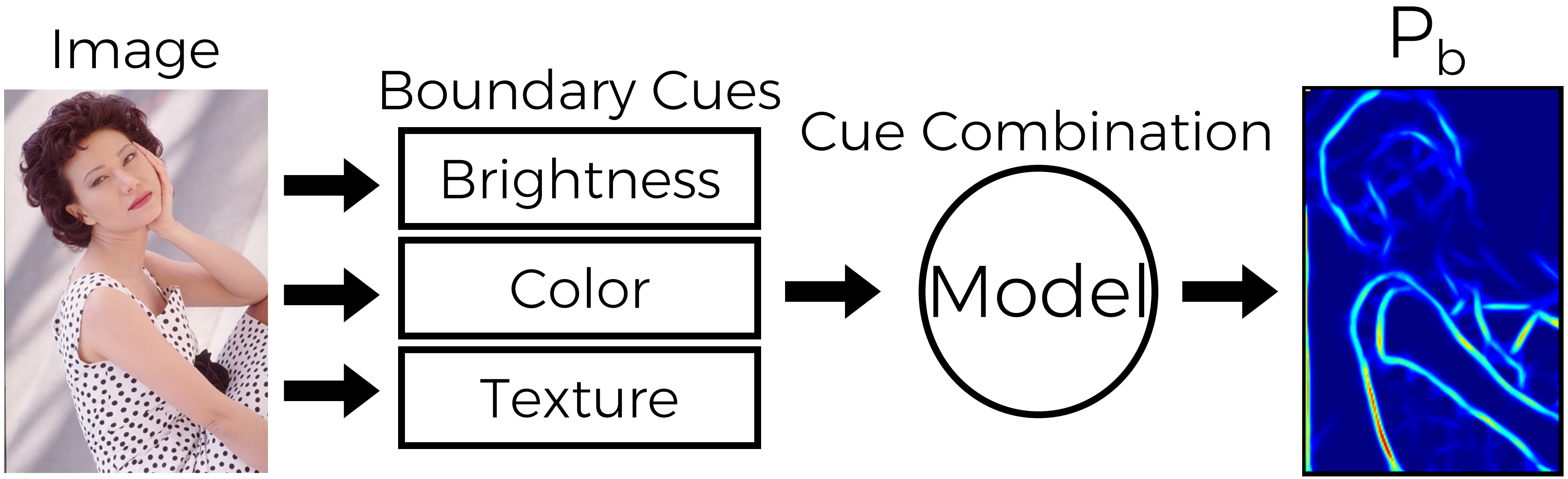
Dataset Summary

- 30 subjects, age 19-23
  - 17 men, 13 women
  - 9 with artistic training
- 8 months
- 1,458 person hours
- 1,020 Corel images
- 11,595 Segmentations
  - 5,555 color,
    5,554 gray,
    486 inverted/negated



Slide credit: J. Hays

Pb Detector



Image

Boundary Cues

Brightness

Color

Texture

Cue Combination

Model

P_b

Challenges: texture cue, cue combination

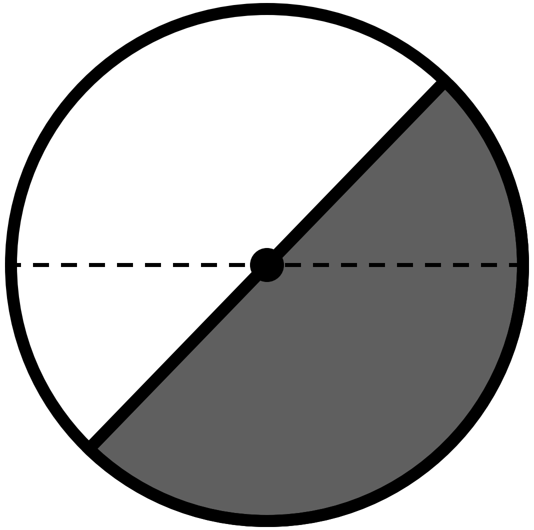
Goal: learn the posterior probability of a boundary P_b from local information only

Slide credit: J. Hays

Brightness and Color Features

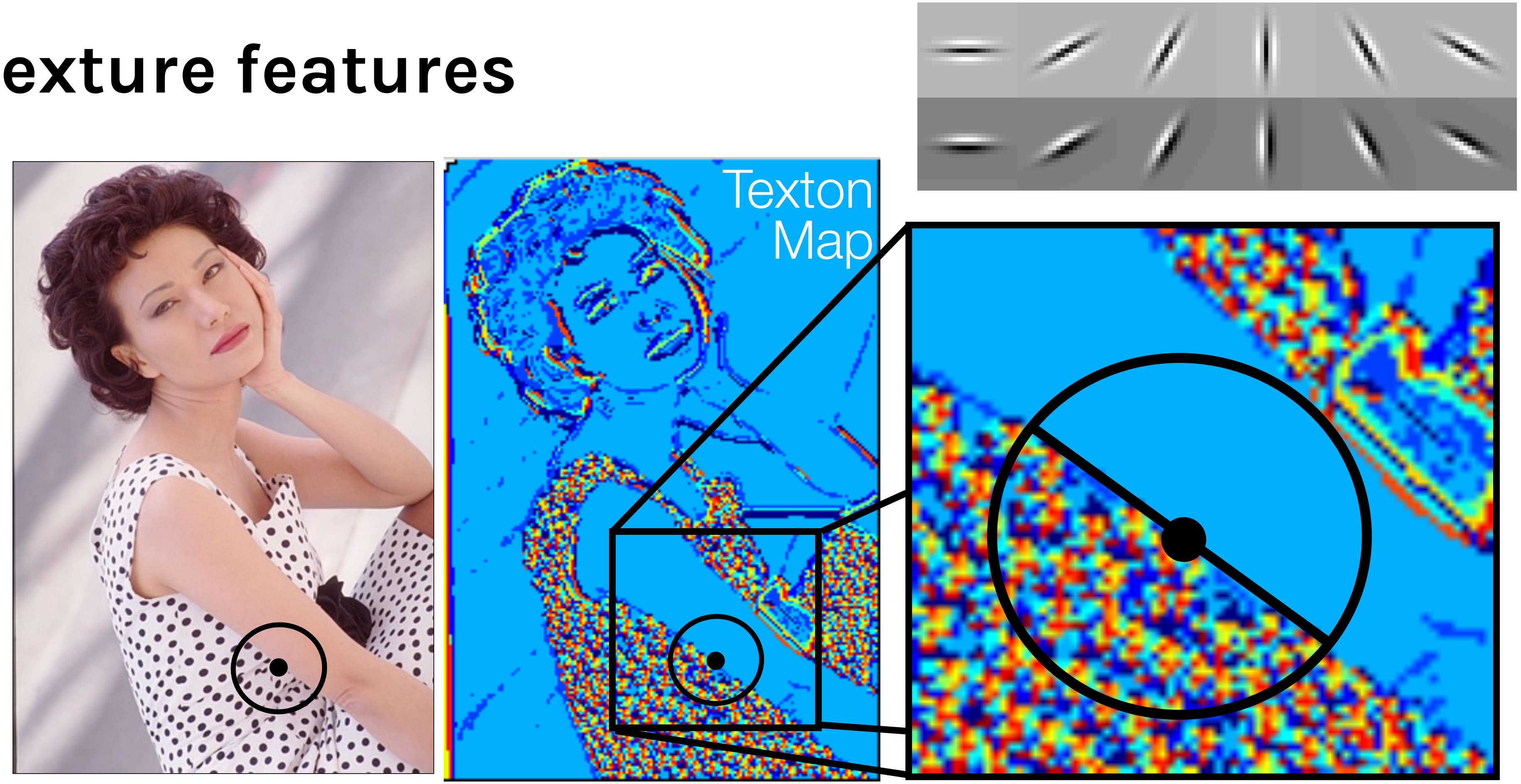
- 1976 CIE L*a*b* colorspace
- Brightness Gradient (B)
  - Chi^2 difference in L* distribution
- Color Gradient (C)
  - Chi^2 difference in a* and b* distributions

$$\chi^2(g,h) = \frac{1}{2}\sum_i \frac{(g_i - h_i)^2}{g_i + h_i}$$



Slide credit: J. Hays

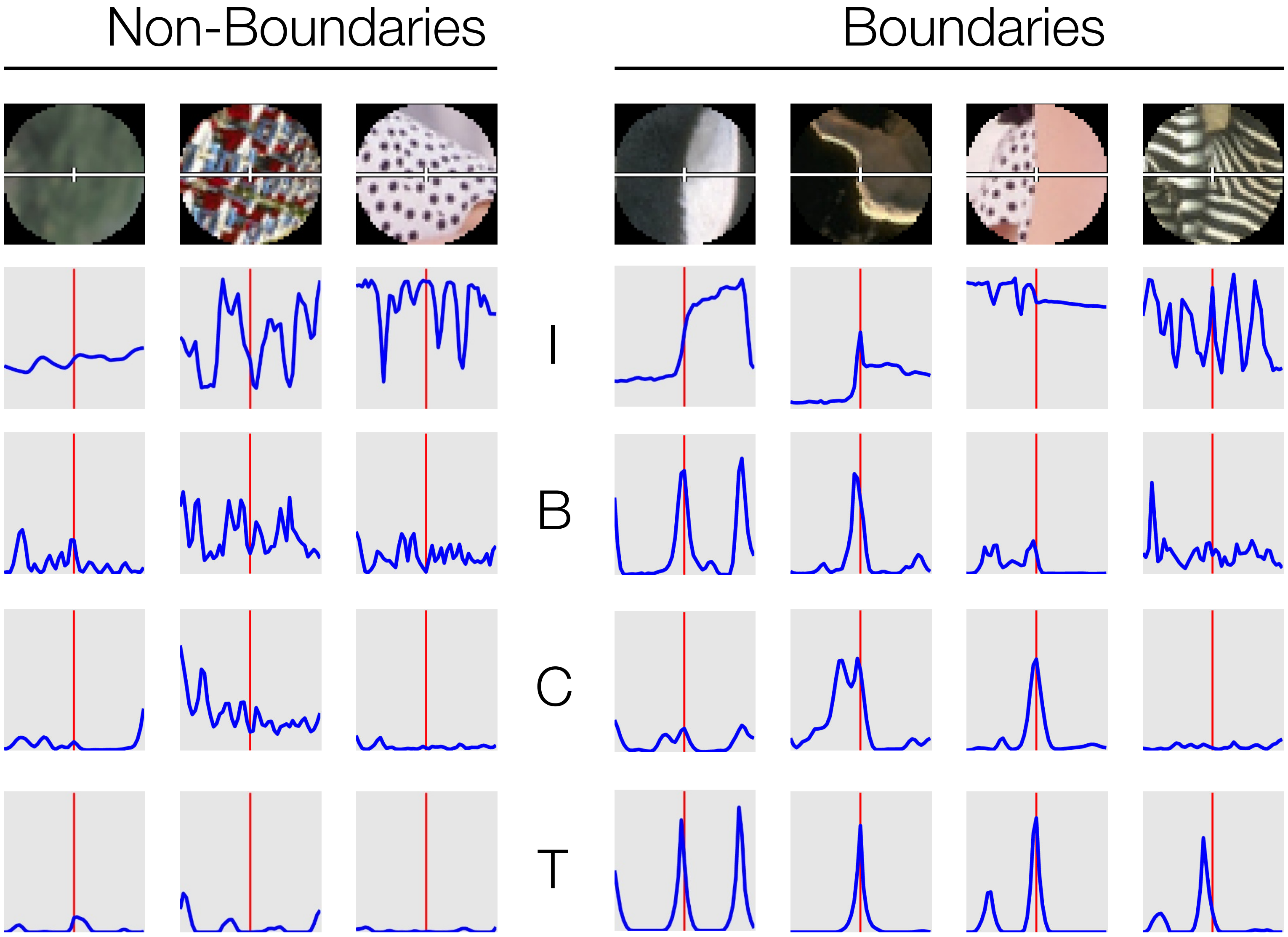
Texture features



- Texture Gradient (T)
- Chi^2 difference of texton histograms
  - – Textons are vector-quantized filter outputs

Slide credit: J. Hays

Texture features



Slide credit: J. Hays

Cue Combination Models

- Classification Trees
  - Top-down splits to maximize entropy, error bounded
- Density Estimation
  - Adaptive bins using k-means
- Logistic Regression, 3 variants
  - Linear and quadratic terms
  - Confidence-rated generalization of AdaBoost (Schapire&Singer)
- Hierarchical Mixtures of Experts (Jordan&Jacobs)
  - Up to 8 experts, initialized top-down, fit with EM
- Support Vector Machines (libsvm, Chang&Lin)

- Range over bias, complexity, parametric/non-parametric

Slide credit: J. Hays

Computing Precision/Recall

Recall = Pr(signal|truth) = fraction of ground truth found by the signal

Precision = Pr(truth|signal) = fraction of signal that is correct

- Always a trade-off between the two
- Standard measures in information retrieval (van Rijsbergen XX)
- ROC from standard signal detection the wrong approach

Strategy

- Detector output (Pb) is a soft boundary map
- Compute precision/recall curve:
  - Threshold Pb at many points t in [0,1]
  - Recall = Pr(Pb>t|seg=1)
  - Precision = Pr(seg=1|Pb>t)

Slide credit: J. Hays

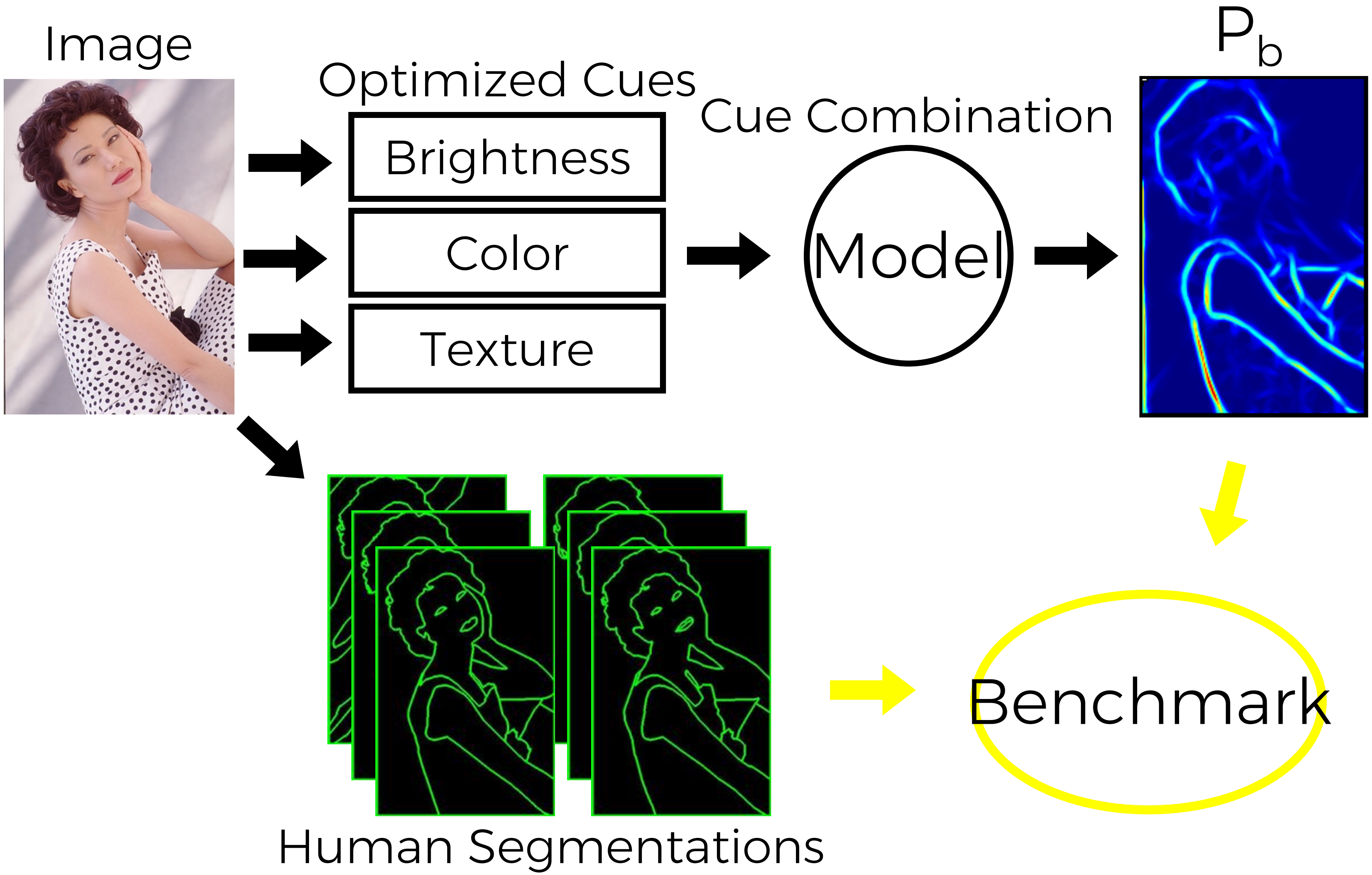
Cue Calibration

- All free parameters optimized on training data
- All algorithmic alternatives evaluated by experiment

- Brightness Gradient
  - Scale, bin/kernel sizes for KDE
- Color Gradient
  - Scale, bin/kernel sizes for KDE, joint vs. marginals
- Texture Gradient
  - Filter bank: scale, multiscale?
  - Histogram comparison
  - Number of textons, Image-specific vs. universal textons
- Localization parameters for each cue

Slide credit: J. Hays

Dataflow



Image

Optimized Cues

Brightness

Color

Texture

Cue Combination

Model

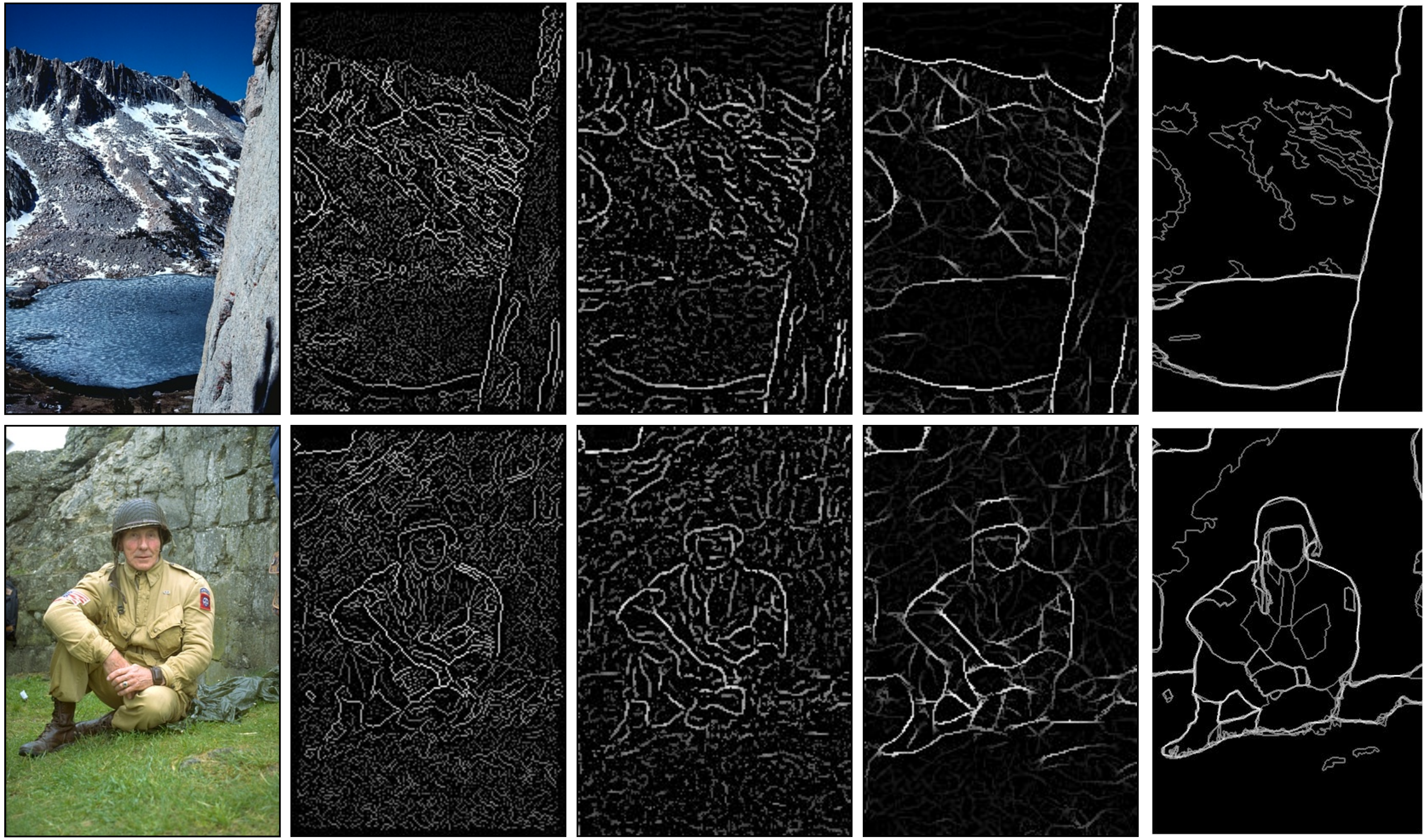
P_b

Human Segmentations

Benchmark

Slide credit: J. Hays

P_b Images



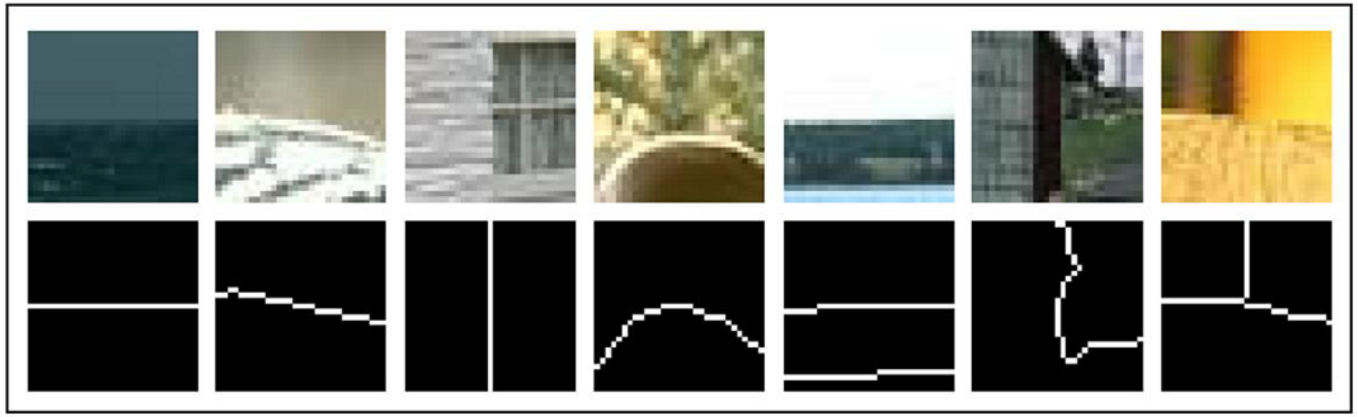
Slide credit: J. Hays

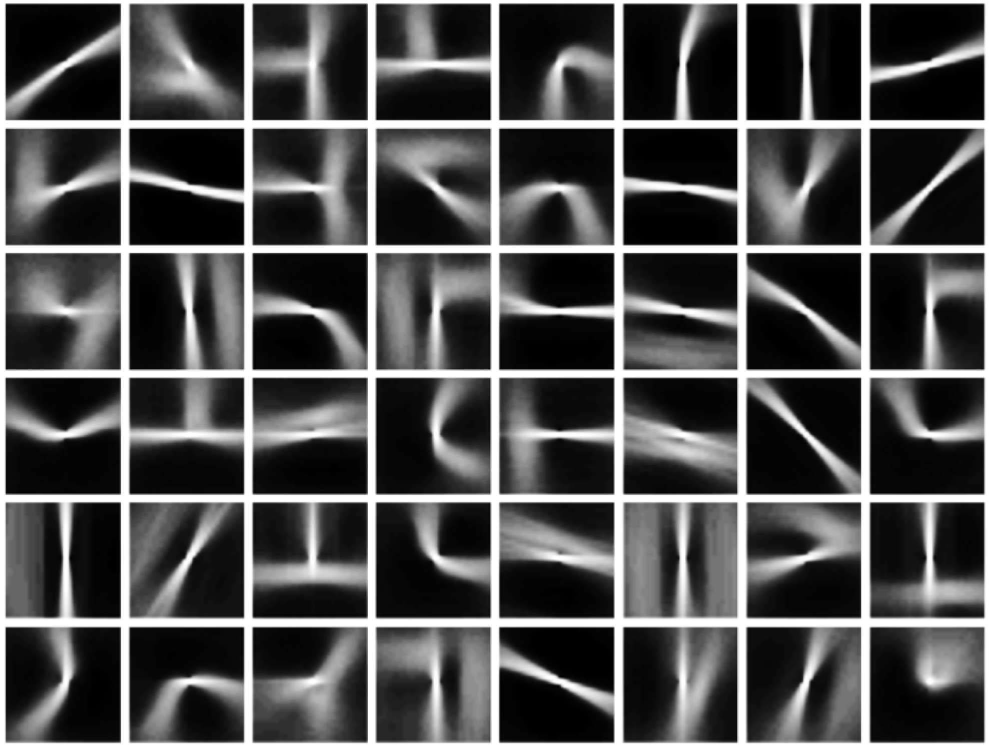
Findings

1. A simple linear model is sufficient for cue combination
   - All cues weighted approximately equally in logistic
2. Proper texture edge model is not optional for complex natural images
   - Texture suppression is not sufficient!
3. Significant improvement over state-of-the-art in boundary detection
4. Empirical approach critical for both cue calibration and cue combination

Slide credit: J. Hays

Sketch Tokens (J. Lim et al., CVPR 2013)





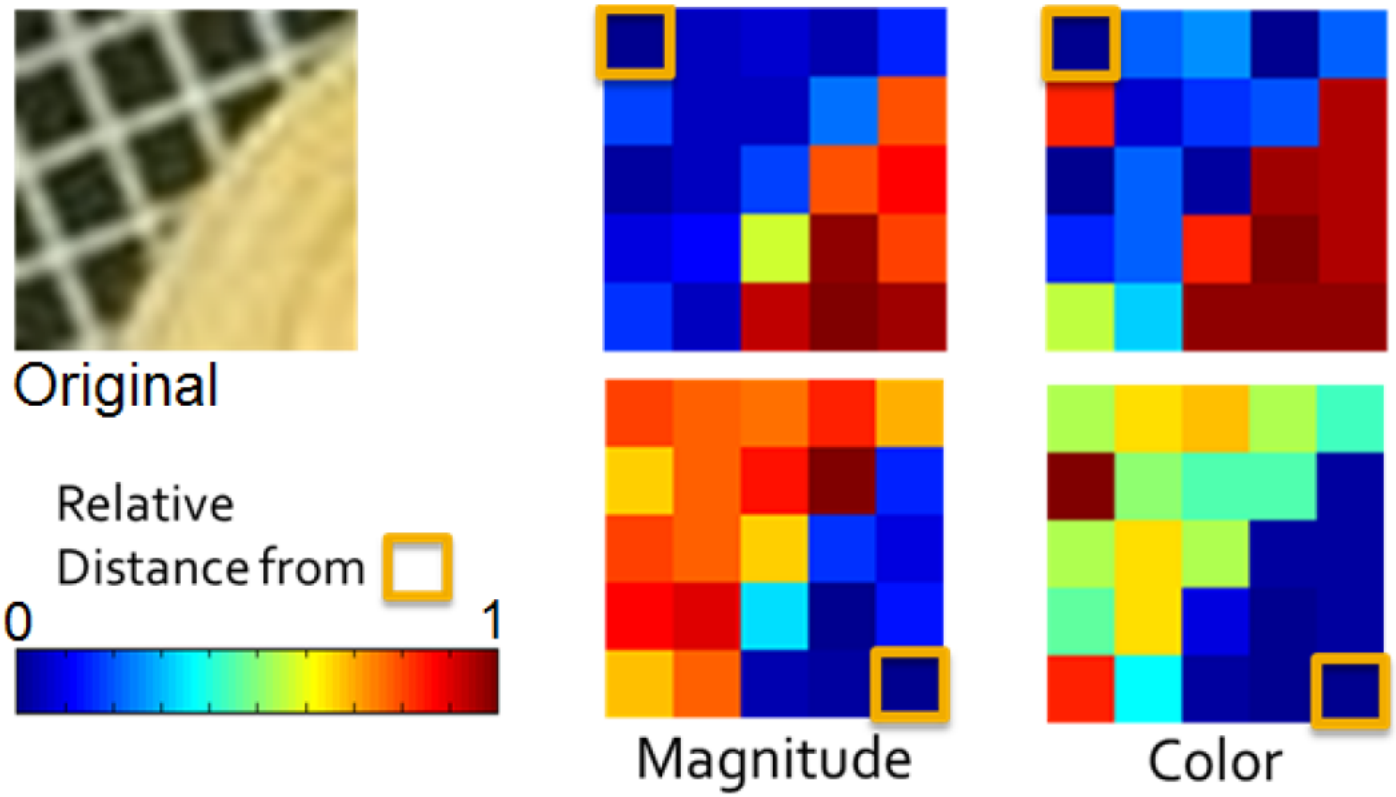
Slide credit: J. Hays

Image Features – 21350 dimensions!

- 35x35 patches centered at every pixel
- 35x35 "channels" of many types:
  - Color (3 channels)
  - Gradients (3 unoriented + 8 oriented channels)
    - Sigma = 0, T heta = 0, pi/2, pi, 3pi/2
    - Sigma = 1.5, Theta = 0, pi/2, pi, 3pi/2
    - Sigma = 5
  - Self Similarity
    - 5x5 maps of self similarity within the above channels for a particular anchor point.

Slide credit: J. Hays

Self-similarity features

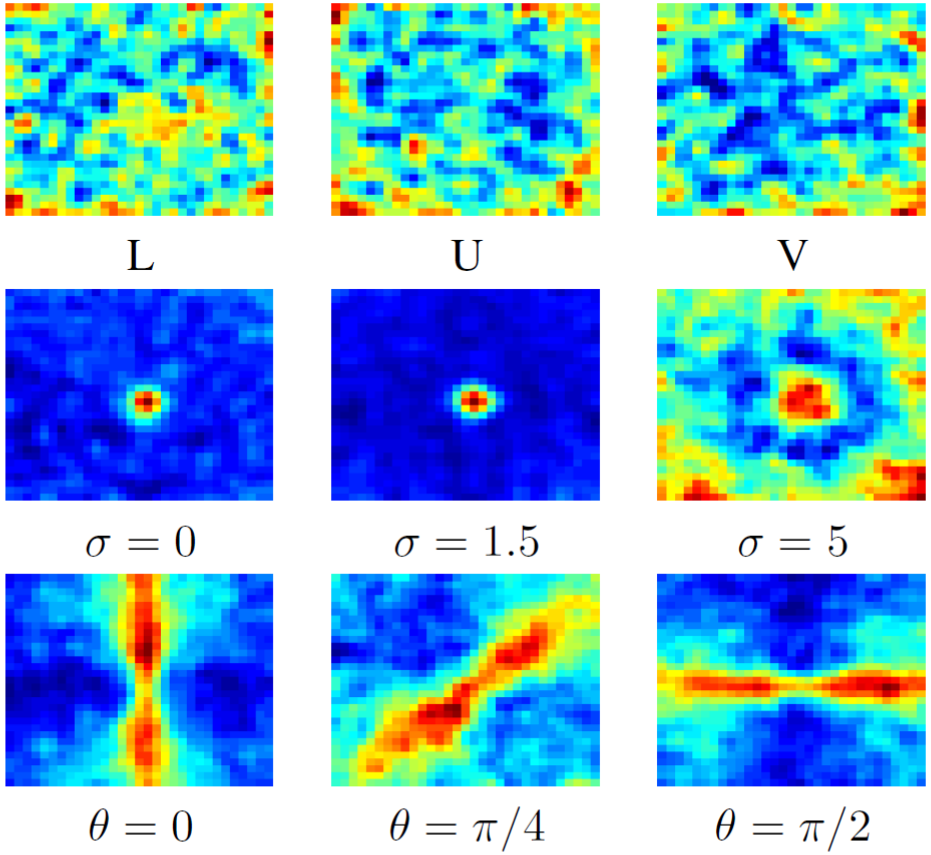


Self-similarity features: The L1 distance from the anchor cell (yellow box) to the other 5 x 5 cells are shown for color and gradient magnitude channels. The original patch is shown to the left.

Slide credit: J. Hays

Learning

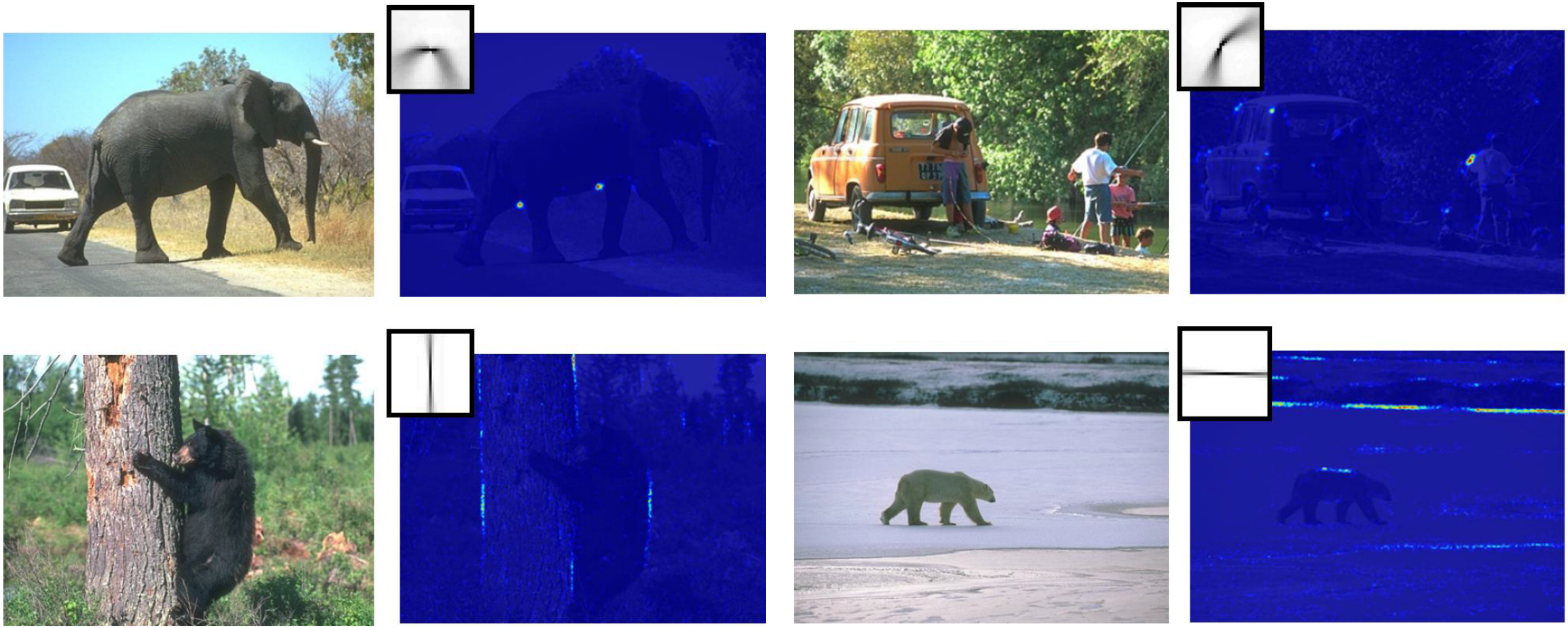
- Random Forest Classifiers, one for each sketch token + background, trained 1-vs-all
- Advantages:
  - Fast at test time, especially for a non-linear classifier.
  - Don't have to explicitly compute independent descriptors for every patch. Just look up what the decision tree wants to know at each branch.



Frequency of example features being selected by the random forest: (first row) color channels, (second row) gradient magnitude channels, (third row) selected orientation channels.

Slide credit: J. Hays

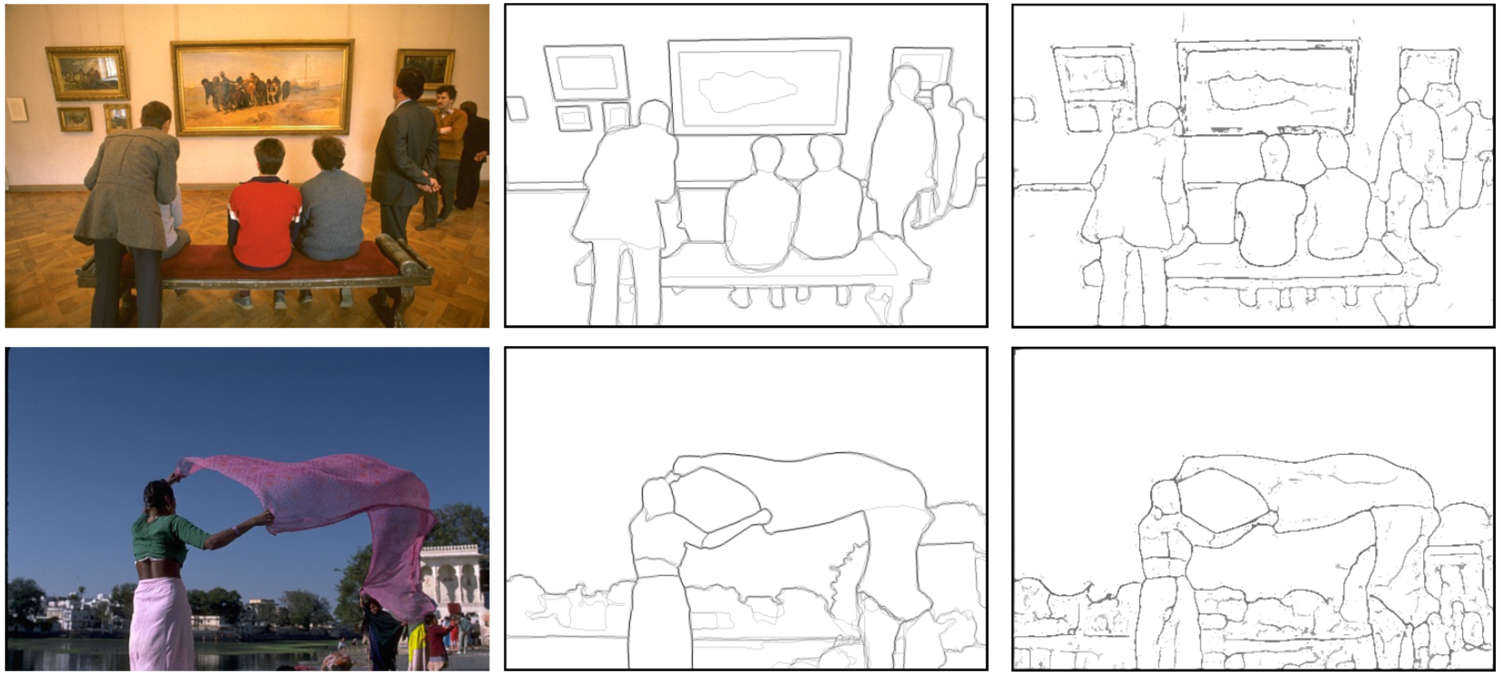
Detections of individual sketch tokens



Slide credit: J. Hays

Combining sketch token detections

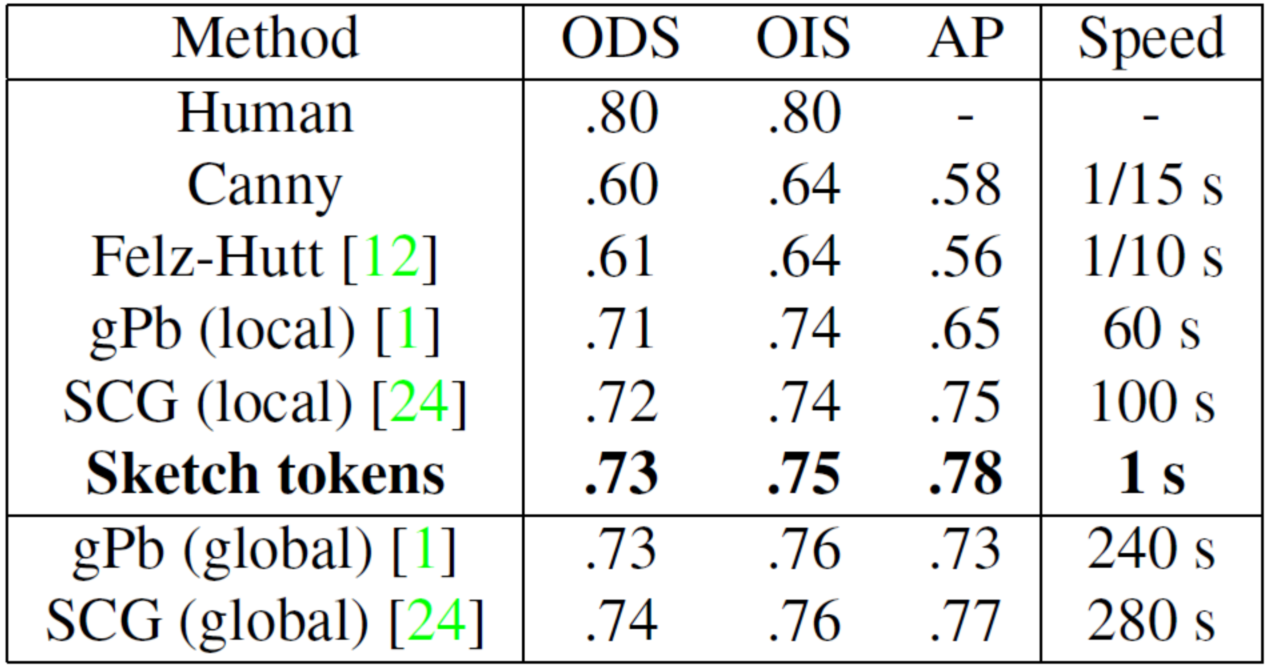
- Simply add the probability of all non-background sketch tokens
- Free parameter: number of sketch tokens
  - k = 1 works poorly, k = 16 and above work OK.



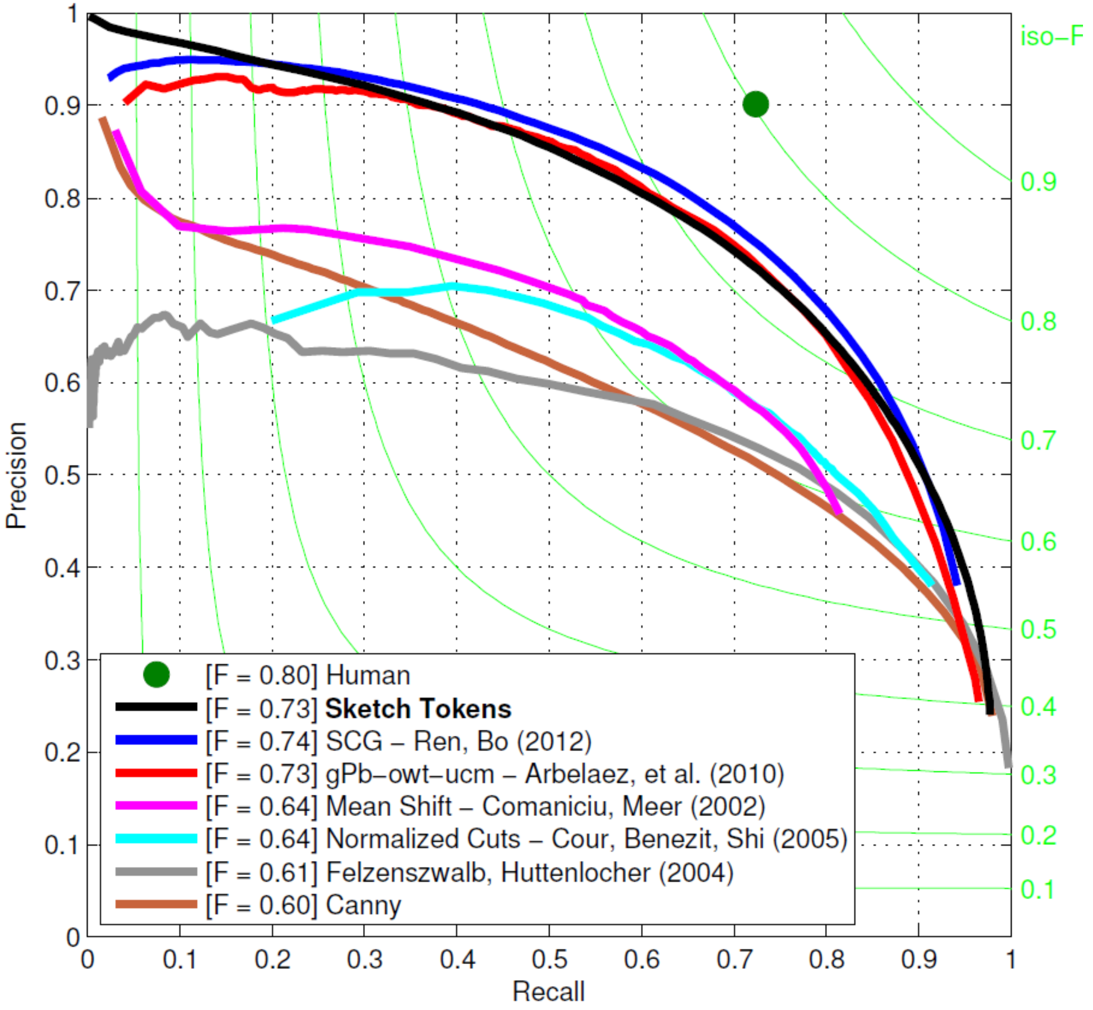
Input Image Ground Truth Sketch Tokens

Slide credit: J. Hays

Evaluation on BSDS



Method	ODS	OIS	AP	Speed
Human	.80	.80	-	-
Canny	.60	.64	.58	1/15 s
Felz-Hutt [12]	.61	.64	.56	1/10 s
gPb (local) [1]	.71	.74	.65	60 s
SCG (local) [24]	.72	.74	.75	100 s
Sketch tokens	**.73**	**.75**	**.78**	**1 s**
gPb (global) [1]	.73	.76	.73	240 s
SCG (global) [24]	.74	.76	.77	280 s



Slide credit: J. Hays

Summary of Sketch Tokens

- Distinct from previous work, cluster the *human annotations* to discover the mid-level structures that you want to detect.
- Train a classifier for every sketch token.
- Is as accurate as any other method while being 200 times faster and using no global information.

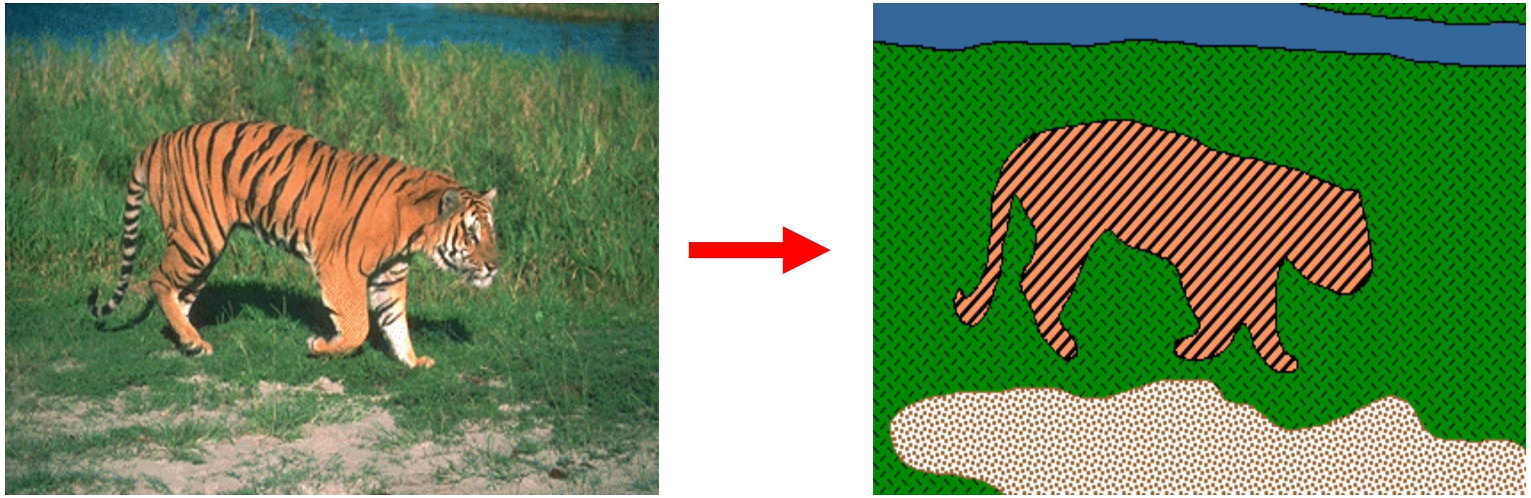
Slide credit: J. Hays

Today

- Linear Filtering
  - Gauss filter, Linear diffusion
- Edge Detection
  - Derivative filters, Laplacian of Gaussian, Canny edge detector
- Boundary Detection
- Segmentation
  - K-means clustering, Graph-theoretic segmentation

Image segmentation

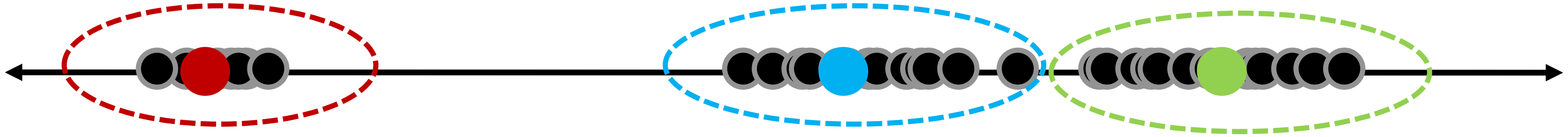
- Goal: identify groups of pixels that go together



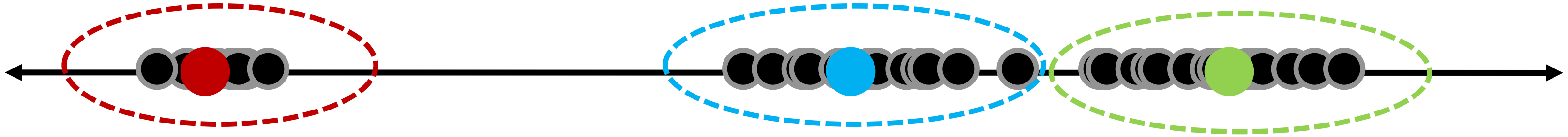
Slide credit: S. Seitz, K. Grauman

Clustering

- With this objective, it is a “chicken and egg” problem:
  - If we knew the **cluster centers**, we could allocate points to groups by assigning each to its closest center.



  - If we knew the **group memberships**, we could get the centers by computing the mean per group.



Slide credit: K. Grauman

Common similarity/distance measures

- P-norms
  - City Block (L1)
  - Euclidean (L2)
  - L-infinity

$$\|\mathbf{x}\|_p := \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}$$

$$\|\boldsymbol{x}\|_1 := \sum_{i=1}^{n} |x_i|$$

$$\|\boldsymbol{x}\| := \sqrt{x_1^2 + \cdots + x_n^2}$$

$$\|\mathbf{x}\|_\infty := \max\left(|x_1|, \ldots, |x_n|\right)$$

Here x_i is the distance btw. two points

- Mahalanobis
  - Scaled Euclidean

$$d(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{N} \frac{(x_i - y_i)^2}{\sigma_i^2}}$$

- Cosine distance

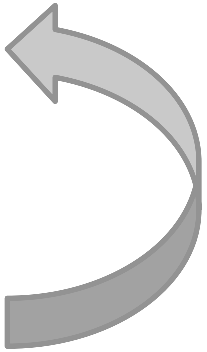
$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

Slide credit: D. Hoiem

K-means clustering

- Basic idea: randomly initialize the *k* cluster centers, and iterate between the two steps we just saw.

1. Randomly initialize the cluster centers, c_1, …, c_K
2. Given cluster centers, determine points in each cluster
   - For each point p, find the closest c_i. Put p into cluster i
3. Given points in each cluster, solve for c_i
   - Set c_i to be the mean of points in cluster i
4. If c_i have changed, repeat Step 2



Properties

- Will always converge to *some* solution
- can be a “local minimum”, it does not always find the global minimum of objective function:

$$\sum_{\text{clusters } i} \sum_{\text{points p in cluster } i} ||p - c_i||^2$$

Slide credit: S. Seitz

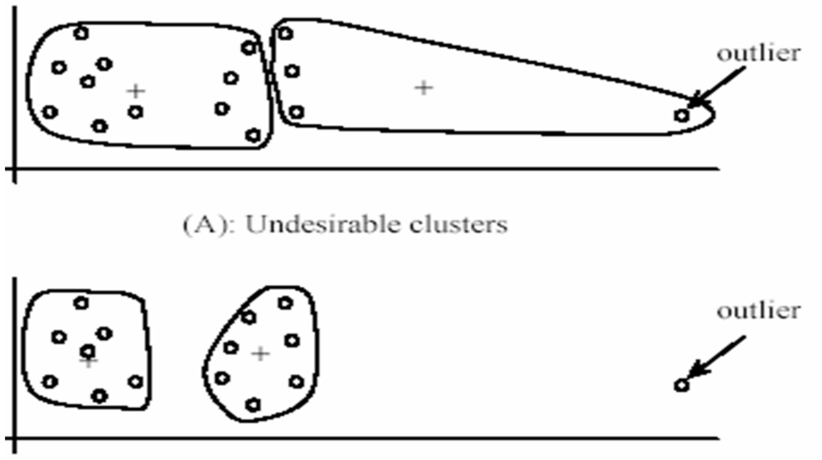
K-means clustering

Pros

- Simple, fast to compute
- Converges to local minimum of within-cluster squared error

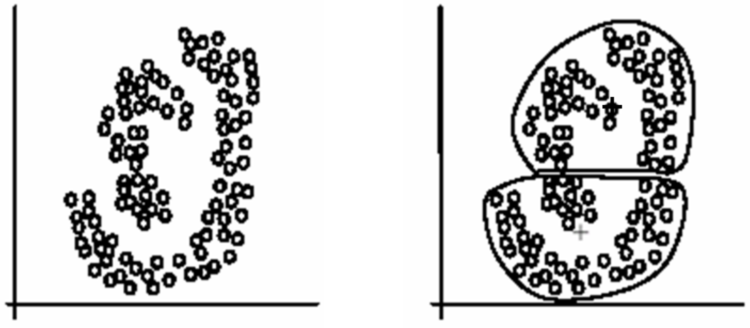
Cons/issues

- Setting k?
- Sensitive to initial centers
- Sensitive to outliers
- Detects spherical clusters
- Assuming means can be computed



(A): Undesirable clusters

(B): Ideal clusters



(A): Two natural clusters

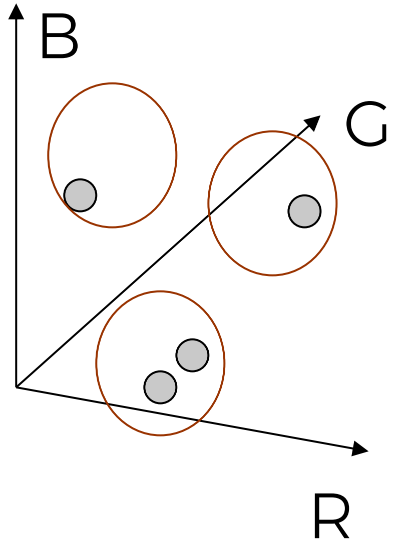
(B): *k*-means clusters

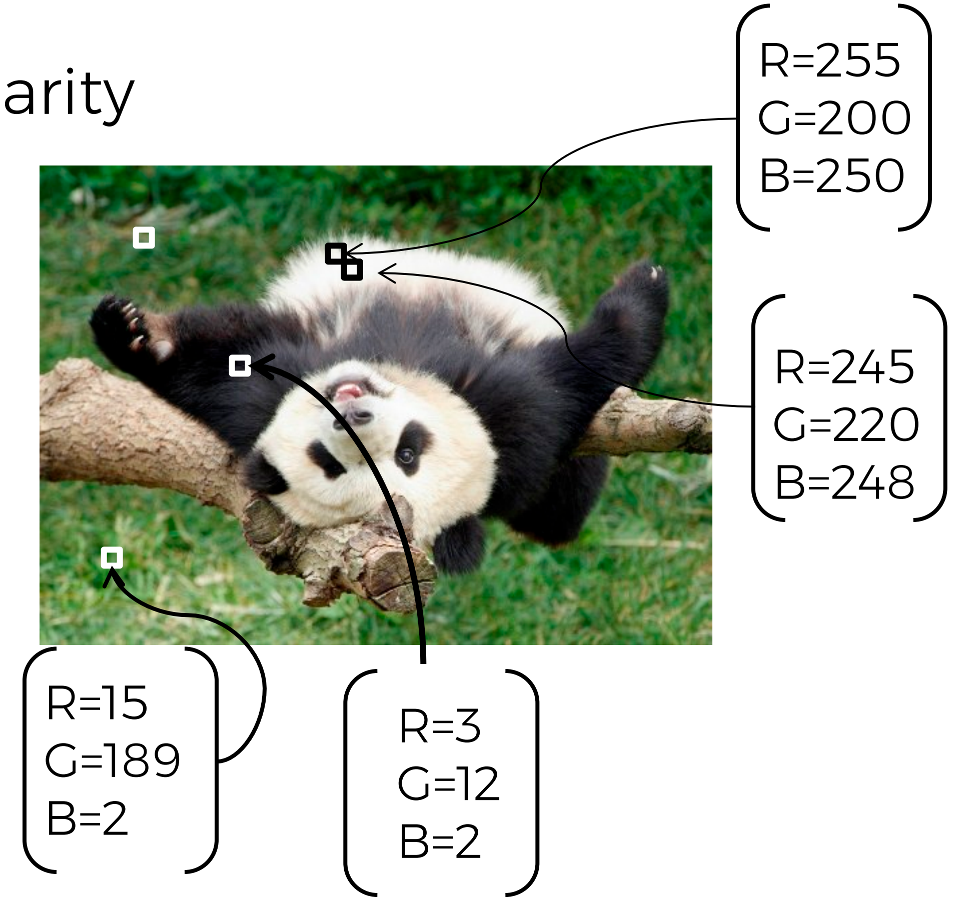
Slide credit: K. Grauman

Segmentation as clustering

Depending on what we choose as the feature space, we can group pixels in different ways.

Grouping pixels based on color similarity





R=255
G=200
B=250

R=245
G=220
B=248

R=15
G=189
B=2

R=3
G=12
B=2

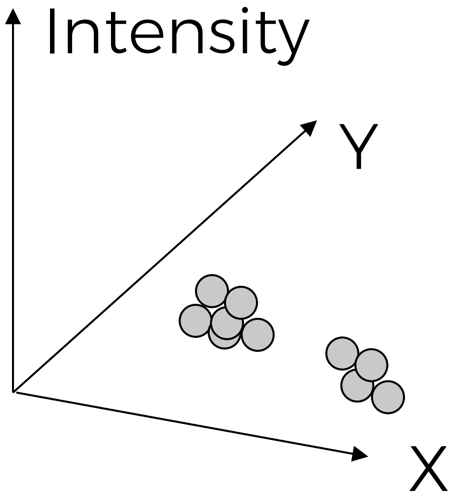
Feature space: color value (3-d)

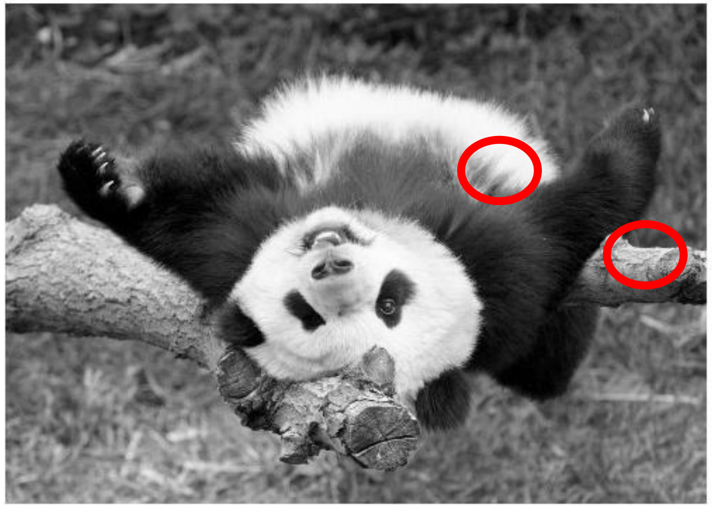
Slide credit: K. Grauman

Segmentation as clustering

Depending on what we choose as the feature space, we can group pixels in different ways.

Grouping pixels based on intensity+position similarity

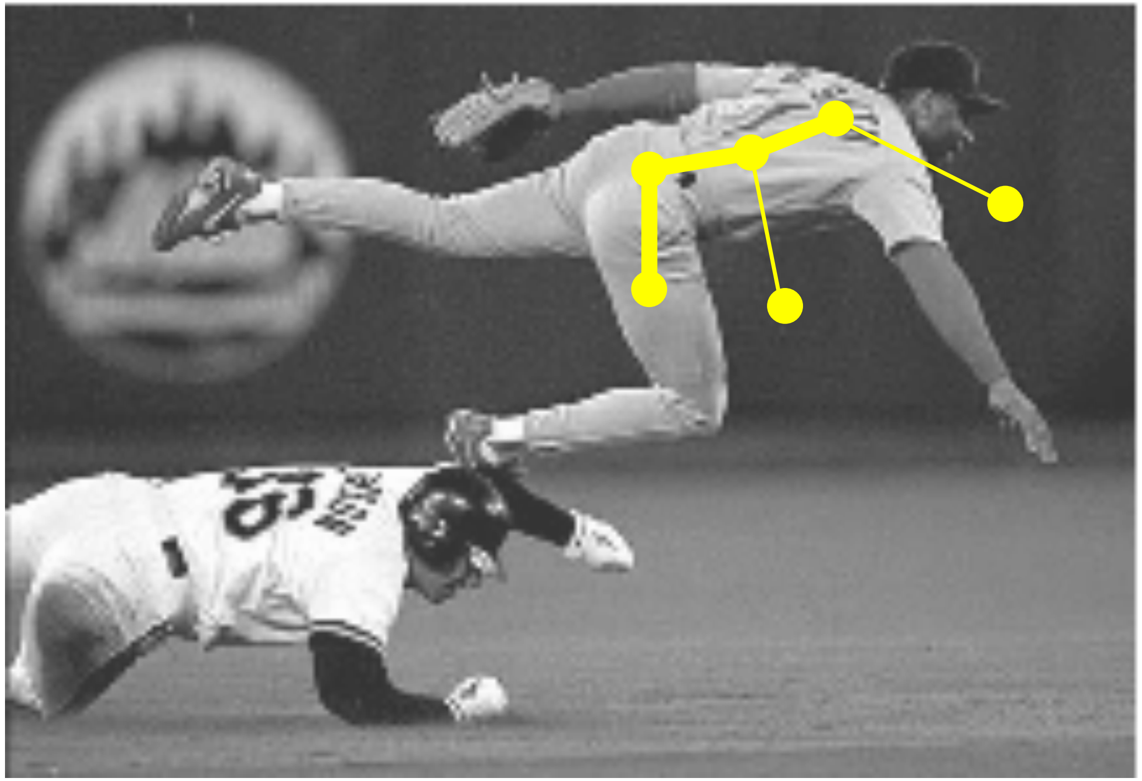




Both regions are black, but if we also include position (x,y), then we could group the two into distinct segments; way to encode both similarity & proximity.

Slide credit: K. Grauman

Graph-Theoretic Image Segmentation



Build a weighted graph G=(V,E) from image

V: image pixels

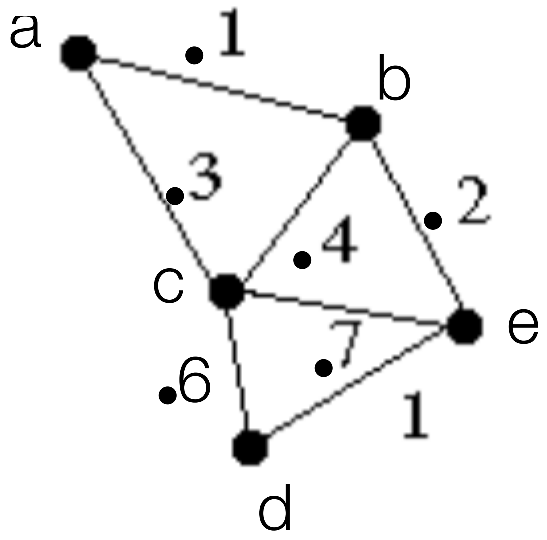
E: connections between pairs of nearby pixels

W_{ij}: probability that i & j belong to the same region

Segmentation = graph partition

Slide credit: B. Freeman and A. Torralba

A Weighted Graph and its Representation



Affinity Matrix

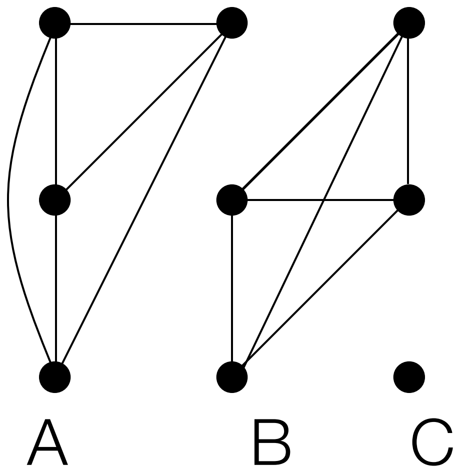
$$W = \begin{bmatrix} 1 & .1 & .3 & 0 & 0 \\ .1 & 1 & .4 & 0 & .2 \\ .3 & .4 & 1 & .6 & .7 \\ 0 & 0 & .6 & 1 & 1 \\ 0 & .2 & .7 & 1 & 1 \end{bmatrix}$$

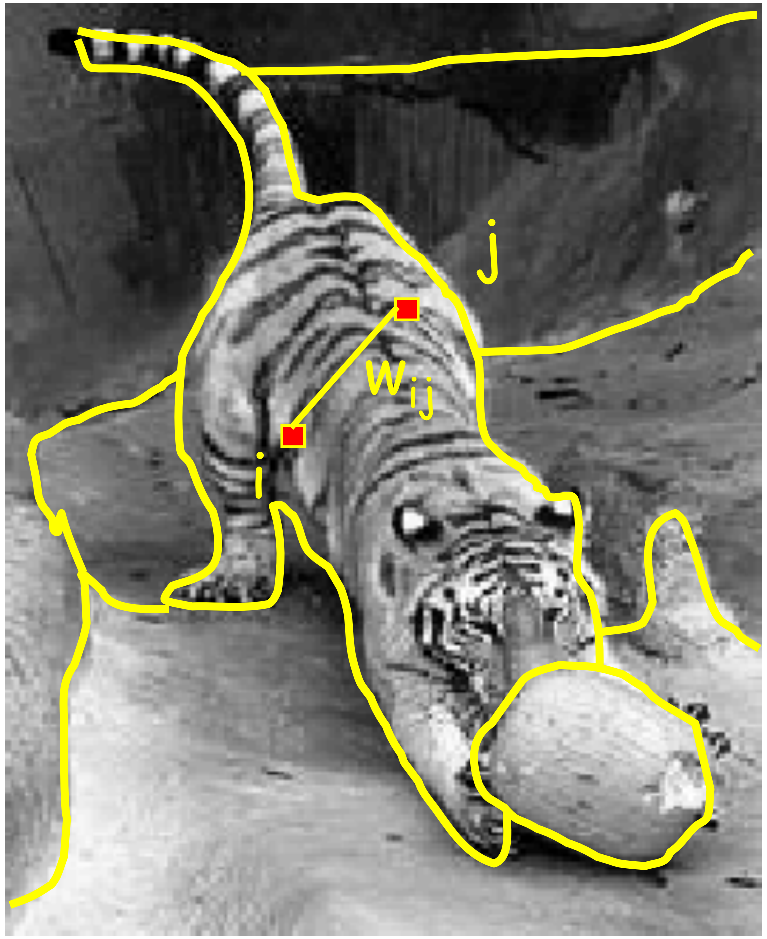
W_{ij}: probability that i & j belong to the same region

* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003

Slide credit: B. Freeman and A. Torralba

Segmentation by graph partitioning





- Break graph into segments
  - Delete links that cross between segments
  - Easiest to break links that have low affinity
    - similar pixels should be in the same segments
    - dissimilar pixels should be in different segments

Slide credit: S. Seitz

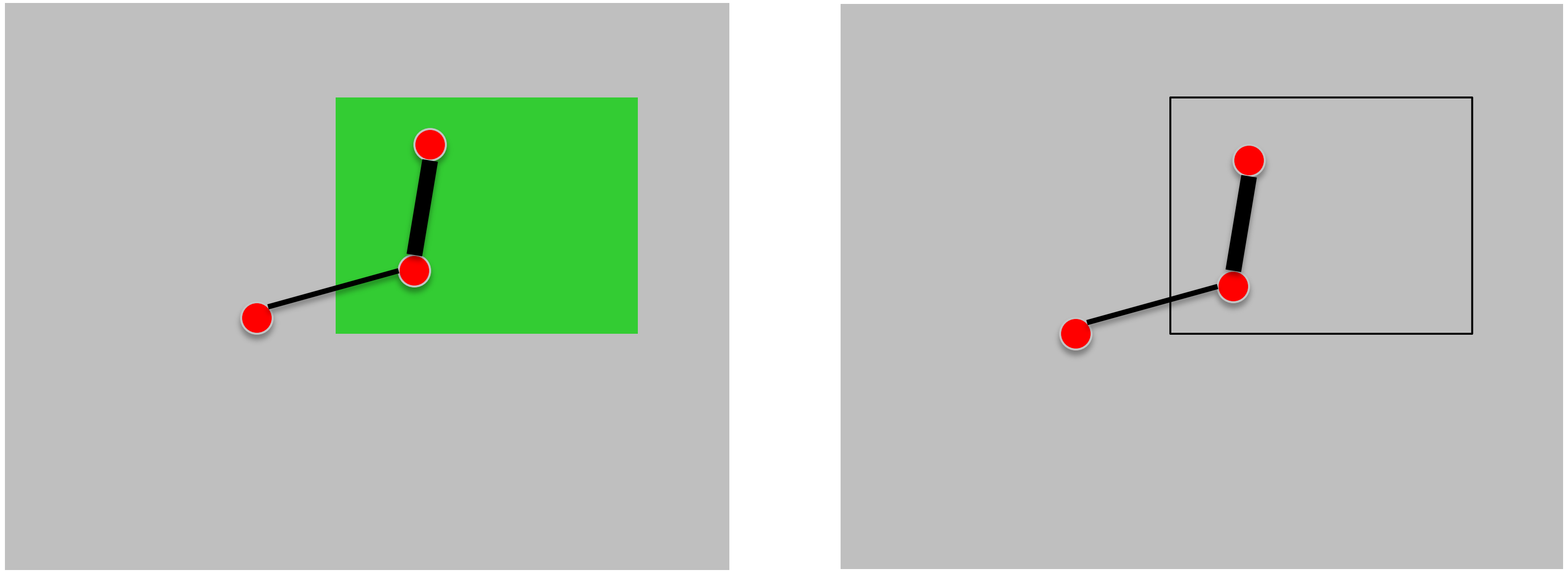
Affinity between pixels

Similarities among pixel descriptors $W_{ij} = \exp(-\| z_i - z_j \|^2 / \sigma^2)$

↖ σ = Scale factor…
it will hunt us later

Interleaving edges $W_{ij} = 1 - \max_{\text{Line between i and j}} Pb$

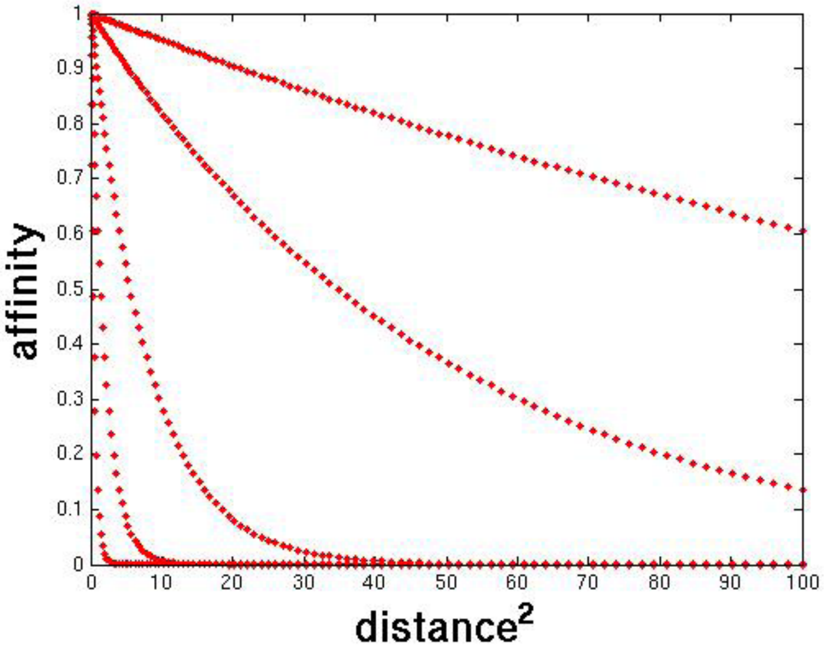
With Pb = probability of boundary

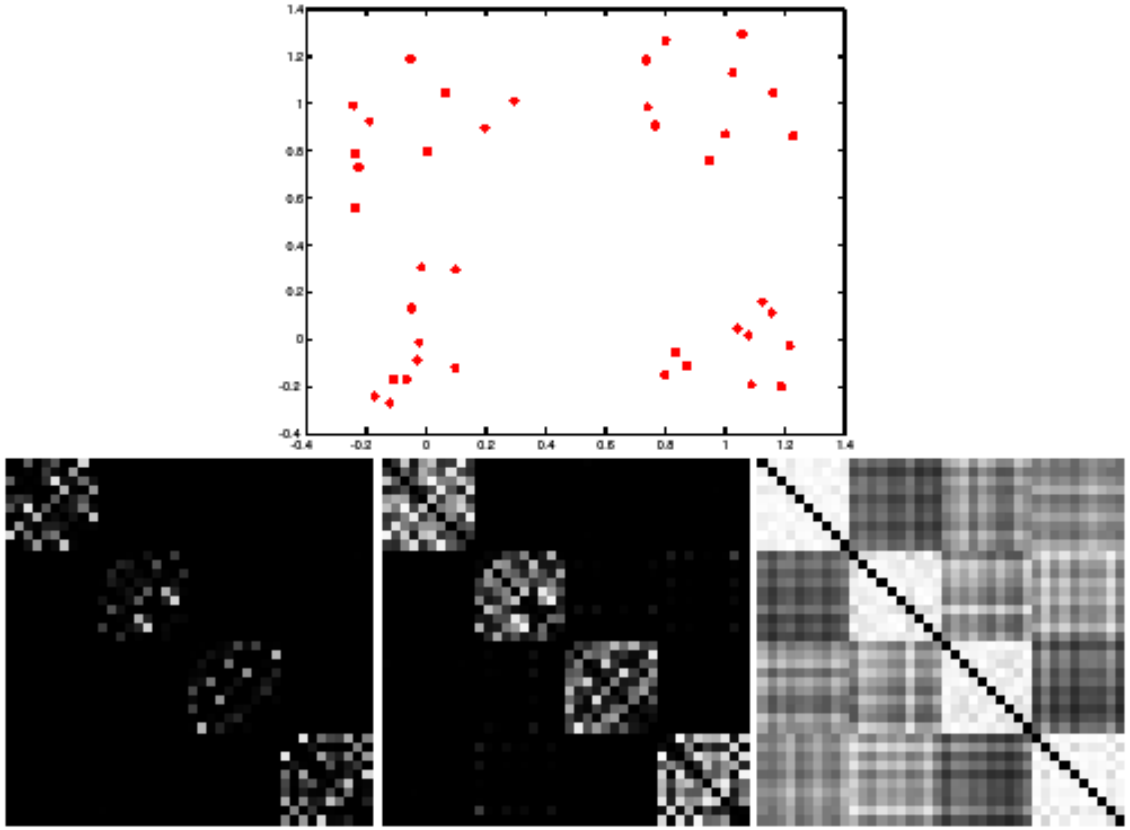


Slide credit: B. Freeman and A. Torralba

Scale affects affinity

- Small σ: group only nearby points
- Large σ: group far-away points

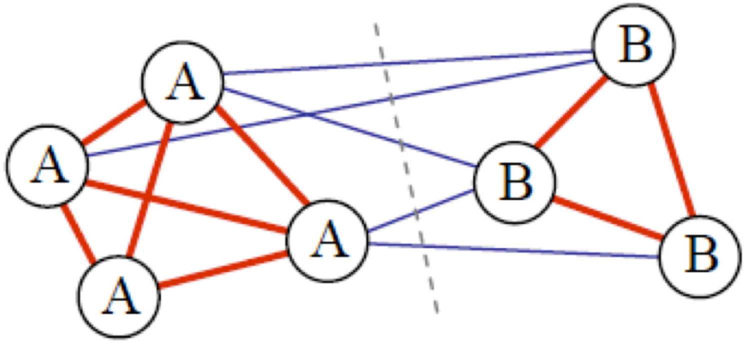




Slide credit: S. Lazebnik

Normalized cut

Write graph as V, one cluster as A and the other as B



$$\text{Ncut}(A,B) = \frac{\text{cut}(A,B)}{\text{assoc}(A,V)} + \frac{\text{cut}(A,B)}{\text{assoc}(B,V)}$$

cut(A,B) is sum of weights with one end in A and one end in B

assoc(A,V) is sum of all edges with one end in A.

$$cut(\mathrm{A},\mathrm{B}) = \sum_{u \in \mathrm{A}, v \in \mathrm{B}} \mathrm{W}(u,v),$$

with $A \cap B = \varnothing$

$$assoc(\mathrm{A},\mathrm{B}) = \sum_{u \in \mathrm{A}, v \in \mathrm{B}} \mathrm{W}(u,v)$$

A and B not necessarily disjoint

Slide credit: B. Freeman and A. Torralba

Normalized cut

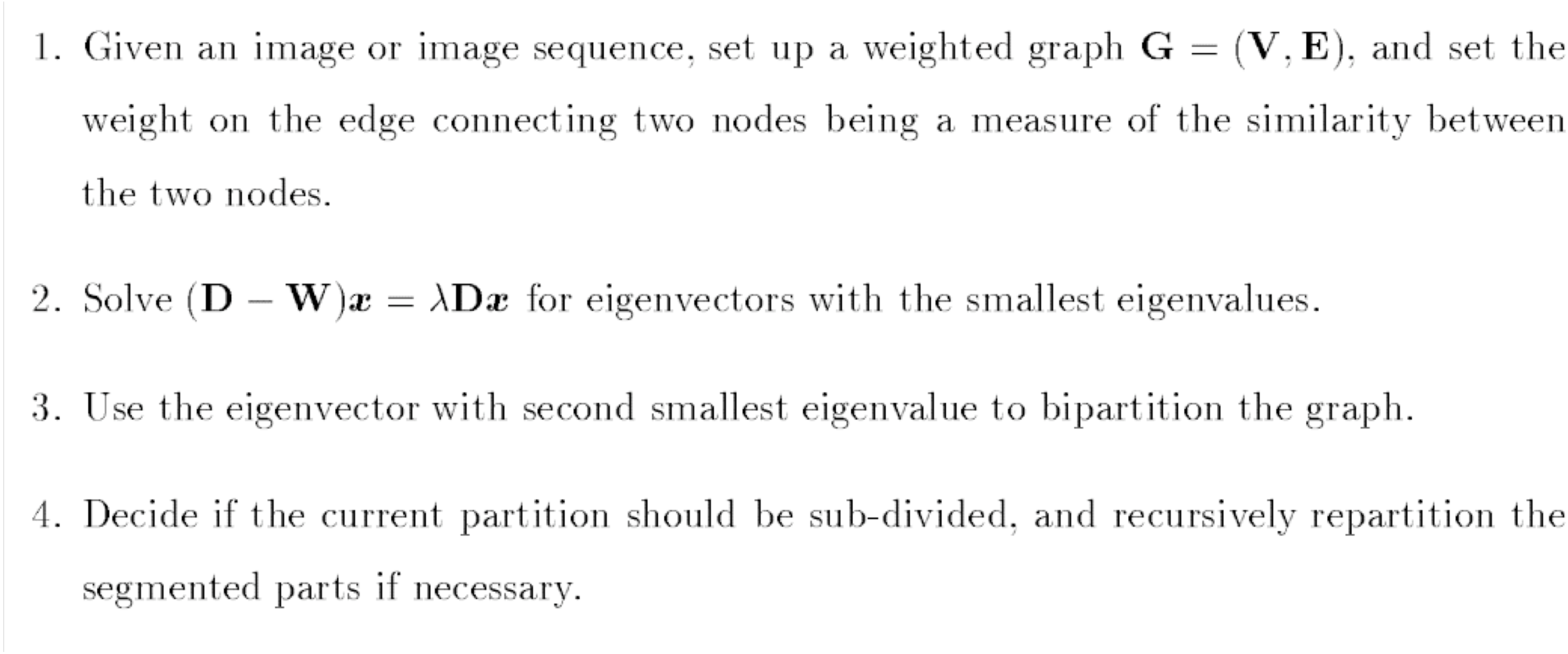
- Let *W* be the adjacency matrix of the graph
- Let *D* be the diagonal matrix with diagonal entries
  $D(i, i) = \Sigma_j W(i, j)$
- Then the normalized cut cost can be written as $$\frac{y^T(D-W)y}{y^T D y}$$

where *y* is an indicator vector whose value should be 1 in the *i*th position if the *i*th feature point belongs to A and a negative constant otherwise

J. Shi and J. Malik. Normalized cuts and image segmentation. PAMI 2000

Slide credit: S. Lazebnik

Normalized cut algorithm



1. Given an image or image sequence, set up a weighted graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, and set the weight on the edge connecting two nodes being a measure of the similarity between the two nodes.
2. Solve $(\mathbf{D} - \mathbf{W})\boldsymbol{x} = \lambda \mathbf{D}\boldsymbol{x}$ for eigenvectors with the smallest eigenvalues.
3. Use the eigenvector with second smallest eigenvalue to bipartition the graph.
4. Decide if the current partition should be sub-divided, and recursively repartition the segmented parts if necessary.

- In this formulation, the segmentation becomes a global process.
- Decisions about what is a boundary are not local (as in Canny edge detector)

Slide credit: B. Freeman and A. Torralba

Boundaries of image regions defined by a number of attributes

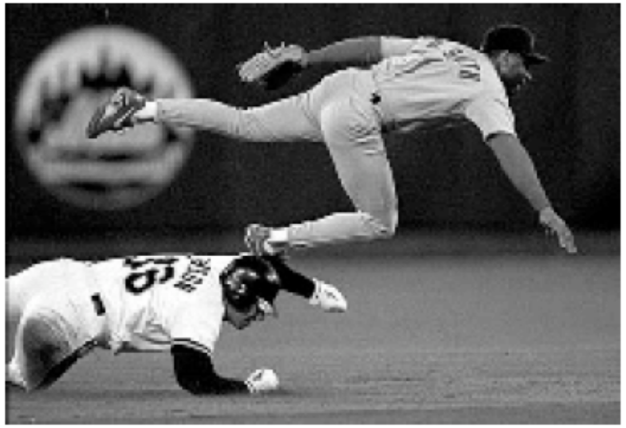
- Brightness/color
- Texture
- Motion
- Stereoscopic depth
- Familiar configuration

Slide credit: B. Freeman and A. Torralba

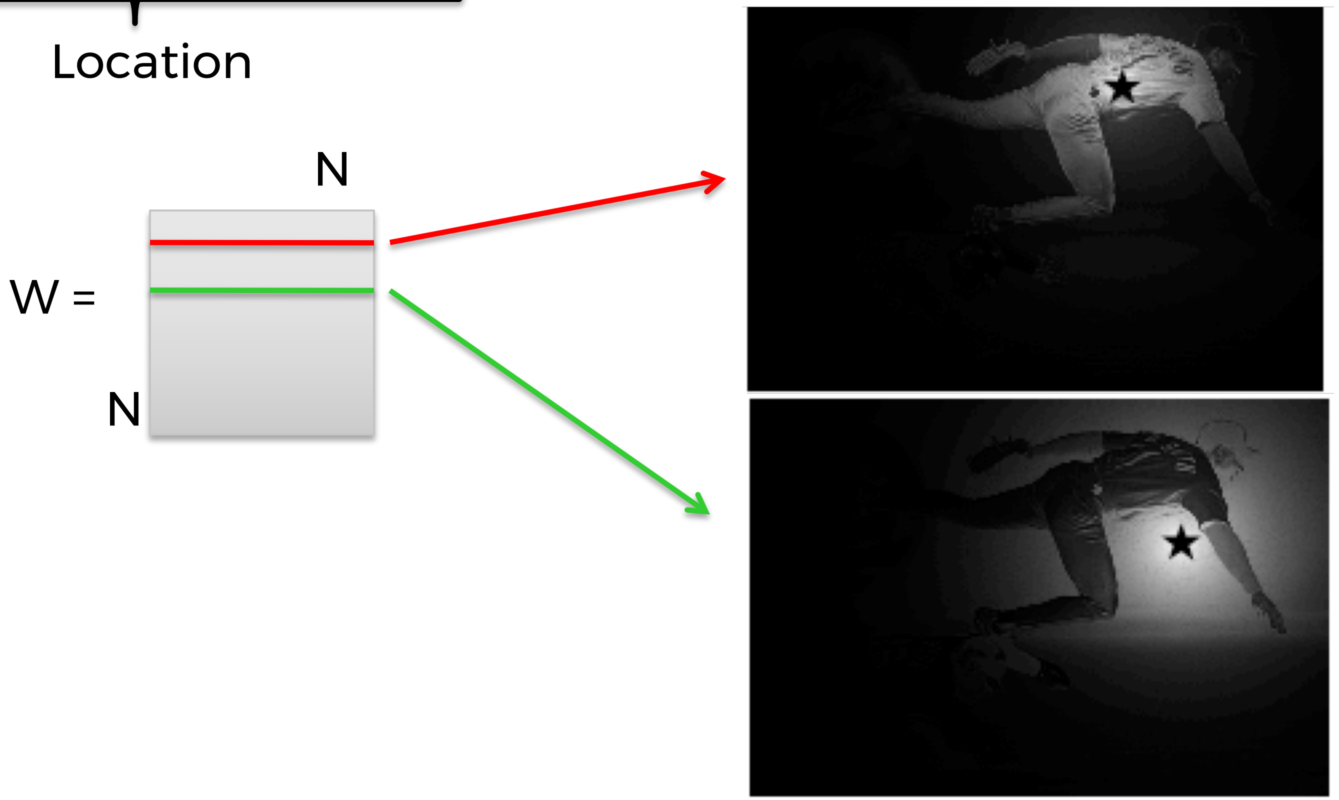
Example

Affinity:

$$w_{ij} = \underbrace{e^{\frac{-\|\boldsymbol{F}_{(i)}-\boldsymbol{F}_{(j)}\|_2^2}{\sigma_I}}}_{\text{brightness}} * \underbrace{\begin{cases} e^{\frac{-\|\boldsymbol{X}_{(i)}-\boldsymbol{X}_{(j)}\|_2^2}{\sigma_X}} & \text{if } \|\boldsymbol{X}(i)-\boldsymbol{X}(j)\|_2 < r \\ 0 & \text{otherwise} \end{cases}}_{\text{Location}}$$

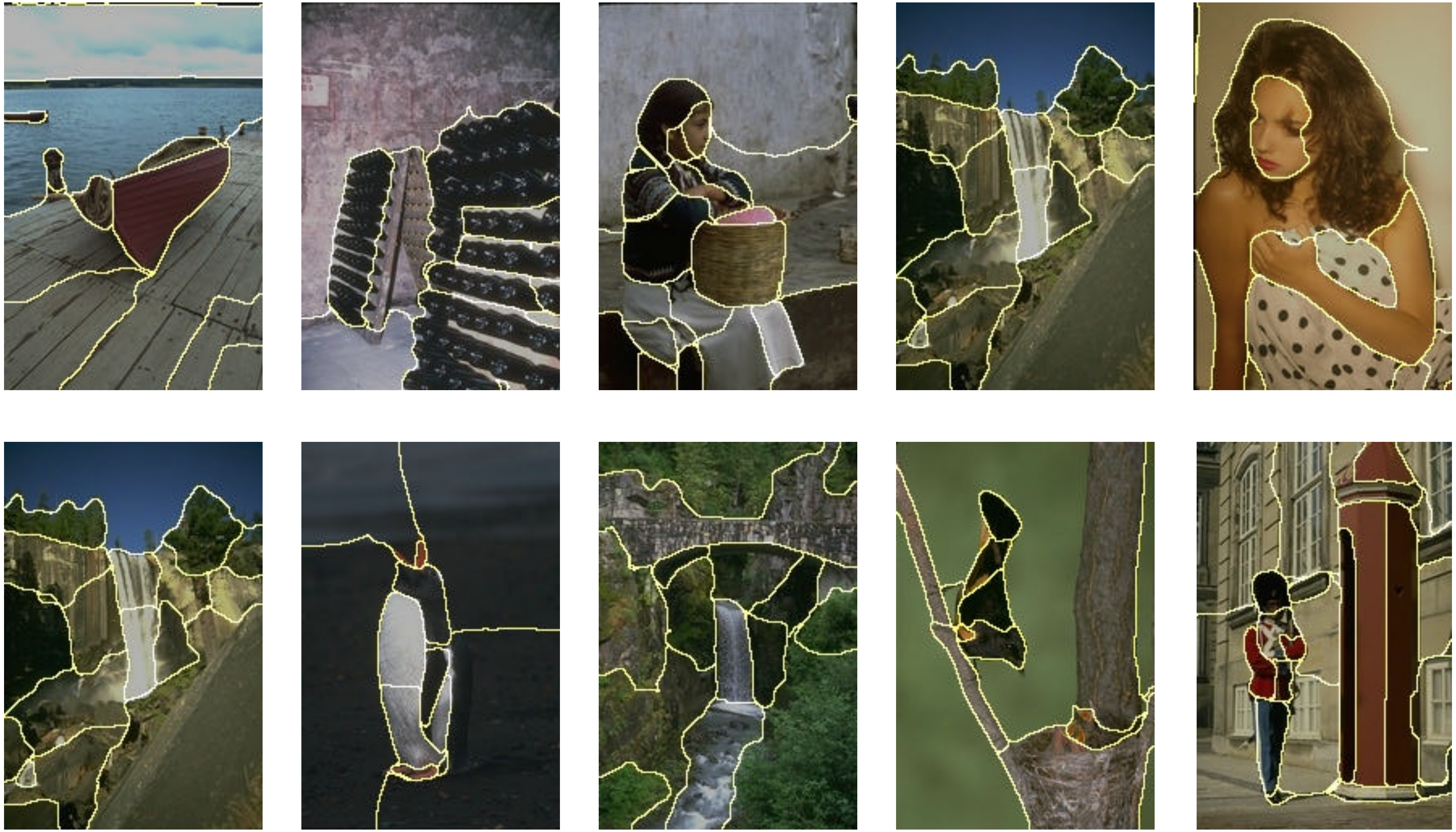


N pixels = ncols * nrows



Slide credit: B. Freeman and A. Torralba

Results: Berkeley Segmentation Engine



http://www.cs.berkeley.edu/~fowlkes/BSE/

Slide credit: S. Lazebnik

Normalized cuts: Pro and con

- Pros
  - Generic framework, can be used with many different features and affinity formulations
- Cons
  - High storage requirement and time complexity
  - Bias towards partitioning into equal segments

Slide credit: S. Lazebnik