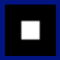
# Edges, Orientation, HOG and SIFT

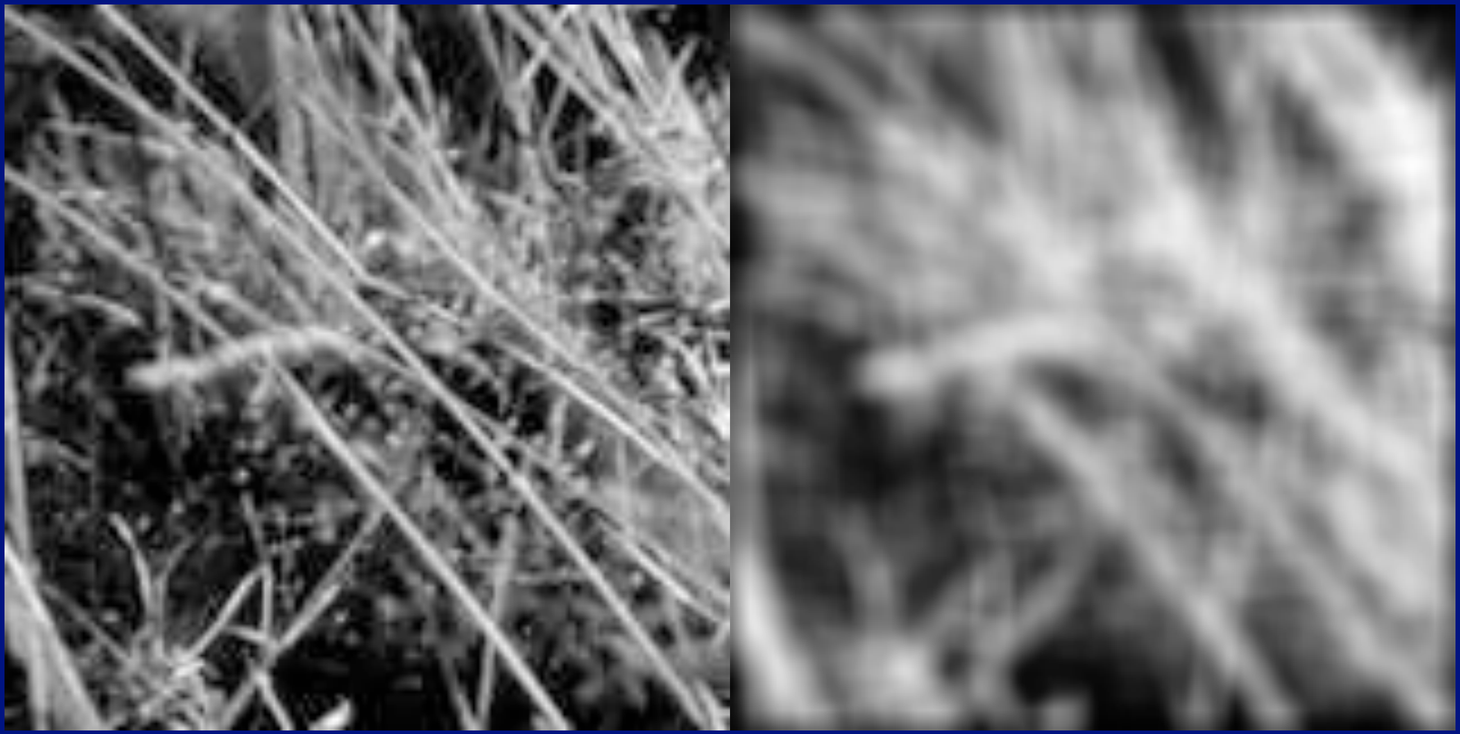
D.A. Forsyth

## Linear Filters

- Example: smoothing by averaging
  - form the average of pixels in a neighbourhood
- Example: smoothing with a Gaussian
  - form a weighted average of pixels in a neighbourhood
- Example:  finding a derivative
  - form a weighted average of pixels in a neighbourhood

## Smoothing by Averaging



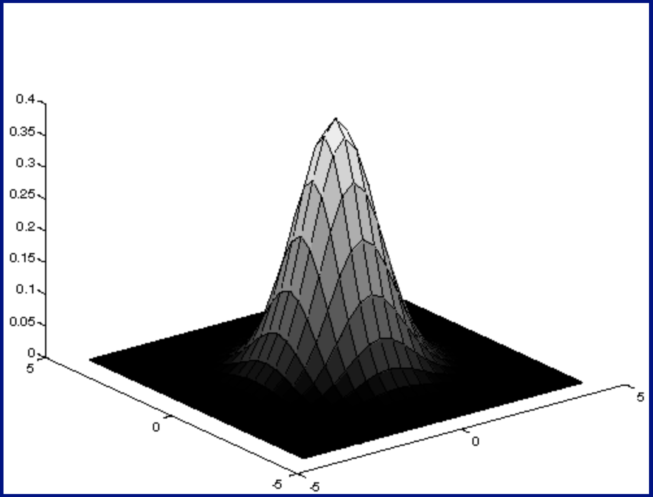


$$N_{ij} = \frac{1}{N} \Sigma_{uv} O_{i+u,j+v}$$

where u, v, is a window of N pixels in total centered at 0, 0

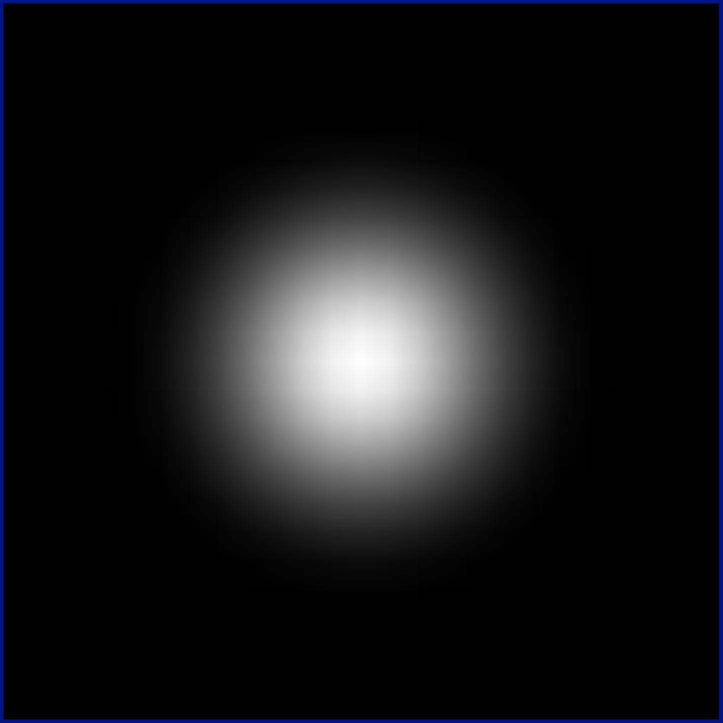
# Smoothing with a Gaussian

- Notice "ringing"
  - apparently, a grid is superimposed
- Smoothing with an average actually doesn't compare at all well with a defocussed lens
  - what does a point of light produce?



- A Gaussian gives a good model of a fuzzy blob

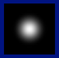
## Gaussian filter kernel

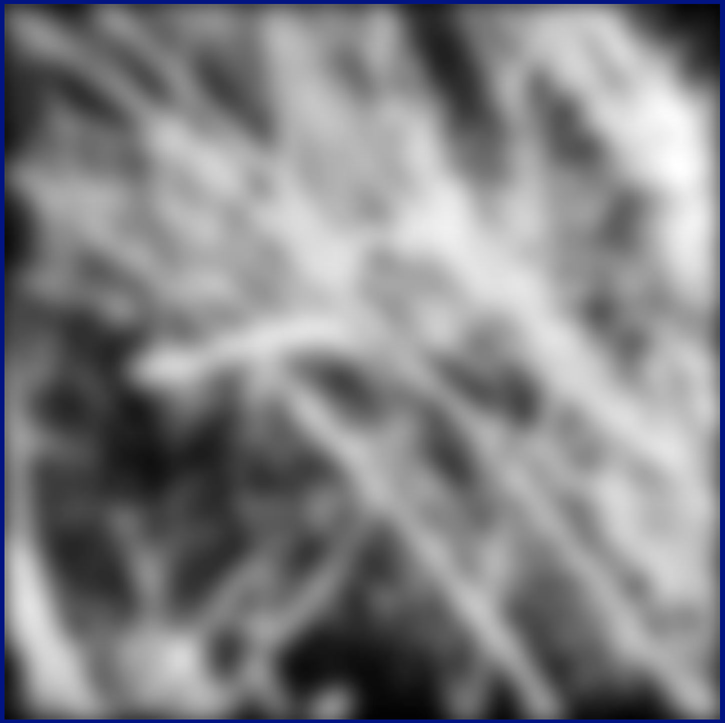


$$K_{uv} = \left(\frac{1}{2\pi\sigma^2}\right) \exp\left(\frac{-\left[u^2 + v^2\right]}{2\sigma^2}\right)$$

We're assuming the index can take negative values

## Smoothing with a Gaussian

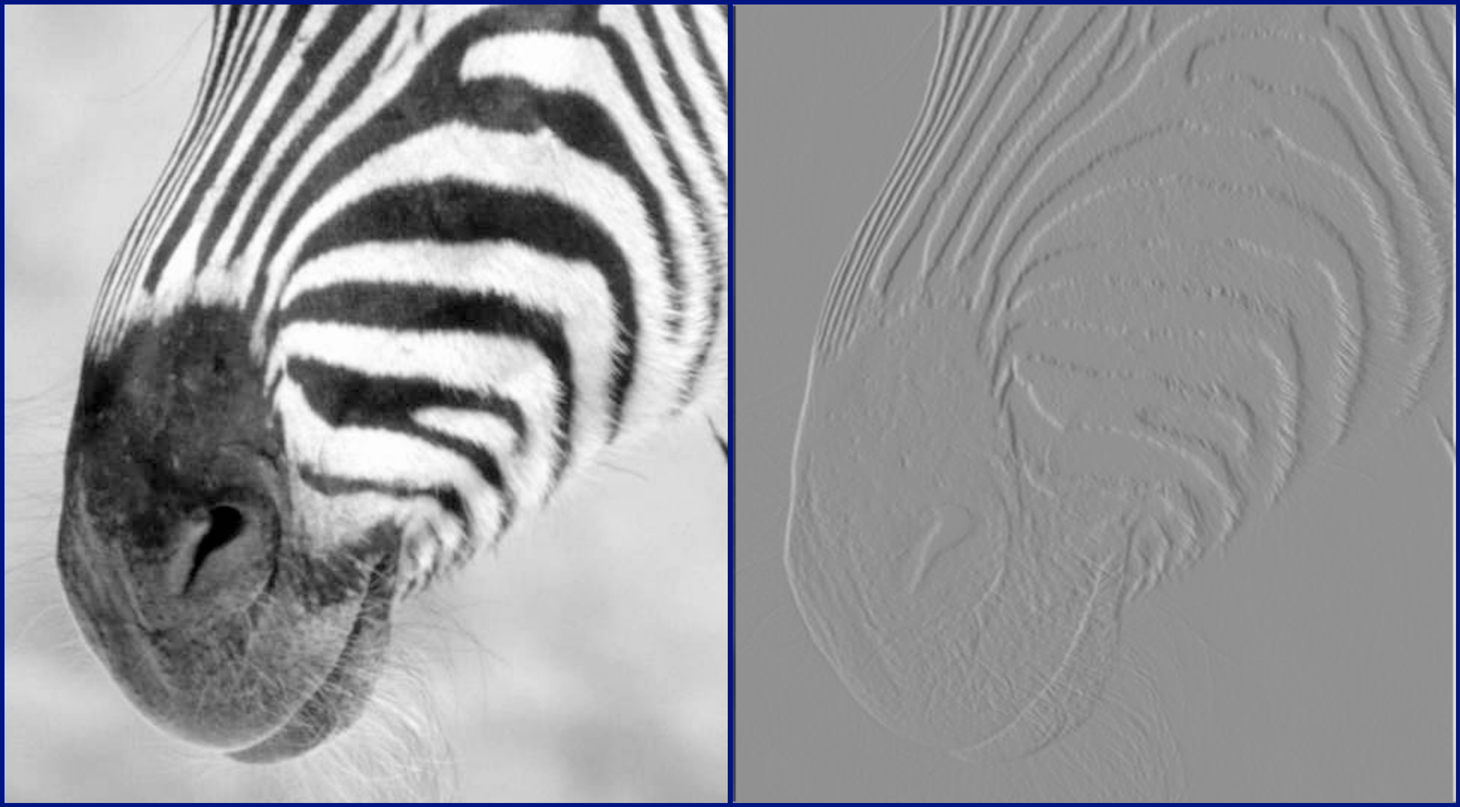




$$N_{ij} = \sum_{uv} O_{i-u,j-v} K_{uv}$$

Notice the curious looking form

## Finding derivatives



$$N_{ij} = \frac{1}{\Delta x}(I_{i+1,j} - I_{ij})$$

# Convolution

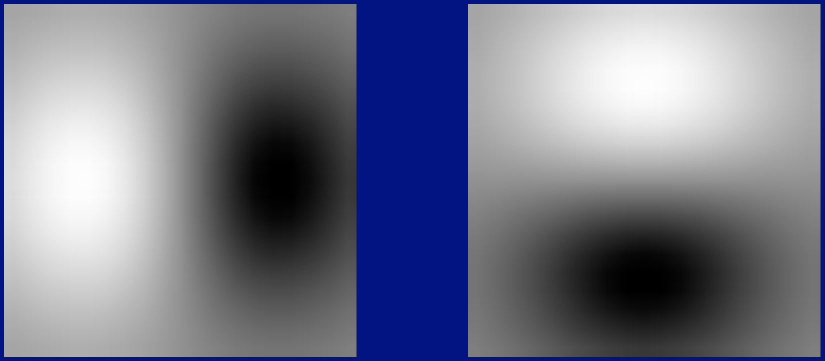
- Each of these involves a weighted sum of image pixels
- The set of weights is the same
    - we represent these weights as an image, H
    - H is usually called the kernel
- Operation is called convolution
    - it's associative
- Any linear shift-invariant operation can be represented by convolution
    - linear:  G(k f)=k G(f)
    - shift invariant:  G(Shift(f))=Shift(G(f))
    - Examples:
        - smoothing, differentiation, camera with a reasonable, defocussed lens system

$$N_{ij} = \sum_{uv} H_{uv} O_{i-u,j-v}$$

# Filters are templates

$$N_{ij} = \sum_{uv} H_{uv} O_{i-u,j-v}$$

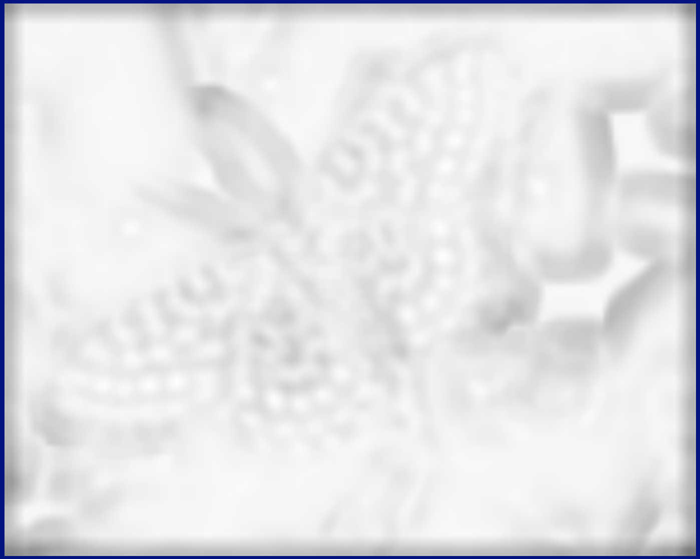
- At one point
  - output of convolution is a (strange) dot-product
- Filtering the image involves a dot product at each point
- Insight
  - filters look like the effects they are intended to find
  - filters find effects they look like

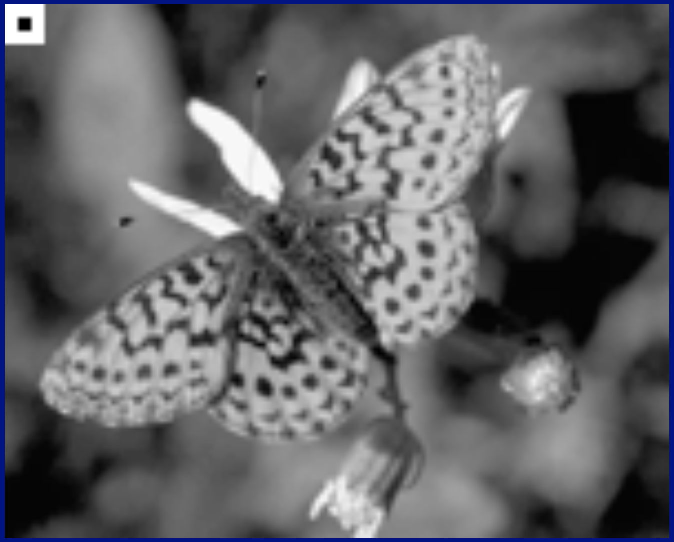
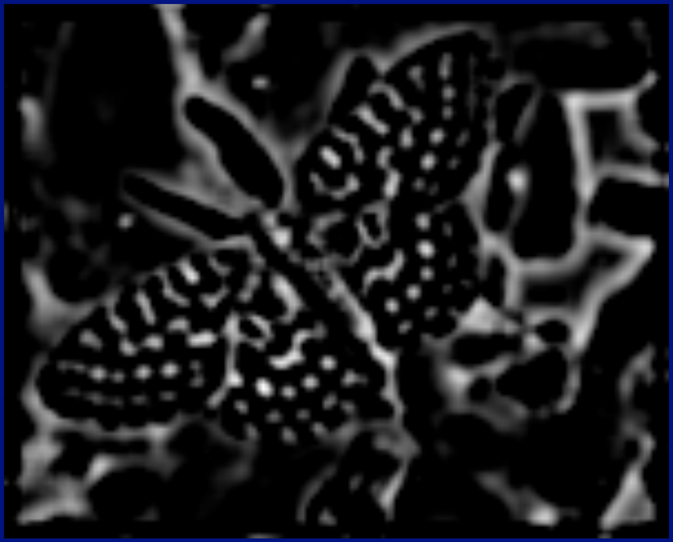


## Normalised correlation

- Think of filters of a dot product
  - now measure the angle
  - i.e normalised correlation output is filter output, divided by root sum of squares of values over which filter lies
  - Tricks:
    - ensure that filter has a zero response to a constant region
      - helps reduce response to irrelevant background
    - subtract image average when computing the normalising constant
    - absolute value deals with contrast reversal

normalised correlation
with non-zero mean filter

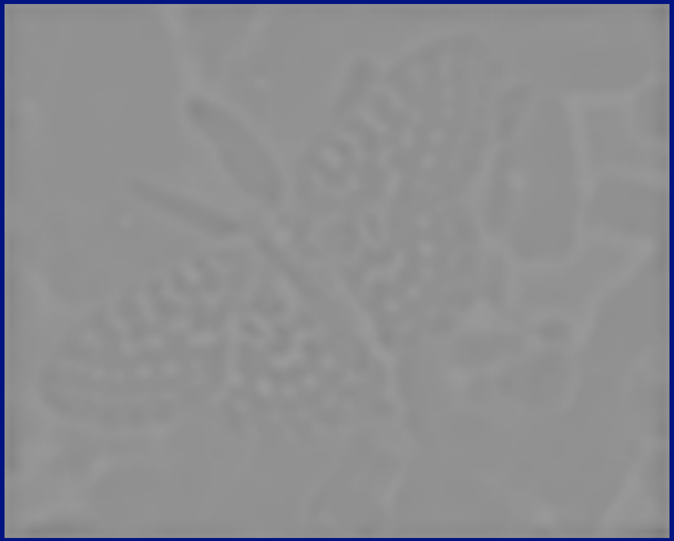
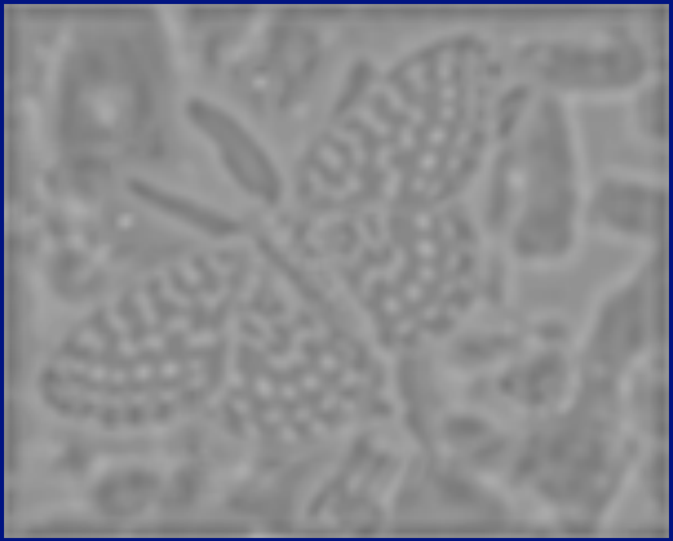


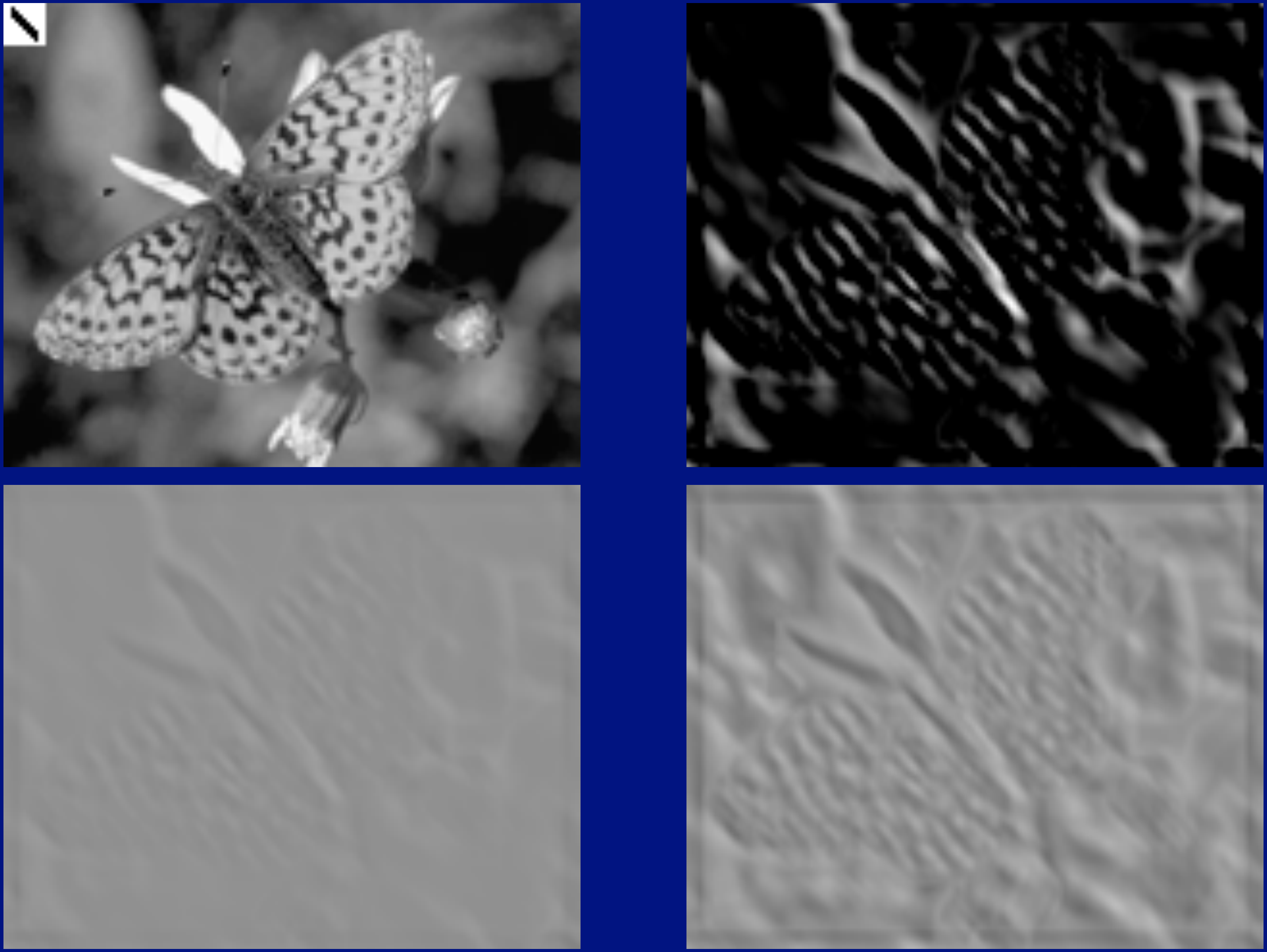



Positive responses

Zero mean image, -1:1 scale

Zero mean image, -max:max scale



# Finding hands

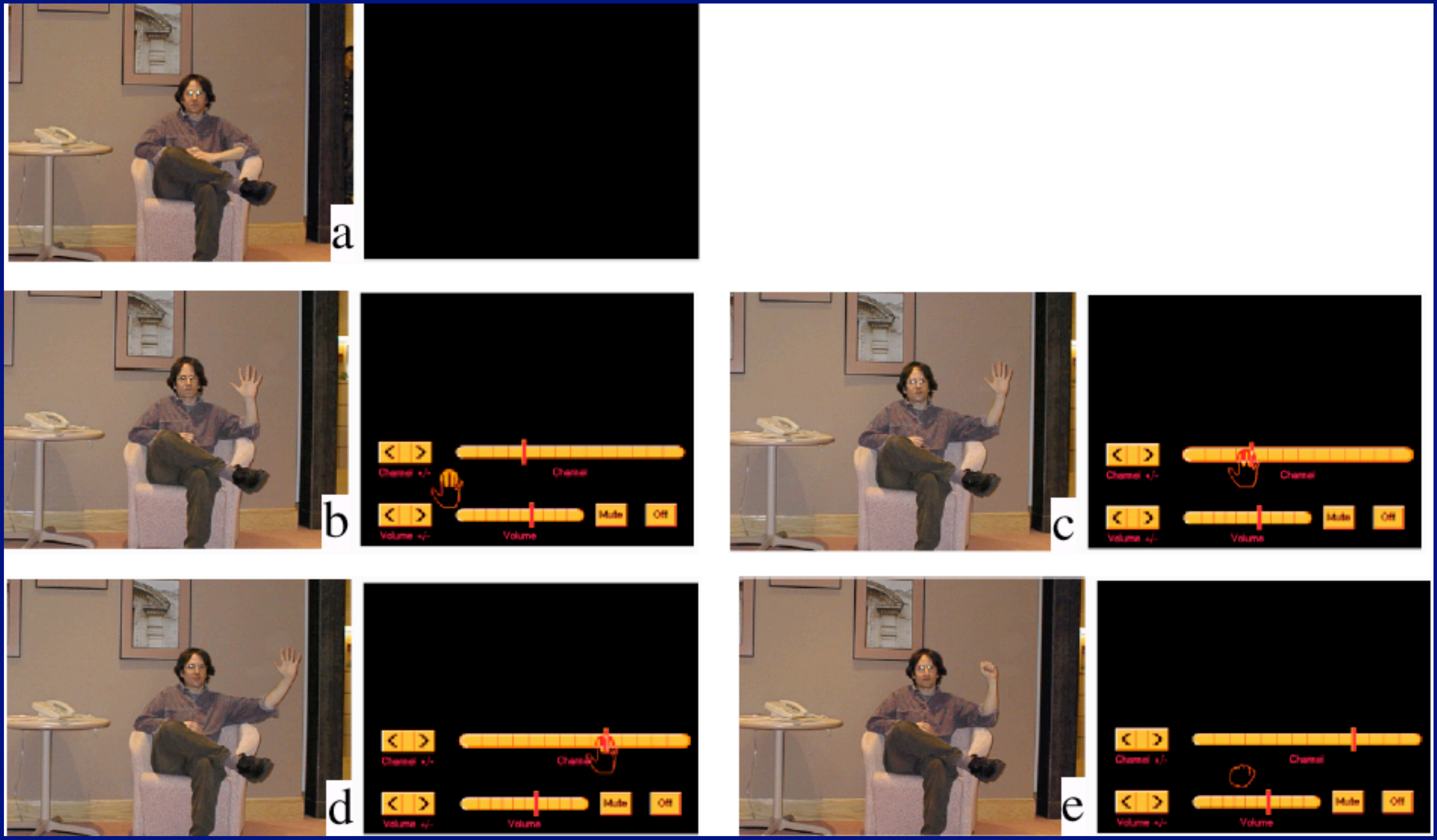
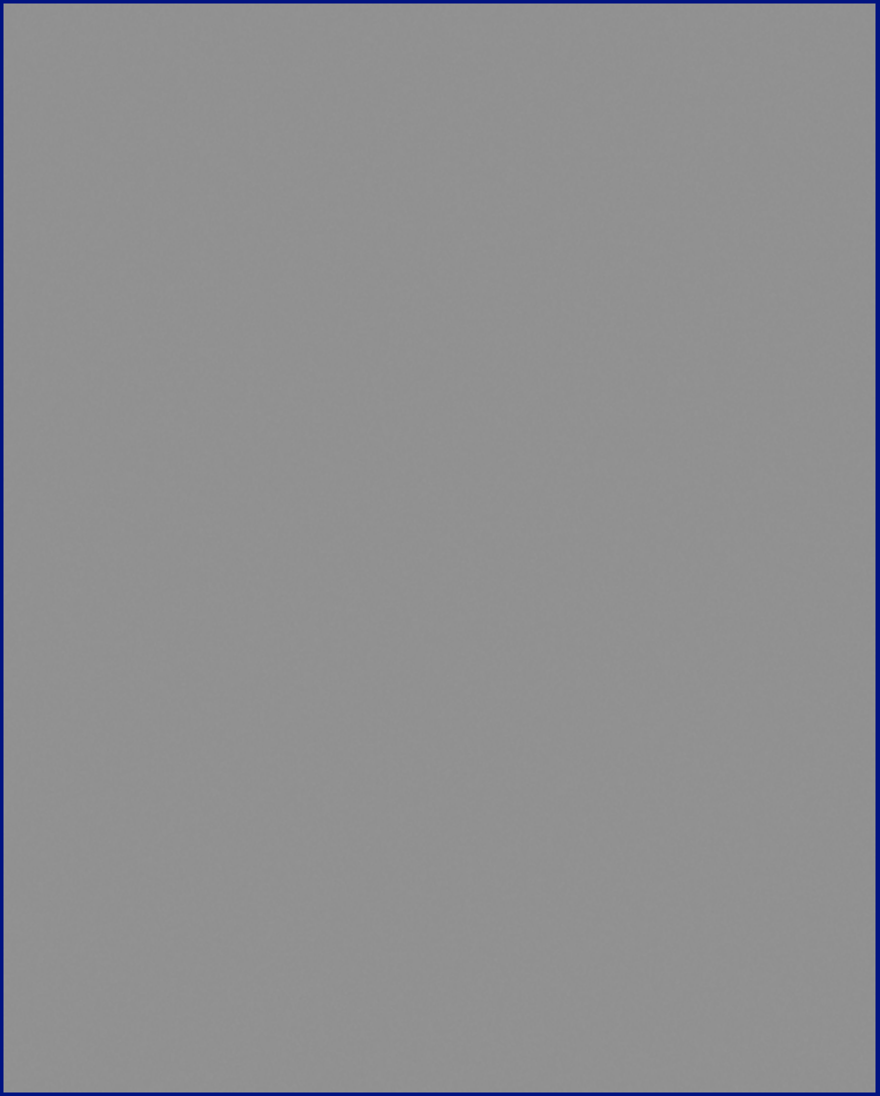


Figure from “Computer Vision for Interactive Computer Graphics,” W.Freeman et al, IEEE Computer Graphics and Applications, 1998

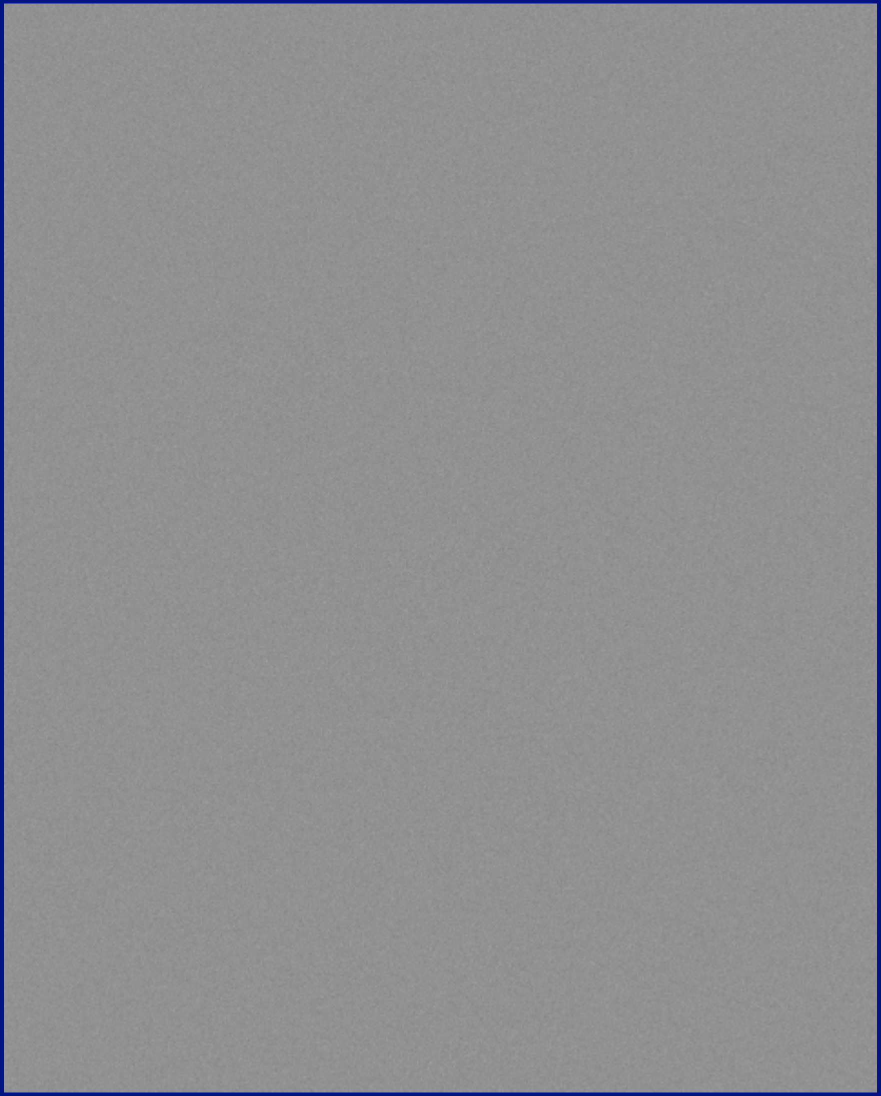
## Noise

- Simplest noise model
  - independent stationary additive Gaussian noise
  - the noise value at each pixel is given by an independent draw from the same normal probability distribution
- Issues
  - allows values greater than maximum camera output or less than zero
    - for small standard deviations, this isn't too much of a problem
  - independence may not be justified (e.g. damage to lens)
  - may not be stationary (e.g. thermal gradients in the ccd)

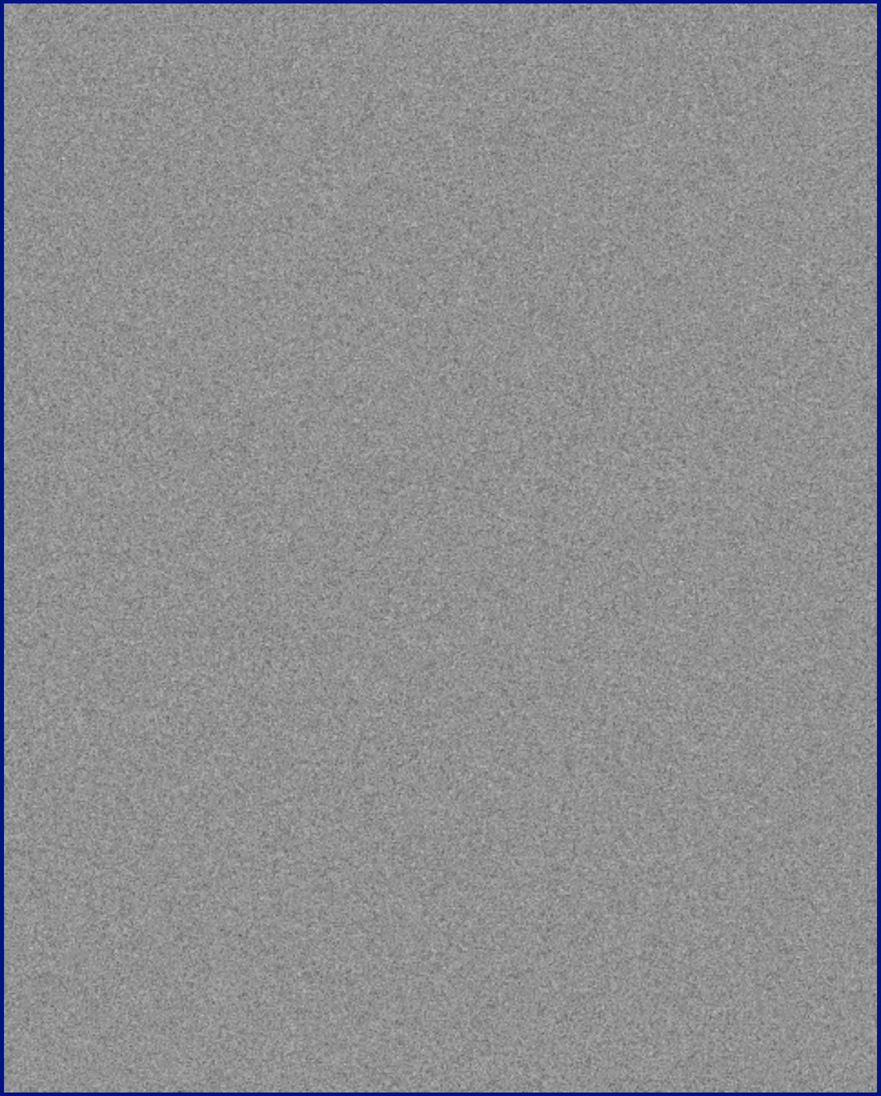
sigma=1



sigma=4



sigma=
16



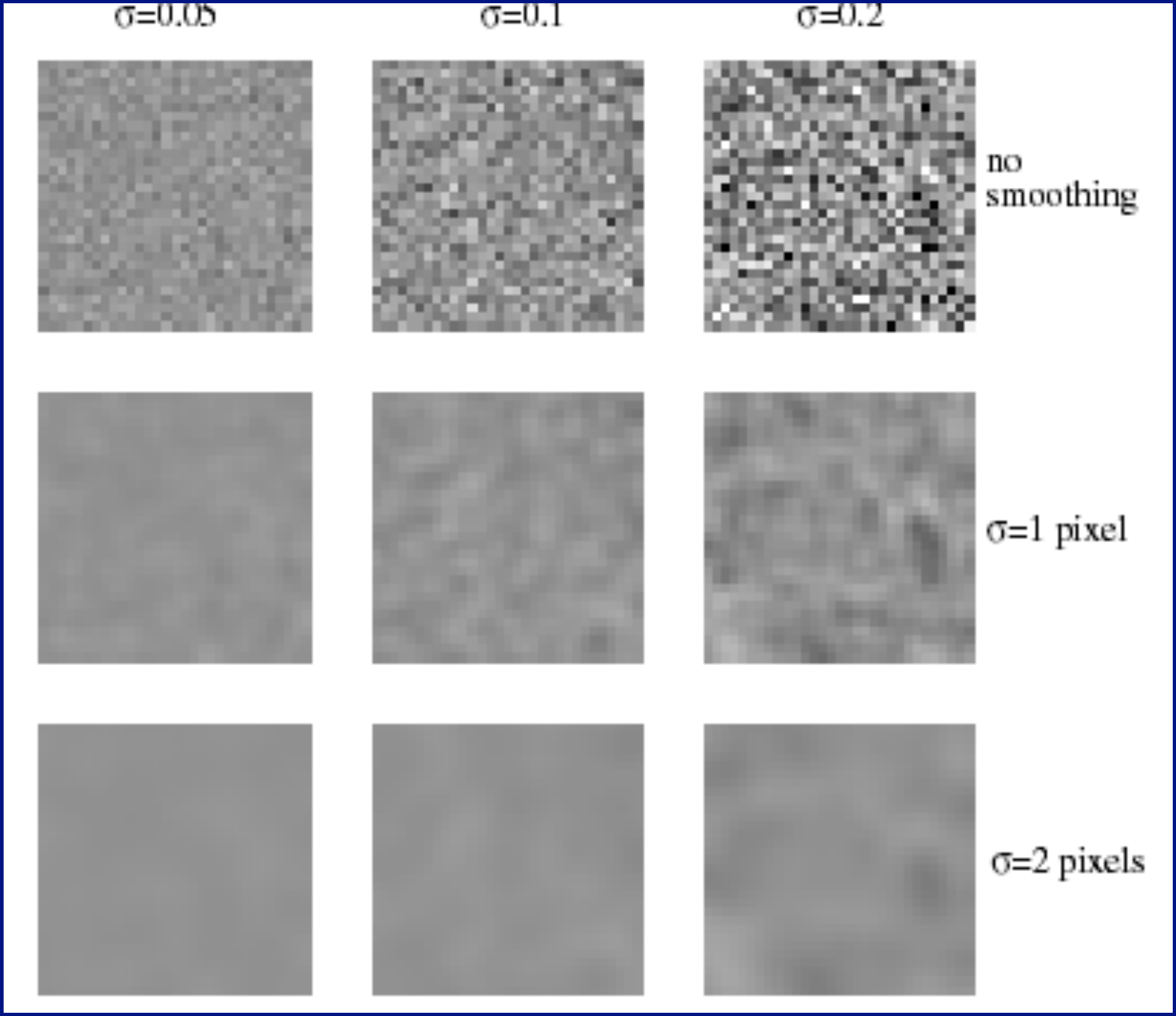
sigma=1

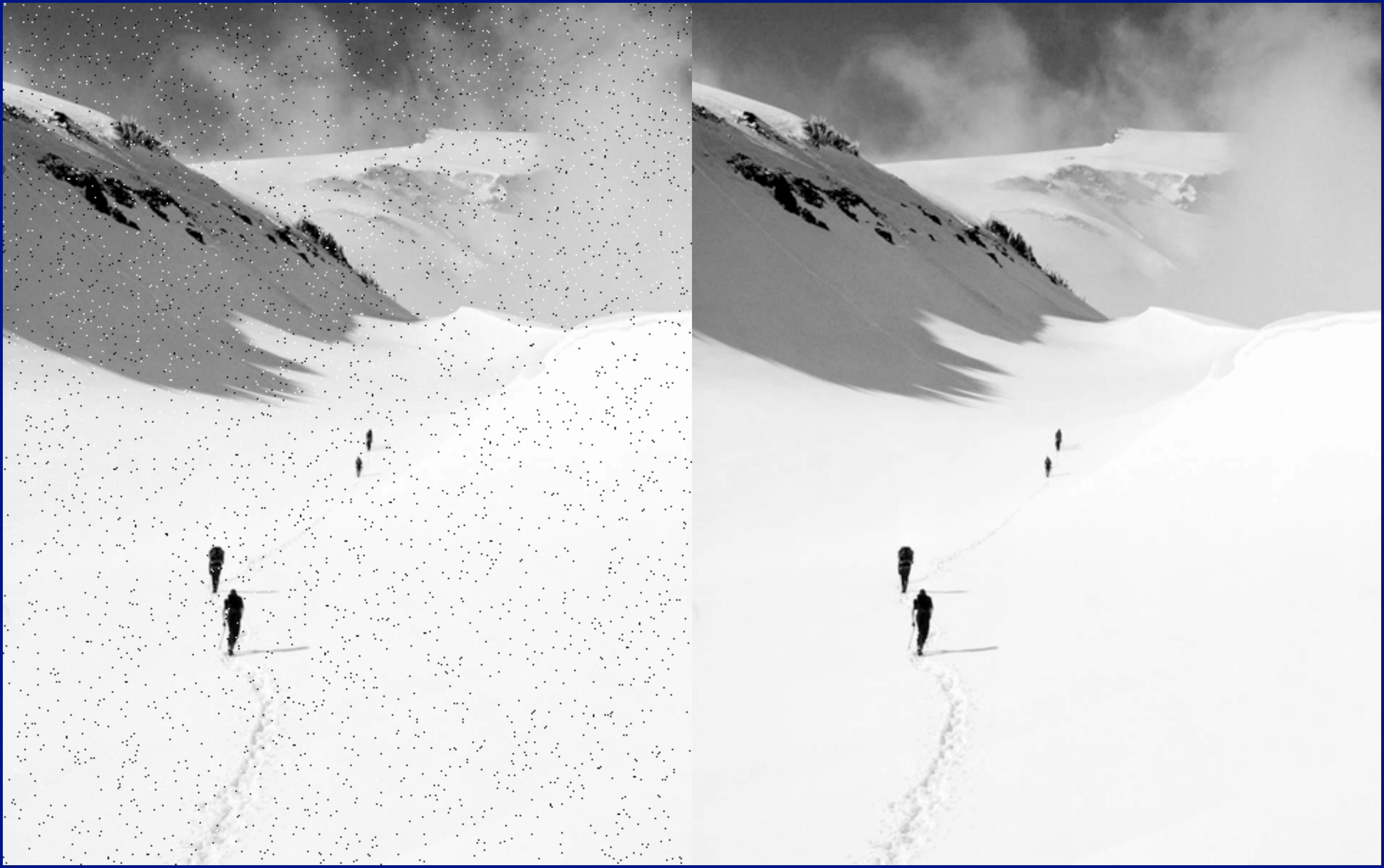
sigma=16

## Smoothing reduces noise

- Generally expect pixels to “be like” their neighbours
  - surfaces turn slowly
  - relatively few reflectance changes
- Expect noise to be independent from pixel to pixel
  - Implies that smoothing suppresses noise, for appropriate noise models
- Scale
  - the parameter in the symmetric Gaussian
  - as this parameter goes up, more pixels are involved in the average
  - and the image gets more blurred
  - and noise is more effectively suppressed

$$K_{uv} = \left(\frac{1}{2\pi\sigma^2}\right)\exp\left(\frac{-\left[u^2+v^2\right]}{2\sigma^2}\right)$$


σ=0.05
σ=0.1
σ=0.2
no smoothing
σ=1 pixel
σ=2 pixels



# Smoothing and scales

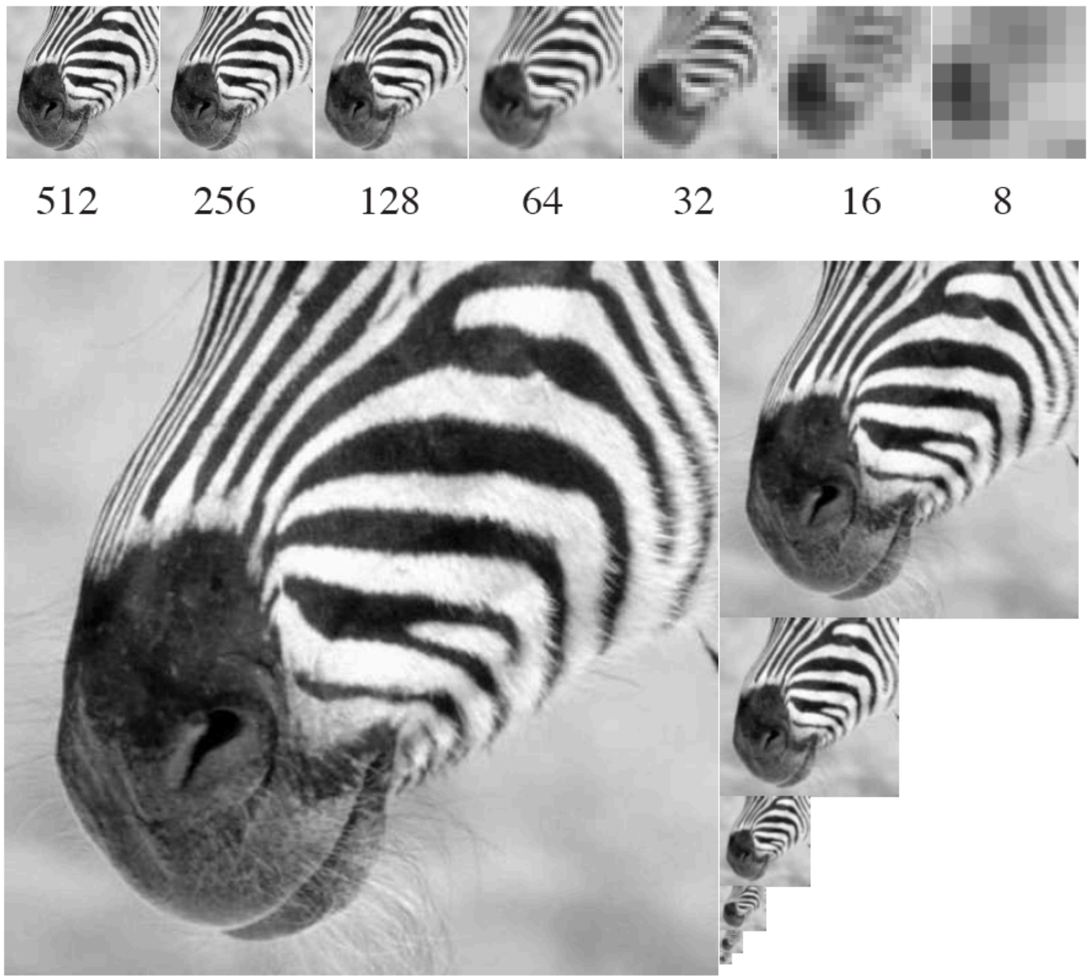
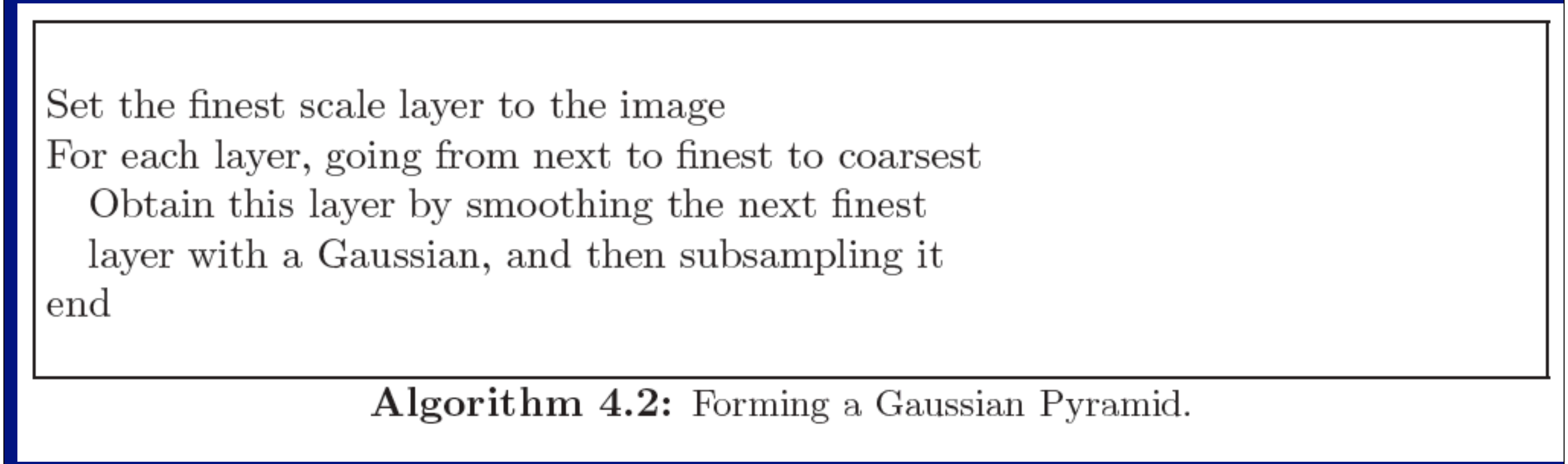


FIGURE 4.17: A Gaussian pyramid of images running from 512x512 to 8x8. On the top row, we have shown each image at the same size (so that some have bigger pixels than others), and the lower part of the figure shows the images to scale. Notice that if we convolve each image with a fixed-size filter, it responds to quite different phenomena. An 8x8 pixel block at the finest scale might contain a few hairs; at a coarser scale, it might contain an entire stripe; and at the coarsest scale, it contains the animal's muzzle.

## Smoothing and scales



```
Set the finest scale layer to the image
For each layer, going from next to finest to coarsest
   Obtain this layer by smoothing the next finest
   layer with a Gaussian, and then subsampling it
end
```

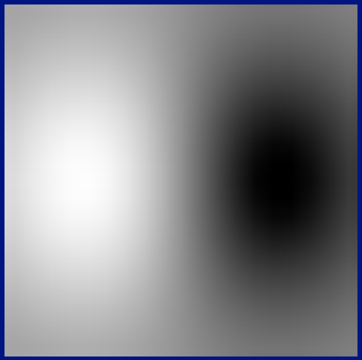
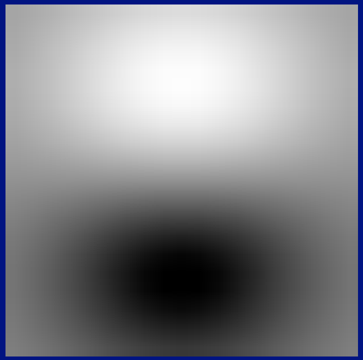
**Algorithm 4.2:** Forming a Gaussian Pyramid.

## Representing image changes: Edges

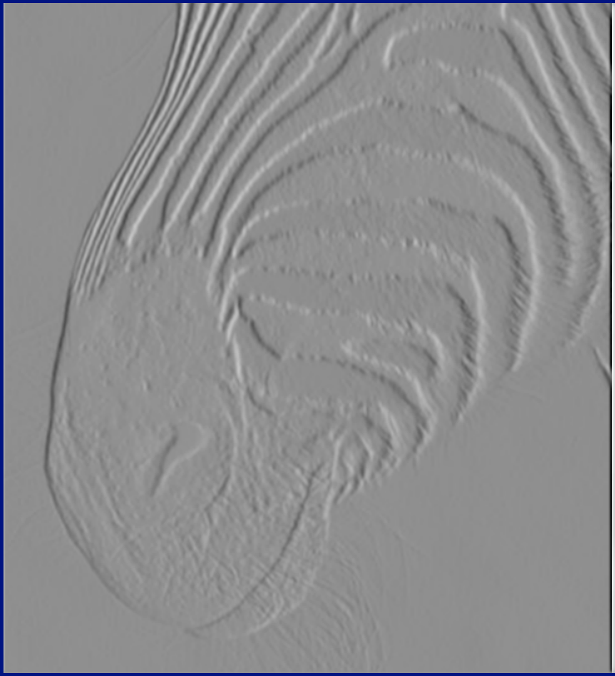
- Idea:
  - points where image value change very sharply are important
    - changes in surface reflectance
    - shadow boundaries
    - outlines
- Finding Edges:
  - Estimate gradient magnitude using appropriate smoothing
  - Mark points where gradient magnitude is
    - Locally biggest and
    - big

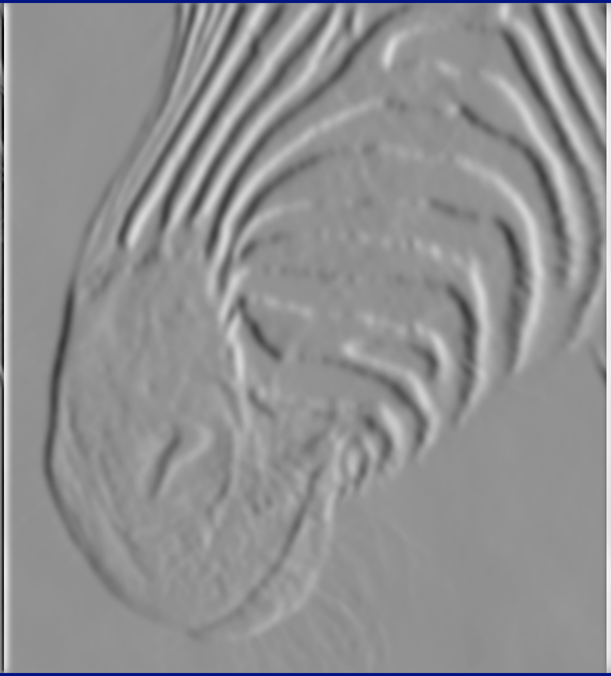
## Smoothing and Differentiation

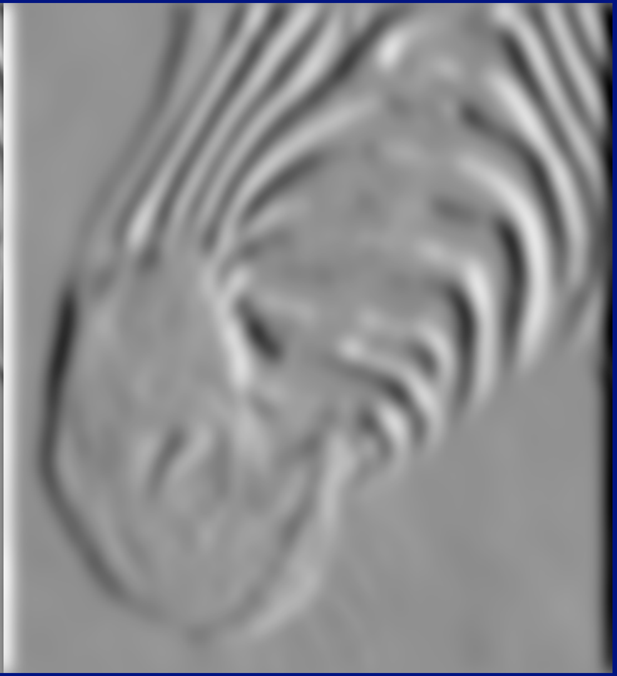
- Issue: noise
  - smooth before differentiation
  - two convolutions to smooth, then differentiate?
  - actually, no - we can use a derivative of Gaussian filter

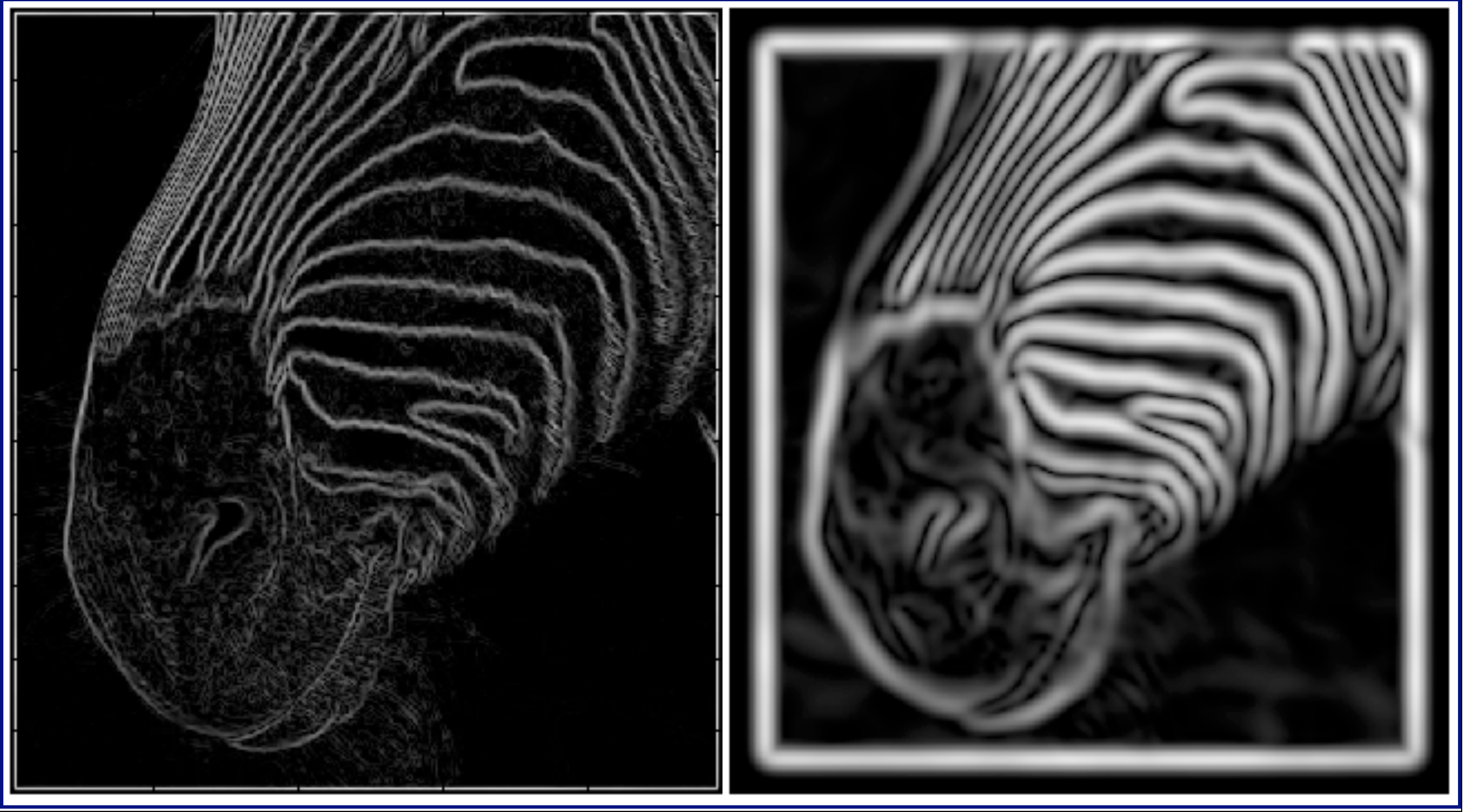
## Scale affects derivatives


1 pixel

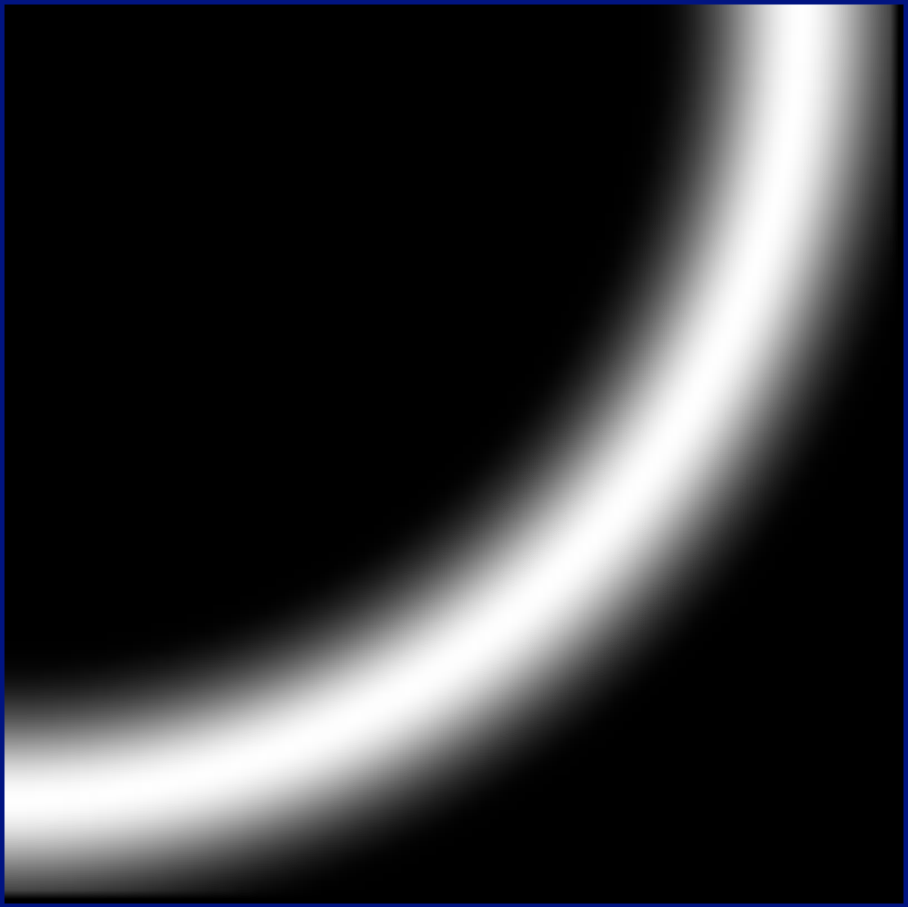

3 pixels

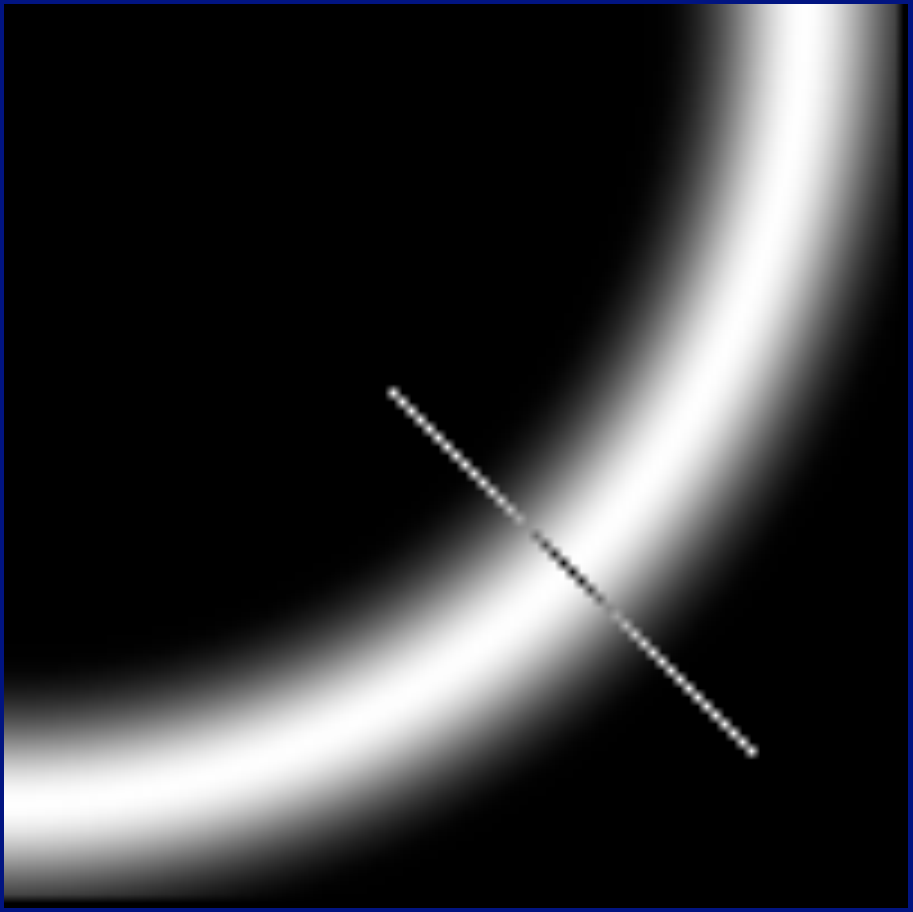

7 pixels

## Scale affects gradient magnitude

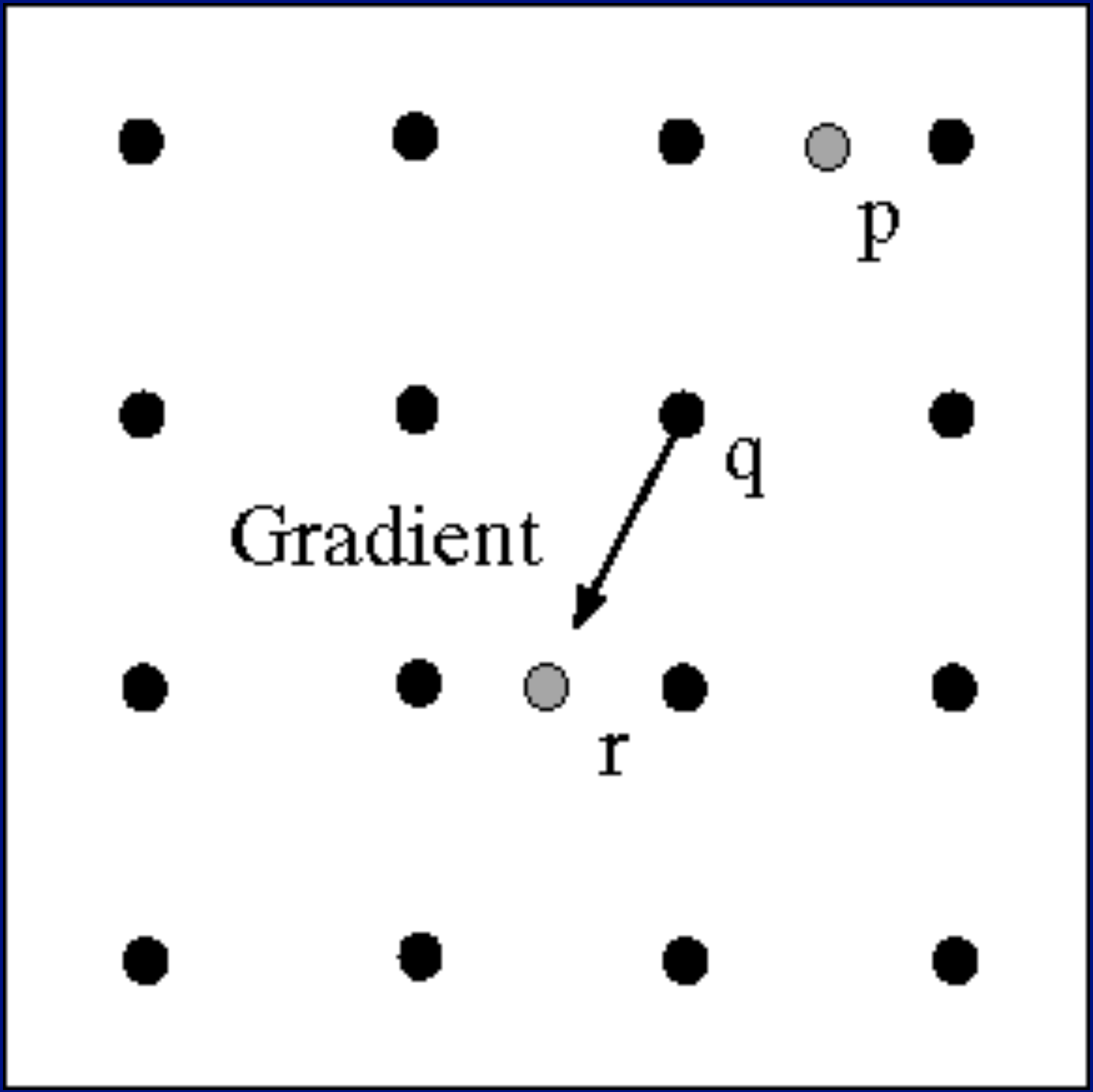


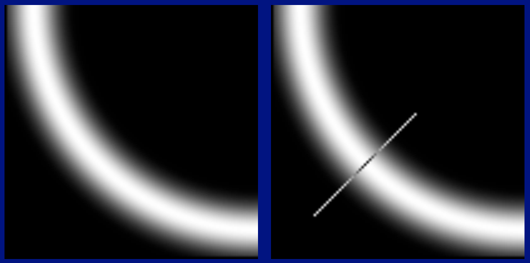
# Marking the points



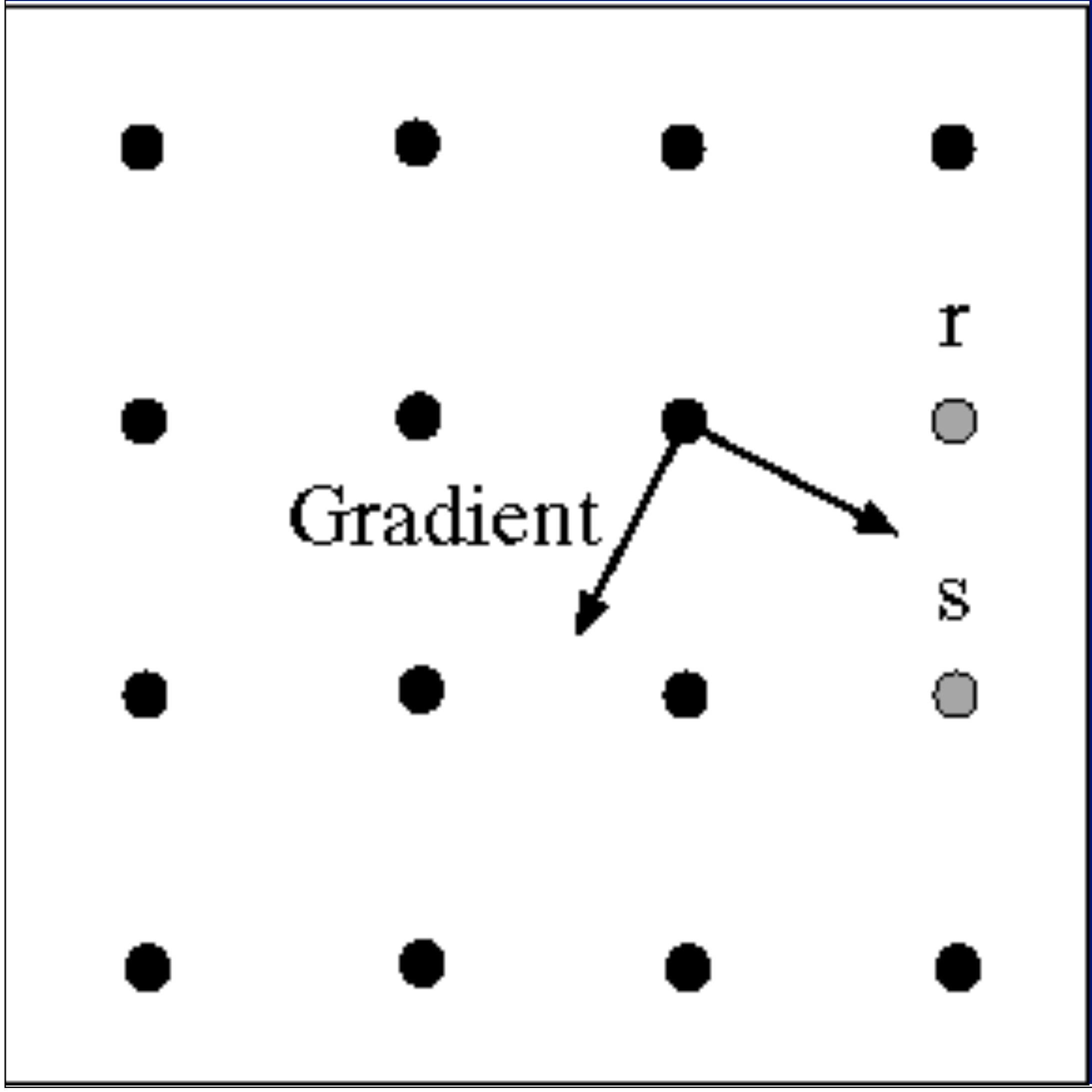


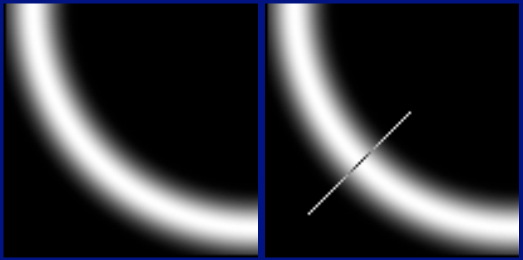
## Non-maximum suppression





# Predicting the next edge point

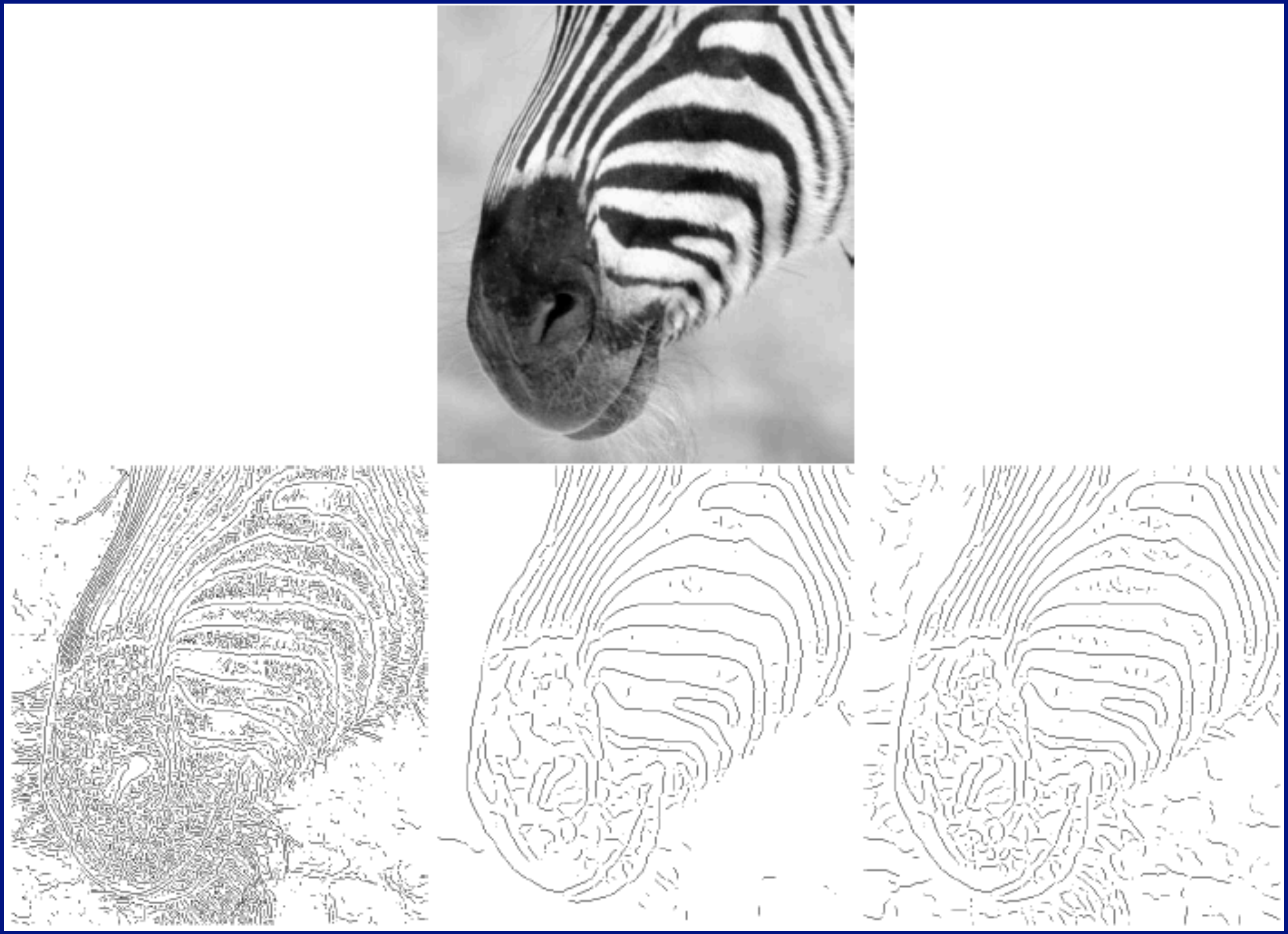




## Remaining issues

- Check maximum value of gradient value is sufficiently large
    - drop-outs?
        - use hysteresis

# Typical edges



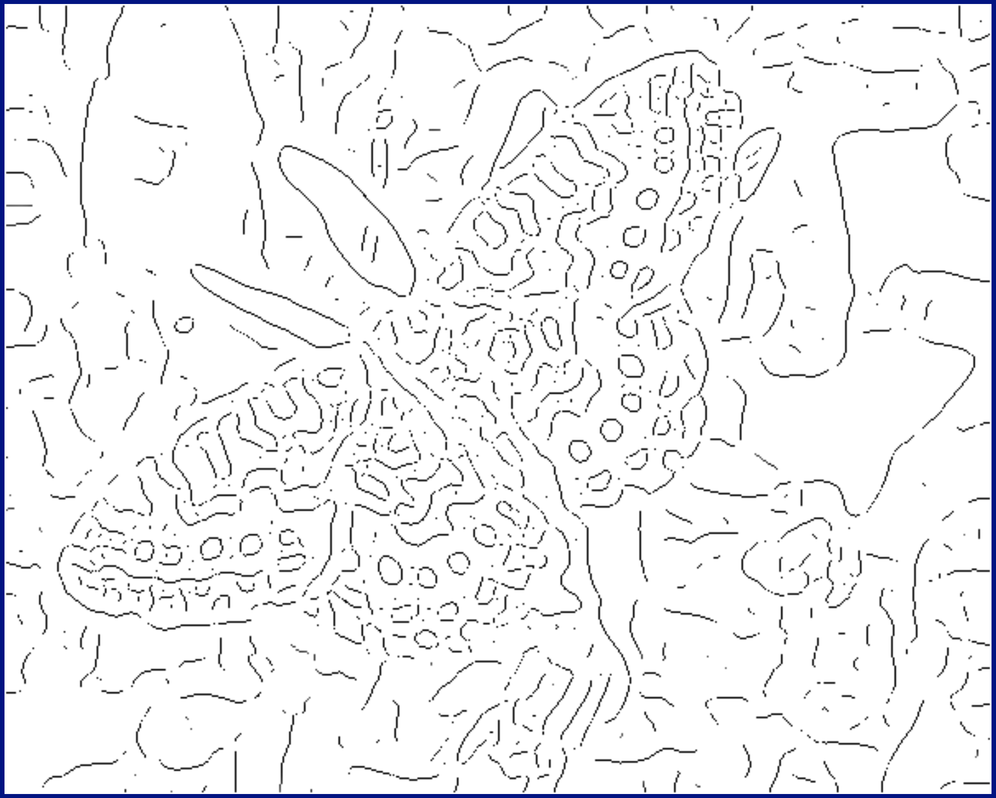
little smoothing, low threshold

heavy smoothing, high threshold

heavy smoothing, low threshold

## Notice

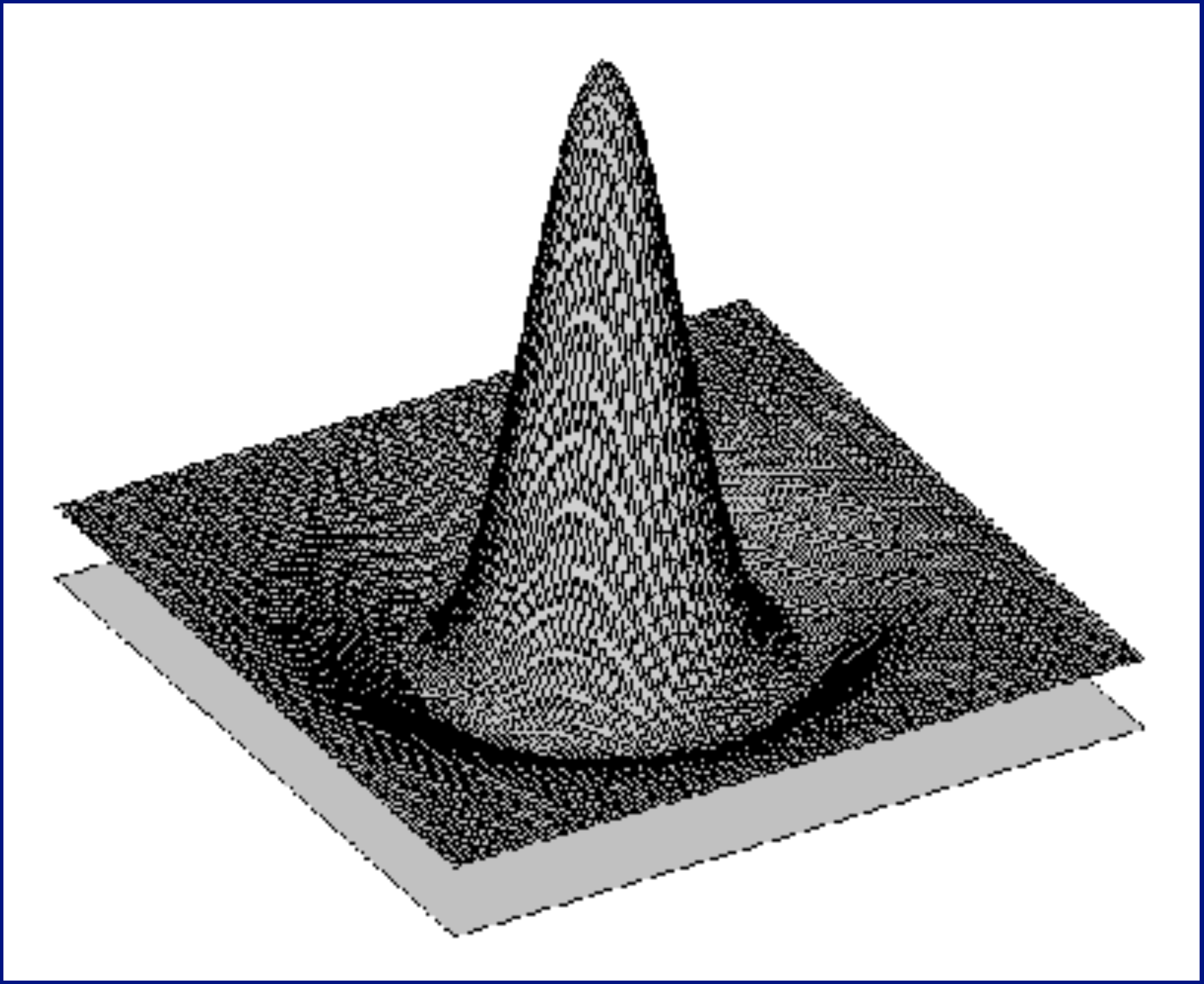
- Something nasty is happening at corners
- Scale affects contrast
- Edges aren't bounding contours



## The Laplacian of Gaussian

- Another way to detect an extremal first derivative is to look for a zero second derivative
- Appropriate 2D analogy is rotation invariant
  - Laplacian
- Edges are zero crossings
  - Bad idea to apply a Laplacian without smoothing
  - smooth with Gaussian, apply Laplacian
  - this is the same as filtering with a Laplacian of Gaussian filter
  - Now mark the zero points where
    - there is a sufficiently large derivative,
    - and enough contrast

## The Laplacian of Gaussian

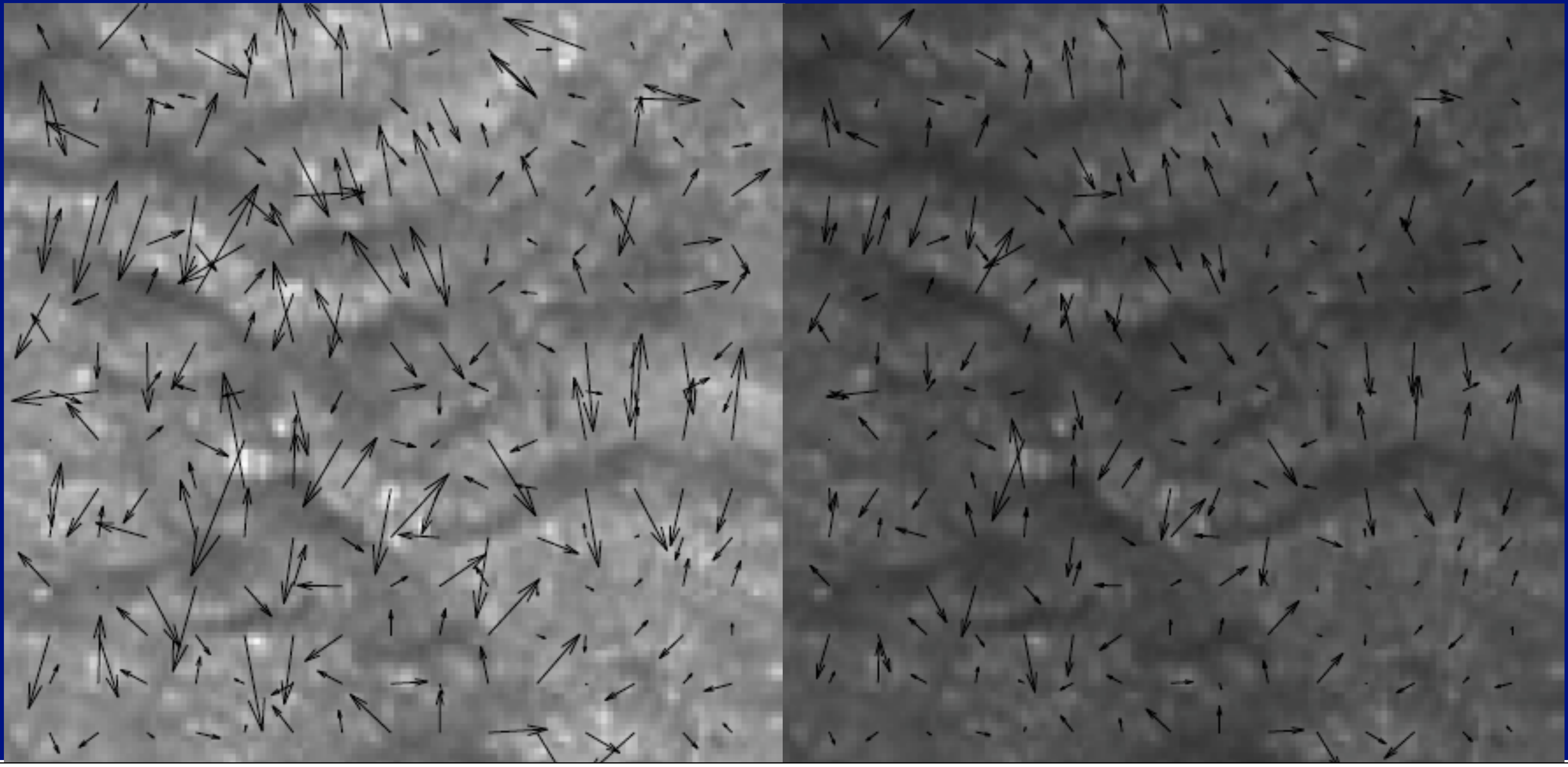


## Filters and Edges:  Crucial points

- Filters are simple detectors
  - they look like patterns they find
- Smoothing suppresses noise
  - because pixels tend to agree
- Sharp changes are interesting
  - because pixels tend to agree
  - easy to find - edges

# Orientation representations

- Gradient magnitude is affected by illumination changes
  - but it’s direction isn’t
- Describe image patches by gradient direction



# Orientations

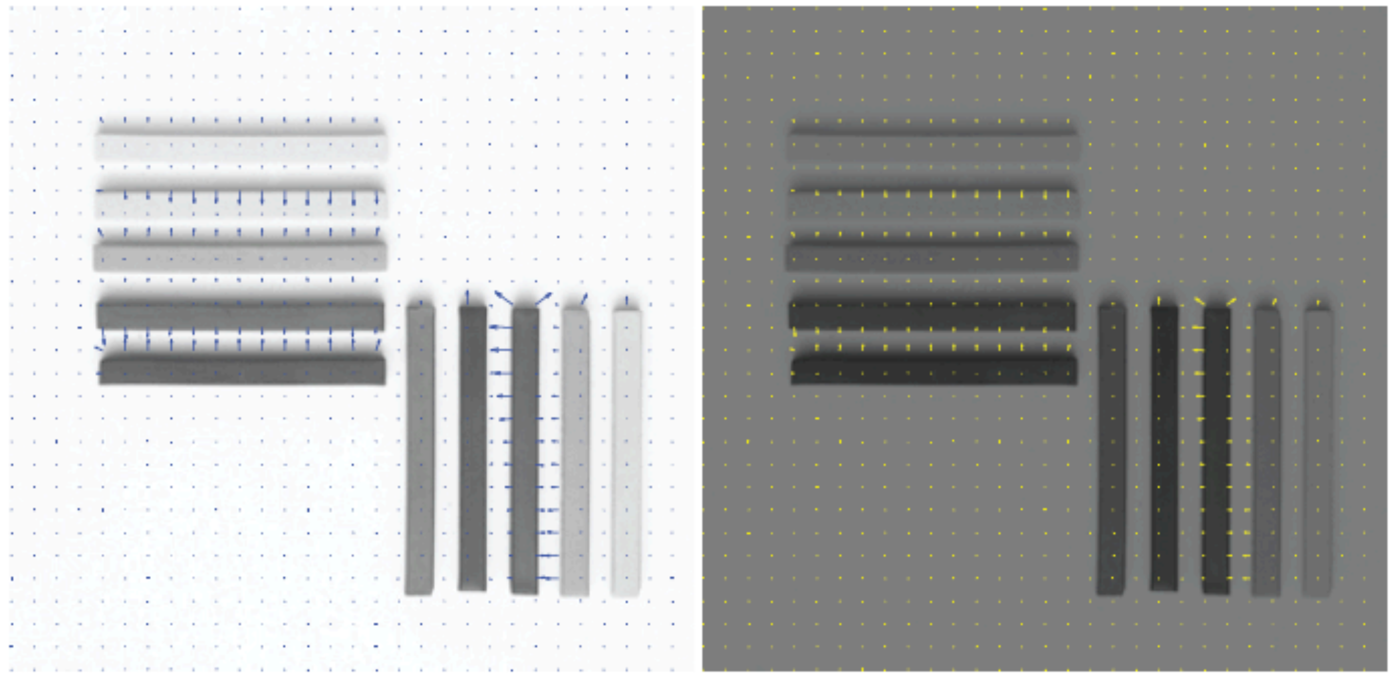
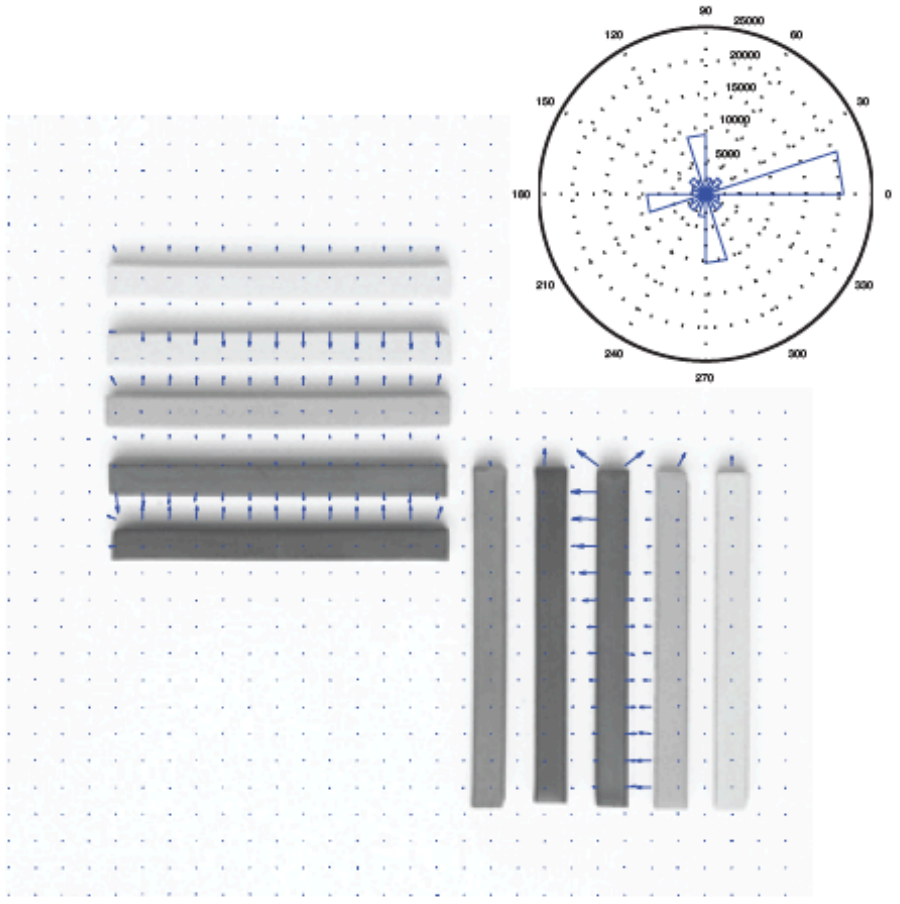
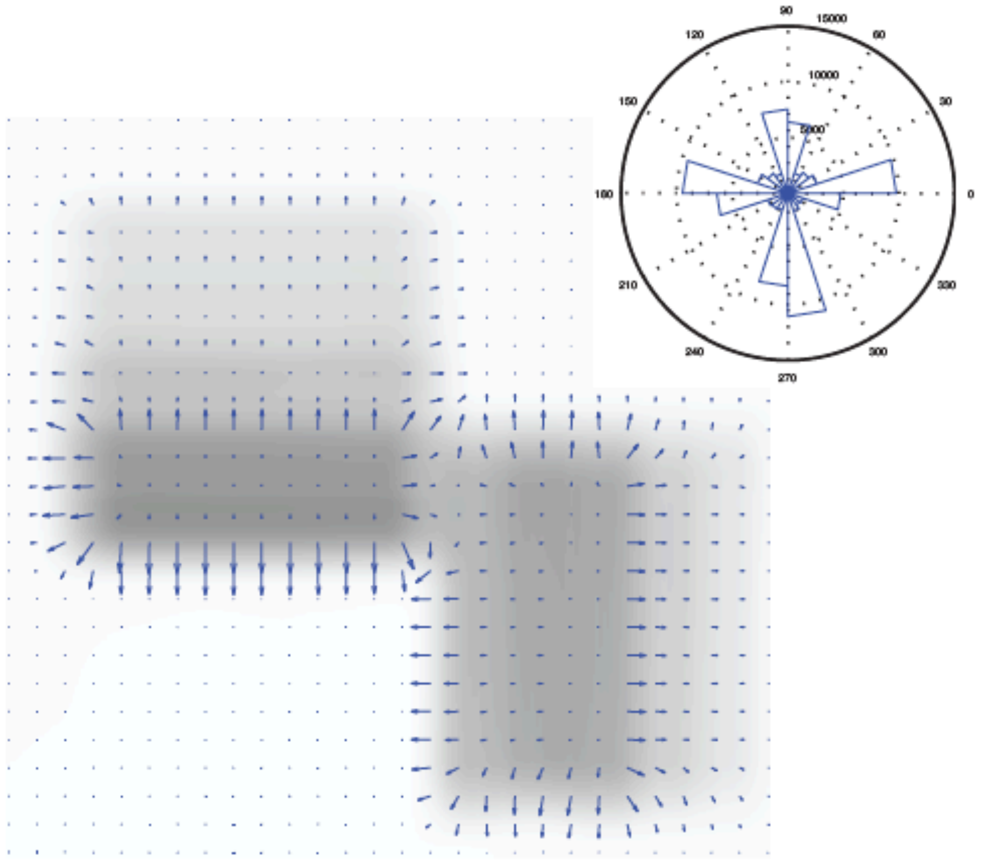


FIGURE 5.7: The magnitude of the image gradient changes when one increases or decreases the intensity. The orientation of the image gradient does not change; we have plotted every 10th orientation arrow, to make the figure easier to read. Note how the directions of the gradient arrows are fixed, whereas the size changes. *Philip Gatward © Dorling Kindersley, used with permission.*

## Orientations at different scales





## Orientation histograms vary

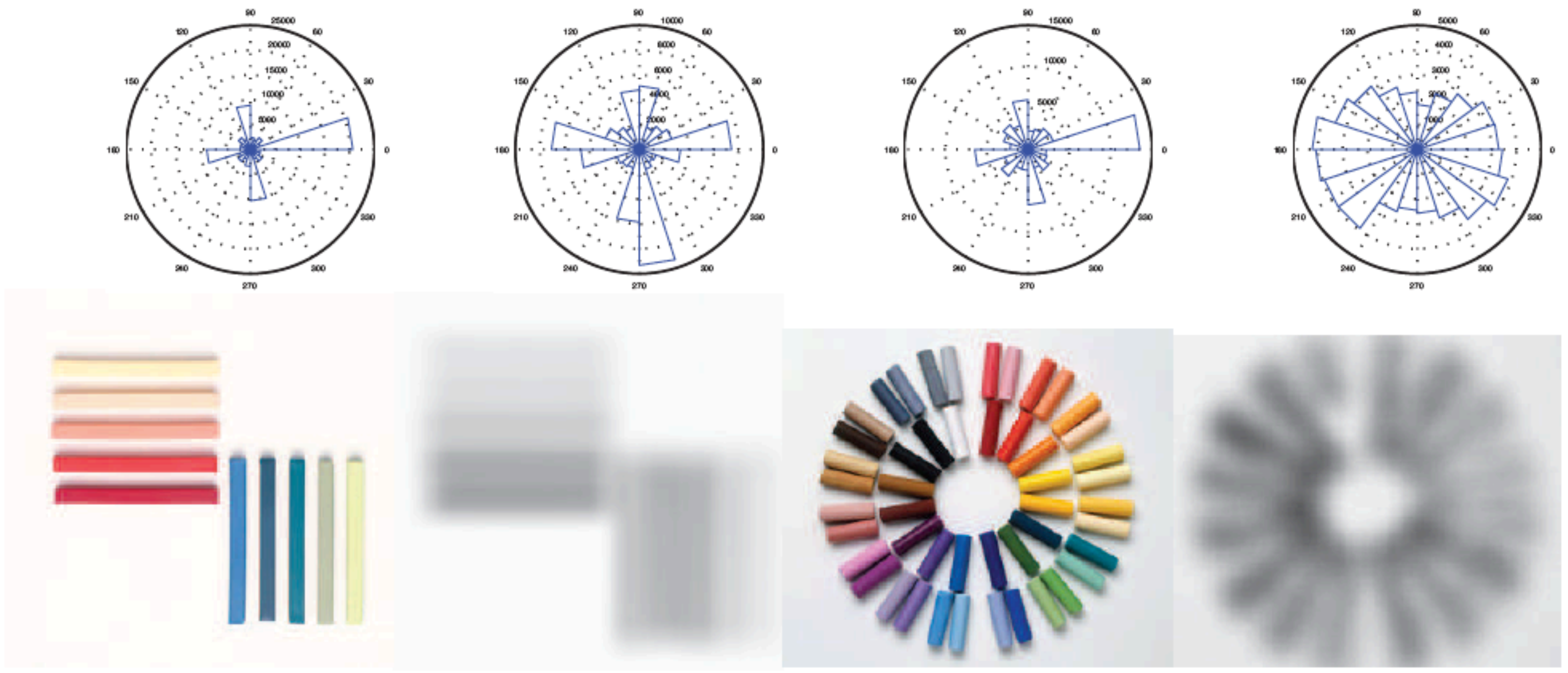
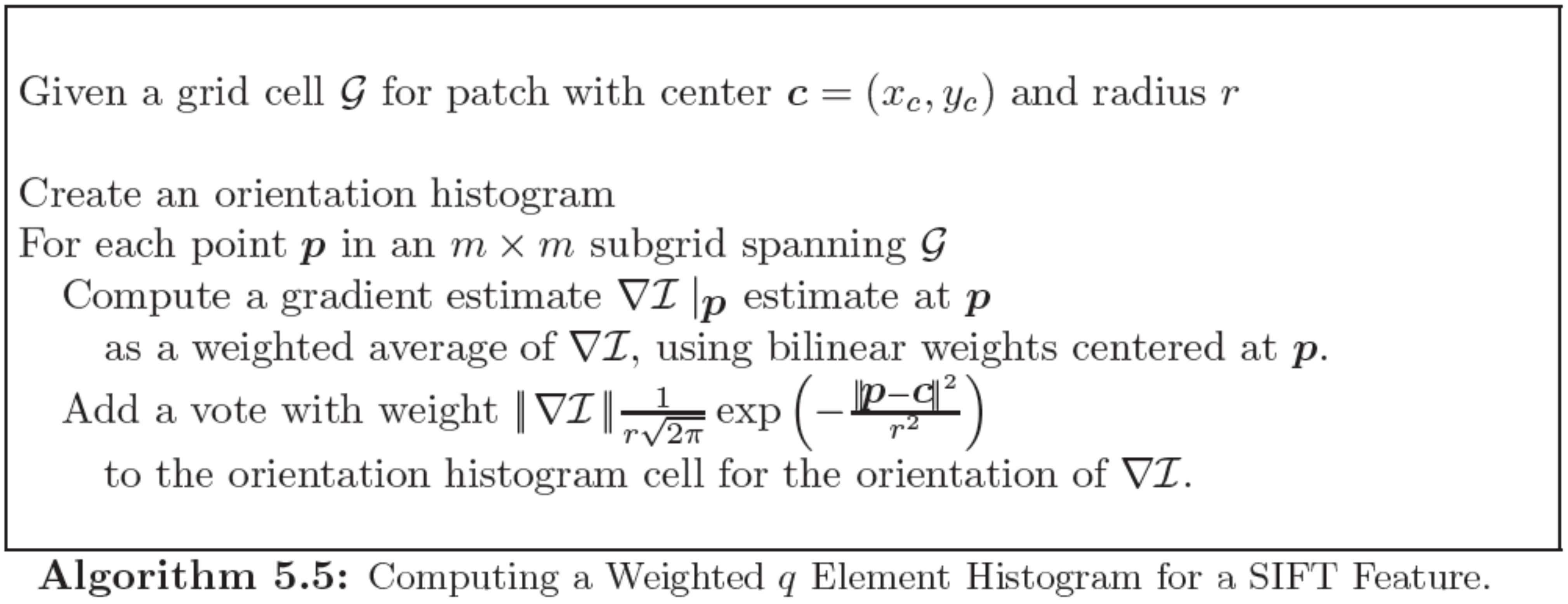


FIGURE 5.9: Different patterns have quite different orientation histograms. The **left** shows rose plots and images for a picture of artists pastels at two different scales; the **right** shows rose plots and images for a set of pastels arranged into a circular pattern. Notice how the pattern of orientations at a particular scale, and also the changes across scales, are quite different for these two very different patterns. *Philip Gatward © Dorling Kindersley, used with permission.*

## Histograms of oriented gradients

- Strategy:
  - break patch up into blocks
  - construct histogram representing gradients in that block
    - which won’t change much if the patch moves slightly
- Variants
  - histogram of angles
  - histogram of gradient vectors, length normalized by block averages

## HOG features

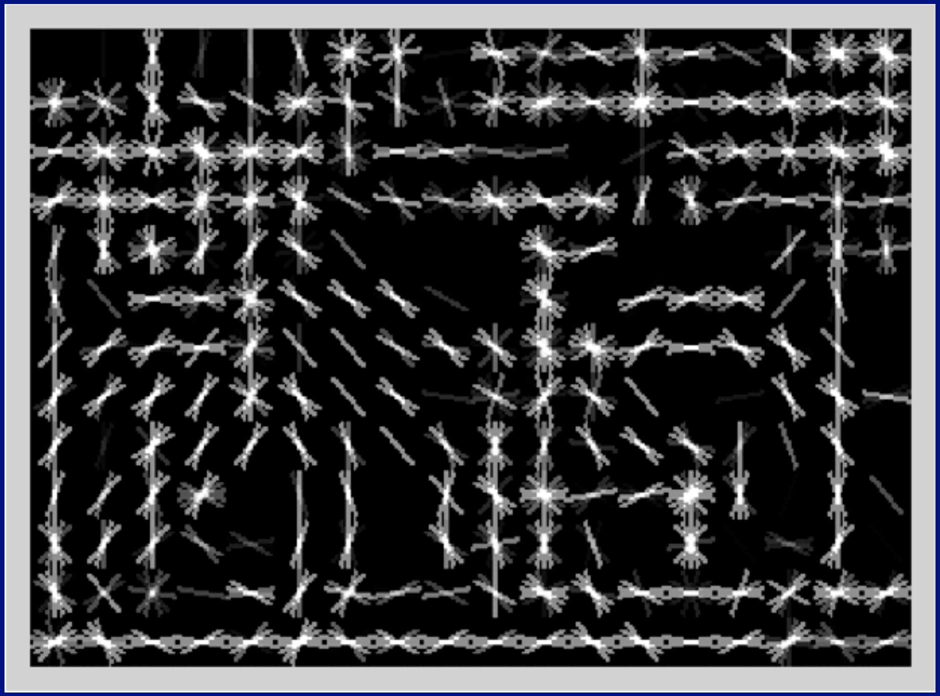


```
Given a grid cell $\mathcal{G}$ for patch with center $\boldsymbol{c} = (x_c, y_c)$ and radius $r$

Create an orientation histogram
For each point $\boldsymbol{p}$ in an $m \times m$ subgrid spanning $\mathcal{G}$
    Compute a gradient estimate $\nabla\mathcal{I}\,|_{\boldsymbol{p}}$ estimate at $\boldsymbol{p}$
        as a weighted average of $\nabla\mathcal{I}$, using bilinear weights centered at $\boldsymbol{p}$.
    Add a vote with weight $\|\nabla\mathcal{I}\|\frac{1}{r\sqrt{2\pi}}\exp\left(-\frac{\|\boldsymbol{p}-\boldsymbol{c}\|^2}{r^2}\right)$
        to the orientation histogram cell for the orientation of $\nabla\mathcal{I}$.
```

**Algorithm 5.5:** Computing a Weighted $q$ Element Histogram for a SIFT Feature.

# Histograms of oriented gradients



From Deva Ramanan’s lake Como slides

## HOG features

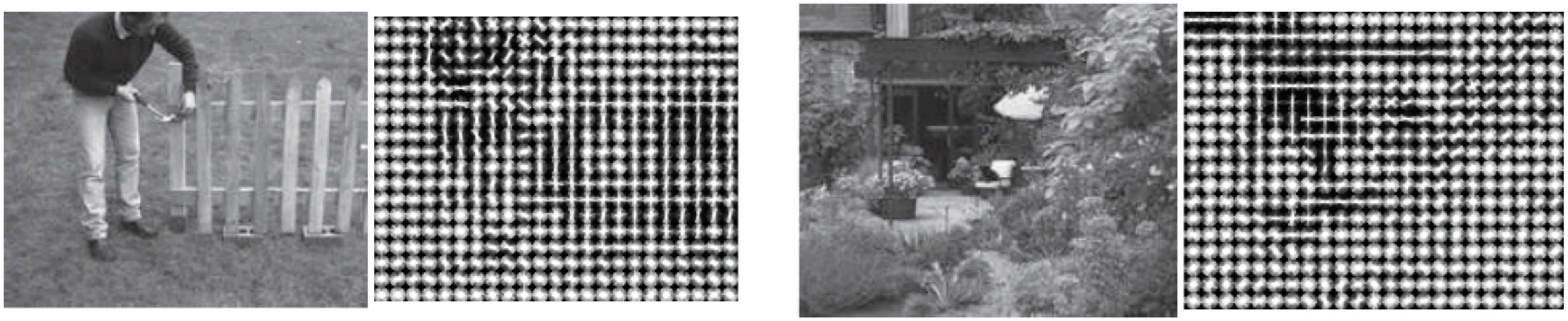
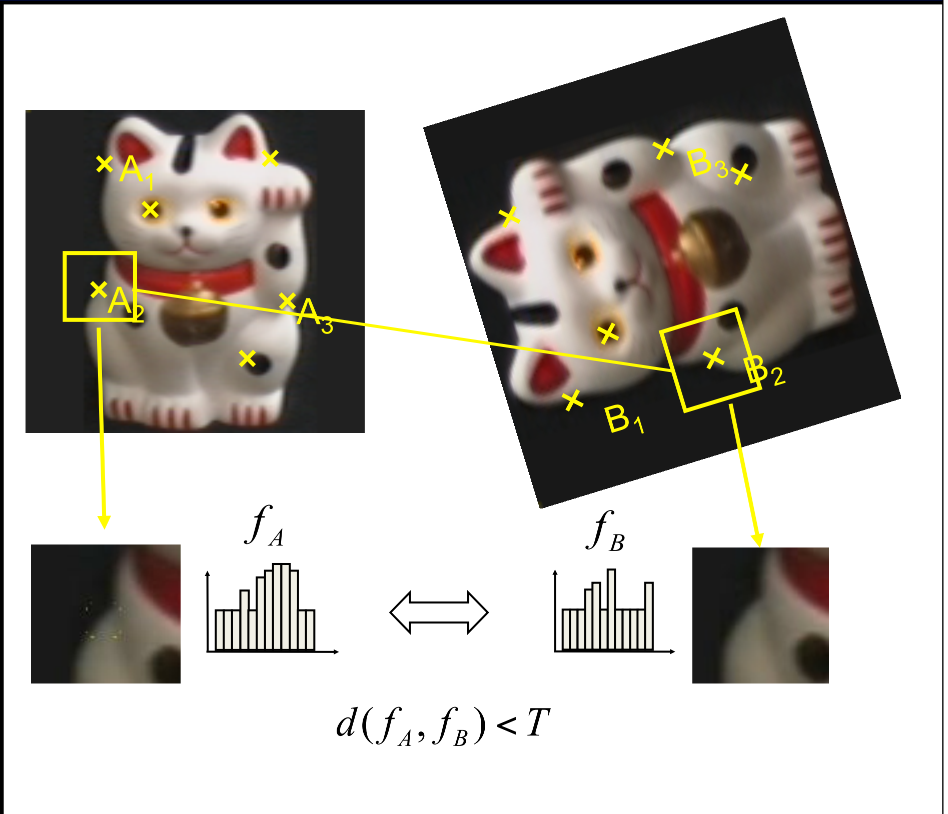


FIGURE 5.15: The HOG features for each the two images shown here have been visualized by a version of the rose diagram of Figures 5.7–5.9. Here each of the cells in which the histogram is taken is plotted with a little rose in it; the direction plotted is at right angles to the gradient, so you should visualize the overlaid line segments as edge directions. Notice that in the textured regions the edge directions are fairly uniformly distributed, but strong contours (the gardener, the fence on the **left**; the vertical edges of the french windows on the **right**) are very clear. This figure was plotted using the toolbox of Dollár and Rabaud. *Left: © Dorling Kindersley, used with permission. Right: Geoff Brightling © Dorling Kindersley, used with permission.*

# Interest points

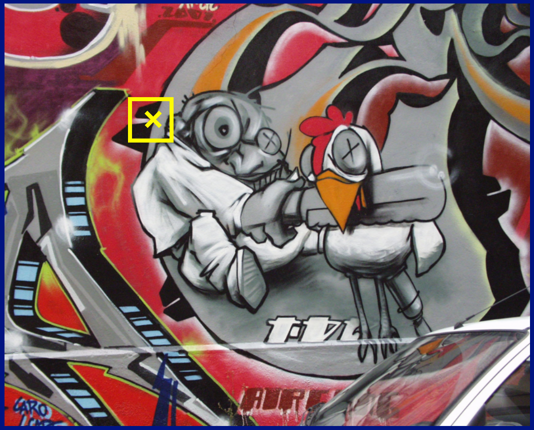
- Automatic patch construction
  - HOG works if we know the patch
    - but what patches should we use?
      - sliding windows
- We then
  - find patches
  - make descriptions
  - match patches
- Matches for
  - making mosaics
  - spotting near duplicates
  - detection
  - reconstruction

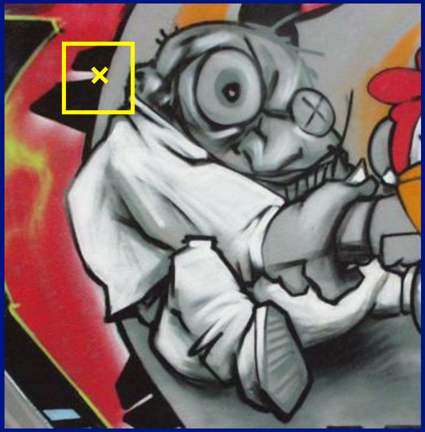


K. Grauman, B. Leibe

## Interest points

- For image, find center/radius of circles “worth describing”
  - these should be “stable”
    - if the image is panned, the centers should pan
    - if the image is scaled, the centers should scale



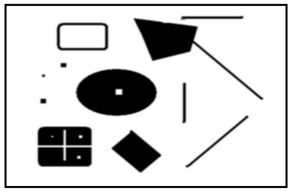


## Interest points: locating centers

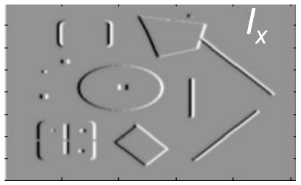
- We use a corner detector (Harris, 88)
  - at a corner there are
    - strong gradients
    - in different directions
- Use second moments of derivatives

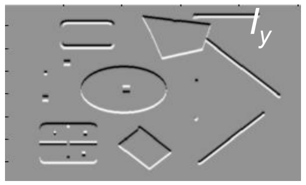
$$\mu(\sigma_I,\sigma_D) = g(\sigma_I) * \begin{bmatrix} I_x^2(\sigma_D) & I_x I_y(\sigma_D) \\ I_x I_y(\sigma_D) & I_y^2(\sigma_D) \end{bmatrix}$$

# Interest points: locating centers

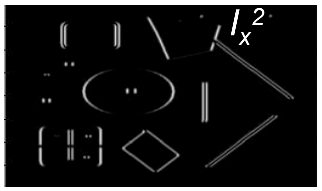


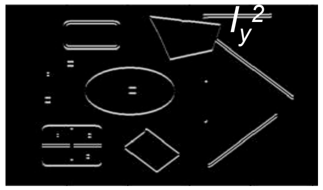
1. Image derivatives

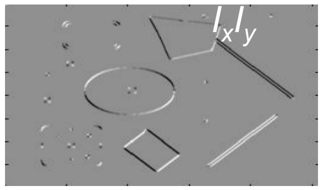




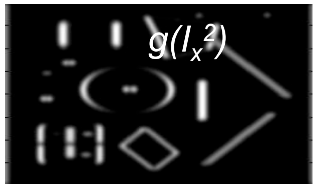
2. Square of derivatives

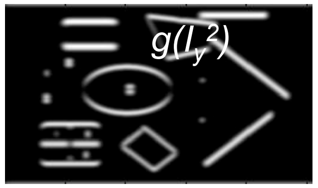


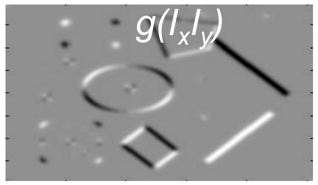




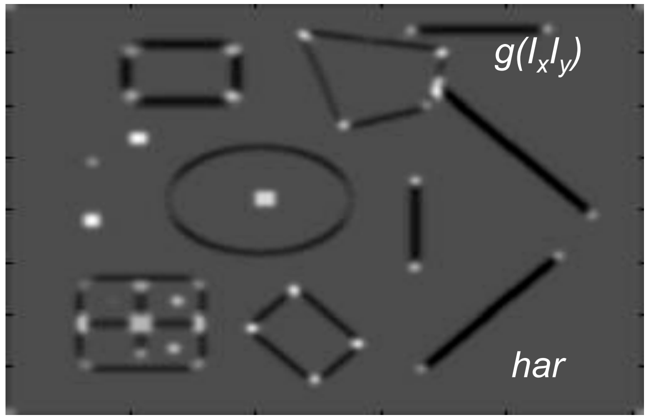
3. Gaussian filter $g(\sigma_I)$



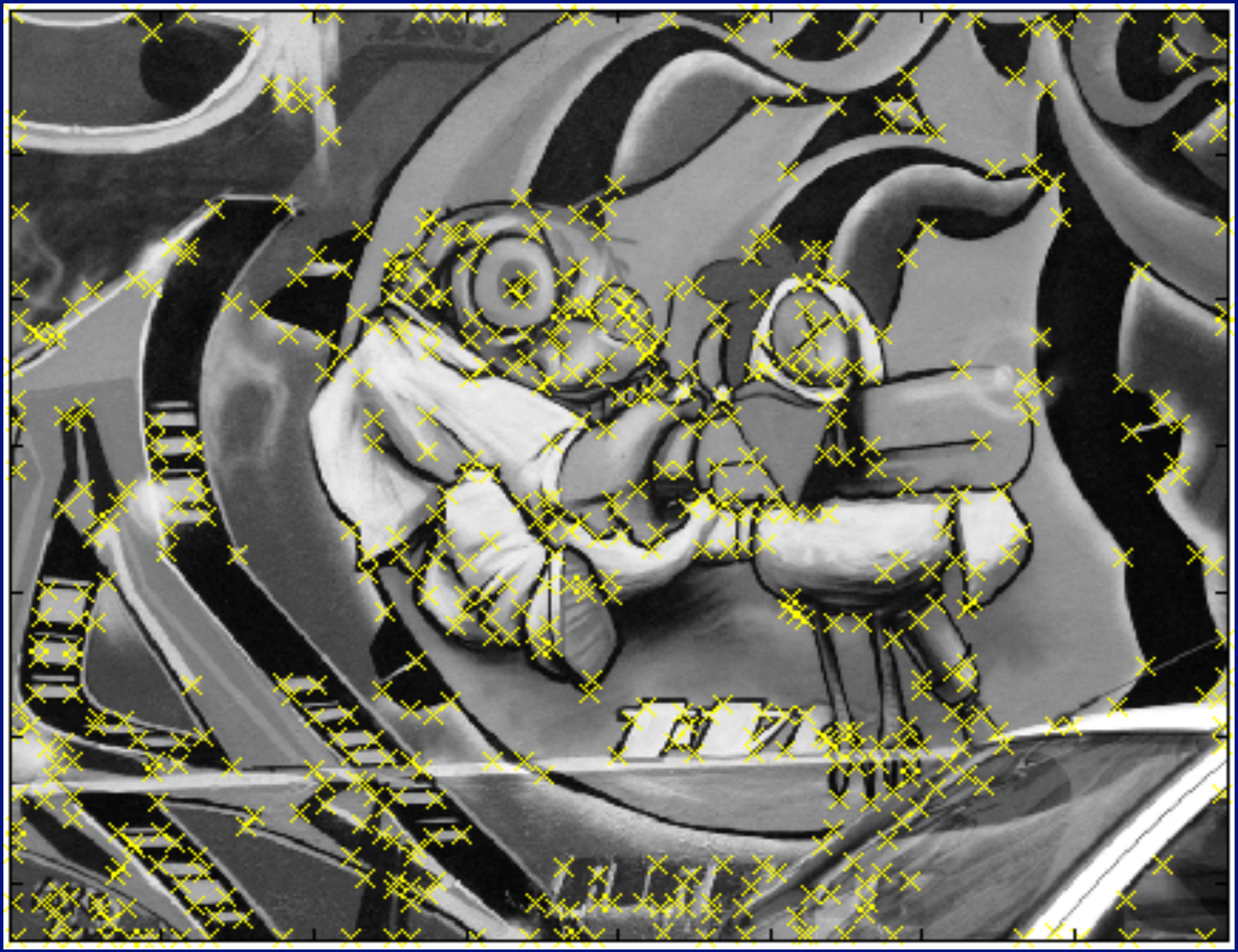




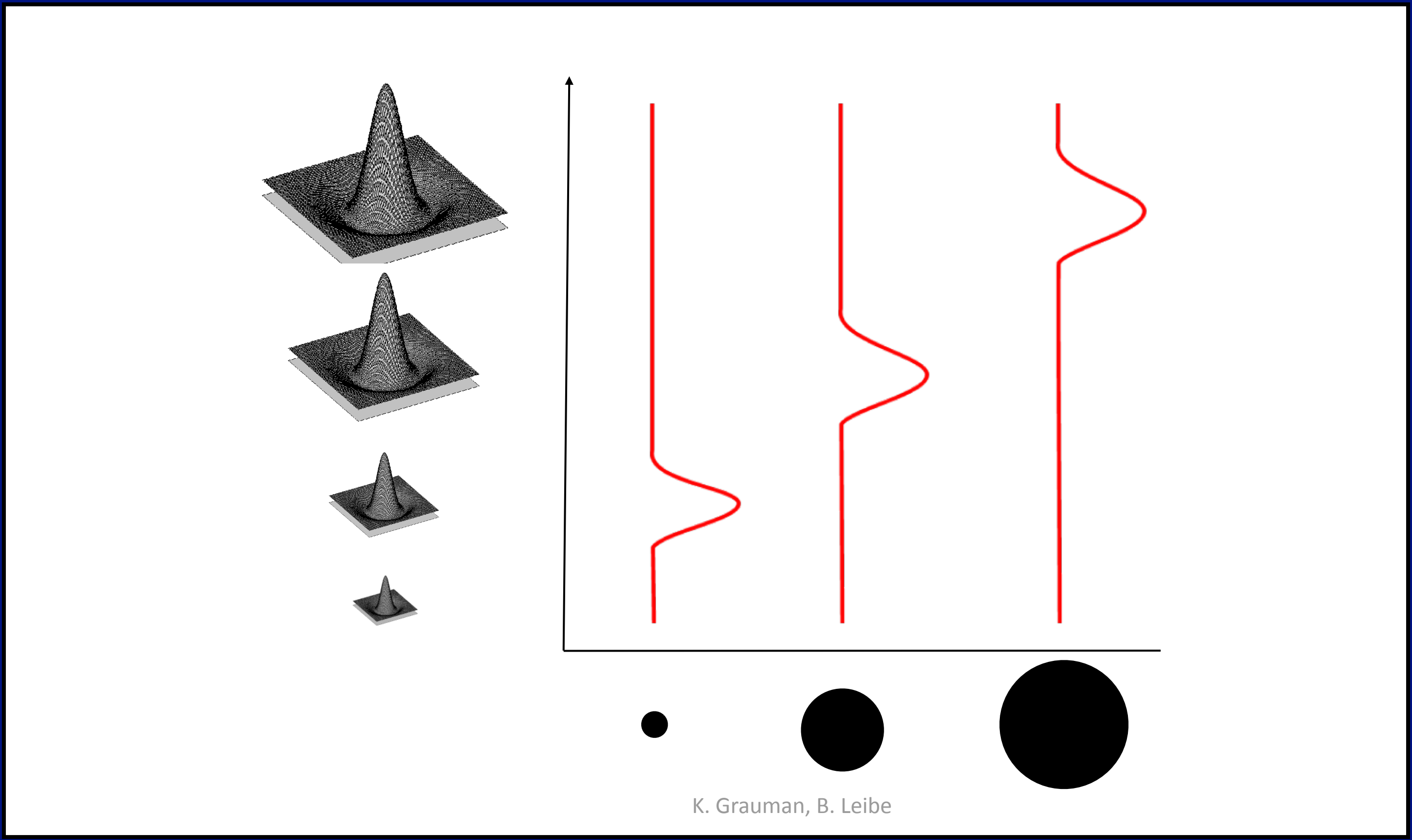
$$har = \det[\mu(\sigma_I,\sigma_D)] - \alpha[\operatorname{trace}(\mu(\sigma_I,\sigma_D))] =$$
$$g(I_x^2)g(I_y^2) - [g(I_xI_y)]^2 - \alpha[g(I_x^2)+g(I_y^2)]^2$$



## Interest points: locating centers

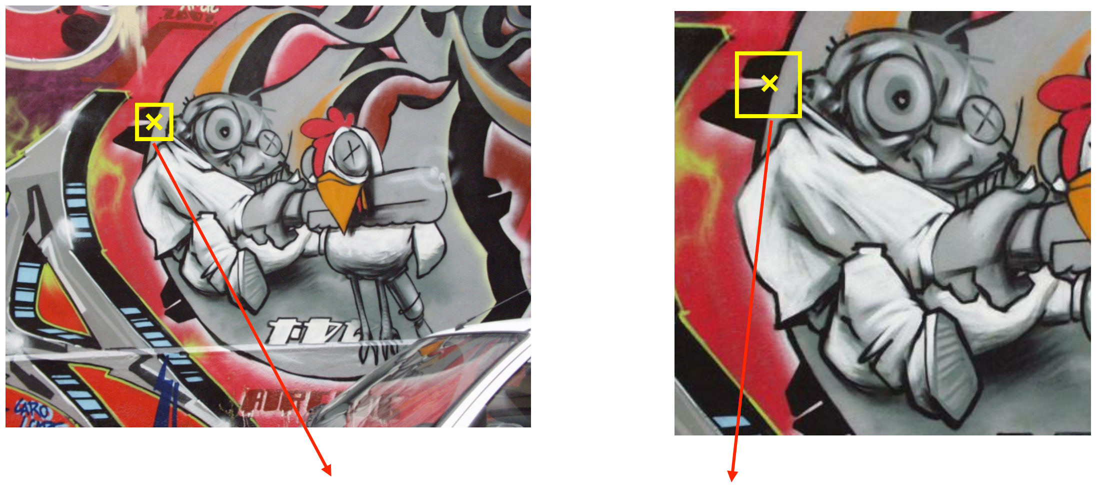


## Interest points: finding the radius



Laplacian of Gaussian:  radius or blob detector

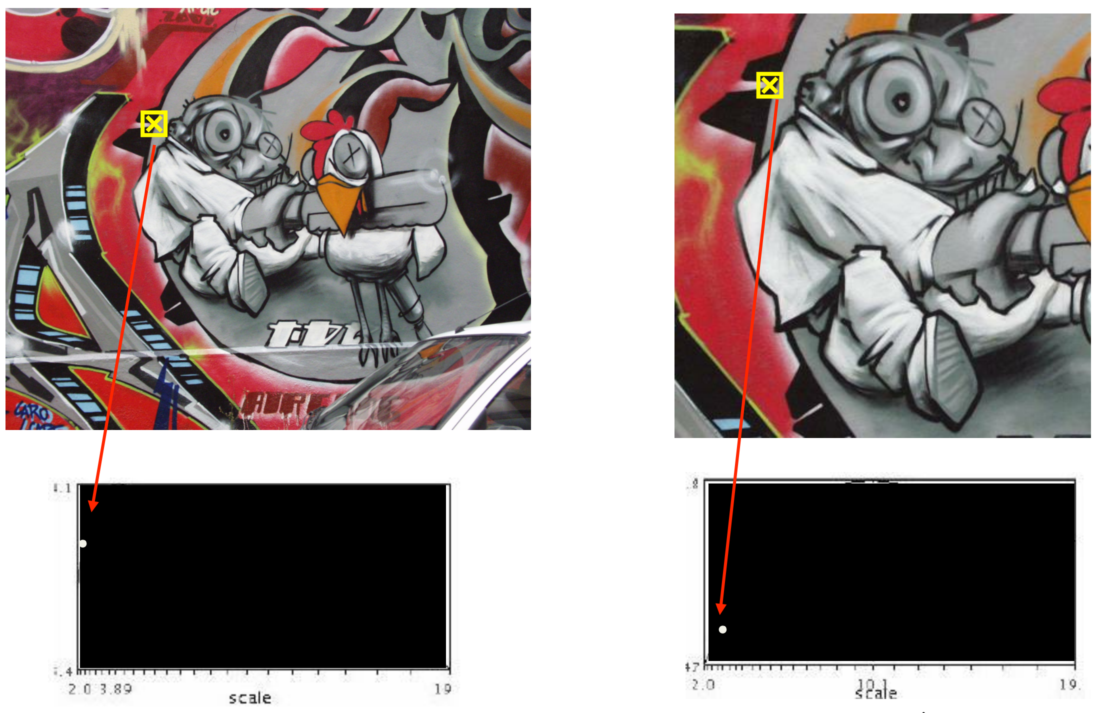
# Interest points: finding the radius



$$f(I_{i_1 \ldots i_m}(x,\sigma)) = f(I_{i_1 \ldots i_m}(x',\sigma'))$$

K. Grauman, B. Leibe

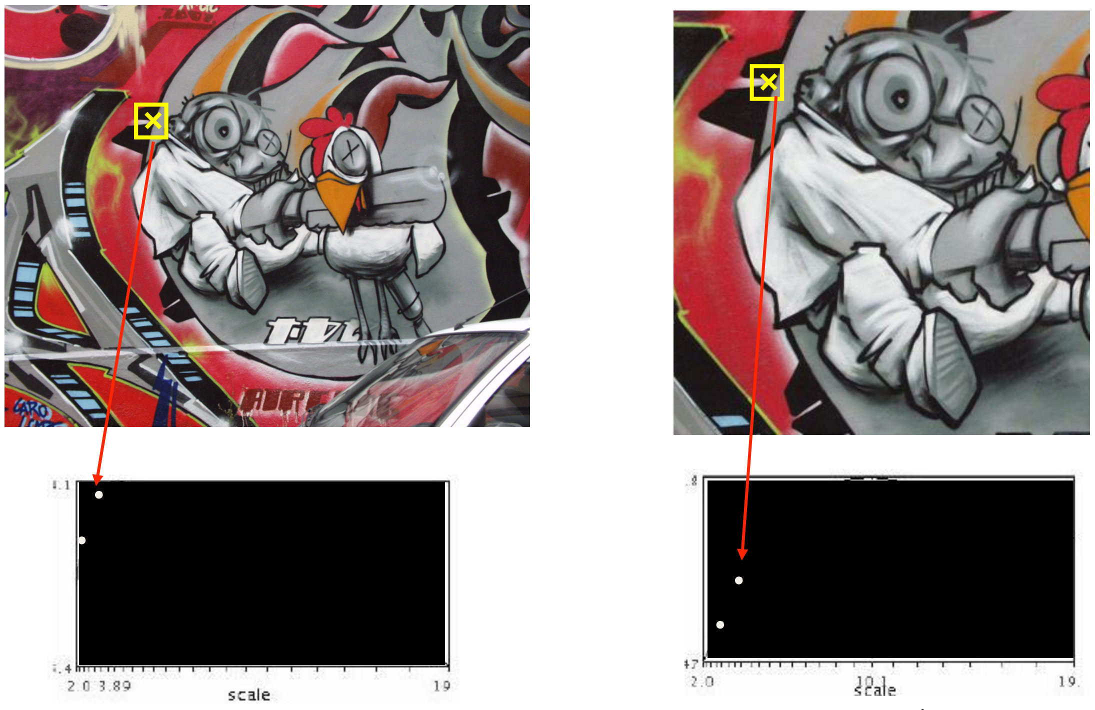
# Interest points: finding the radius



$f(I_{i_1 \ldots i_m}(x,\sigma))$ $f(I_{i_1 \ldots i_m}(x',\sigma))$

K. Grauman, B. Leibe

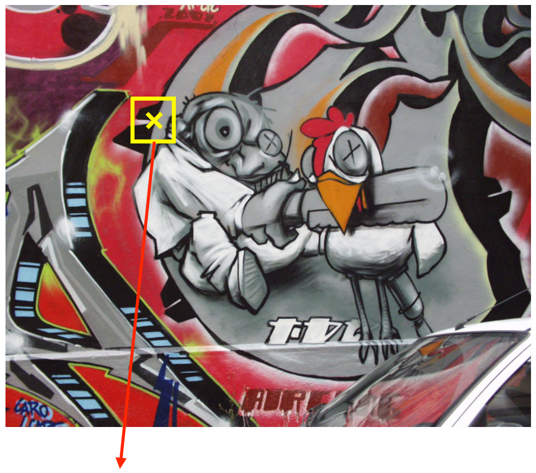
# Interest points: finding the radius

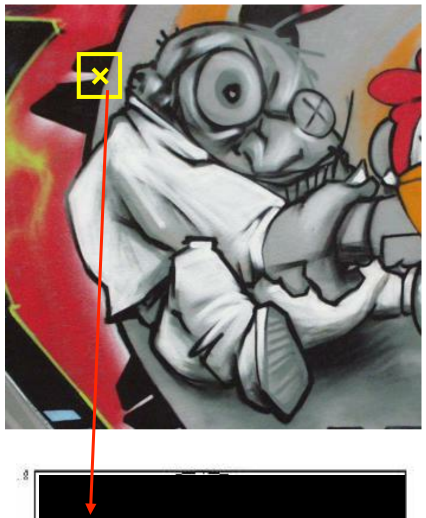


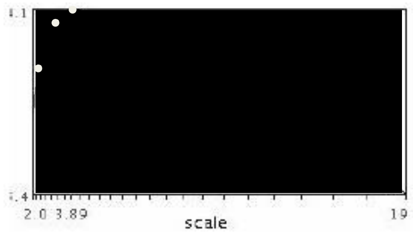
$f(I_{i_1 \ldots i_m}(x,\sigma))$ $f(I_{i_1 \ldots i_m}(x',\sigma))$

K. Grauman, B. Leibe

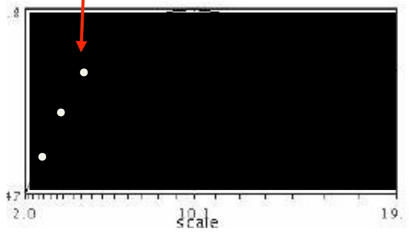
## Interest points: finding the radius







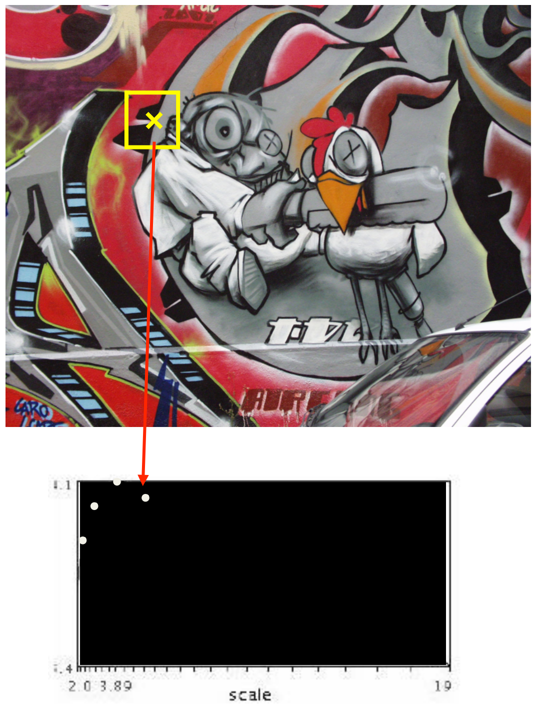
$f(I_{i_1 \dots i_m}(x,\sigma))$



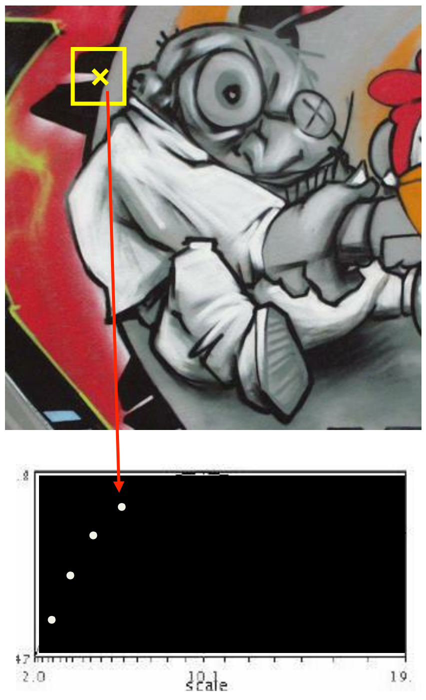
$f(I_{i_1 \dots i_m}(x',\sigma))$

K. Grauman, B. Leibe

# Interest points: finding the radius



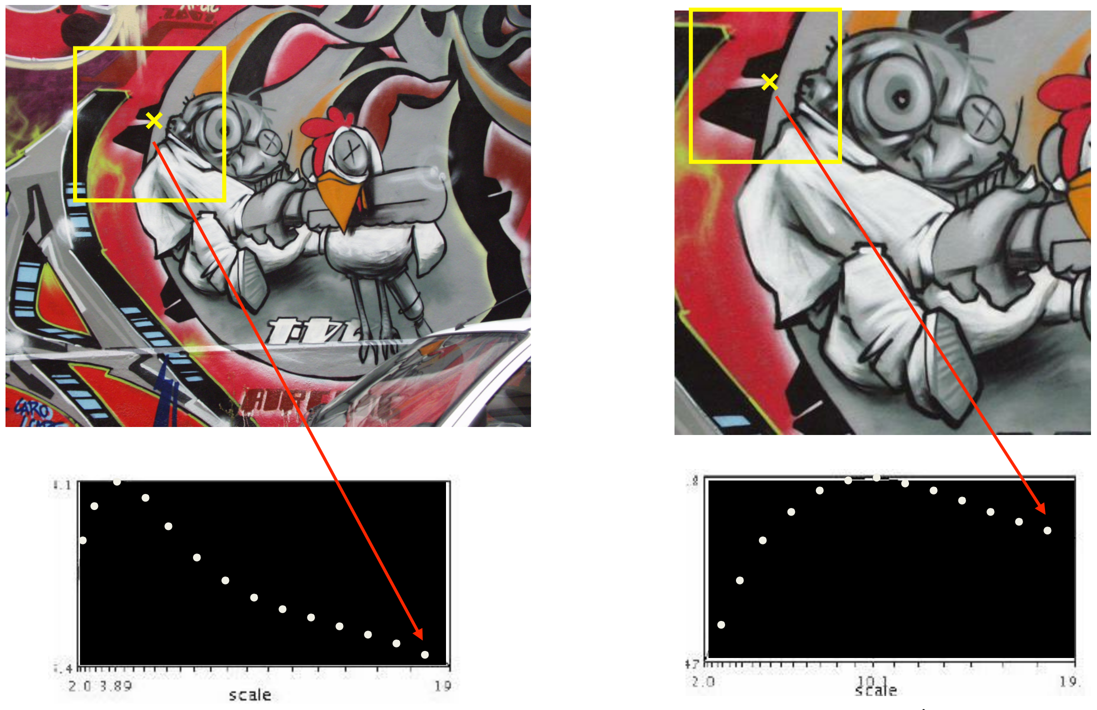
$f(I_{i_1 \ldots i_m}(x,\sigma))$



$f(I_{i_1 \ldots i_m}(x',\sigma))$

K. Grauman, B. Leibe

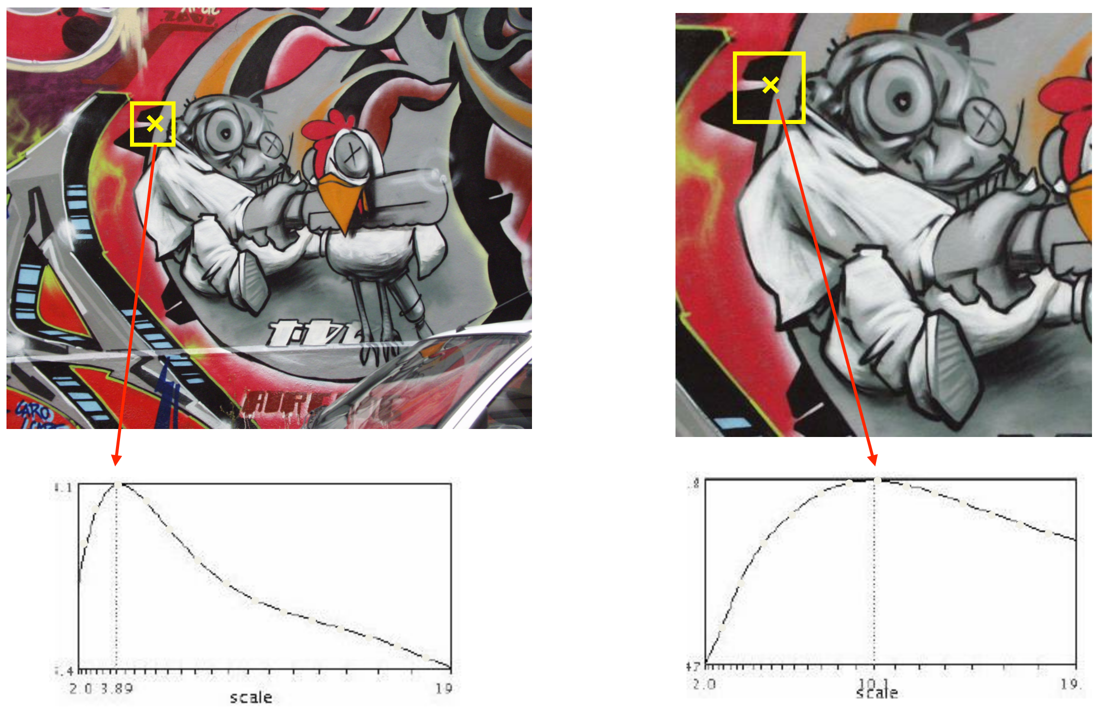
# Interest points: finding the radius



$f(I_{i_1\ldots i_m}(x,\sigma))$ $f(I_{i_1\ldots i_m}(x',\sigma))$

K. Grauman, B. Leibe

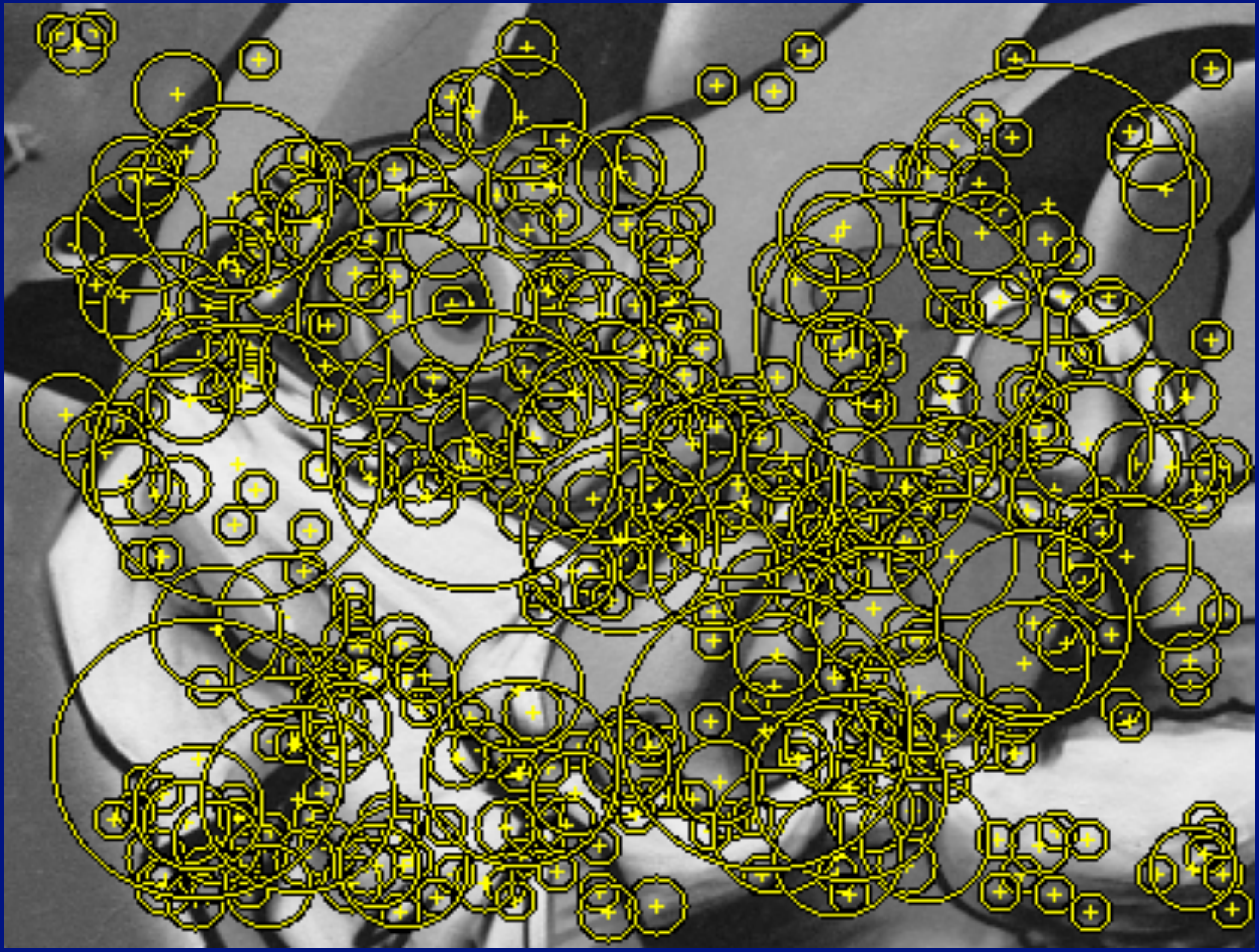
# Interest points: finding the radius



$f(I_{i_1 \ldots i_m}(x, \sigma))$ $f(I_{i_1 \ldots i_m}(x', \sigma'))$

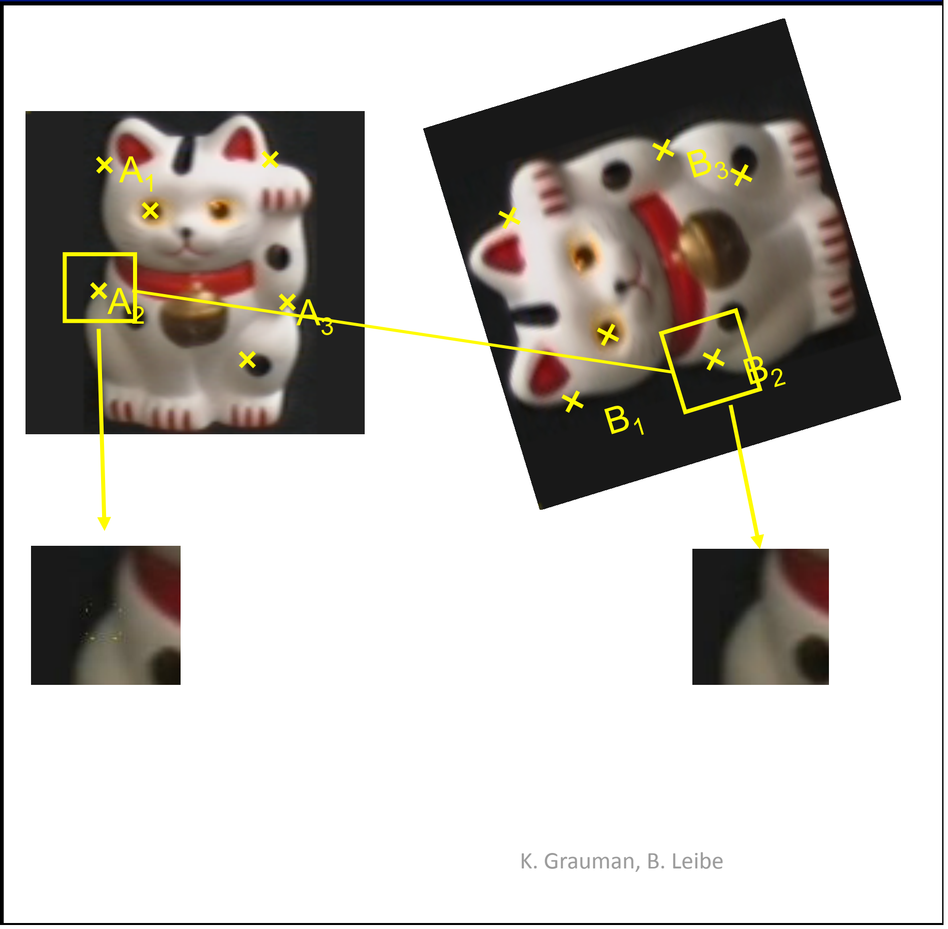
K. Grauman, B. Leibe

## Interest points: finding the radius



# Orientation of the patch

- We would like to know how the patch is rotated
  - to compare, compute features, etc.
- Strategy
  - compute orientation histogram
  - select most common orientation
    - this is 0 degrees



## Describing patches

- Various histograms of orientation
  - HOG
  - SIFT
  - SURF
  - etc.

# Lowe's SIFT features

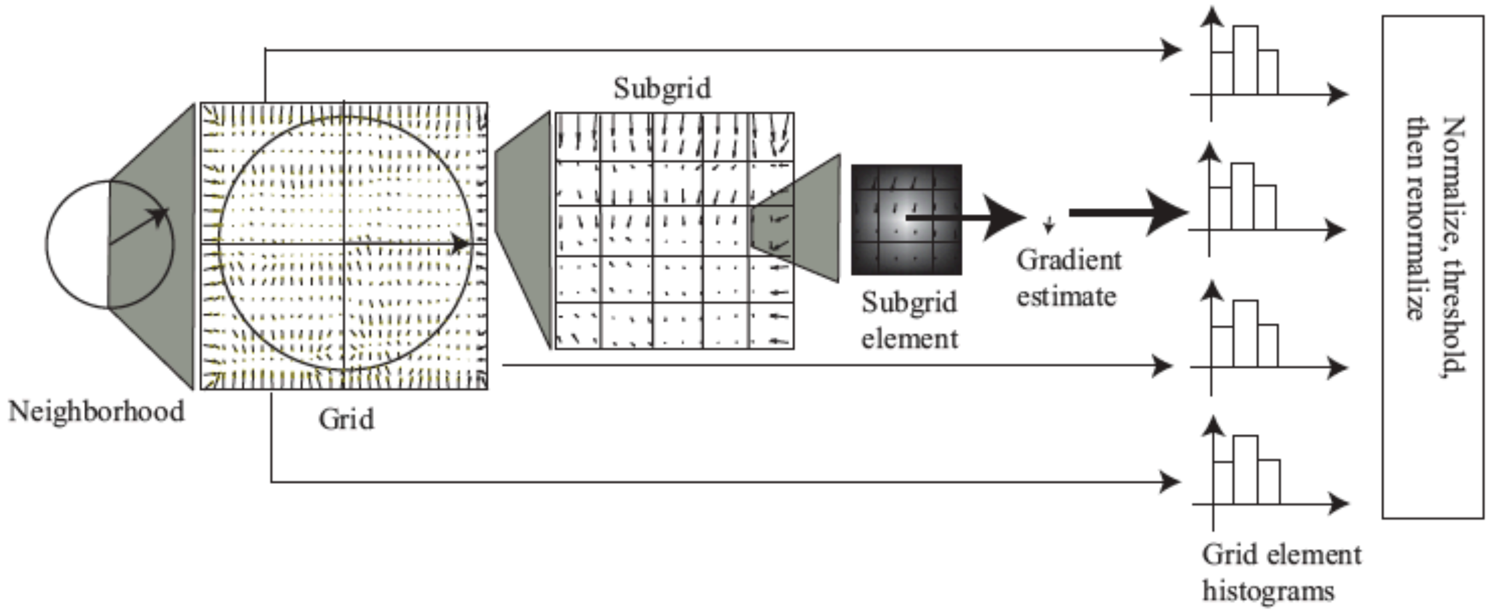
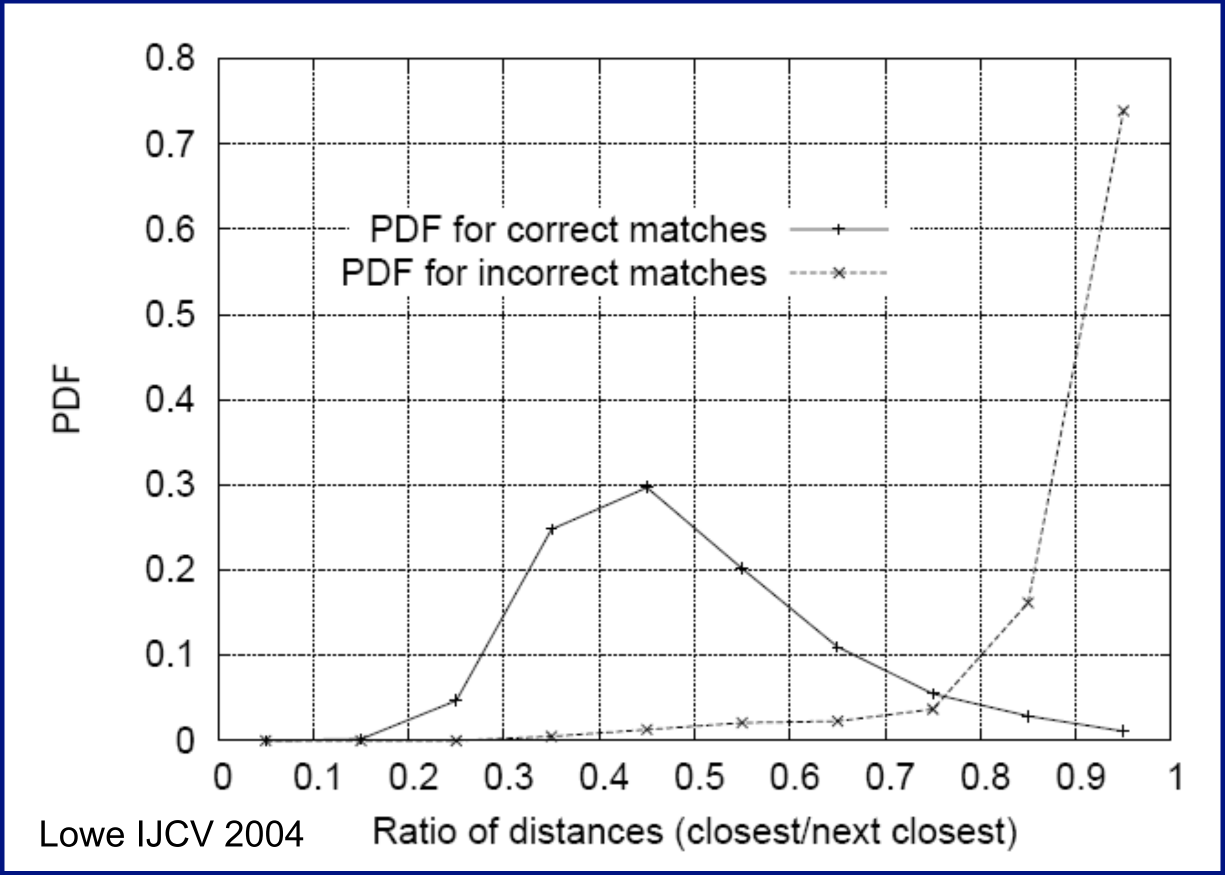


FIGURE 5.14: To construct a SIFT descriptor for a neighborhood, we place a grid over the rectified neighborhood. Each grid is divided into a subgrid, and a gradient estimate is computed at the center of each subgrid element. This gradient estimate is a weighted average of nearby gradients, with weights chosen so that gradients outside the subgrid cell contribute. The gradient estimates in each subgrid element are accumulated into an orientation histogram. Each gradient votes for its orientation, with a vote weighted by its magnitude and by its distance to the center of the neighborhood. The resulting orientation histograms are stacked to give a single feature vector. This is normalized to have unit norm; then terms in the normalized feature vector are thresholded, and the vector is normalized again.

# Matching SIFT features

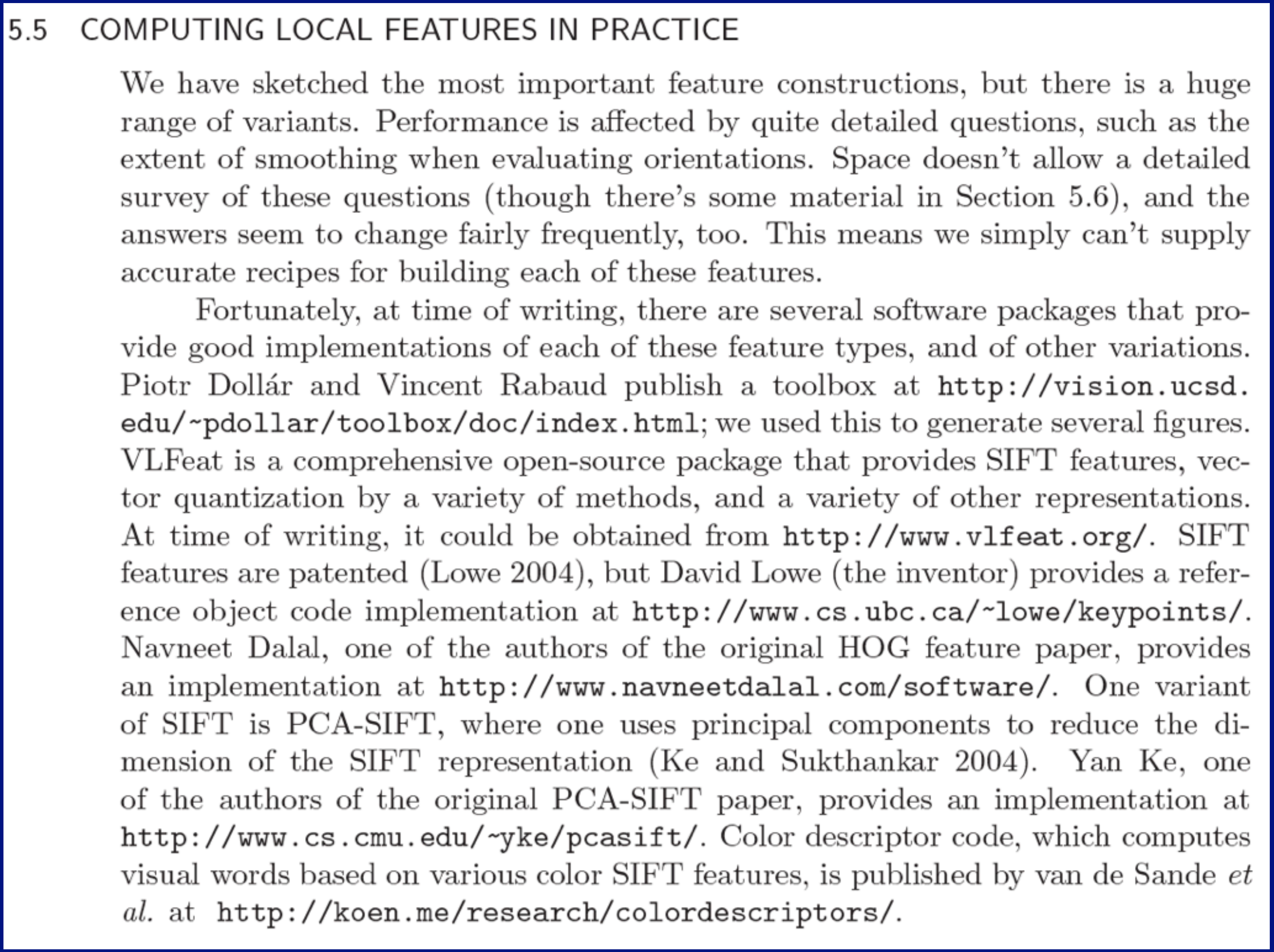
- Can be compared with Euclidean distance
  - test: (dist to closest)/(dist to second closest)



## HOG and SIFT - Crucial points

- Orientation based descriptors are very powerful
  - because robust to changes in brightness
- HOG feature
  - known window, make histogram of orientations
- SIFT feature
  - find domain
    - patch center and radius
  - compute descriptor
    - histogram of orientations
- Numerous powerful variants
- Software available

# Software, etc



## 5.5 COMPUTING LOCAL FEATURES IN PRACTICE

We have sketched the most important feature constructions, but there is a huge range of variants. Performance is affected by quite detailed questions, such as the extent of smoothing when evaluating orientations. Space doesn't allow a detailed survey of these questions (though there's some material in Section 5.6), and the answers seem to change fairly frequently, too. This means we simply can't supply accurate recipes for building each of these features.

Fortunately, at time of writing, there are several software packages that provide good implementations of each of these feature types, and of other variations. Piotr Dollár and Vincent Rabaud publish a toolbox at `http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html`; we used this to generate several figures. VLFeat is a comprehensive open-source package that provides SIFT features, vector quantization by a variety of methods, and a variety of other representations. At time of writing, it could be obtained from `http://www.vlfeat.org/`. SIFT features are patented (Lowe 2004), but David Lowe (the inventor) provides a reference object code implementation at `http://www.cs.ubc.ca/~lowe/keypoints/`. Navneet Dalal, one of the authors of the original HOG feature paper, provides an implementation at `http://www.navneetdalal.com/software/`. One variant of SIFT is PCA-SIFT, where one uses principal components to reduce the dimension of the SIFT representation (Ke and Sukthankar 2004). Yan Ke, one of the authors of the original PCA-SIFT paper, provides an implementation at `http://www.cs.cmu.edu/~yke/pcasift/`. Color descriptor code, which computes visual words based on various color SIFT features, is published by van de Sande *et al.* at `http://koen.me/research/colordescriptors/`.