Lecture 4: Smoothing

Related text is T&V Section 2.3.3 and Chapter 3

Summary about Convolution

Computing a linear operator in neighborhoods centered at each pixel. Can be thought of as sliding a kernel of fixed coefficients over the image, and doing a weighted sum in the area of overlap.

<u>things to take note of:</u>

full : compute a value for any overlap between kernel and image (resulting image is bigger than the original)

same: compute values only when center pixel of kernel aligns with a pixel in the image (resulting image is same size as original)

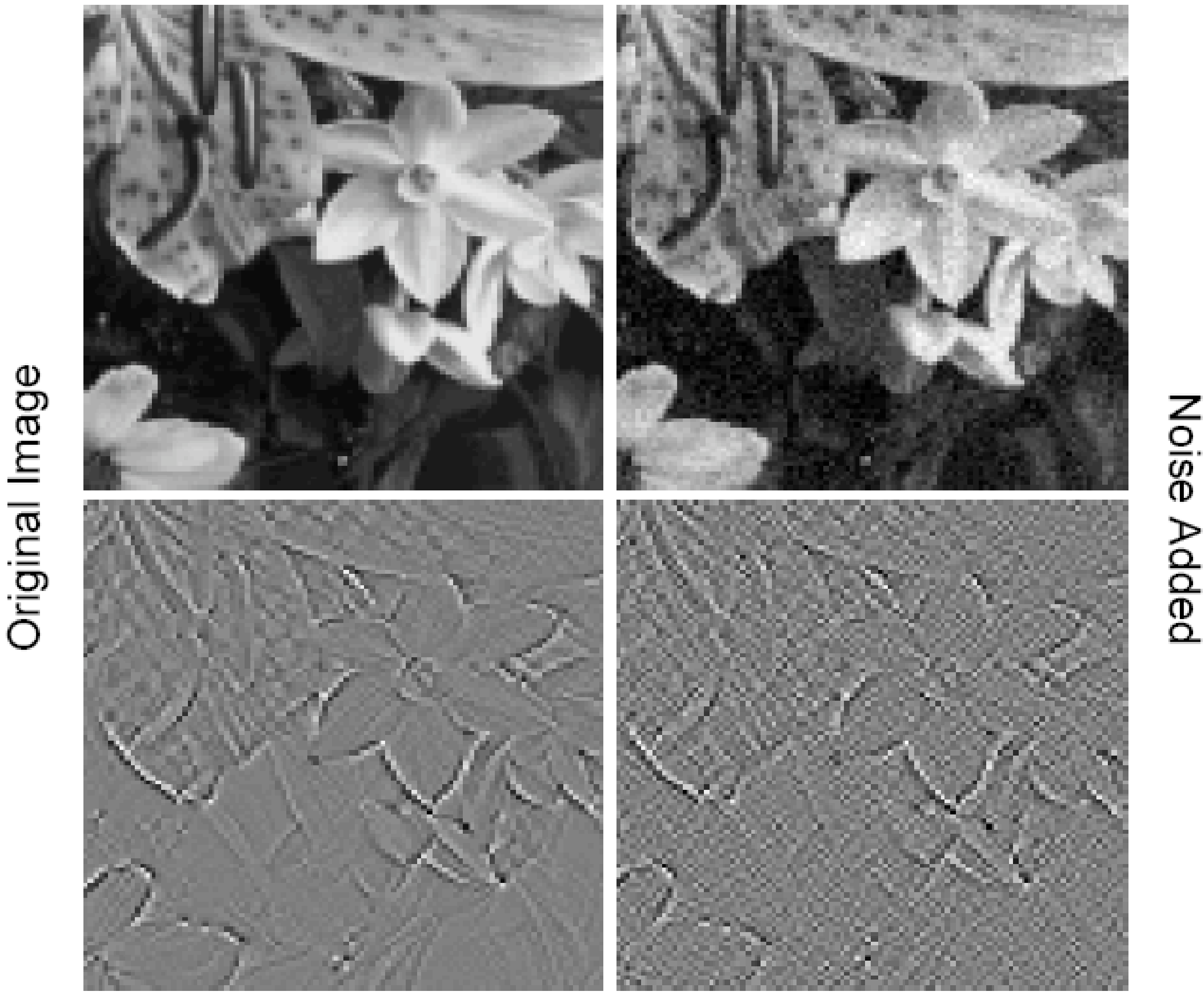
convolution : kernel gets rotated 180 degrees before sliding over the image

cross-correlation: kernel does not get rotated first

border handling methods : defining values for pixels off the image

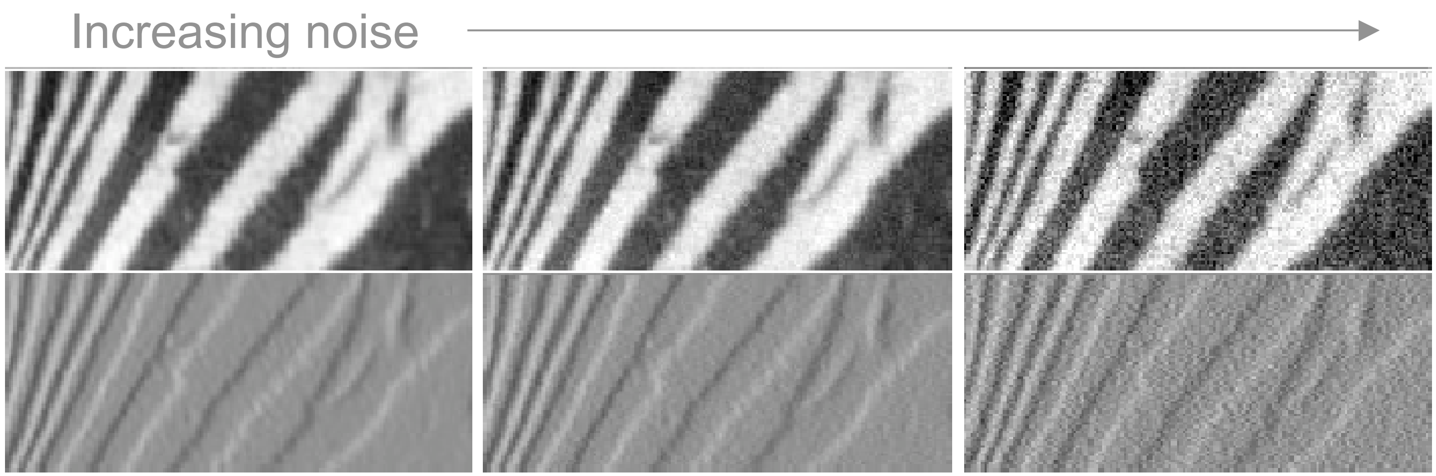
Robert Collins
CSE486, Penn State

Problem: Derivatives and Noise



Problem: Derivatives and Noise

- First derivative operator is affected by noise



- Numerical derivatives can amplify noise!
  (particularly higher order derivatives)

Image Noise

- Fact: Images are noisy
- Noise is anything in the image that we are not interested in
- Examples:
  - Light fluctuations
  - Sensor noise
  - Quantization effects
  - Finite precision

Modeling Image Noise

Simple model: additive RANDOM noise

$I(x,y) = s(x,y) + n_i$

Where $s(x,y)$ is the deterministic signal

n_i is a random variable

Common Assumptions:

n is i.i.d for all pixels

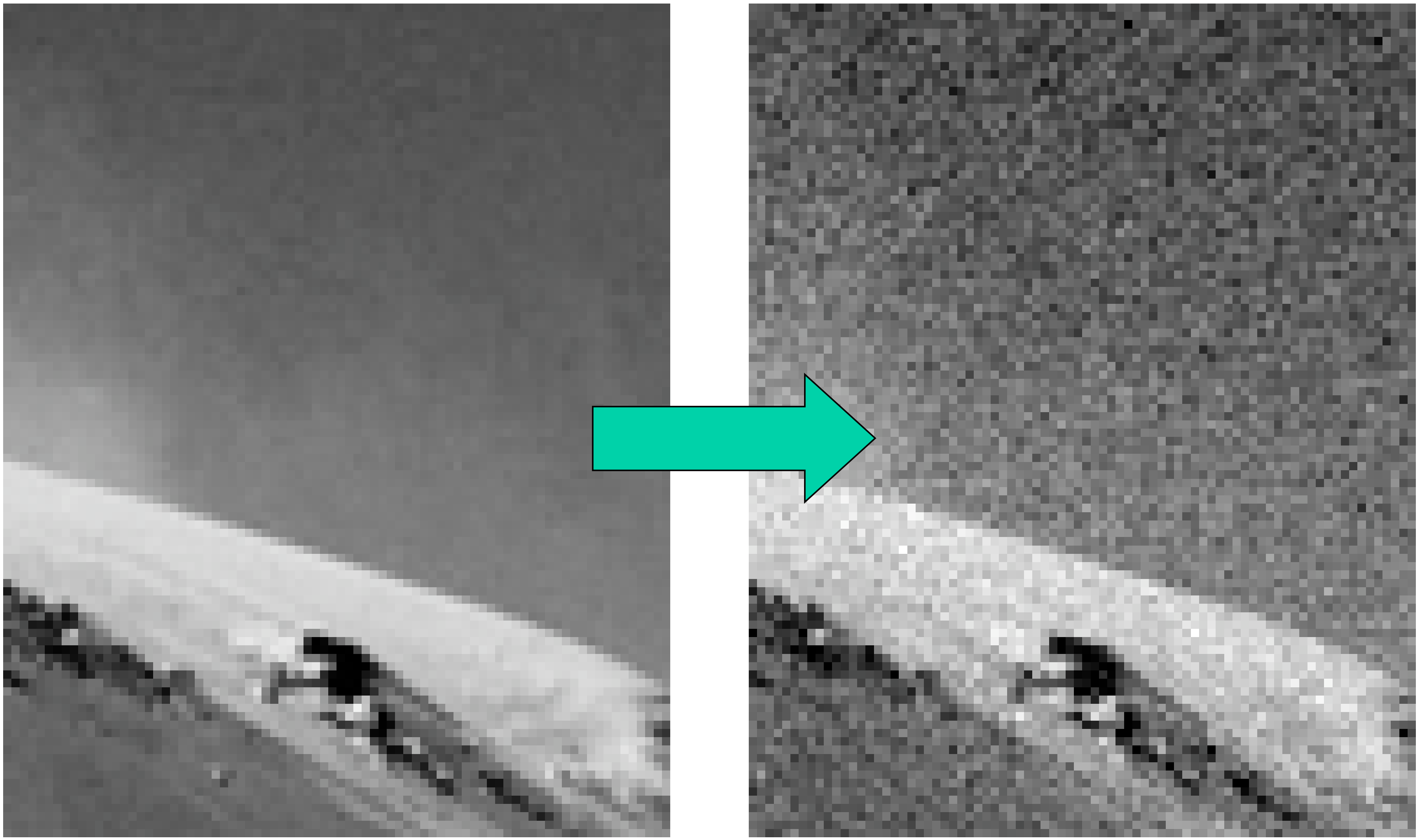
n is zero-mean Gaussian (normal)

$E(n) = 0 \quad var(n) = \sigma^2$

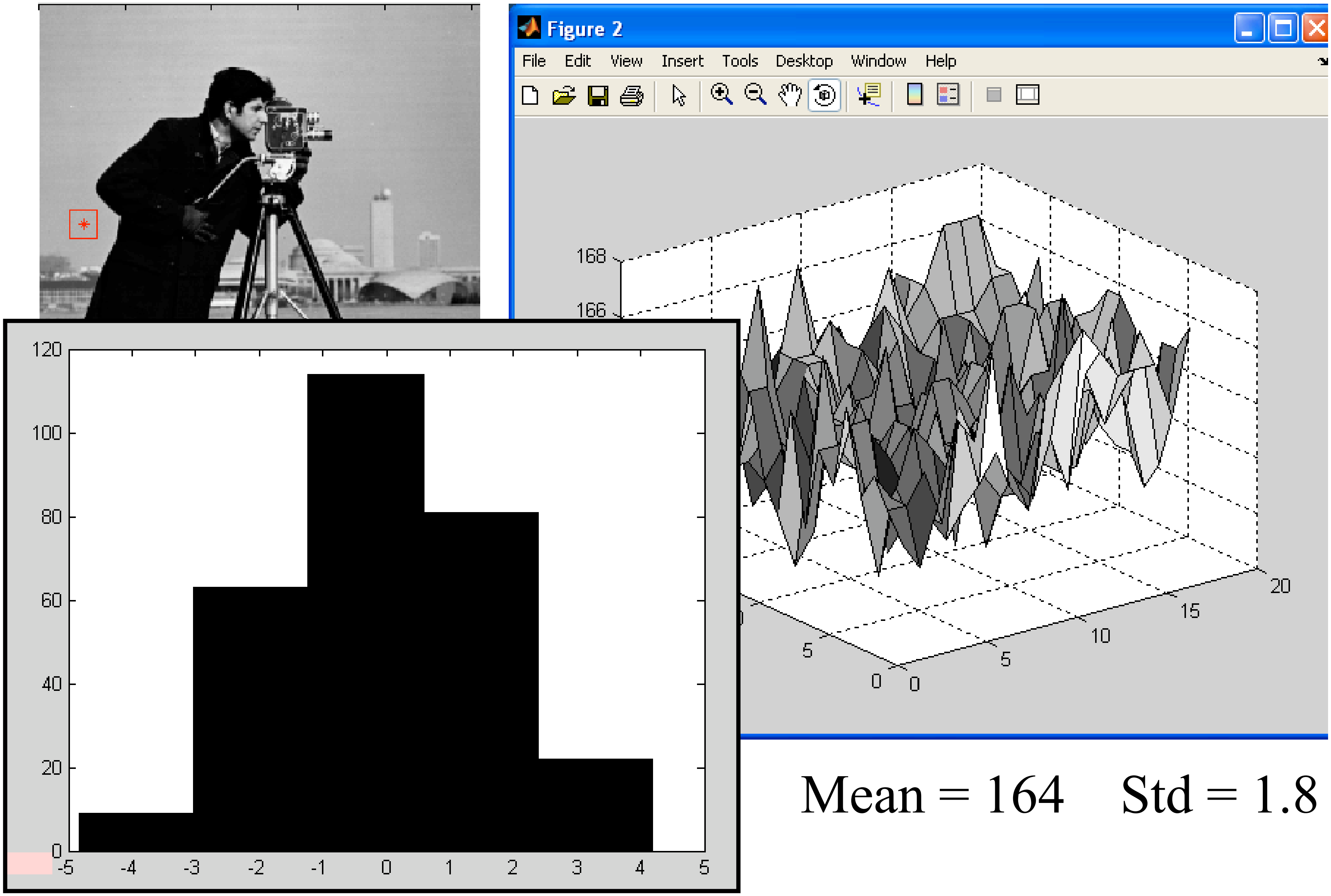
$E(n_i\, n_j) = 0$ (independence)

Note: This really only models the sensor noise.

Example: Additive Gaussian Noise
mean 0, sigma = 16



Empirical Evidence



Mean = 164   Std = 1.8

Other Noise Models

We won't cover them. Just be aware they exist!!

- Multiplicative noise: $I(i,j) = \hat{I}(i,j) \cdot N(i,j)$

- Impulse ("shot") noise (aka salt and pepper):

$$I(i,j) = \begin{cases} \hat{I}(i,j) & \text{if } x < l \\ i_{\min} + y(i_{\max} - i_{\min}) & x \geq l \end{cases}$$

See Textbook

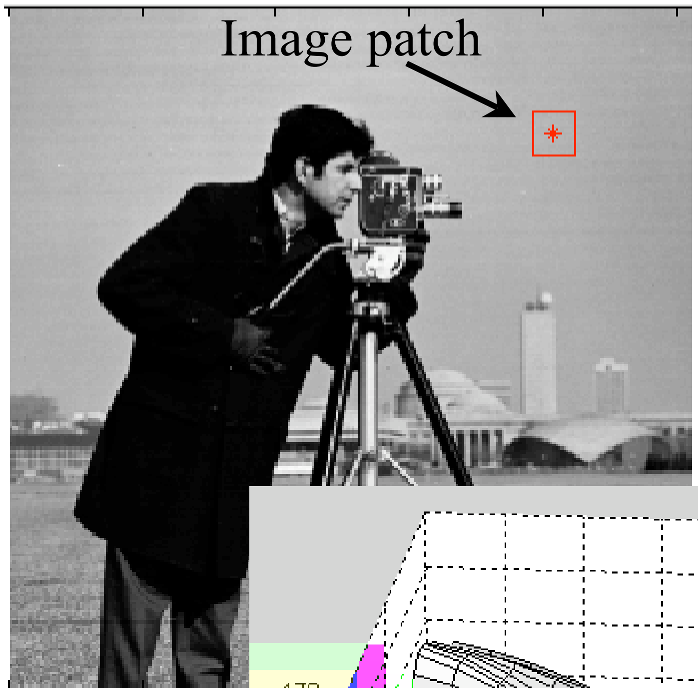
Smoothing Reduces Noise

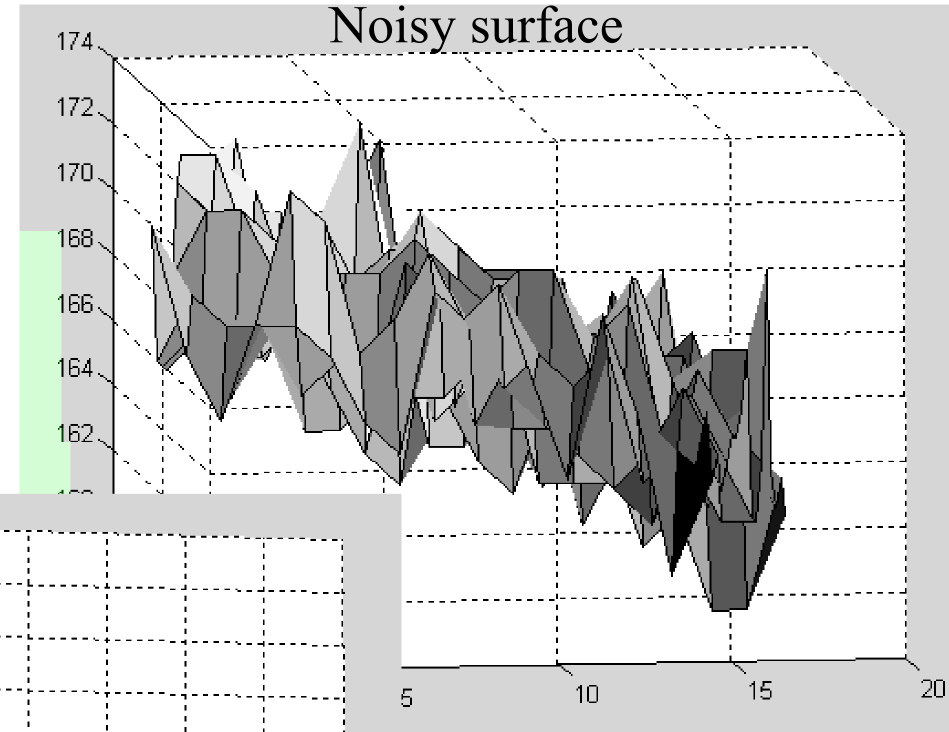
From Numerical Recipes in C:

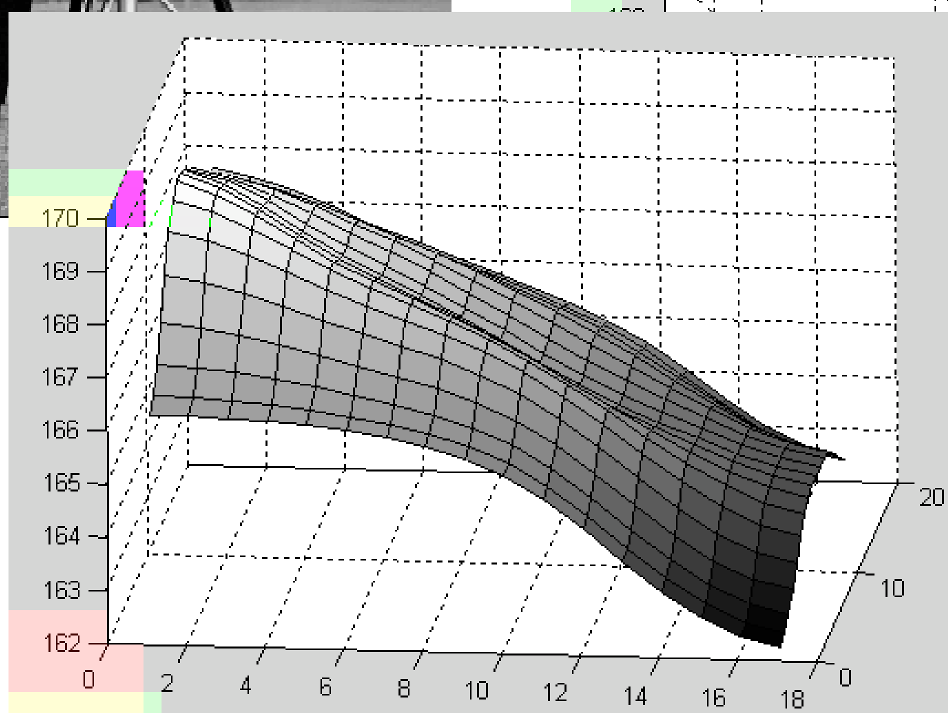
The premise of data smoothing is that one is measuring a variable that is both slowly varying and also corrupted by random noise. Then it can sometimes be useful to replace each data point by some kind of local average of surrounding data points. Since nearby points measure very nearly the same underlying value, averaging can reduce the level of noise without (much) biasing the value obtained.

Robert Collins
CSE486, Penn State

Today: Smoothing Reduces Noise







smoothing reduces noise, giving us (perhaps) a more accurate intensity surface.

Preview

- We will talk about two smoothing filters
  - Box filter (simple averaging)
  - Gaussian filter (center pixels weighted more)

Averaging / Box Filter

- Mask with positive entries that sum to 1.
- Replaces each pixel with an average of its neighborhood.
- Since all weights are equal, it is called a BOX filter.

Box filter

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

important point:

since this is a linear operator, we can take the average around each pixel by convolving the image with this 3x3 filter!

Why Averaging Reduces Noise

- Intuitive explanation: variance of noise in the average is smaller than variance of the pixel noise (assuming zero-mean Gaussian noise).

- Sketch of more rigorous explanation:

$$A = \frac{1}{m^2} \sum_{i=1}^{m^2} I_m$$

$I_m = s_m + n_m$ with n being i.i.d. $G(0, \sigma^2)$

$$E(A) = \frac{1}{m^2} \sum s_m$$

$$var(A) = E\left[(A - E(A))^2\right] = \frac{\sigma^2}{m}$$

Robert Collins
CSE486, Penn State

Smoothing with Box Filter

original

Convolved with 11x11 box filter

Drawback: smoothing reduces fine image detail

Important Point about Smoothing

Averaging attenuates noise (reduces the variance), leading to a more "accurate" estimate.

However, the more accurate estimate is of the mean of a local pixel neighborhood! This might not be what you want.

Balancing act: smooth enough to "clean up" the noise, but not so much as to remove important image gradients.

Gaussian Smoothing Filter

- a case of weighted averaging
  - The coefficients are a 2D Gaussian.
  - Gives more weight at the central pixels and less weights to the neighbors.
  - The farther away the neighbors, the smaller the weight.

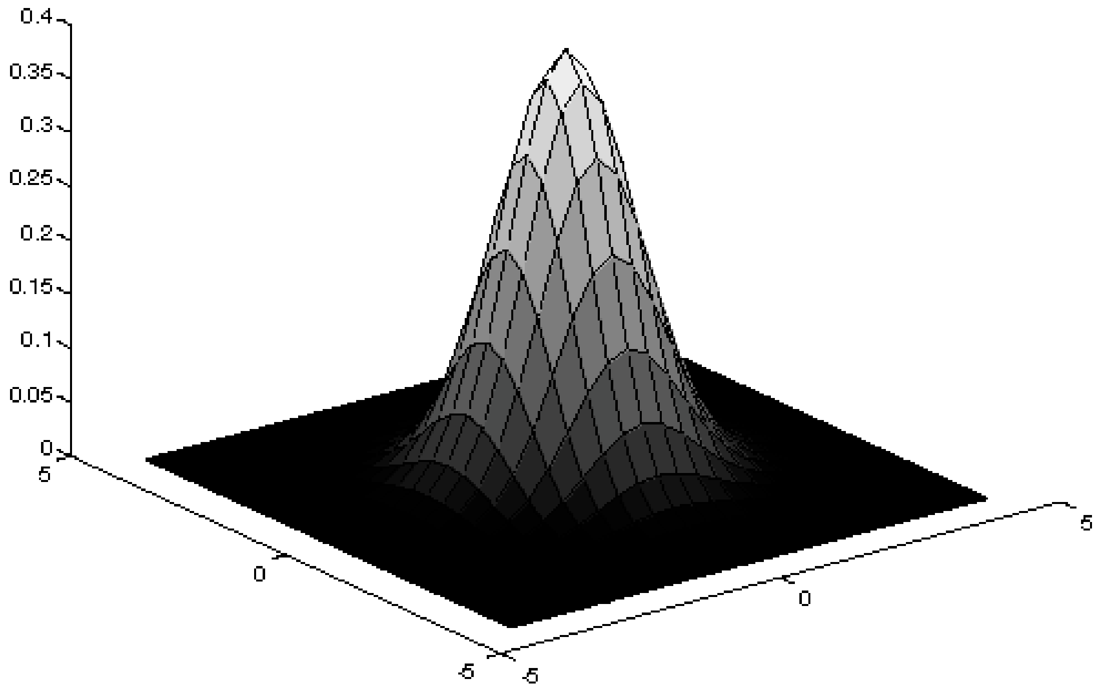
$$G_\sigma \equiv \frac{1}{2\pi\sigma^2} \exp\{-\frac{x^2+y^2}{2\sigma^2}\}$$

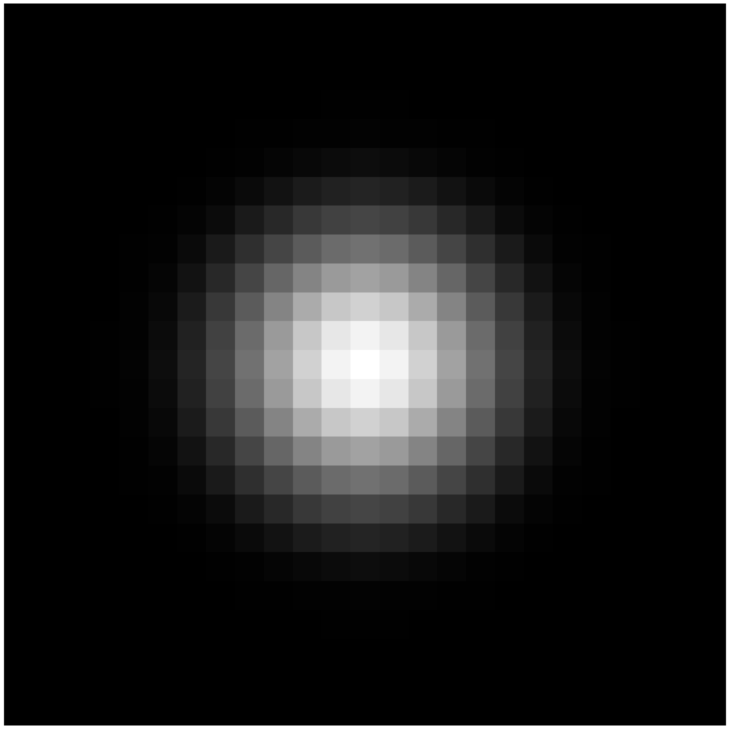
Confusion alert: there are now two Gaussians being discussed here (one for noise, one for smoothing). They are different.

Robert Collins
CSE486, Penn State

Gaussian Smoothing Filter

An isotropic (circularly symmetric) Gaussian:





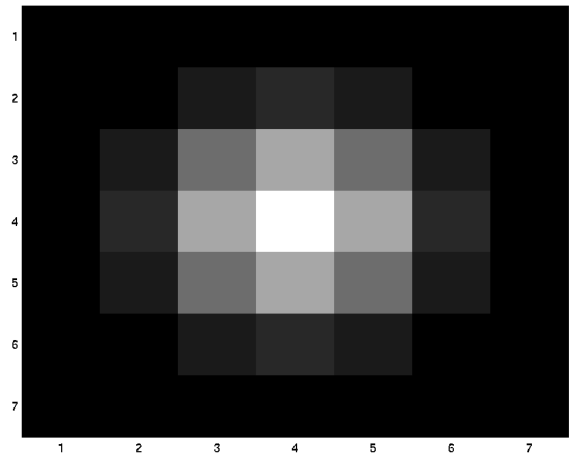
In Matlab

```
>> sigma = 1

sigma =

     1

>> halfwid = 3*sigma

halfwid =

     3

>> [xx,yy] = meshgrid(-halfwid:halfwid, -halfwid:halfwid);
>> tmp = exp(-1/(2*sigma^2) * (xx.^2 + yy.^2))

tmp =

    0.0001    0.0015    0.0067    0.0111    0.0067    0.0015    0.0001
    0.0015    0.0183    0.0821    0.1353    0.0821    0.0183    0.0015
    0.0067    0.0821    0.3679    0.6065    0.3679    0.0821    0.0067
    0.0111    0.1353    0.6065    1.0000    0.6065    0.1353    0.0111
    0.0067    0.0821    0.3679    0.6065    0.3679    0.0821    0.0067
    0.0015    0.0183    0.0821    0.1353    0.0821    0.0183    0.0015
    0.0001    0.0015    0.0067    0.0111    0.0067    0.0015    0.0001
```

Note: we have not included the normalization constant. Values of a Gaussian should sum to one. However, it's OK to ignore the constant since you can divide by it later.

Robert Collins
CSE486, Penn State

Gaussian Smoothing Filter




```
>> sigma = 1

sigma =

     1

>> halfwid = 3*sigma

halfwid =

     3
>> [xx,yy] = meshgrid(-halfwid:halfwid, -halfwid:halfwid);
>> gau = exp(-1/(2*sigma^2) * (xx.^2 + yy.^2))

gau =

    0.0001    0.0015    0.0067    0.0111    0.0067    0.0015    0.0001
    0.0015    0.0183    0.0821    0.1353    0.0821    0.0183    0.0015
    0.0067    0.0821    0.3679    0.6065    0.3679    0.0821    0.0067
    0.0111    0.1353    0.6065    1.0000    0.6065    0.1353    0.0111
    0.0067    0.0821    0.3679    0.6065    0.3679    0.0821    0.0067
    0.0015    0.0183    0.0821    0.1353    0.0821    0.0183    0.0015
    0.0001    0.0015    0.0067    0.0111    0.0067    0.0015    0.0001
```

Just another linear filter. Performs a weighted average.
Can be convolved with an image to produce a smoother image.

Robert Collins
CSE486, Penn State

Gaussian Smoothing Example

original **sigma = 3**

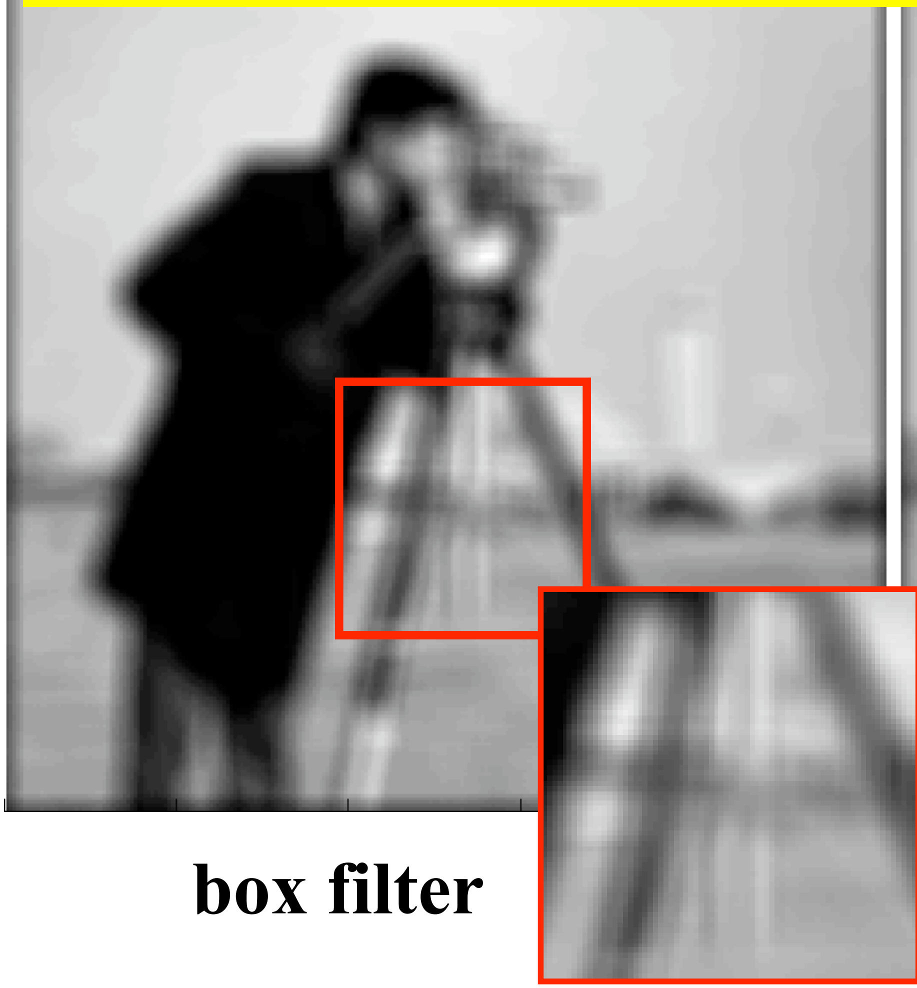
Box vs Gaussian

box filter

gaussian

Box vs Gaussian

Note: Gaussian is a true low-pass filter, so won't cause high frequency artifacts. See T&V Chap3 for more info.



box filter

gaussian

Robert Collins
CSE486, Penn State

Gaussian Smoothing at Different Scales

original

sigma = 1

Gaussian Smoothing at Different Scales

original

sigma = 3

Robert Collins
CSE486, Penn State

Gaussian Smoothing at Different Scales

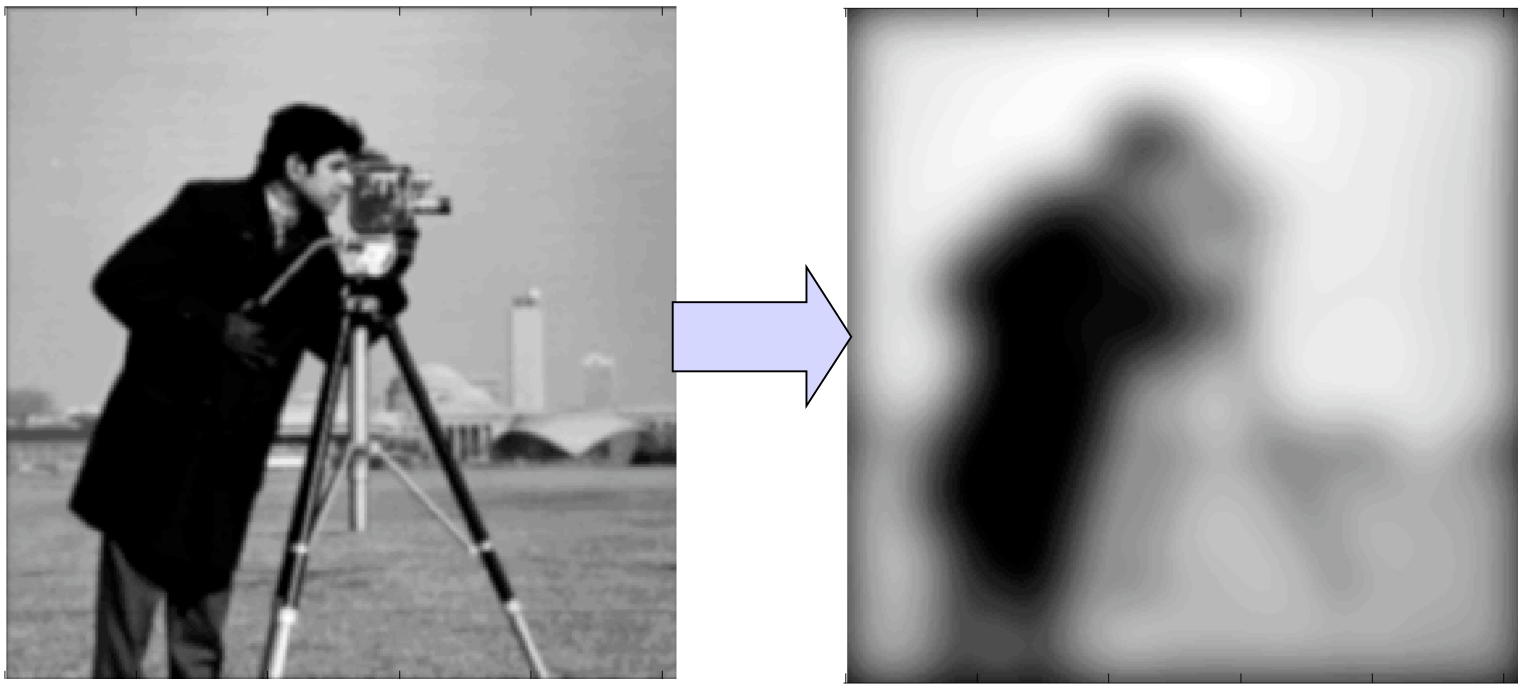
original

sigma = 10

Gaussian Smoothing at Different Scales

Later in the course we will look more deeply into the notions of scale and resolution.

A Small Aside...



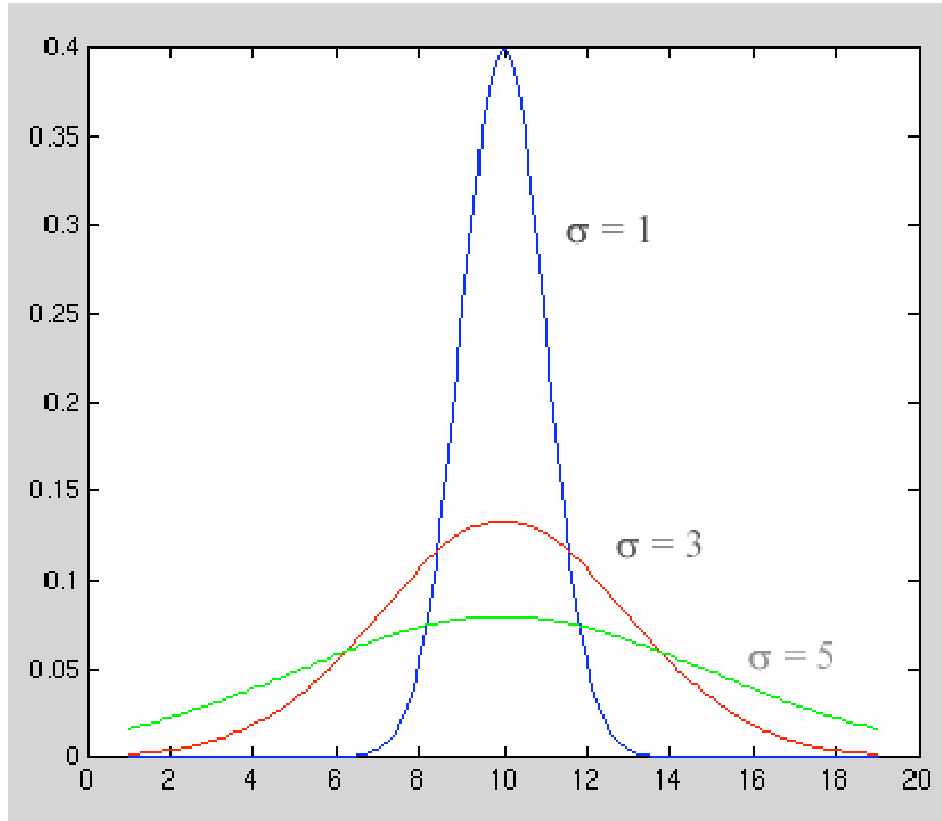
can you guess what
border-handling method
I used when convolving???

Robert Collins
CSE486, Penn State

Implementation Issues

How big should a Gaussian mask be?

- The std. dev σ of the Gaussian determines the amount of smoothing.
- Gaussian theoretically has infinite support, but we need a filter of finite size.
- For a 98.76% of the area, we need +/-2.5σ
- +/- 3σ covers over 99% of the area.



Efficient Implementation

- Both, the Box filter and the Gaussian filter are separable:
  - First convolve each row with a 1D filter
  - Then convolve each column with a 1D filter.

Separable Gaussian: **associativity**

$$G_\sigma * f = \left(g_{\sigma\rightarrow} * g_{\sigma\uparrow}\right) * f \;=\; g_{\sigma\rightarrow} * \left(g_{\sigma\uparrow} * f\right)$$

- Explain why Gaussian can be factored, on the board. (sketch: write out convolution and use identity $e^A e^B = e^{A+B}$)

Efficient Implementation

- Cascaded Gaussians
  - Repeated convolution by a smaller Gaussian to simulate effects of a larger one.
- G*(G*f) = (G*G)*f   [associative]

This is important!

- Note: $G_{\sigma_1} * G_{\sigma_2} = G_{\sigma} \qquad \sigma^2 = \sigma_1^{\,2} + \sigma_2^{\,2}$

- explanation sketch: convolution in spatial domain is multiplication in frequency domain (Fourier space). Fourier transform of Gaussian is

$$\mathcal{F}_x\left[e^{-\frac{x^2}{2\sigma^2}}\right] = e^{-2\pi^2\sigma^2u^2}$$

$$e^{-2\pi^2\sigma_1^2u^2}e^{-2\pi^2\sigma_2^2u^2} = e^{-2\pi^2(\sigma_1^2+\sigma_2^2)u^2}$$

Efficient Implementation

- Cascaded Gaussians
  - Repeated convolution by a smaller Gaussian to simulate effects of a larger one.
- G*(G*f) = (G*G)*f [associativity]

This is important!

- Note: $G_{\sigma_1} * G_{\sigma_2} = G_{\sigma} \quad \sigma^2 = \sigma_1^2 + \sigma_2^2$

Confusion alert!
σ is std.dev
σ^2 is variance

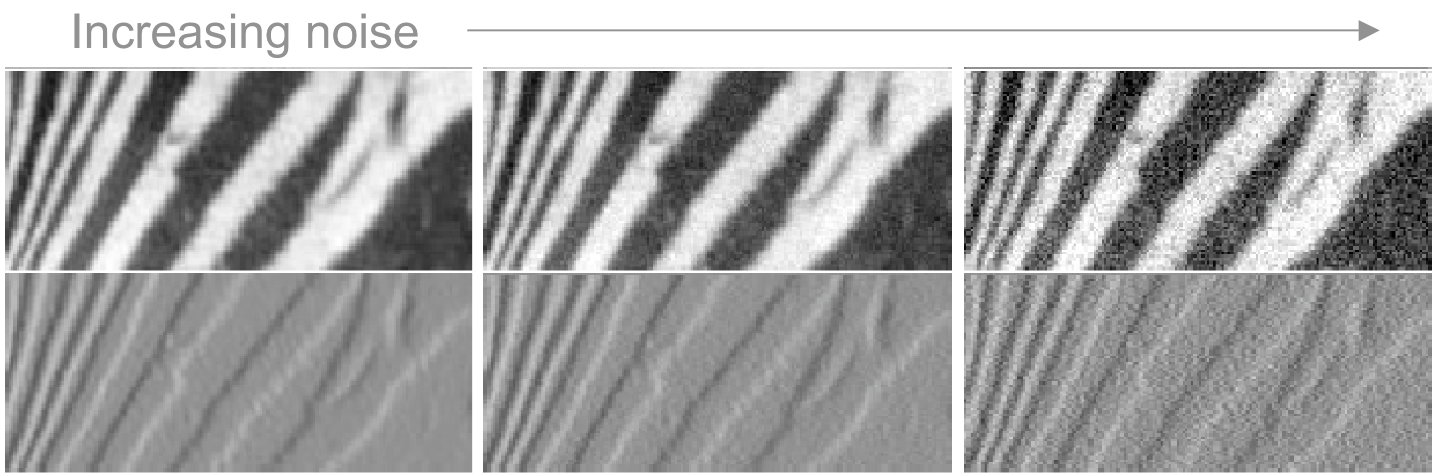
- explanation sketch: convolution in spatial multiplication in frequency domain (Four Fourier transform of Gaussian is

$$\mathcal{F}_x\left[e^{-\frac{x^2}{2\sigma^2}}\right] = e^{-2\pi^2\sigma^2u^2}$$

$$e^{-2\pi^2\sigma_1^2u^2}e^{-2\pi^2\sigma_2^2u^2} = e^{-2\pi^2(\sigma_1^2+\sigma_2^2)u^2}$$

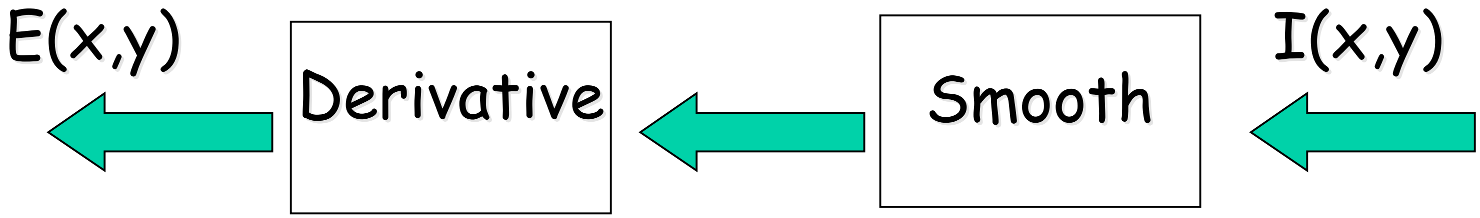
Recall: Derivatives and Noise

- derivative operator is affected by noise



- Numerical derivatives can <u>amplify</u> noise!
  (particularly higher order derivatives)

Solution: Smooth before Applying Derivative Operator!



Question: Do we have to apply two linear operations here (convolutions)?

DerivFilter * (SmoothFilter * I)

Robert Collins
CSE486, Penn State

Smoothing and Differentiation

No, we can combine filters!

By associativity of convolution operator:

$$\text{DerivFilter} * (\text{SmoothFilter} * \text{I})$$

$$= \underbrace{(\text{DerivFilter} * \text{SmoothFilter})}_{\text{we can precompute this part as a single kernel to apply}} * \text{I}$$

Example: Prewitt Operator

Convolve with:

-1	0	1
-1	0	1
-1	0	1

Noise Smoothing

Vertical Edge Detection

This mask is called the (vertical) Prewitt Edge Detector

Example: Prewitt Operator

Convolve with:

-1	-1	-1
0	0	0
1	1	1

Noise Smoothing

Horizontal Edge Detection

This mask is called the (horizontal) Prewitt Edge Detector

Example: Sobel Operator

Convolve with:

-1	0	1
-2	0	2
-1	0	1

and

-1	-2	-1
0	0	0
1	2	1

Gives more weight to the 4-neighbors

Important Observation

Note that a Prewitt operator is a box filter convolved with a derivative operator [using “full” option].

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} = \overbrace{\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}}^{\textbf{Finite diff operator}} * \overbrace{\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}}^{\textbf{Simple box filter}}$$

Also note: a Sobel operator is a [1 2 1] filter convolved with a derivative operator.

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \overbrace{\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}}^{\textbf{Finite diff operator}} * \overbrace{\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}}^{\textbf{Simple Gaussian}}$$

Generalize: Smooth Derivatives

- Solution: First smooth the image by a Gaussian G_s and then take derivatives:

$$\frac{\partial f}{\partial x} \approx \frac{\partial (G_\sigma * f)}{\partial x}$$

- Applying the differentiation property of the convolution:

$$\frac{\partial f}{\partial x} \approx \frac{\partial G_\sigma}{\partial x} * f$$

- Therefore, taking the derivative in x of the image can be done by convolution with the derivative of a Gaussian:

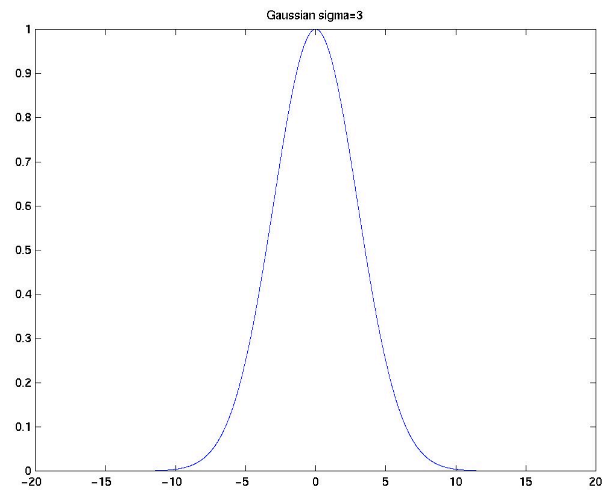
$$G_\sigma^x = \frac{\partial G_\sigma}{\partial x} = xe^{-\frac{x^2+y^2}{2\sigma^2}}$$

- Crucial property: The Gaussian derivative is also separable:

$$G_\sigma^x * f = g_\sigma^x * g_{\sigma\uparrow} * f$$

First (partial) Derivative of a Gaussian

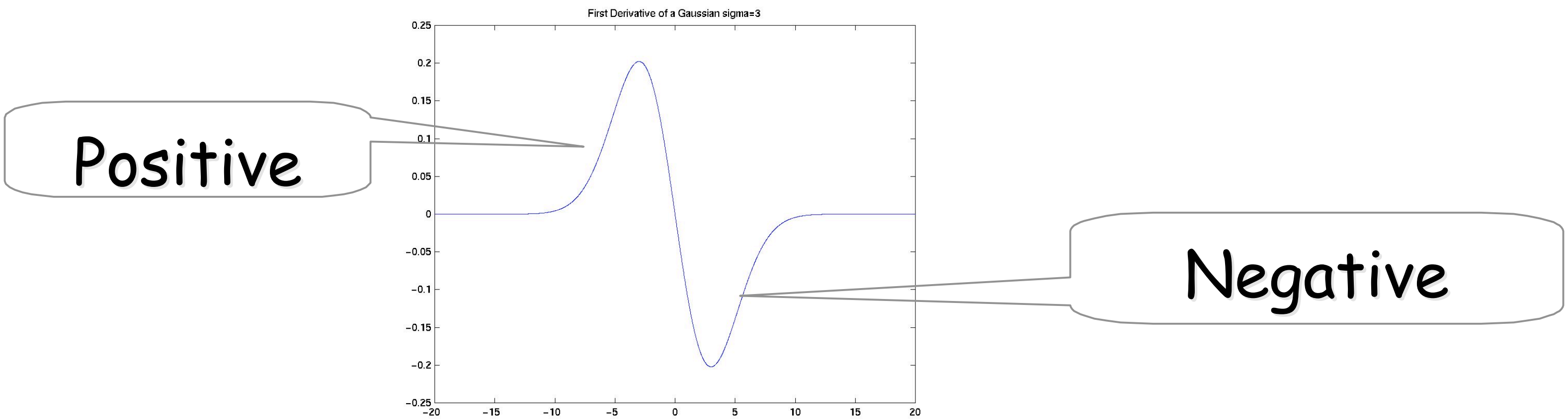
$$g(x) = e^{-\frac{x^2}{2\sigma^2}}$$



$$g'(x) = -\frac{1}{2\sigma^2} 2x e^{-\frac{x^2}{2\sigma^2}} = -\frac{x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}}$$

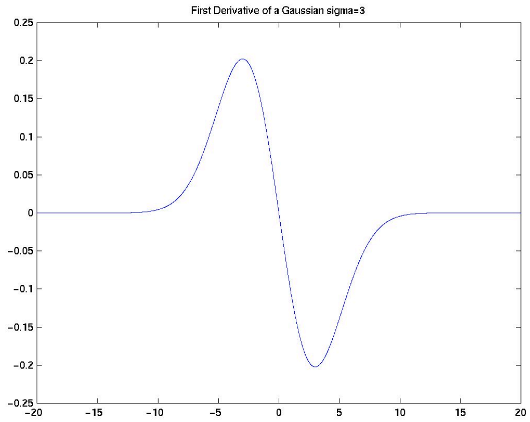
First (partial) Derivative of a Gaussian

$$g'(x) = -\frac{1}{2\sigma^2} 2xe^{-\frac{x^2}{2\sigma^2}} = -\frac{x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}}$$

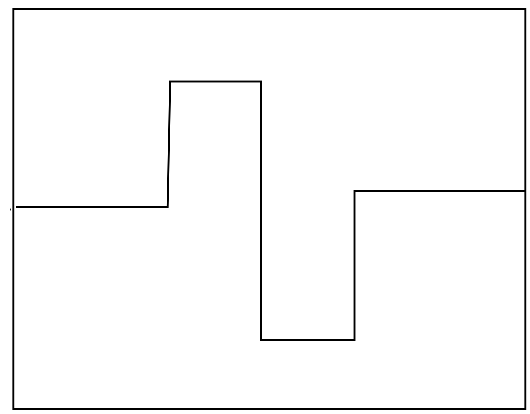


As a mask, it is also computing a difference (derivative)

Compare with finite diff operator



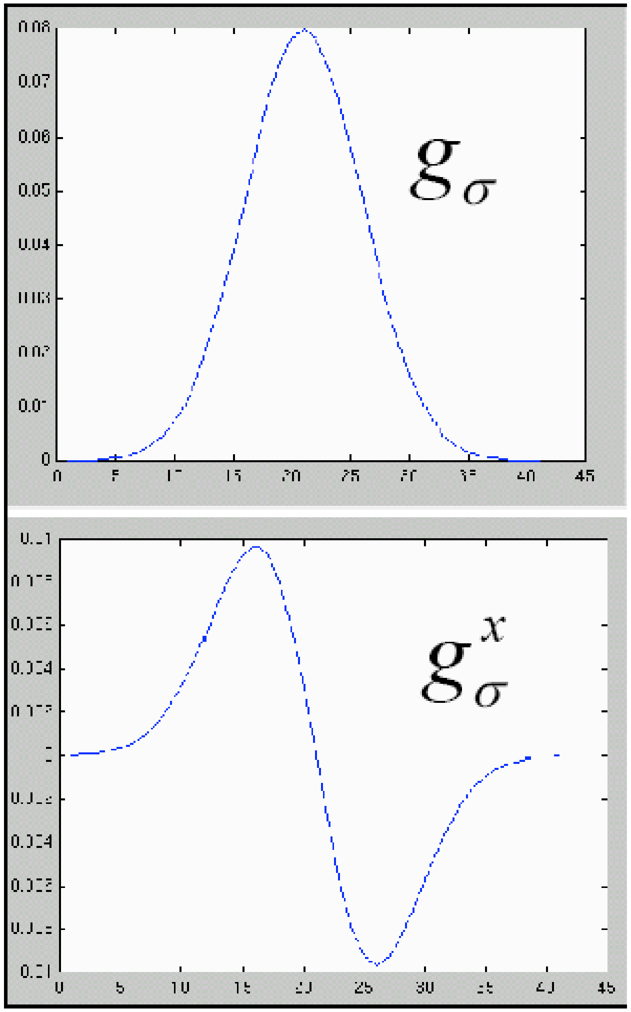
deriv of Gaussian

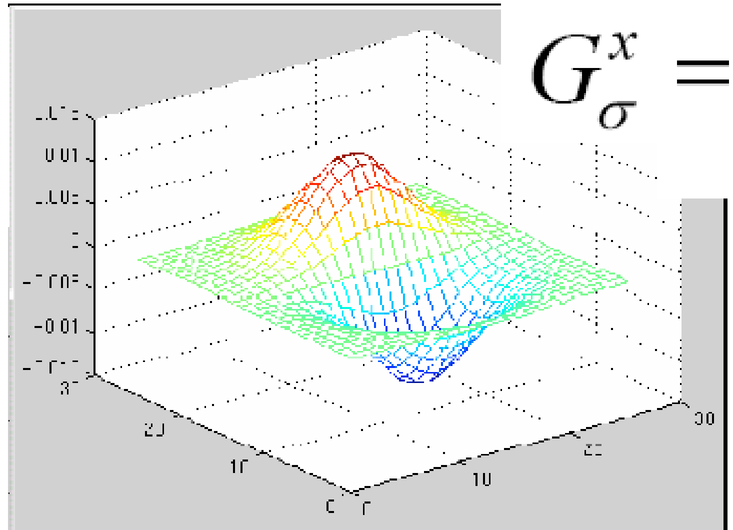


finite diff operator

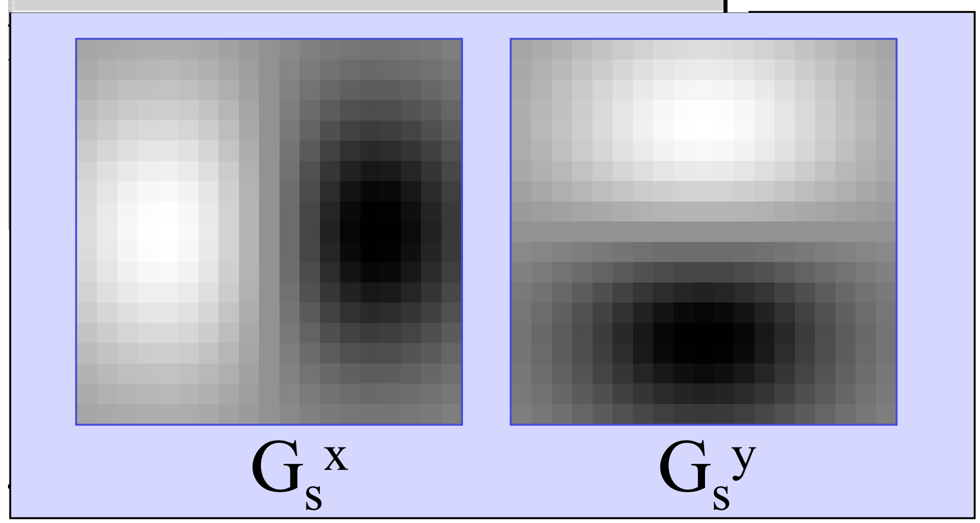
Robert Collins
CSE486, Penn State

Derivative of Gaussian Filter





$$G_\sigma^x = g_\sigma^x * g_\sigma$$



Robert Collins
CSE486, Penn State

Summary: Smooth Derivatives