Computational Photography

Prof. Feng Liu

Spring 2022

http://www.cs.pdx.edu/~fliu/courses/cs510/

04/05/2022

Last Time

- Digital Camera
  - History of Camera
  - Controlling Camera
- Photography Concepts

Today

- Filters and its applications
- Paper presentation schedule available
  - https://docs.google.com/spreadsheets/d/1a6biKmsEFtEzkrNcGCZLbKF1egnQydiN2svN9uETIGs/edit#gid=0
  - Log in using your pdx.edu to view this link
  - Up to Week 5
  - More will be added

Today

- Filters and its applications

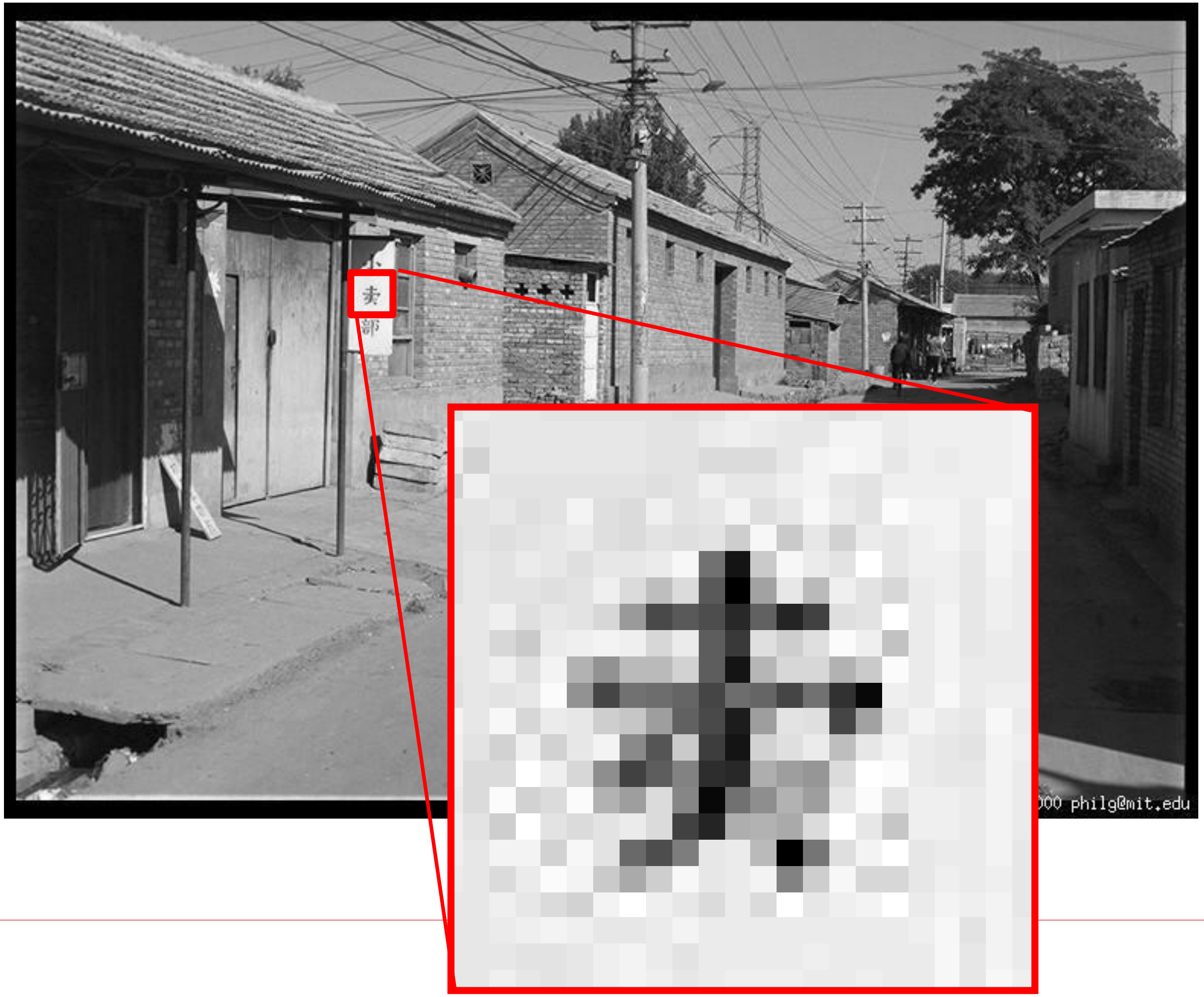
noisy image

naïve denoising
Gaussian blur

better denoising
edge-preserving filter

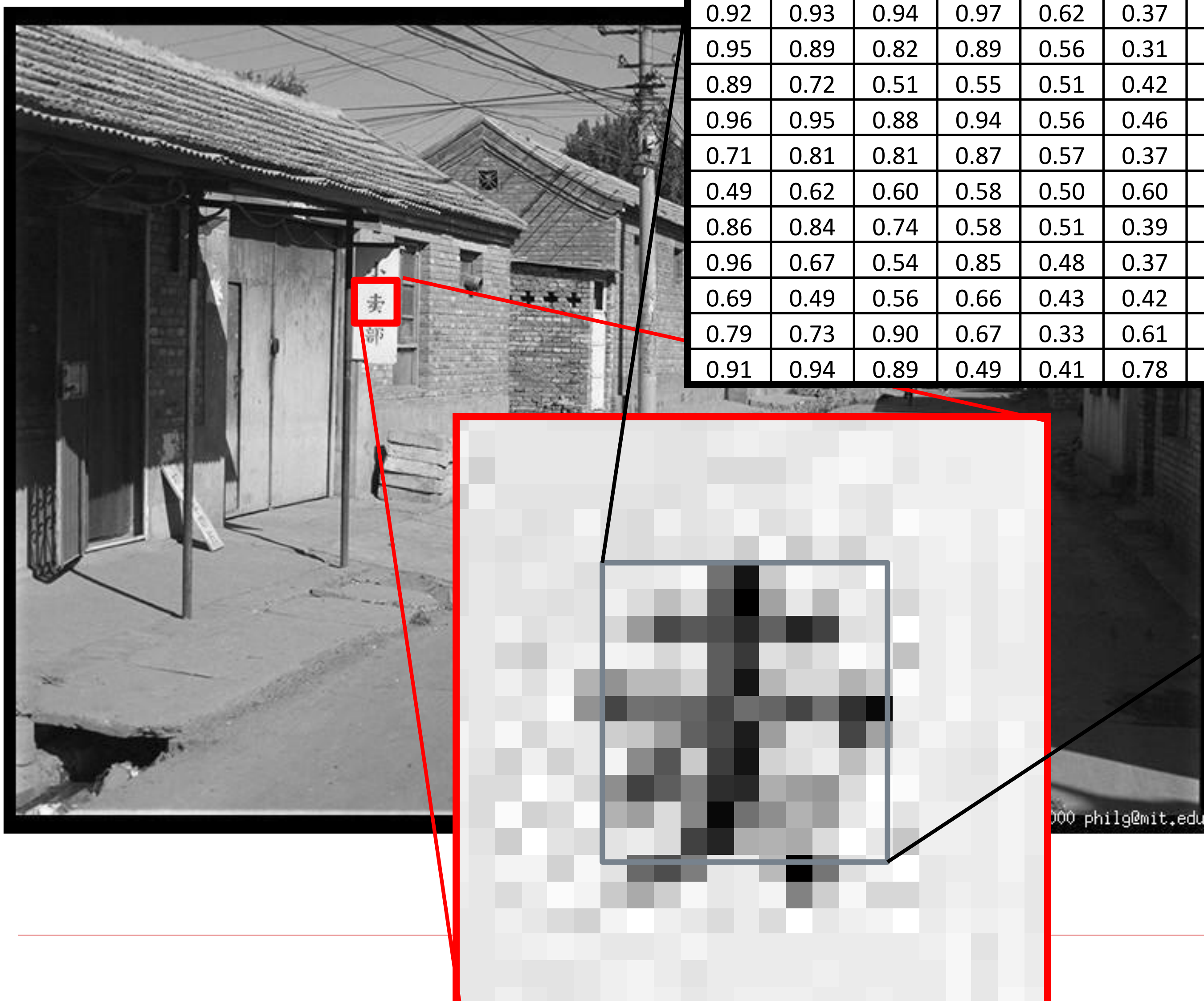
Slide credit: Sylvain Paris and Frédo Durand

The raster image (pixel matrix)



Slide credit: D. Hoiem

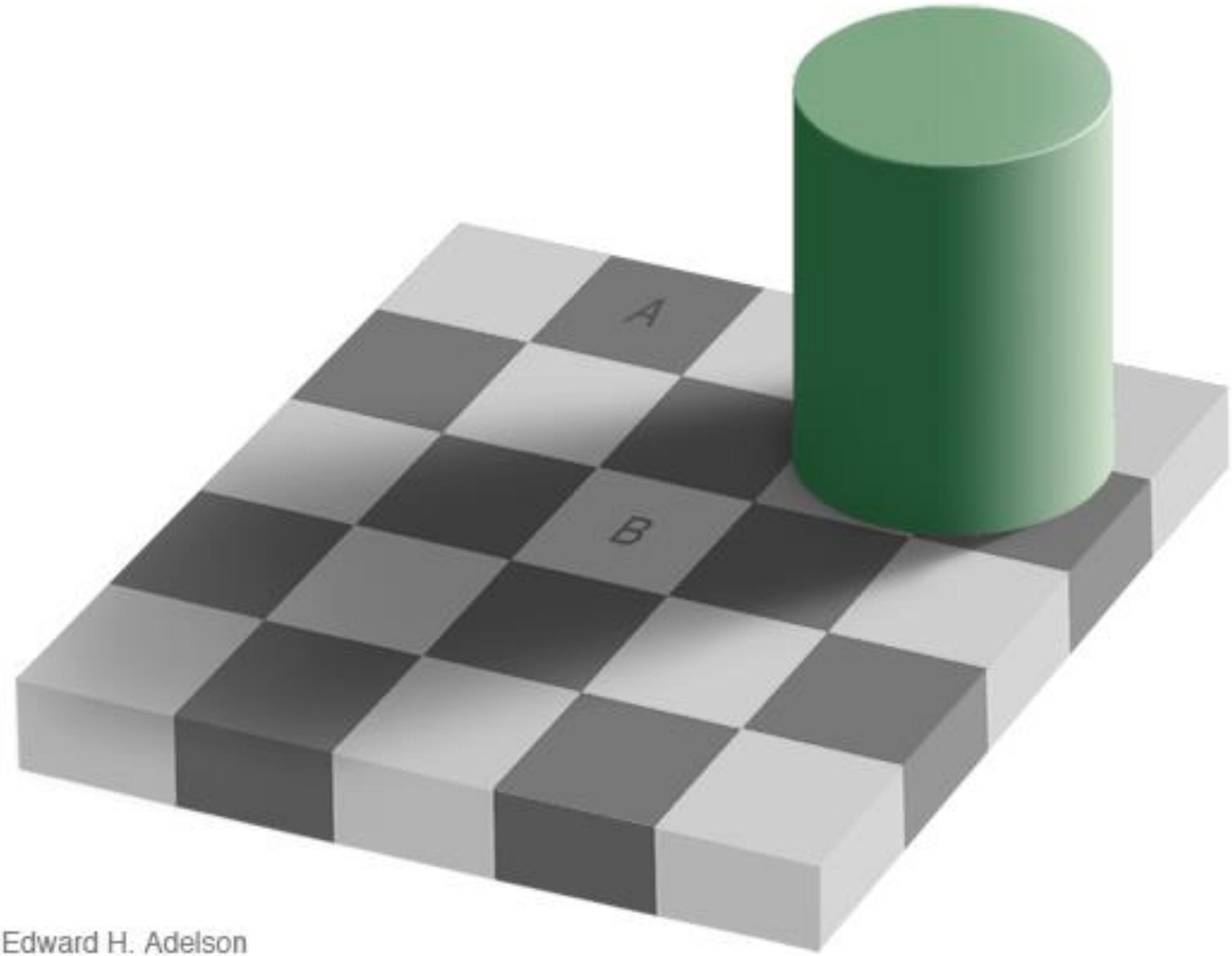
The raster image (pixel matrix)



0.92	0.93	0.94	0.97	0.62	0.37	0.85	0.97	0.93	0.92	0.99
0.95	0.89	0.82	0.89	0.56	0.31	0.75	0.92	0.81	0.95	0.91
0.89	0.72	0.51	0.55	0.51	0.42	0.57	0.41	0.49	0.91	0.92
0.96	0.95	0.88	0.94	0.56	0.46	0.91	0.87	0.90	0.97	0.95
0.71	0.81	0.81	0.87	0.57	0.37	0.80	0.88	0.89	0.79	0.85
0.49	0.62	0.60	0.58	0.50	0.60	0.58	0.50	0.61	0.45	0.33
0.86	0.84	0.74	0.58	0.51	0.39	0.73	0.92	0.91	0.49	0.74
0.96	0.67	0.54	0.85	0.48	0.37	0.88	0.90	0.94	0.82	0.93
0.69	0.49	0.56	0.66	0.43	0.42	0.77	0.73	0.71	0.90	0.99
0.79	0.73	0.90	0.67	0.33	0.61	0.69	0.79	0.73	0.93	0.97
0.91	0.94	0.89	0.49	0.41	0.78	0.78	0.77	0.89	0.99	0.93

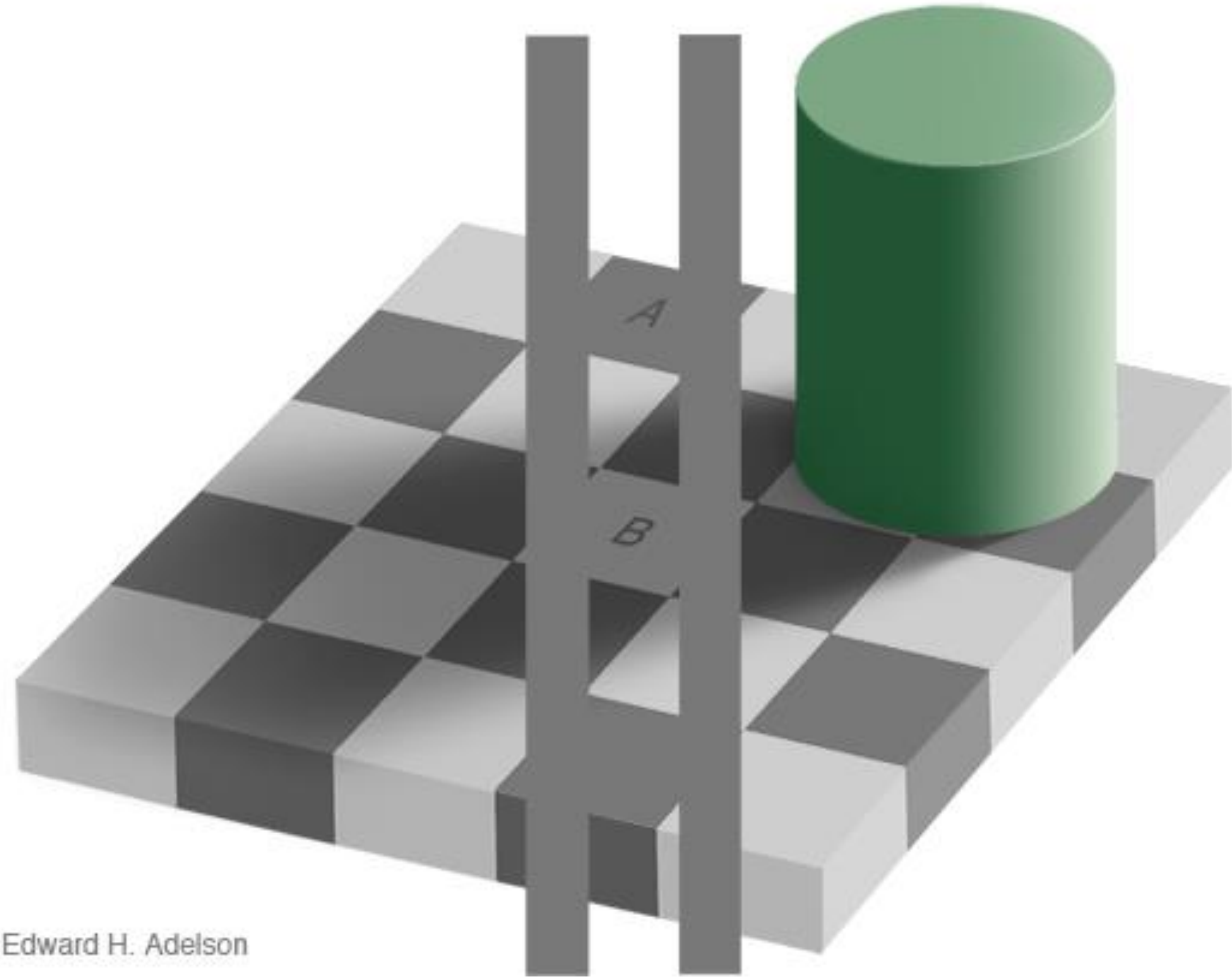
Slide credit: D. Hoiem

Perception of Intensity



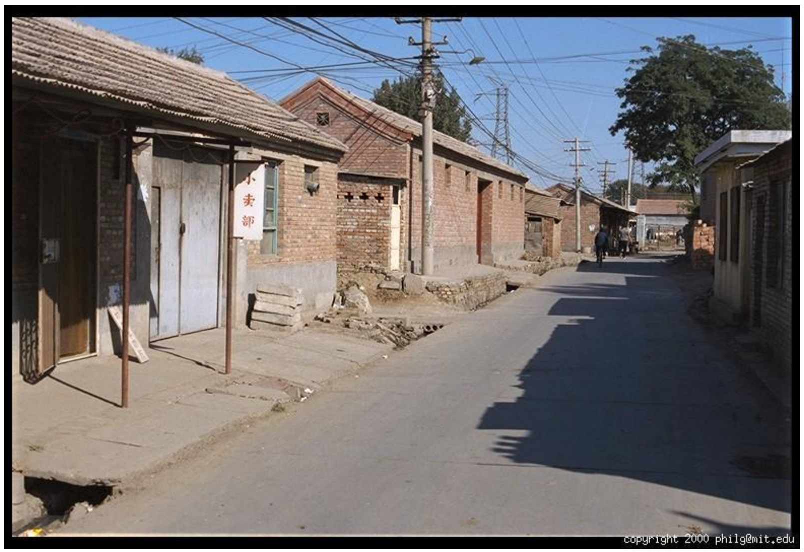
Slide credit: C. Dyer Edward H. Adelson

Perception of Intensity



Slide credit: C. Dyer

Color Image



Slide credit: D. Hoiem

Image Filtering

- Image filtering: compute function of local neighborhood at each pixel position
- One type of “Local operator,” “Neighborhood operator,” “Window operator”

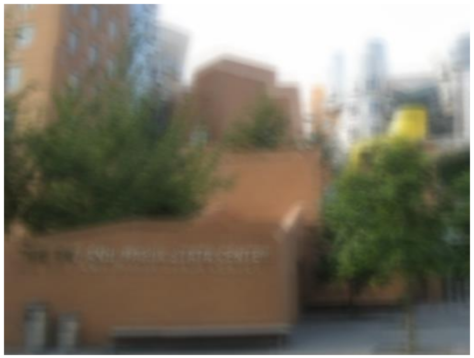
- Useful for:
  - Enhancing images
    - Noise reduction, smooth, resize, increase contrast, etc.
  - Extracting information from images
    - Texture, edges, distinctive points, etc.
  - Detecting patterns
    - Template matching, e.g., eye template

Source: D. Hoiem

Slide credit: C. Dyer

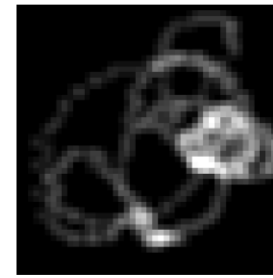
Blurring in the Real World

Camera shake



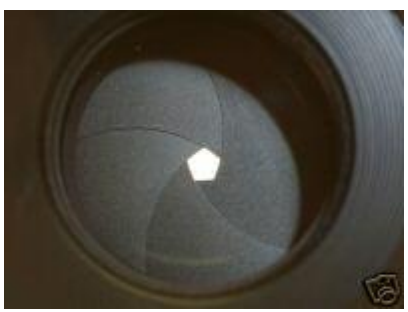
=

*



Source: Fergus, *et al.* "Removing Camera Shake from a Single Photograph", SIGGRAPH 2006

Bokeh: Blur in out-of-focus regions of image



http://lullaby.homepage.dk/diy-camera/bokeh.html

Slide credit: C. Dyer

Image Correlation Filtering

- Select a filter matrix g
  - g is also called a ***filter***, ***mask***, ***kernel***, or ***template***
- Center filter g at each pixel in image f
- Multiply weights by corresponding pixels
- Set resulting value in output image h
- Linear filtering is sum of dot product at each pixel position
- Filtering operation called ***cross-correlation***, and denoted $h = f \otimes g$

Slide credit: C. Dyer

Example: Box Filter

$$g[\cdot\,,\cdot\,]$$

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

Slide credit: David Lowe

Image Filtering

$g[\cdot\,,\cdot\,]\ \frac{1}{9}$

1	1	1
1	1	1
1	1	1

$$f[.,.]$$

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

$$h[m,n] = \sum_{k,l} g[k,l]\ f[m+k, n+l]$$

Credit: S. Seitz

Image Filtering

$$g[\cdot,\cdot]\ \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

$f[.,.]$

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

$h[.,.]$

$$h[m,n]=\sum_{k,l} g[k,l]\ f[m+k,n+l]$$

Credit: S. Seitz

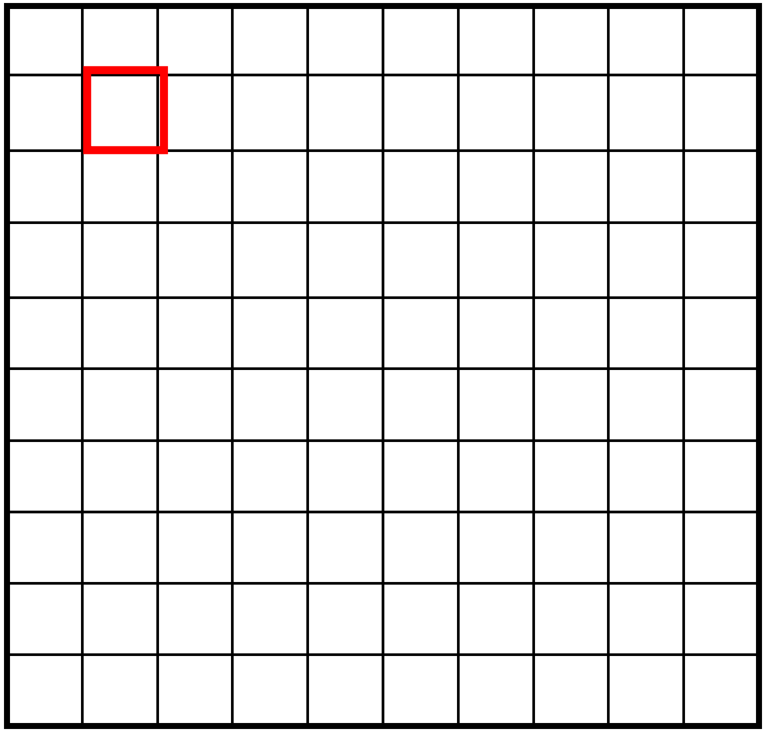
Image Filtering

$$g[\cdot,\cdot]\ \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

$$f[.,.]$$

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

$$h[.,.]$$



$$h[m,n]=\sum_{k,l} g[k,l]\ f[m+k,n+l]$$

Credit: S. Seitz

Image Filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$h[.,.]$

	0	10							

$$h[m,n]=\sum_{k,l} g[k,l]\ f[m+k,n+l]$$

Credit: S. Seitz

Image Filtering

$$g[\cdot,\cdot]\ \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

$f[.,.]$

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

$h[.,.]$

	0	10	20						

$$h[m,n]=\sum_{k,l} g[k,l]\ f[m+k,n+l]$$

Credit: S. Seitz

Image Filtering

$g[\cdot\,,\cdot\,]\ \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

$f[.,.]$

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

$h[.,.]$

	0	10	20	30					

$$h[m,n]=\sum_{k,l} g[k,l]\ f[m+k,n+l]$$

Credit: S. Seitz

Image Filtering

$$g[\cdot,\cdot]\ \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

$f[.,.]$

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

$h[.,.]$

	0	10	20	30	30				

$$h[m,n]=\sum_{k,l} g[k,l]\ f[m+k,n+l]$$

Credit: S. Seitz

Image Filtering

$$g[\cdot,\cdot]\ \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

$$f[.,.]$$

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

$$h[.,.]$$

	0	10	20	30	30				
				?					

$$h[m,n]=\sum_{k,l} g[k,l]\; f[m+k,n+l]$$

Credit: S. Seitz

Image Filtering

$g[\cdot,\cdot]\ \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$

$f[.,.]$

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

$h[.,.]$

	0	10	20	30	30				
						?			
				50					

$$h[m,n]=\sum_{k,l} g[k,l]\ f[m+k,n+l]$$

Credit: S. Seitz

Image Filtering

$$g[\cdot,\cdot]\quad \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

$f[.,.]$

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

$h[.,.]$

	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	

$$h[m,n]=\sum_{k,l} g[k,l]\ f[m+k,n+l]$$

Credit: S. Seitz

Box Filter

What does it do?

- Replaces each pixel with an average of its neighborhood
- Achieves smoothing effect (i.e., removes sharp features)
- Weaknesses:
  - Blocky results
  - Axis-aligned streaks

$$g[\cdot,\cdot]$$

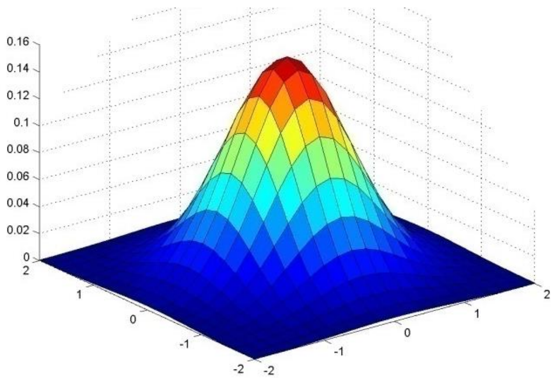
$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

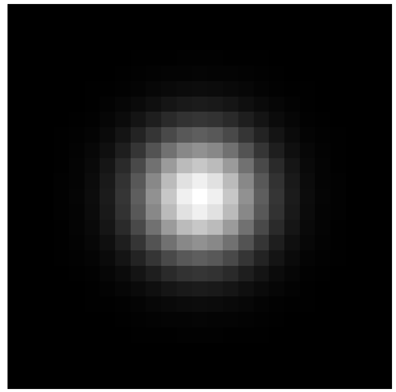
Smoothing with Box Filter

Slide credit: C. Dyer

Gaussian Filtering

- Weight contributions of neighboring pixels by nearness





0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

5 x 5, σ = 1

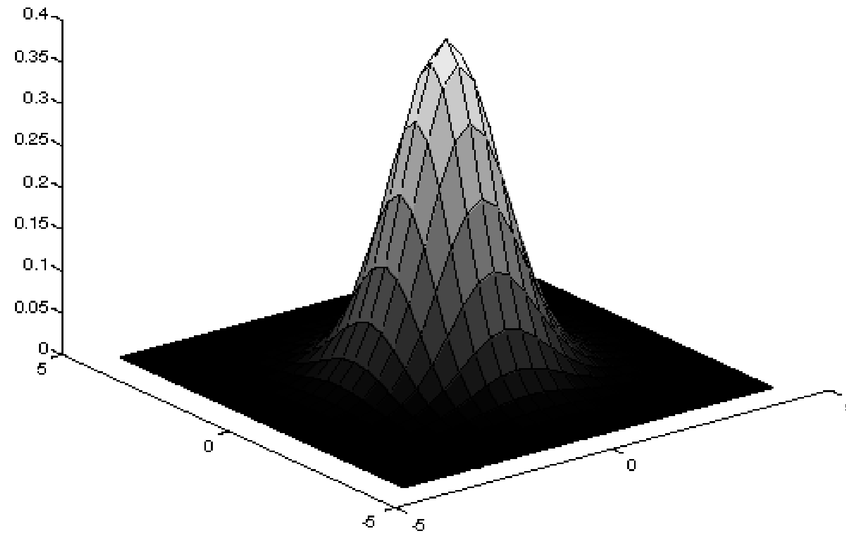
$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

- Constant factor at front makes volume sum to 1
- Convolve each row of image with 1D kernel to produce new image; then convolve each column of new image with same 1D kernel to yield output image

Slide credit: Christopher Rasmussen

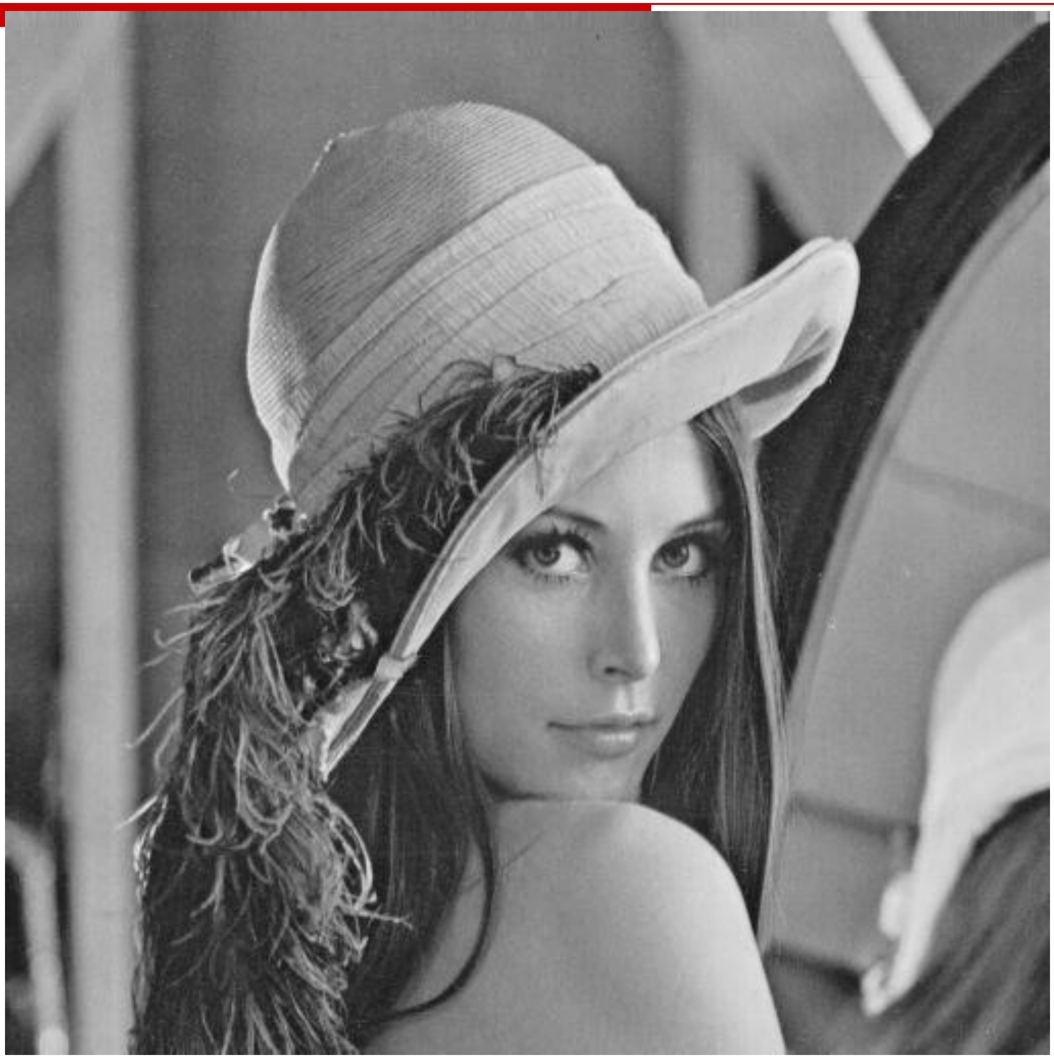
Smoothing with a Gaussian

- Smoothing with a box actually doesn't compare at all well with a defocused lens
- Most obvious difference is that a single point of light viewed in a defocused lens looks like a fuzzy blob; but the averaging process would give a little square
- Gaussian is *isotropic* (i.e., rotationally symmetric)



- A Gaussian gives a good model of a fuzzy blob
- It closely models many physical processes (the sum of many small effects)

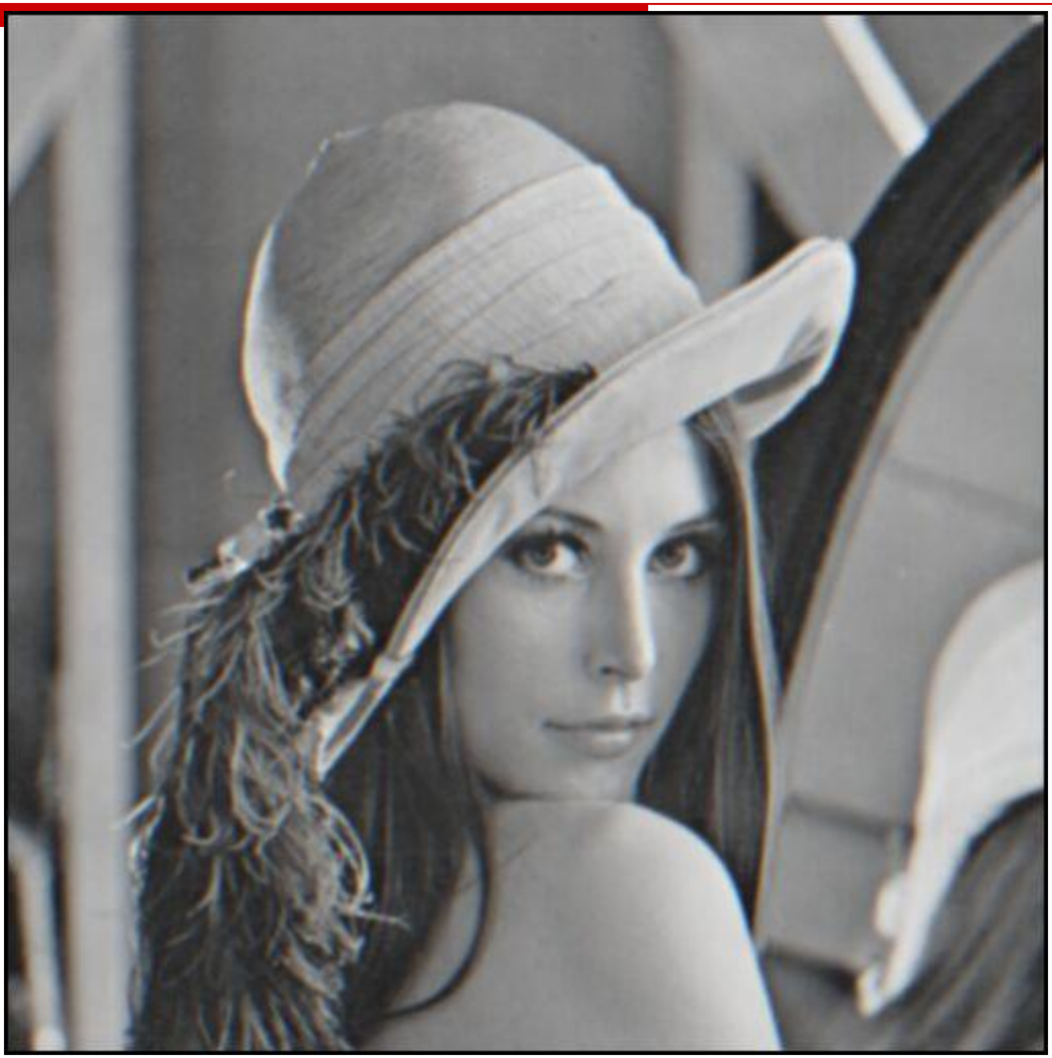
What does Blurring take away?



original

Slide credit: C. Dyer

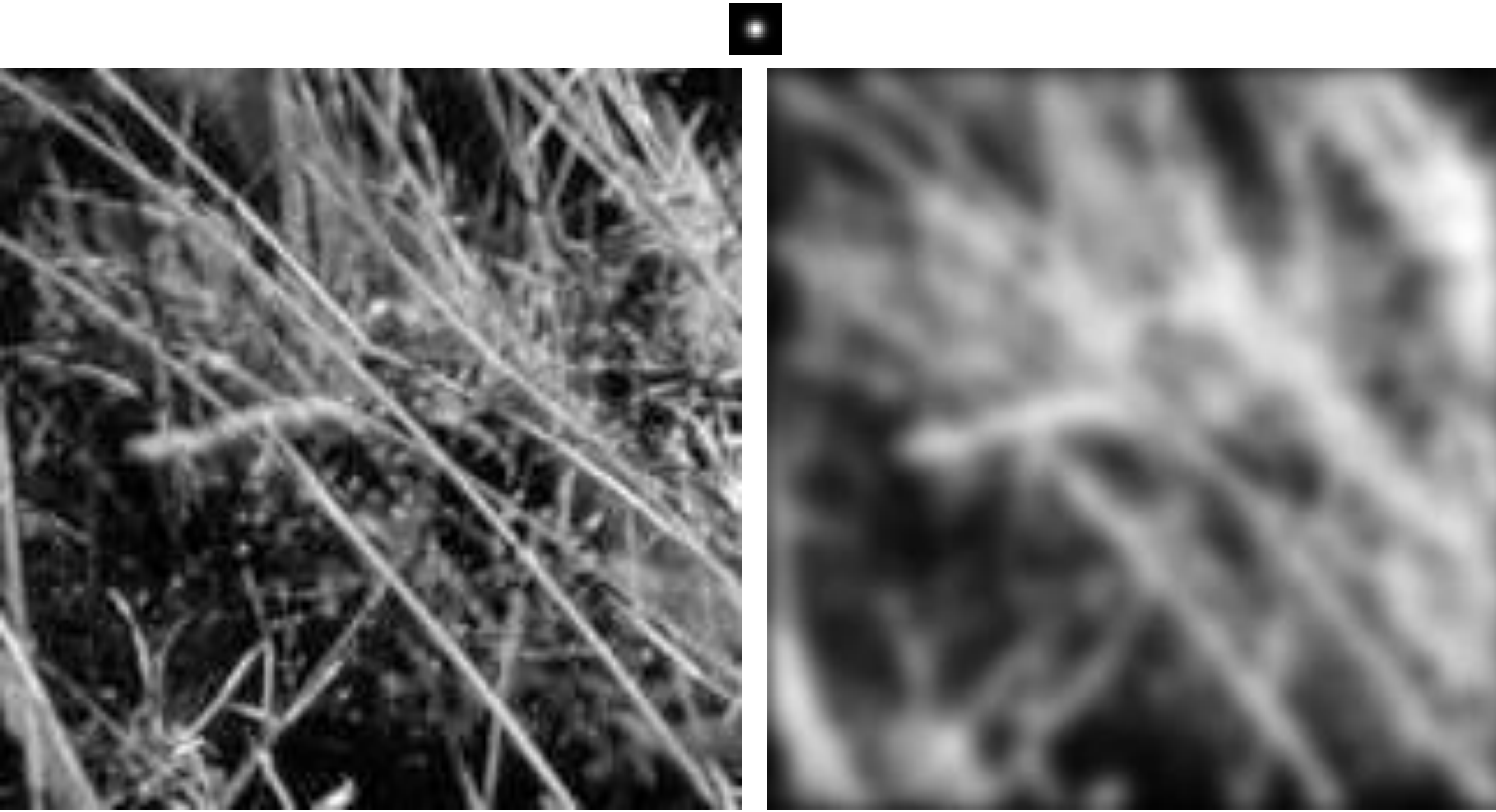
What does Blurring take away?



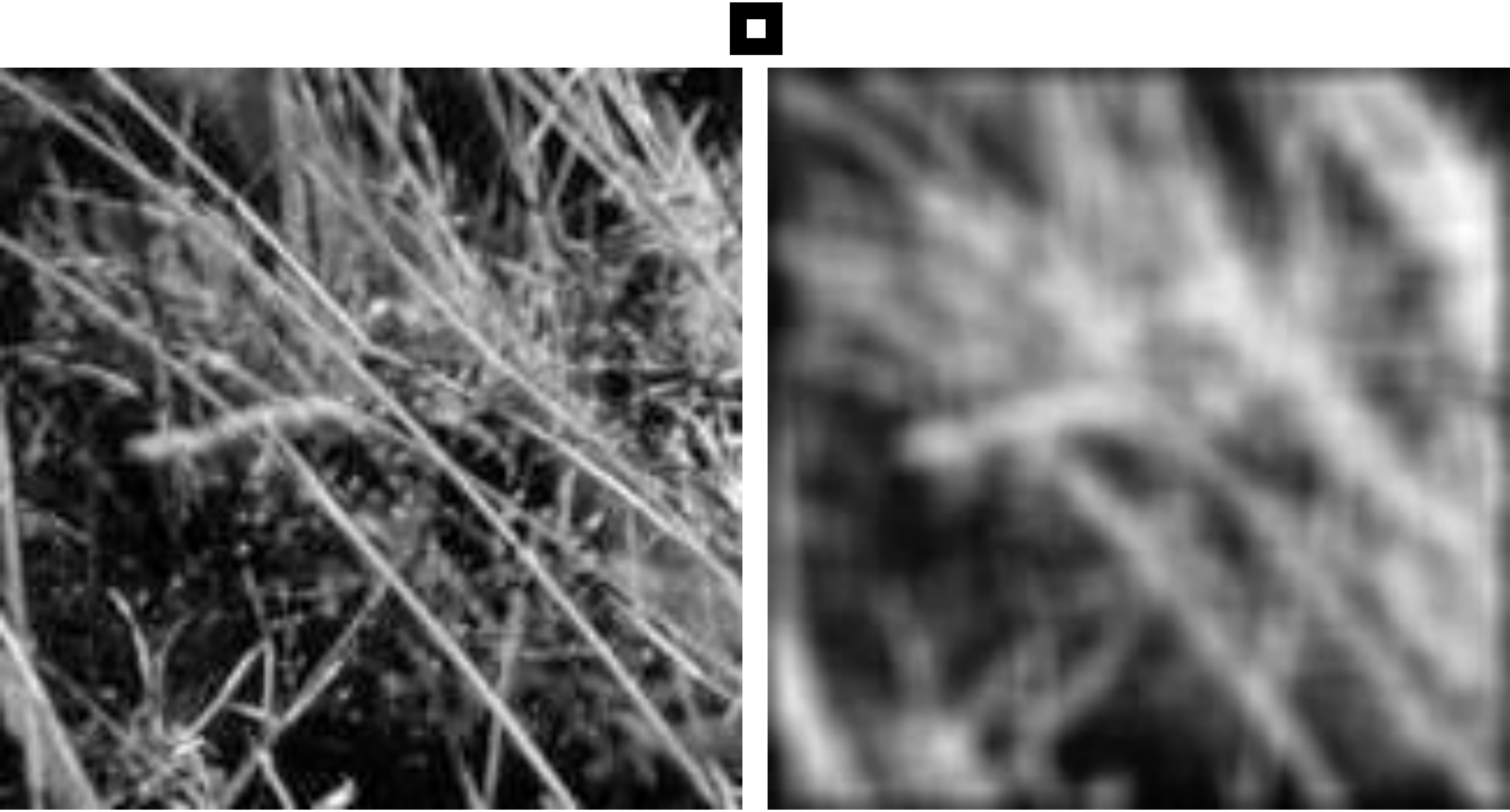
smoothed (5x5 Gaussian)

Slide credit: C. Dyer

Smoothing with Gaussian Filter



Smoothing with Box Filter




input
Slide by S. Paris

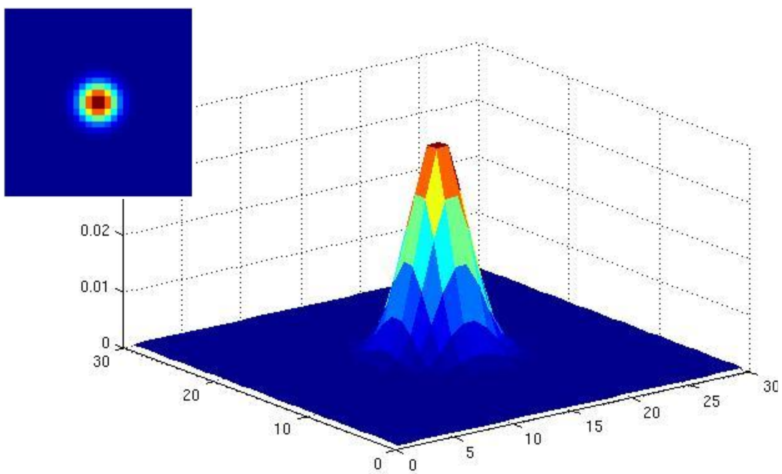
Slide by S. Paris

Gaussian blur

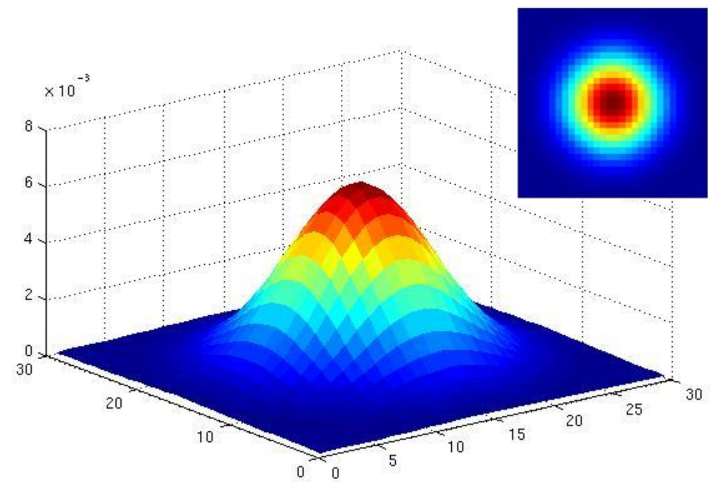
Slide by S. Paris

Gaussian Filters

- What parameters matter here?
- **Standard deviation**, σ, of Gaussian: determines extent of smoothing

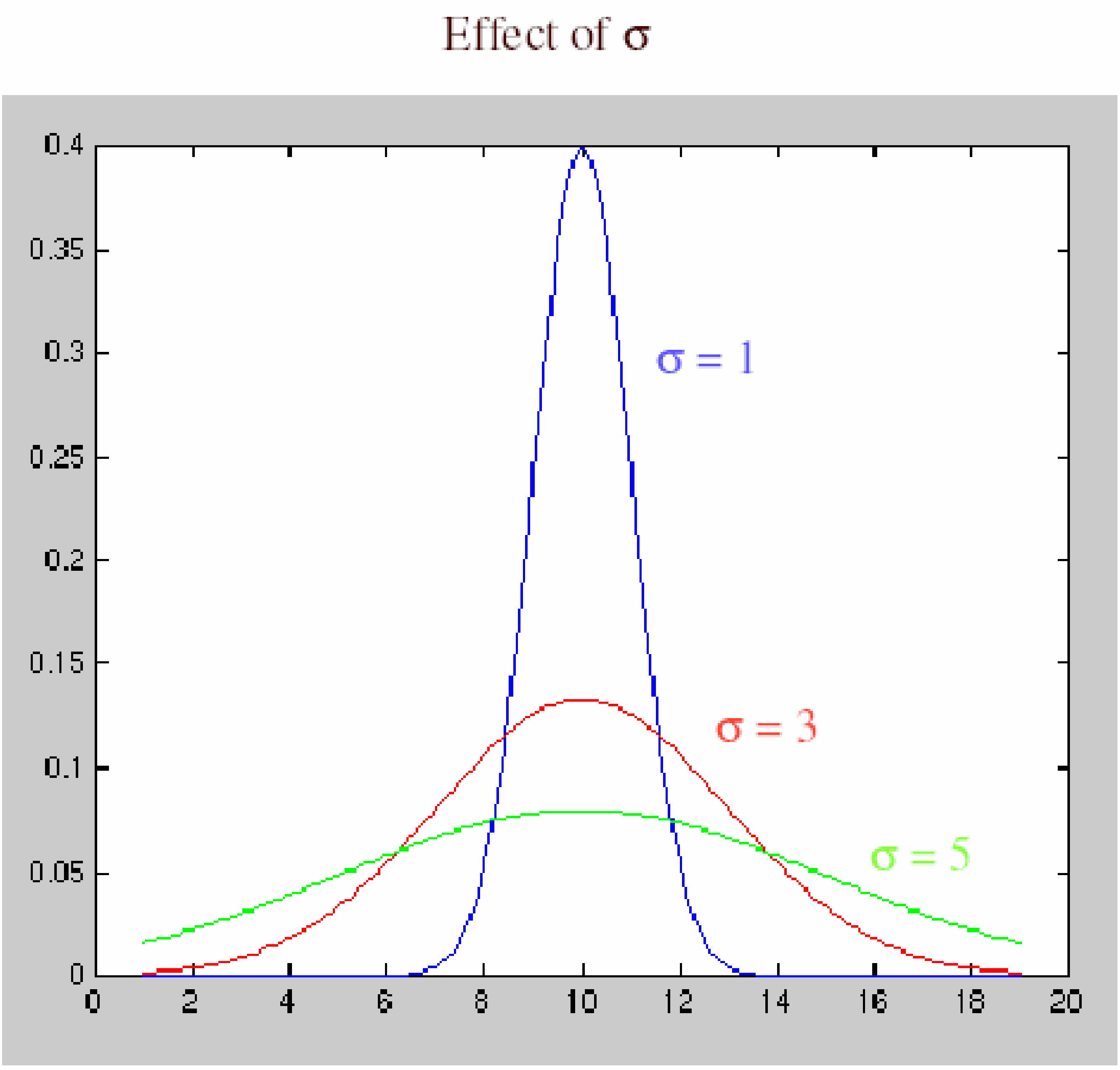


σ = 2 with
30 x 30 kernel



σ = 5 with
30 x 30 kernel

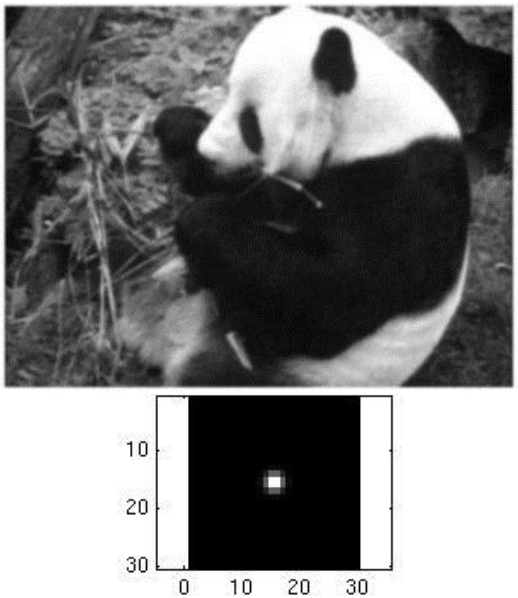
Source: D. Hoiem

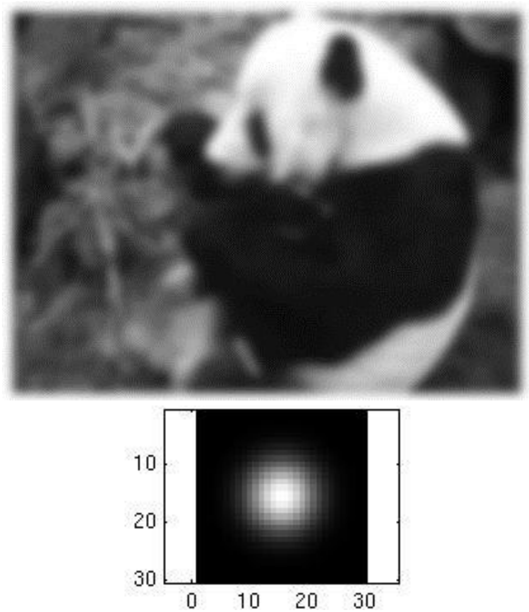


Slide credit: C. Dyer

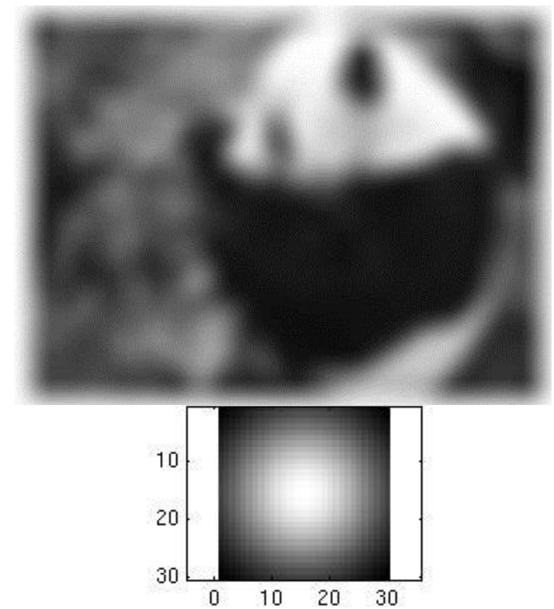
Smoothing with a Gaussian

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





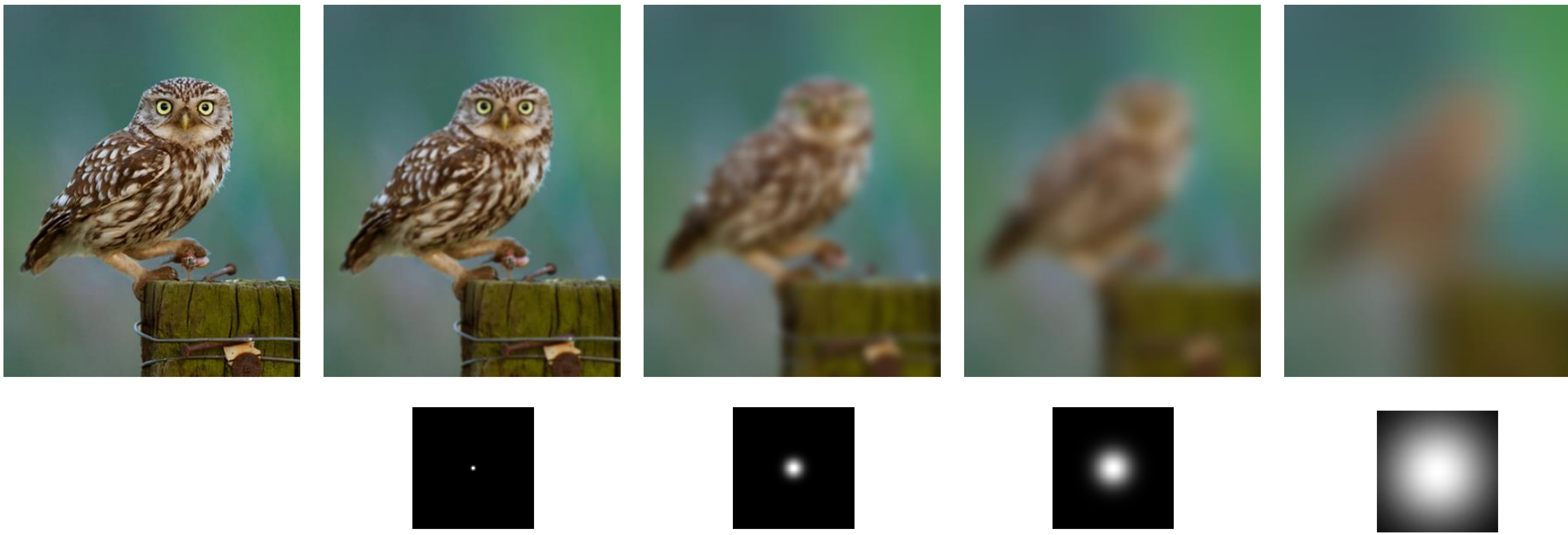
…




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

Slide credit: C. Dyer

Gaussian filters

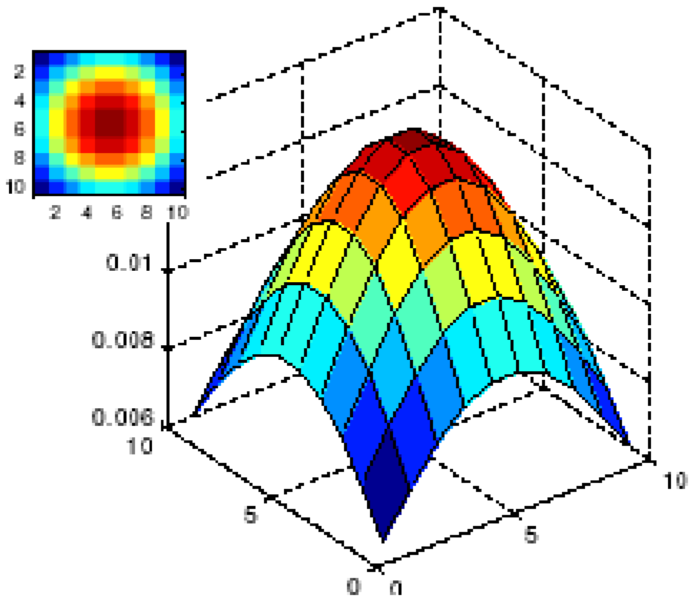


σ = 1 pixel σ = 5 pixels σ = 10 pixels σ = 30 pixels

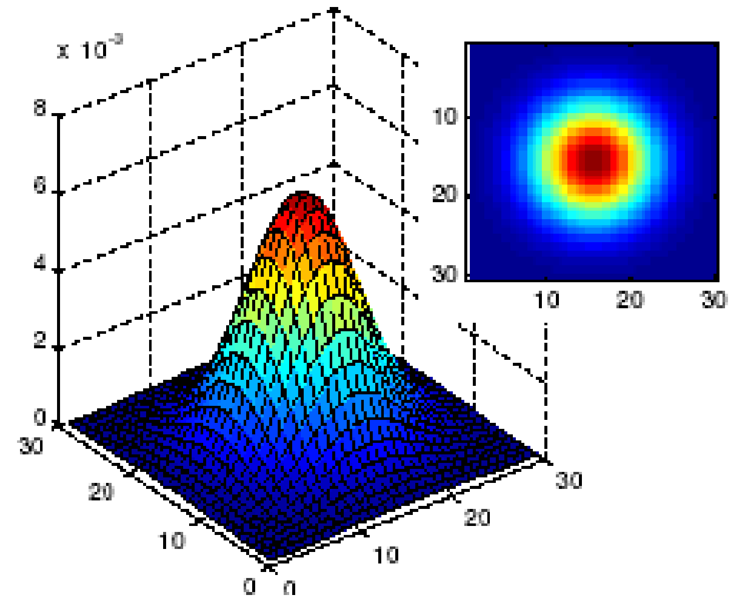
Slide credit: C. Dyer

Gaussian Filters

- What parameters matter here?
- **Size** of kernel or mask



σ = 5 with 10 x 10 kernel

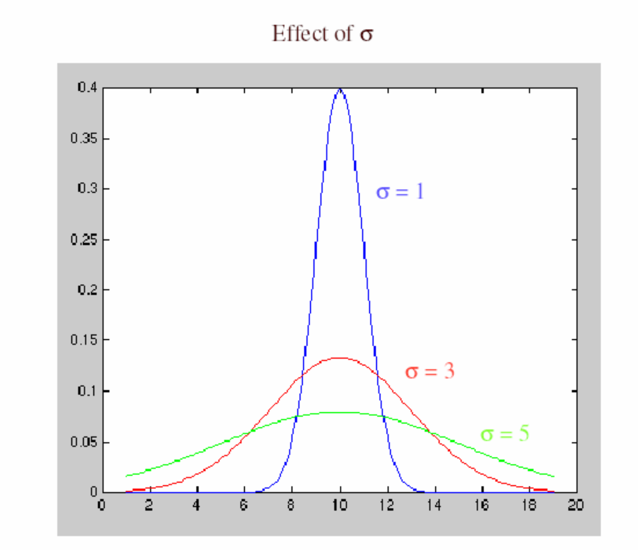


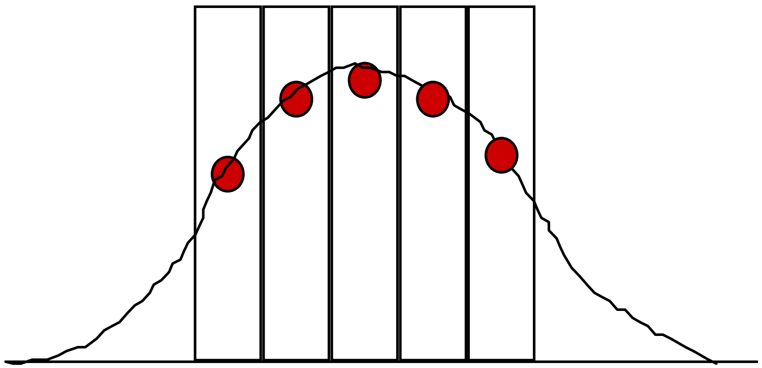
σ = 5 with 30 x 30 kernel

Slide credit: C. Dyer

How big should the filter be?

- Gaussian function has infinite “support” but need a finite-size kernel
- Values at edges should be near 0
- ~98.8% of area under Gaussian in mask of size 5σ x 5σ
- In practice, use mask of size $2k+1$ x $2k+1$ where $k \approx 3\sigma$
- Normalize output by dividing by sum of all weights





Slide credit: C. Dyer

Properties of Smoothing Filters

- Smoothing
  - Values all positive
  - Sum to 1 $\Rightarrow$ constant regions same as input
  - Amount of smoothing proportional to mask size
  - Removes “high-frequency” components
  - “low-pass” filter

Slide credit: C. Dyer

Sharpening Filters

Original

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

?

(Note that filter sums to 1)

Source: D. Lowe

Sharpening Filters

Original

$$\begin{bmatrix}0&0&0\\0&1&0\\0&0&0\end{bmatrix} + \begin{bmatrix}0&0&0\\0&1&0\\0&0&0\end{bmatrix} - \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix} \quad ?$$

(Note that filter sums to 1)

Source: D. Lowe

Sharpening Filters

Original

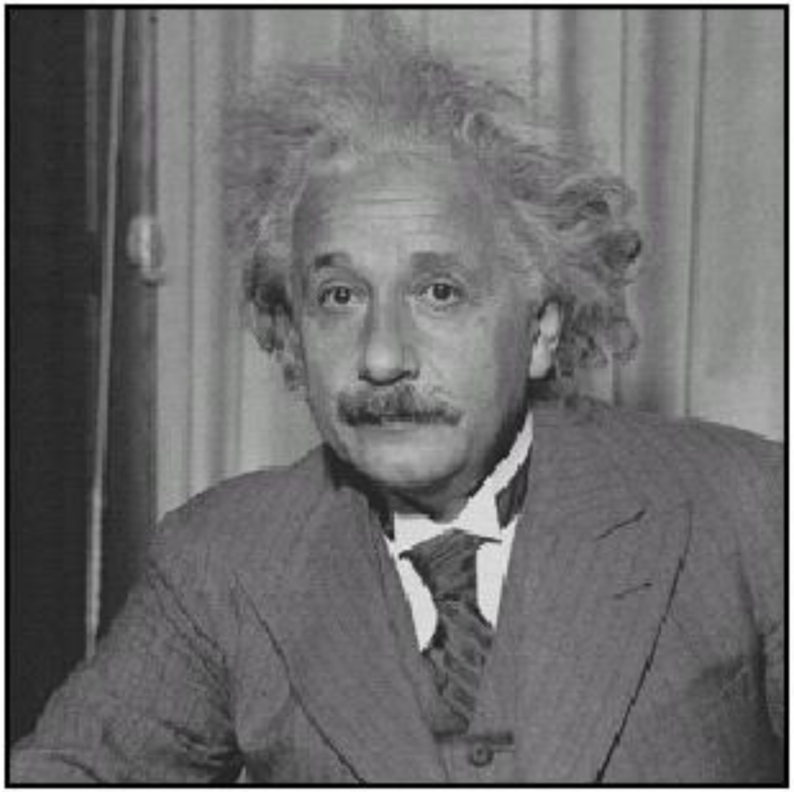
$$\begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 2 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array} - \frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

Sharpening filter

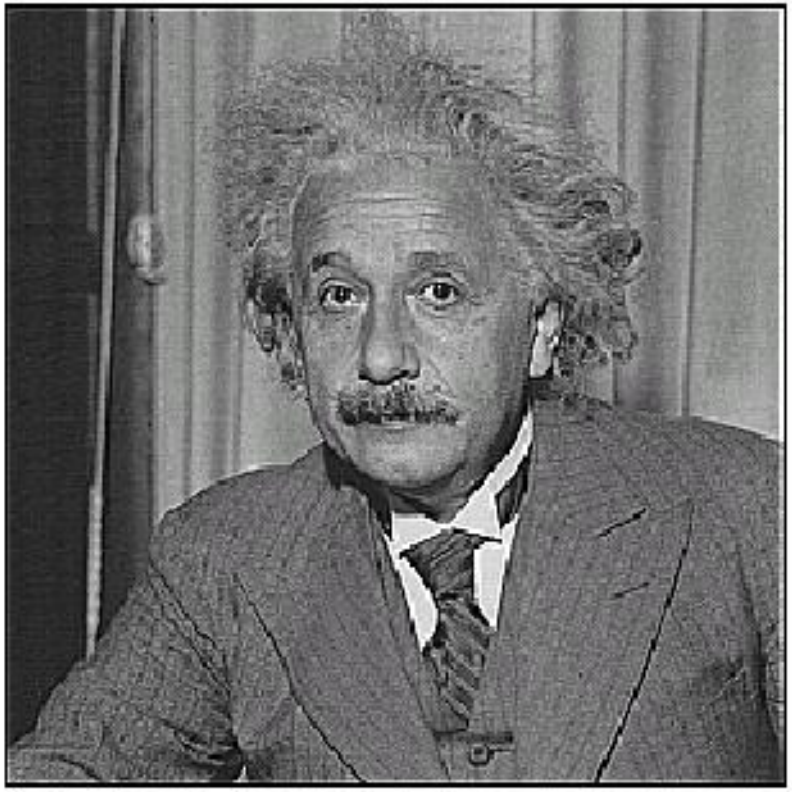
- Sharpen an out of focus image by subtracting a multiple of a blurred version

Source: D. Lowe

Sharpening



before



after

Source: D. Lowe

Sharpening by Unsharp Masking

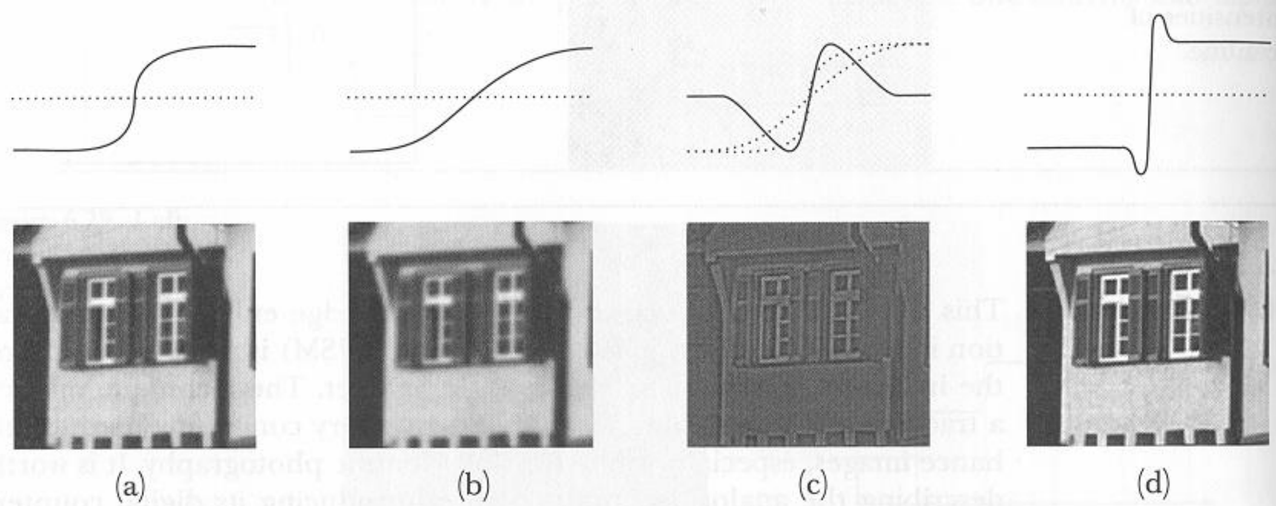
- $h = f + k(f * g)$ where k is a small positive constant and g =

0	1	0
1	-4	1
0	1	0

called a Laplacian mask

- Called *unsharp masking* in photography

Figure 6.32. (a) Original image. (b) Blurred image. (c) Difference between first two. (d) Enhanced image.



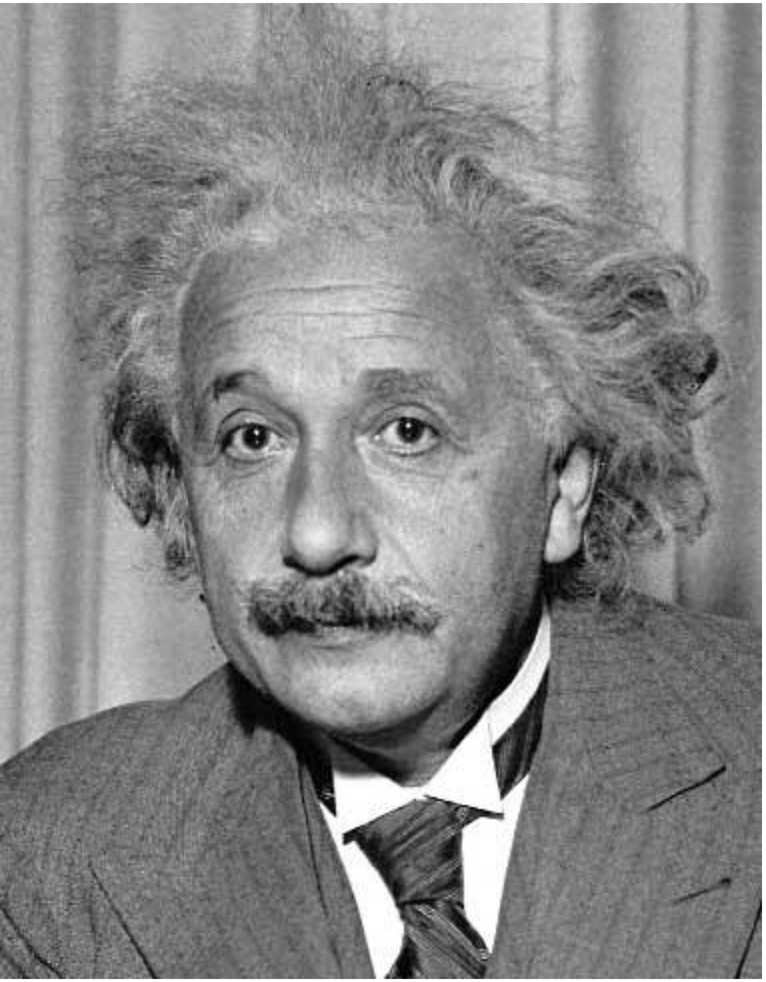
Sharpening using Unsharp Mask Filter

Original

Filtered result

Slide credit: C. Dyer

Application: Edge Detection

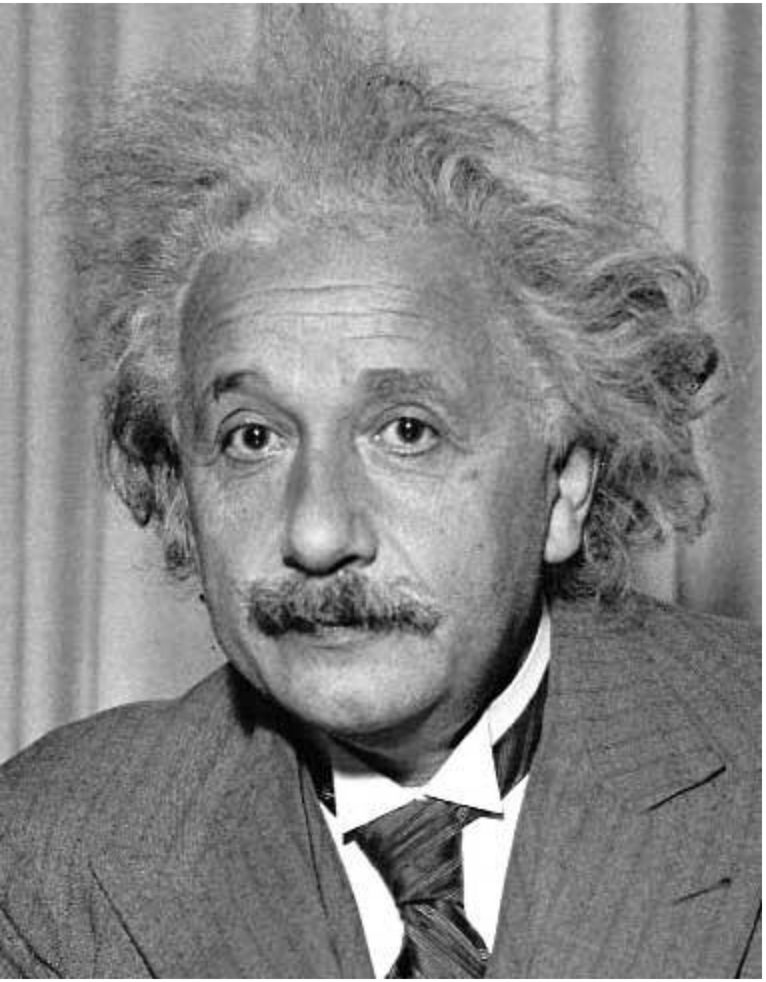


1	0	-1
2	0	-2
1	0	-1

Sobel

Slide credit: C. Dyer

Application: Edge Detection



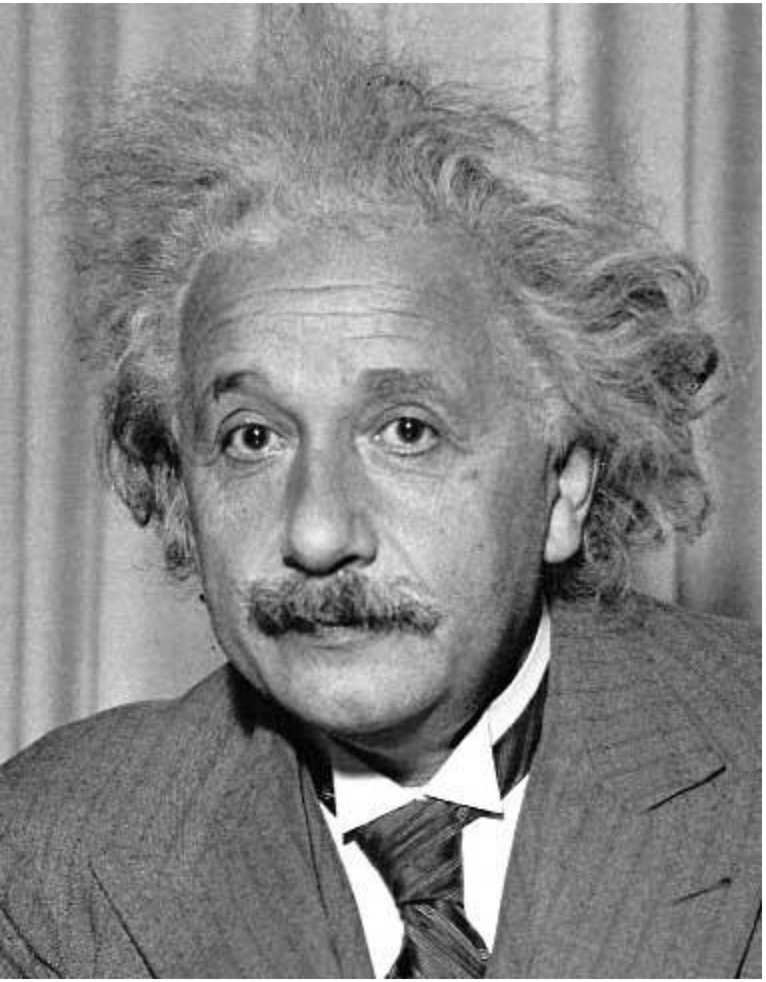
1	0	-1
2	0	-2
1	0	-1

Sobel

Vertical Edge
(absolute value)

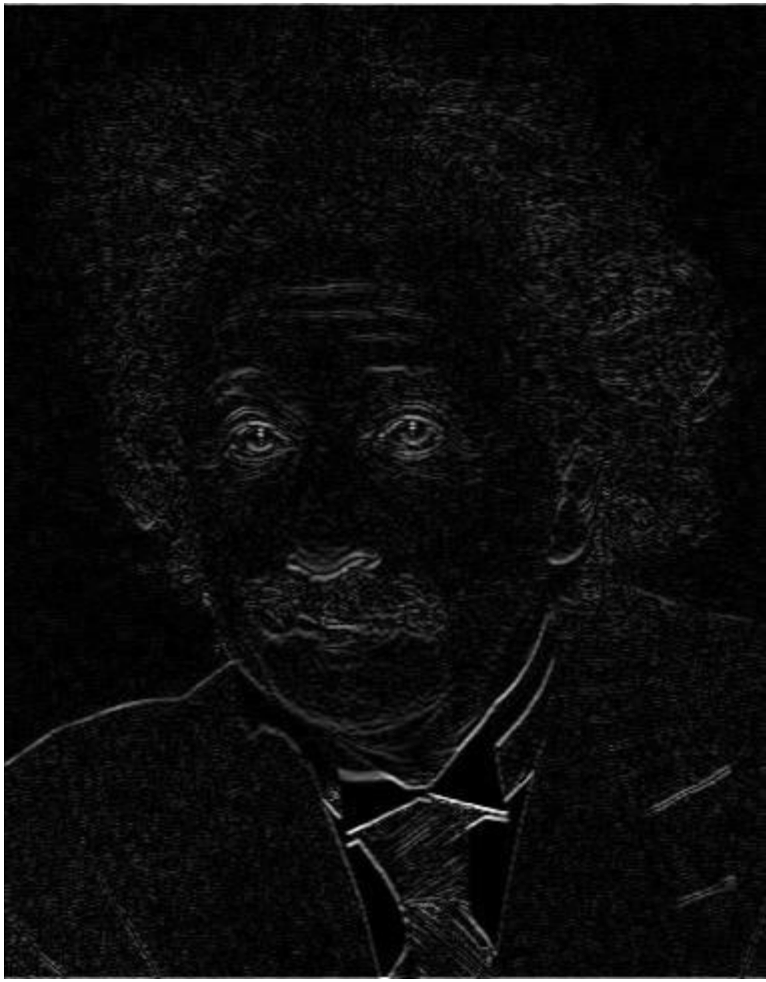
Slide credit: C. Dyer

Application: Edge Detection



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

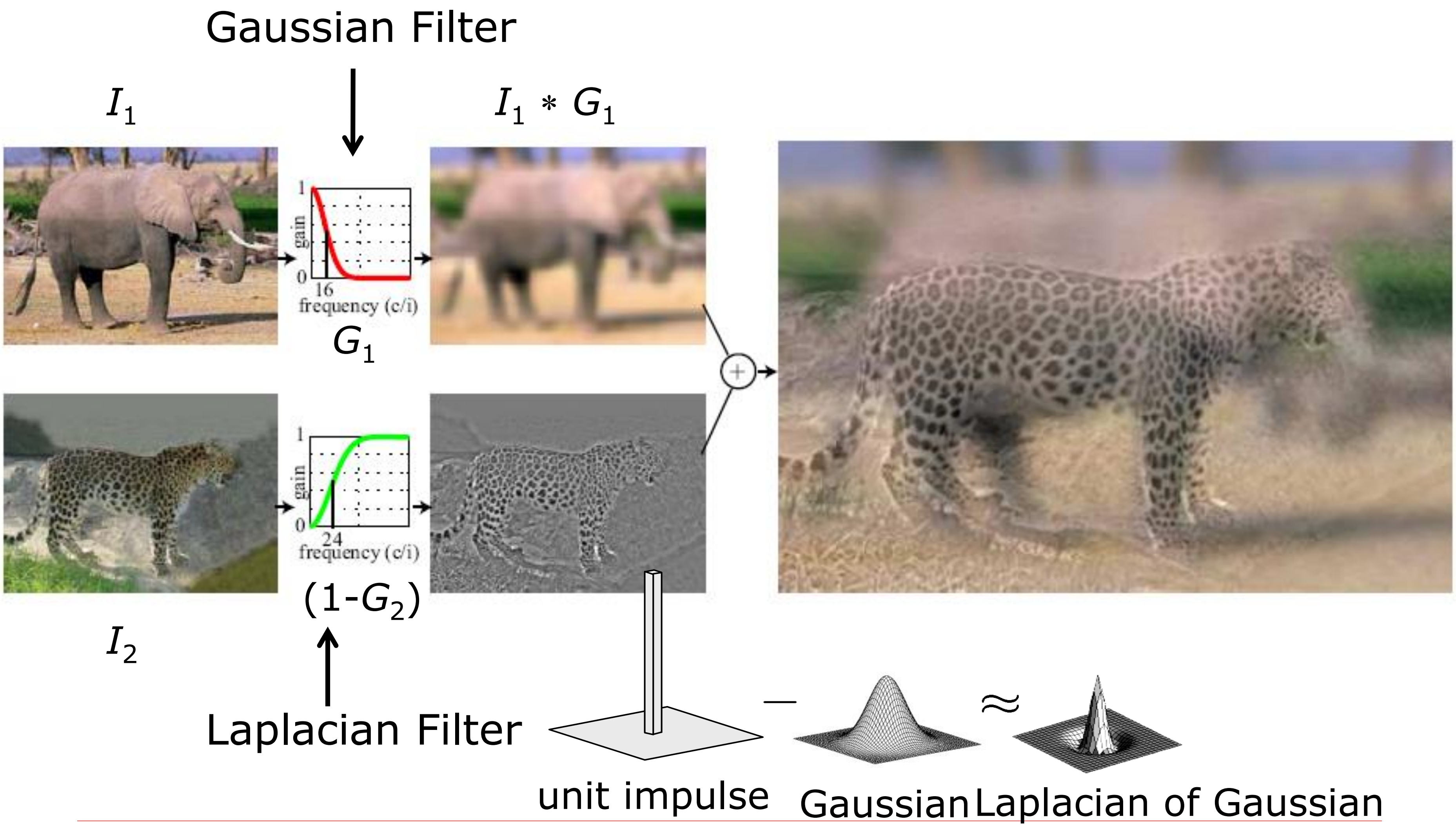
Sobel



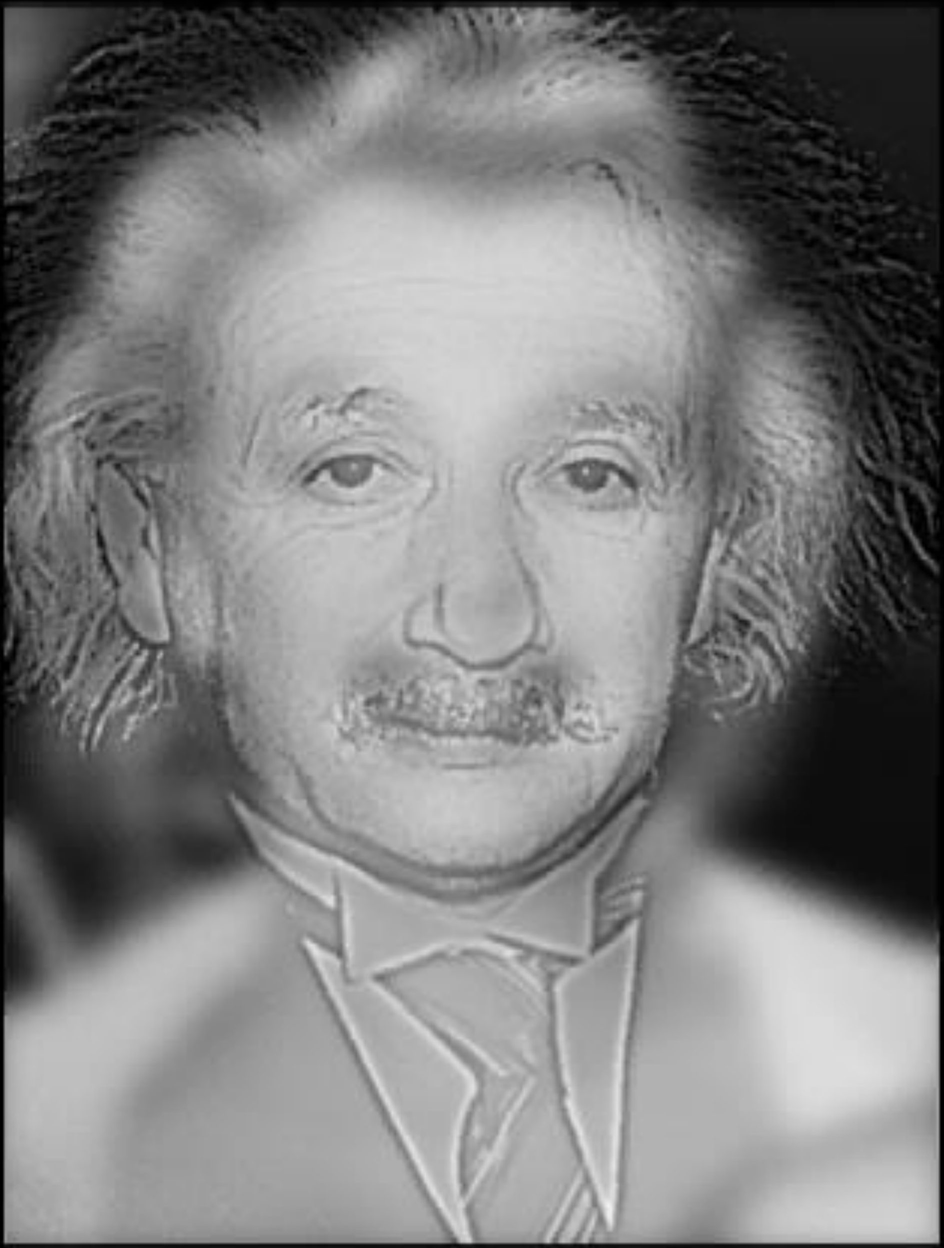
Horizontal Edge
(absolute value)

Slide credit: C. Dyer

Application: Hybrid Images



A. Oliva, A. Torralba, P.G. Schyns, *Hybrid Images*, SIGGRAPH 2006



Application: XDoG Filters

Input

XDoG

Input

XDoG

Gaussian filtering results

$$
\begin{aligned}
D_X(\sigma, k, \tau) &= G(\sigma) - \tau \cdot G(k \cdot \sigma) \\
E_X(\sigma, k, \tau, \epsilon, \varphi) &= \begin{cases} 1, & \text{if } D_X(\sigma, k, \tau) < \epsilon \\ 1 + \tanh(\varphi \cdot (D_X(\sigma, k, \tau)), & \\ \quad \text{otherwise.} & \end{cases}
\end{aligned}
$$

Winnemoller, H., *XDoG: advanced image stylization with eXtended Difference-of-Gaussians* . NPAR 2011

Application: Painterly Filters

- Many methods have been proposed to make a photo look like a painting
- Today we look at one: *Painterly-Rendering with Brushes of Multiple Sizes*
- Basic ideas:
  - Build painting one layer at a time, from biggest to smallest brushes
  - At each layer, add detail missing from previous layer

A. Hertzmann, *Painterly rendering with curved brush strokes of multiple sizes*, SIGGRAPH 1998.

Slide credit: S. Chenney

Algorithm 1

```
function paint(sourceImage,R_1 ... R_n) // take source and several brush sizes
{
        canvas := a new constant color image
        // paint the canvas with decreasing sized brushes
        for each brush radius R_i, from largest to smallest do
        {
        // Apply Gaussian smoothing with a filter of size const * radius
        // Brush is intended to catch features at this scale
                referenceImage = sourceImage * G(fs R_i)
                // Paint a layer
                paintLayer(canvas, referenceImage, Ri)
        }
        return canvas
}
```

Slide credit: S. Chenney

Algorithm 2

procedure paintLayer(canvas,referenceImage, R) // *Add a layer of strokes*
{
    S := a new set of strokes, initially empty
    D := difference(canvas,referenceImage) // *euclidean distance at every pixel*
        for x=0 **to** imageWidth **stepsize** grid **do** // *step in size that depends on brush radius*
            for y=0 **to** imageHeight **stepsize** grid **do** {
            // sum the error near (x,y)
            M := the region *(x-grid/2..x+grid/2, y-grid/2..y+grid/2)*
                areaError := sum($D_{i,j}$ for i,j in M) / grid^2
                if (areaError > *T*) **then** {
                    // find the largest error point
                    (x1,y1) := max $D_{i,j}$ in M
                    s :=makeStroke(R,x1,y1,referenceImage)
                    add s to S
                }
            }
        paint all strokes in S on the canvas, in random order
}

Results


Original


Biggest brush


Medium brush added


Finest brush added

Slide credit: S. Chenney

Next Time

- More Filters
- De-noise
- Student paper presentation
  - Accelerating Spatially Varying Gaussian Filters. Baek, J., Jacobs, D. E., SIGGRAPH Asia 2010
  - By Dave Howell