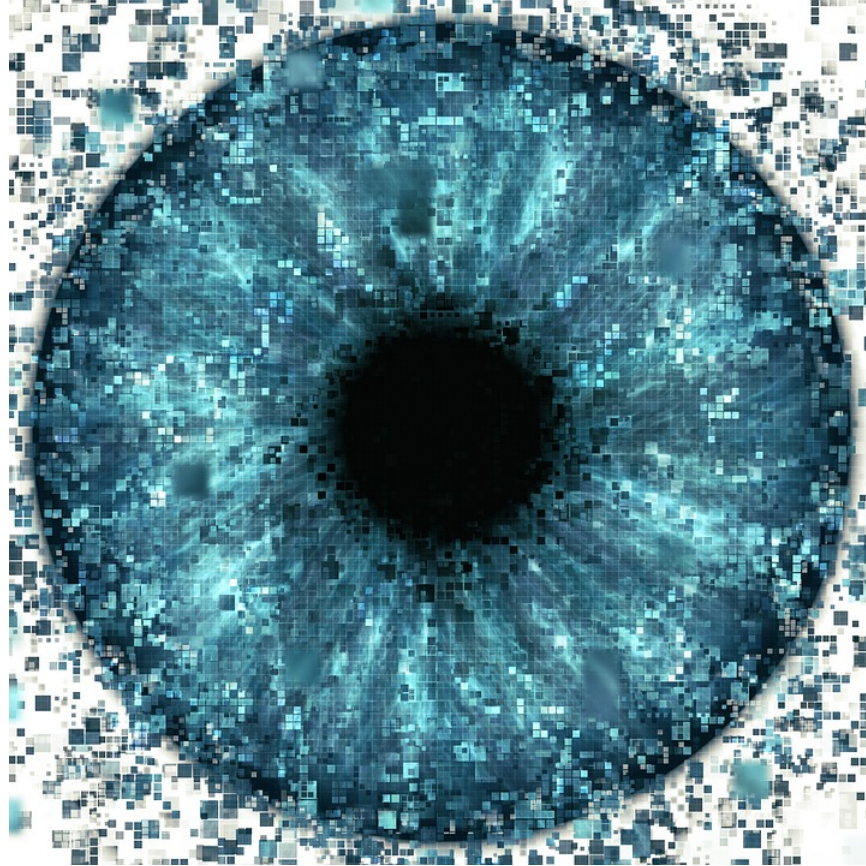


Pixels and Image Filtering



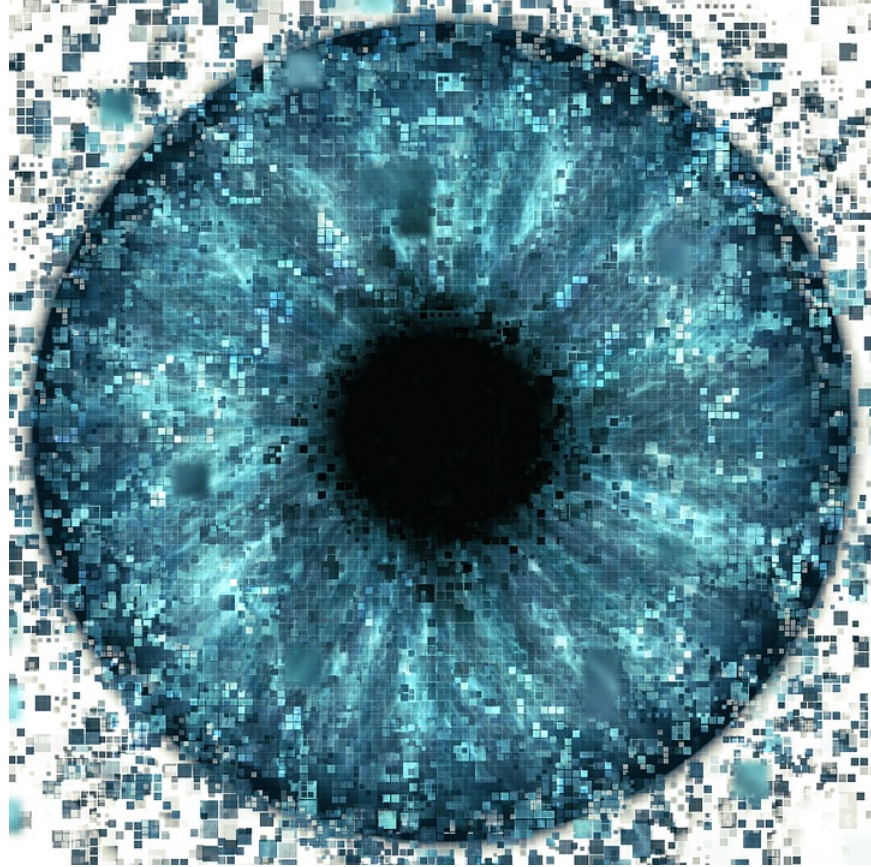
Computational Photography

Derek Hoiem

Administrative stuff

- Any questions?
- Next week reminders
 - Labor day on Monday
 - Derek is out Wed. Jae will teach. Recording from earlier year available.
 - Yuan will give tutorial Thurs at 5pm in DCL 1320

Pixels and Image Filtering



Computational Photography

Derek Hoiem

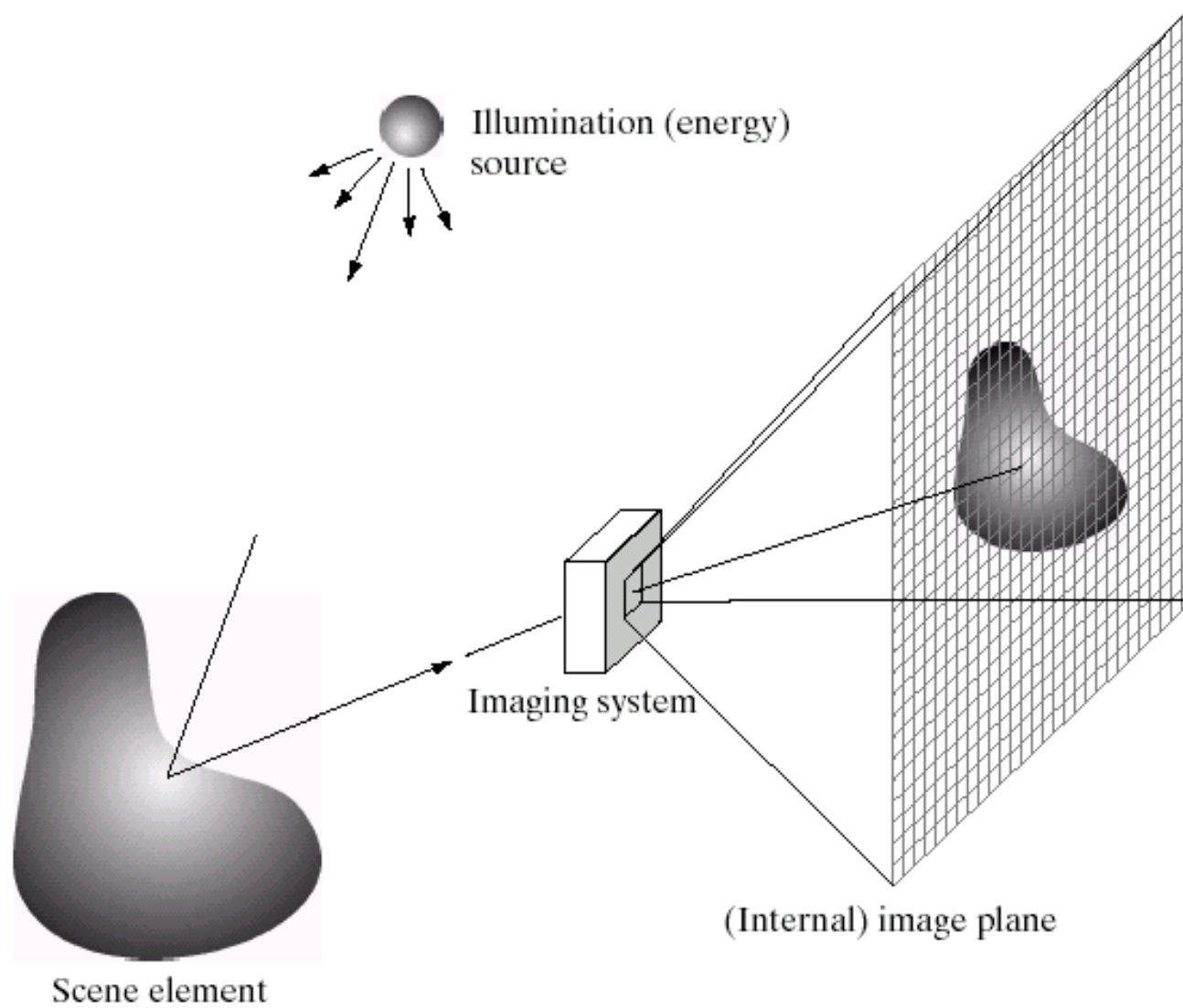
Today's Class: Pixels and Linear Filters

- What is a pixel? How is an image represented?
- What is image filtering and how do we do it?
- Introduce Project 1: Hybrid Images

Next three classes

- Image filters in spatial domain
 - Smoothing, sharpening, measuring texture
- Image filters in the frequency domain
 - Denoising, sampling, image compression
- Templates and Image Pyramids
 - Detection, coarse-to-fine registration

Image Formation



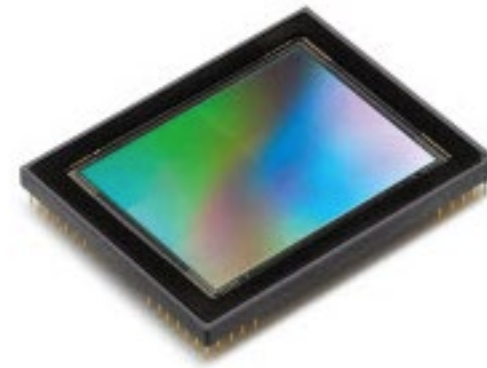
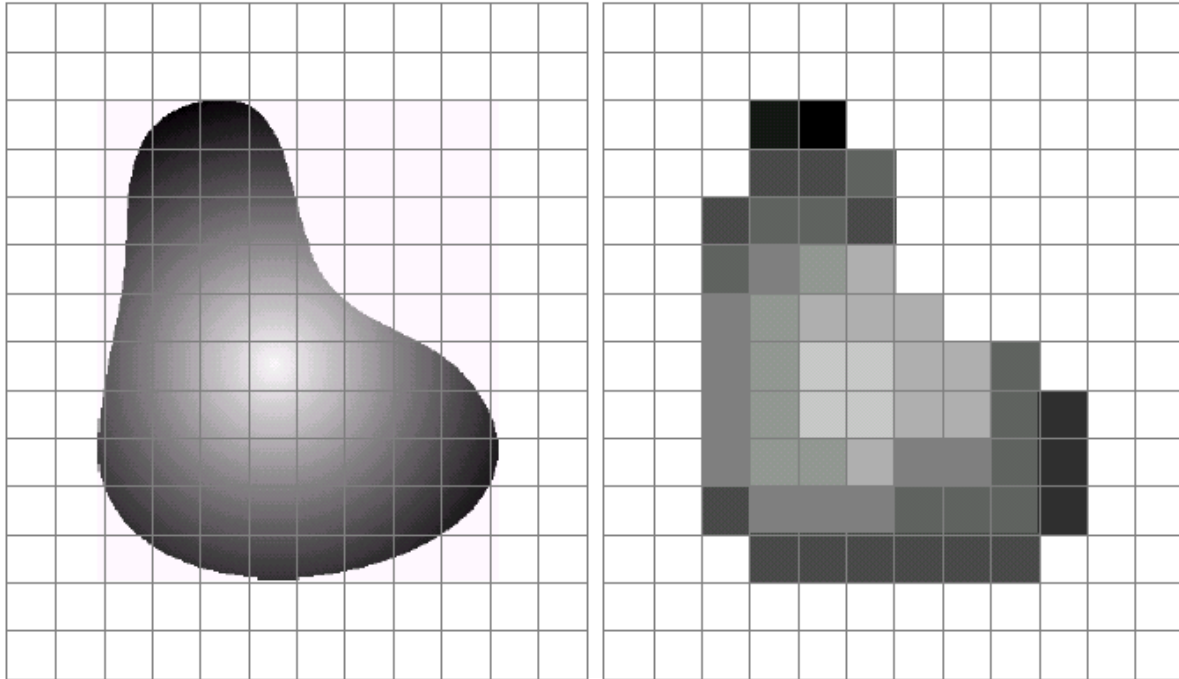
Digital camera



Digital camera replaces film with a sensor array

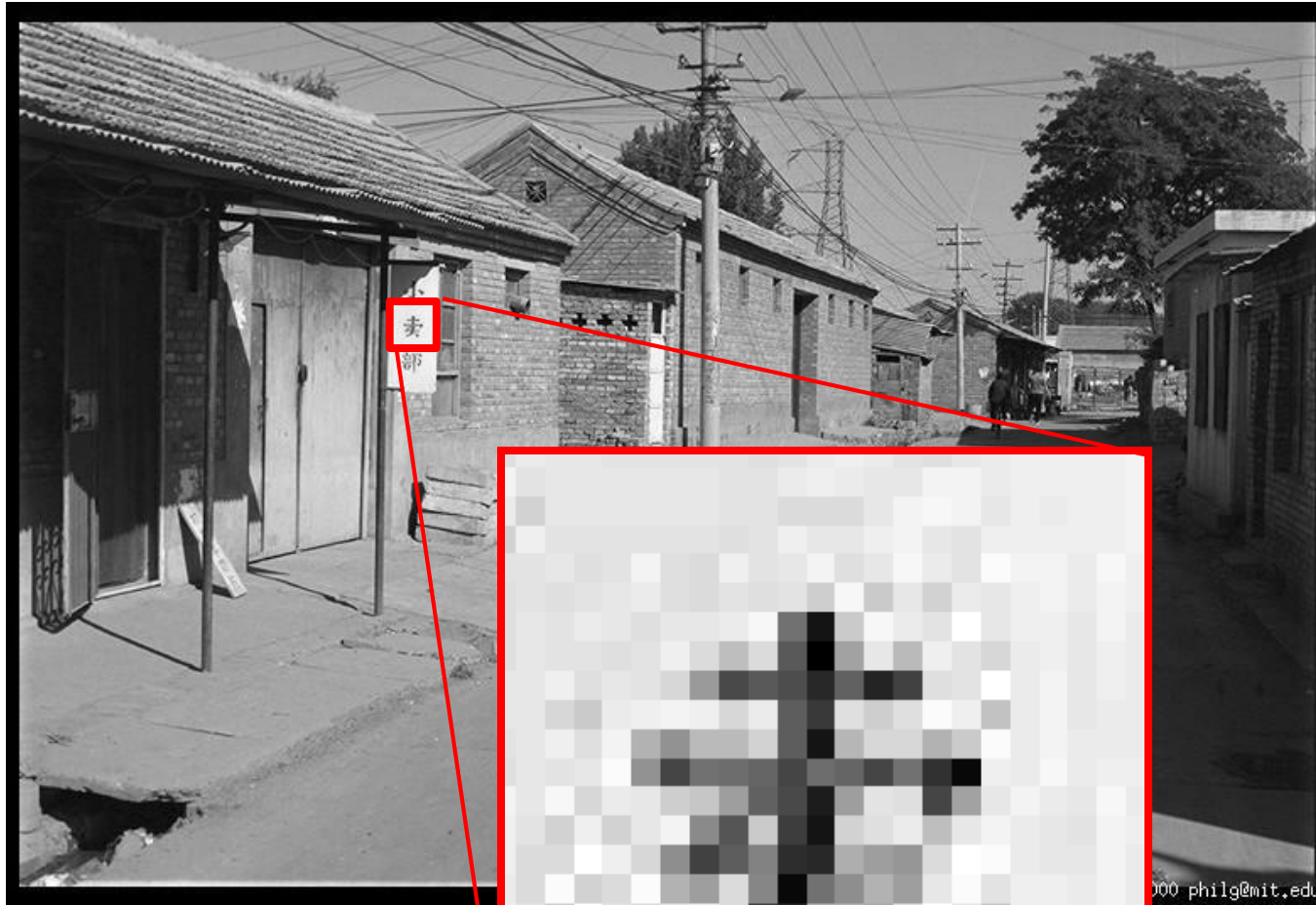
- Each cell in the array is light-sensitive diode that converts photons to electrons
- <http://electronics.howstuffworks.com/digital-camera.htm>

Sensor Array

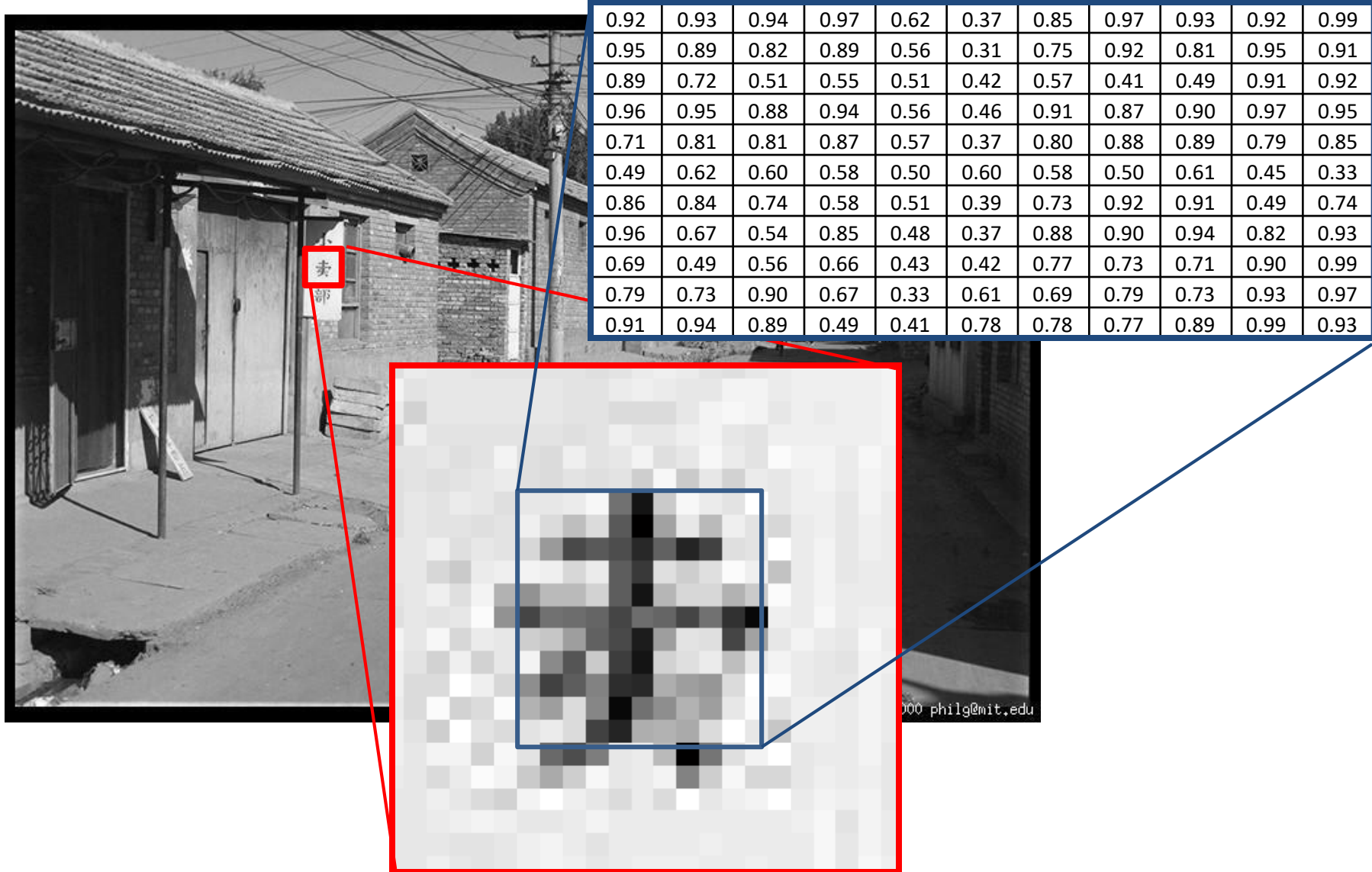


CCD sensor

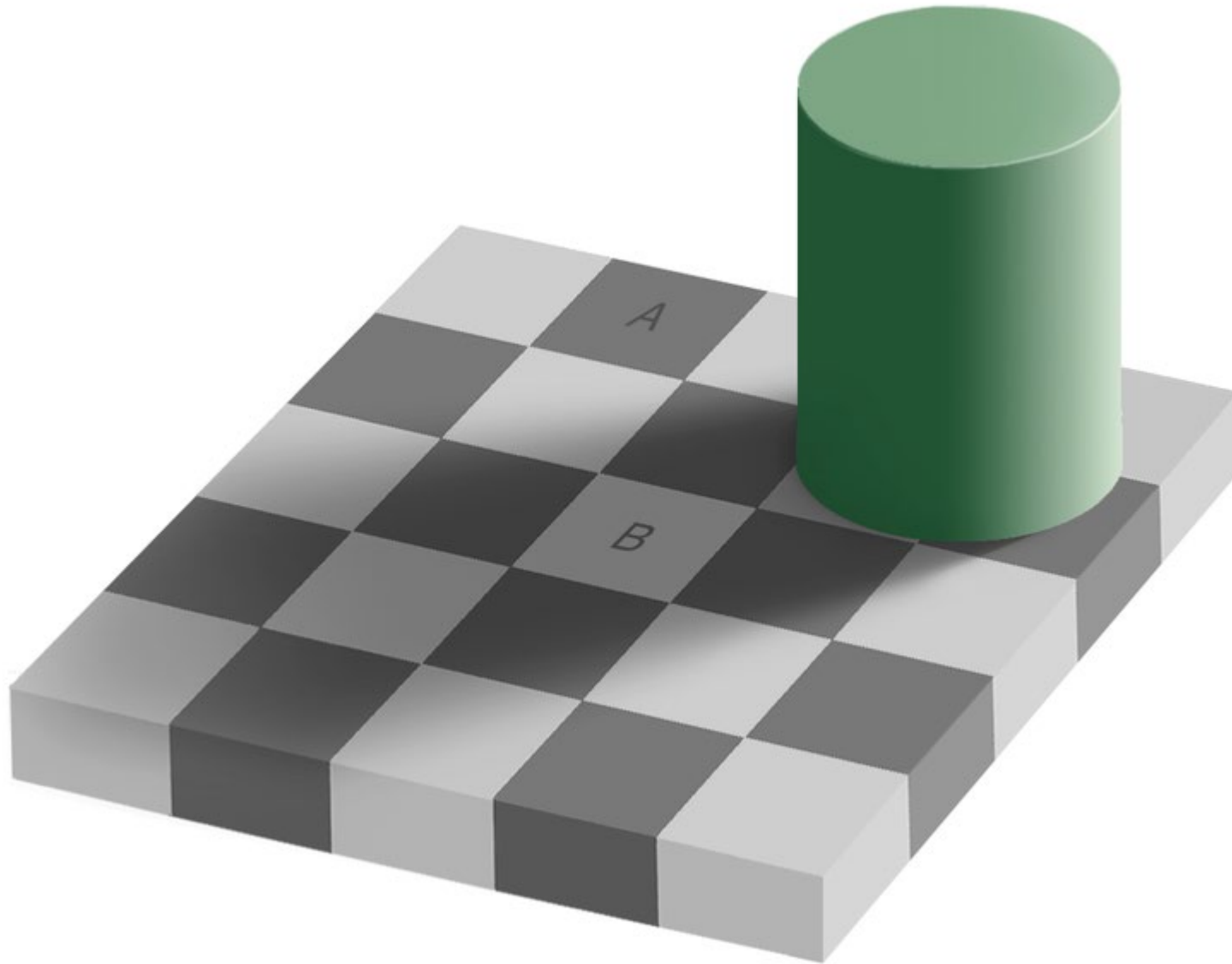
The raster image (pixel matrix)



The raster image (pixel matrix)

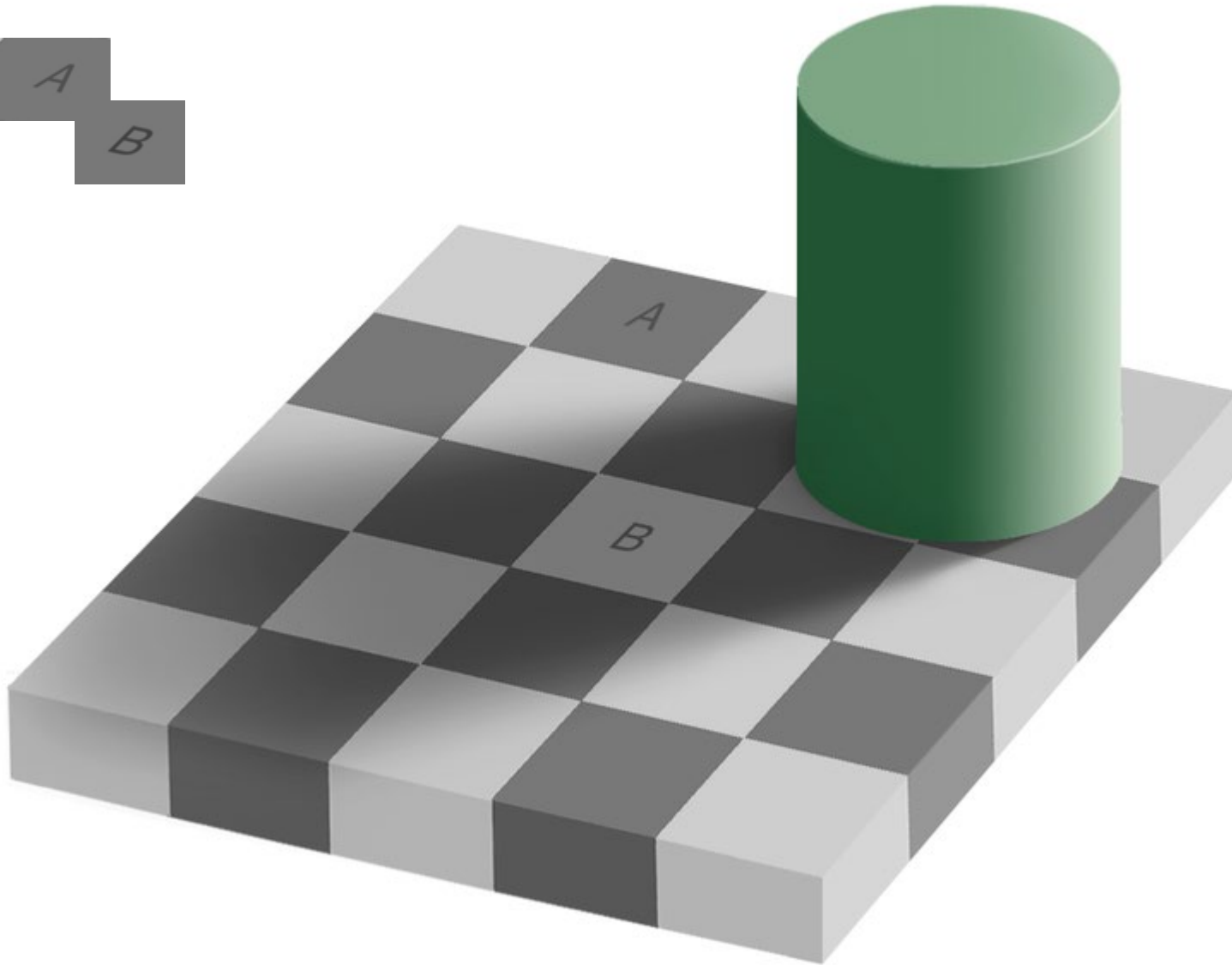


Perception of Intensity



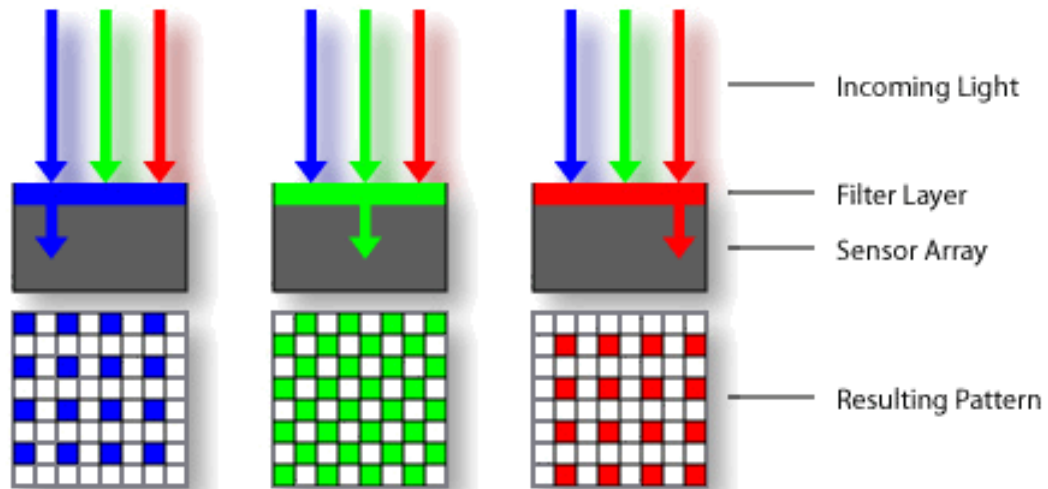
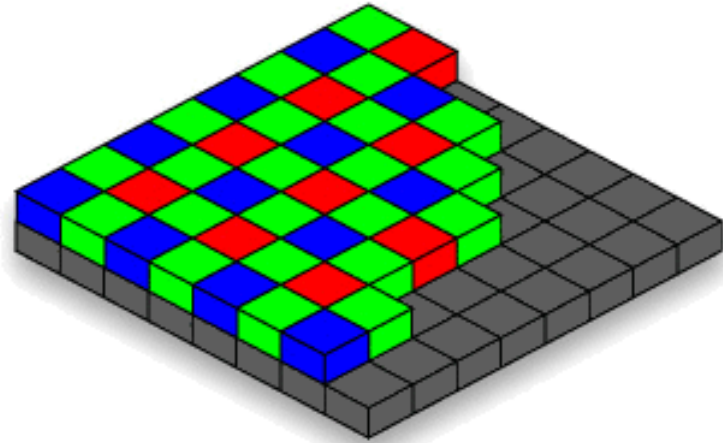
from Ted Adelson

Perception of Intensity



from Ted Adelson

Digital Color Images



Color Image

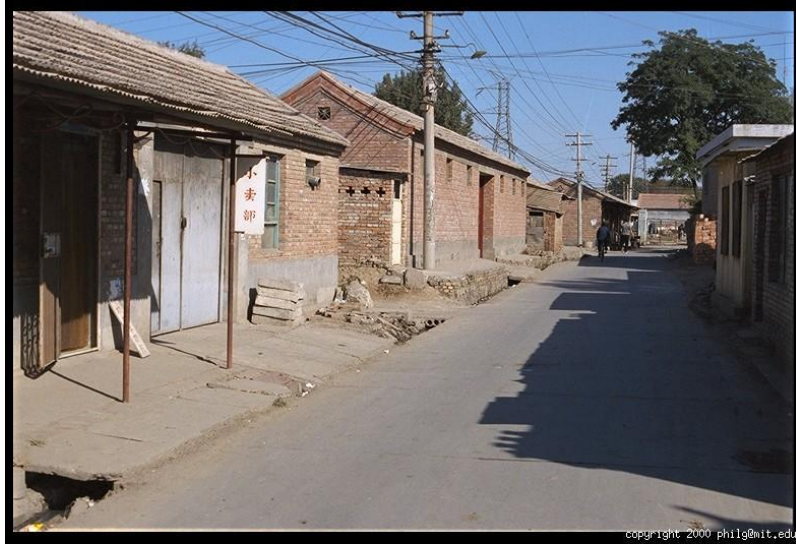


Image filtering

- Image filtering: compute function of local neighborhood at each position
- Really important!
 - Enhance images
 - Denoise, resize, increase contrast, etc.
 - Extract information from images
 - Texture, edges, distinctive points, etc.
 - Detect patterns
 - Template matching

Example: box filter

$g[\cdot, \cdot]$

1	1	1	1
9	1	1	1
9	1	1	1

Image filtering

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

1	1	1
1	1	1
1	1	1

$f[\cdot, \cdot]$

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

$h[\cdot, \cdot]$

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

Image filtering

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

1	1	1
1	1	1
1	1	1

$f[\cdot, \cdot]$

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

$h[\cdot, \cdot]$

	0	10							

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

Image filtering

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

$f[\cdot, \cdot]$

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

$h[\cdot, \cdot]$

	0	10	20						

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

Image filtering

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

1	1	1
1	1	1
1	1	1

$f[\cdot, \cdot]$

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

$h[\cdot, \cdot]$

	0	10	20	30					

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

Image filtering

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

1	1	1
1	1	1
1	1	1

$f[\cdot, \cdot]$

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

$h[\cdot, \cdot]$

	0	10	20	30	30				

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

Image filtering

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

$f[\cdot, \cdot]$

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

$h[\cdot, \cdot]$

	0	10	20	30	30				

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

Image filtering

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

1	1	1
1	1	1
1	1	1

$f[\cdot, \cdot]$

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

$h[\cdot, \cdot]$

	0	10	20	30	30				
						?			
				50					

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

Image filtering

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

1	1	1
1	1	1
1	1	1

$f[\cdot, \cdot]$

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

$h[\cdot, \cdot]$

	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	

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

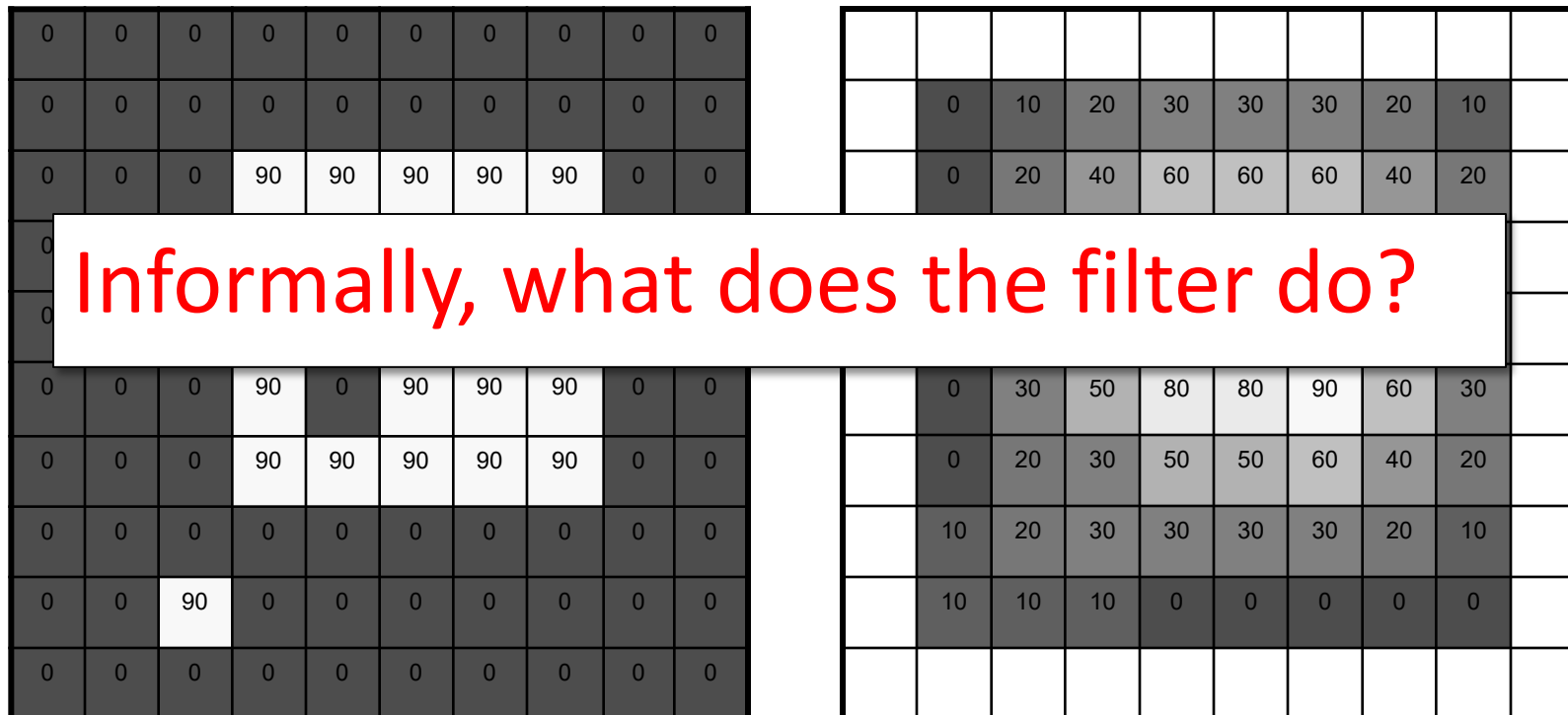
Image filtering

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

1	1	1
1	1	1
1	1	1

$f[\cdot, \cdot]$

$h[\cdot, \cdot]$



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

Box Filter

What does it do?

- Replaces each pixel with an average of its neighborhood
- Achieve smoothing effect (remove sharp features)

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

Smoothing with box filter



One more by hand...

0	1	1	0
1	2	2	0
0	0	0	1
0	1	1	2

*

1	0	0
0	1	0
0	0	1

=

0	1	1	0
1	2	2	0
0	0	0	1
0	1	1	2

Practice with linear filters

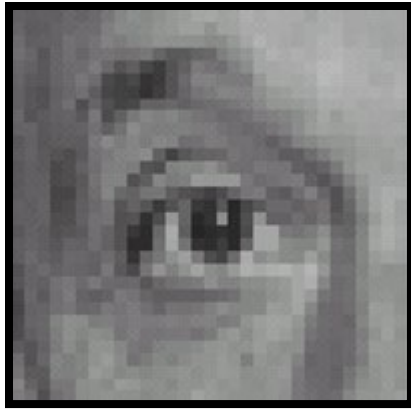


Original

0	0	0
0	1	0
0	0	0

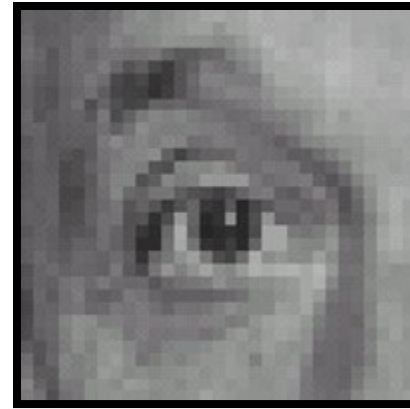
?

Practice with linear filters



Original

0	0	0
0	1	0
0	0	0



Filtered
(no change)

Practice with linear filters



Original

0	0	0
0	0	1
0	0	0

?

Practice with linear filters



Original

0	0	0
0	0	1
0	0	0



Shifted left
By 1 pixel

Practice with linear filters



Original

0	0	0
0	2	0
0	0	0

-

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

?

(Note that filter sums to 1)

Practice with linear filters



Original

0	0	0
0	2	0
0	0	0

-

$\frac{1}{9}$

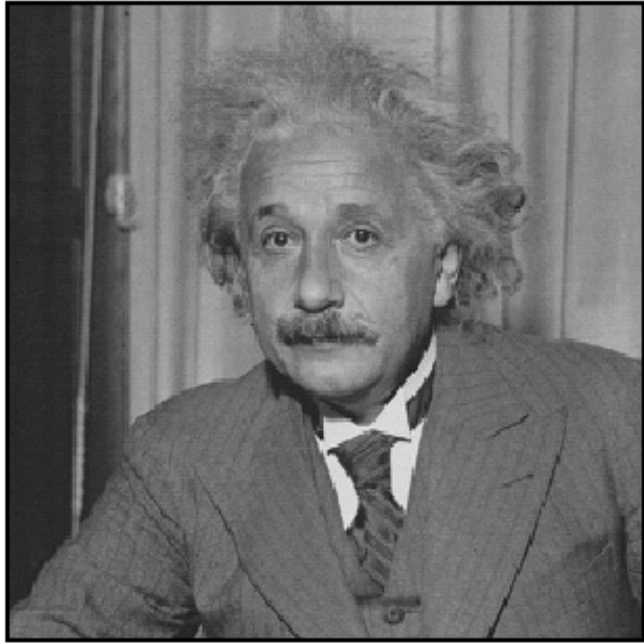
1	1	1
1	1	1
1	1	1



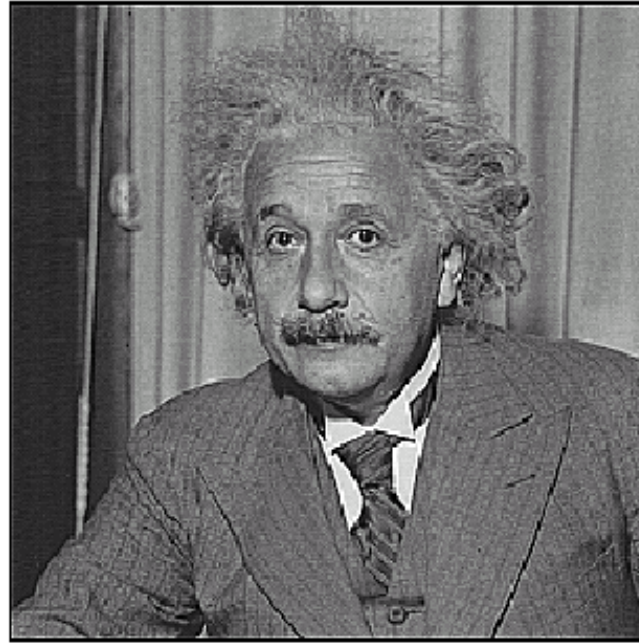
Sharpening filter

- Accentuates differences with local average

Sharpening

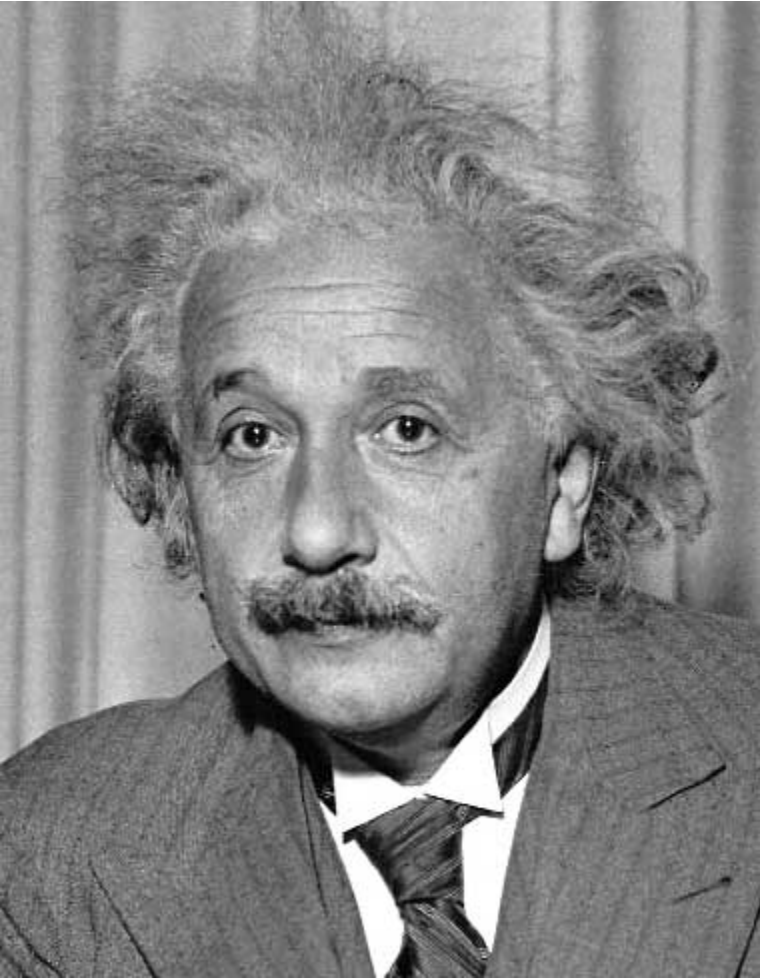


before



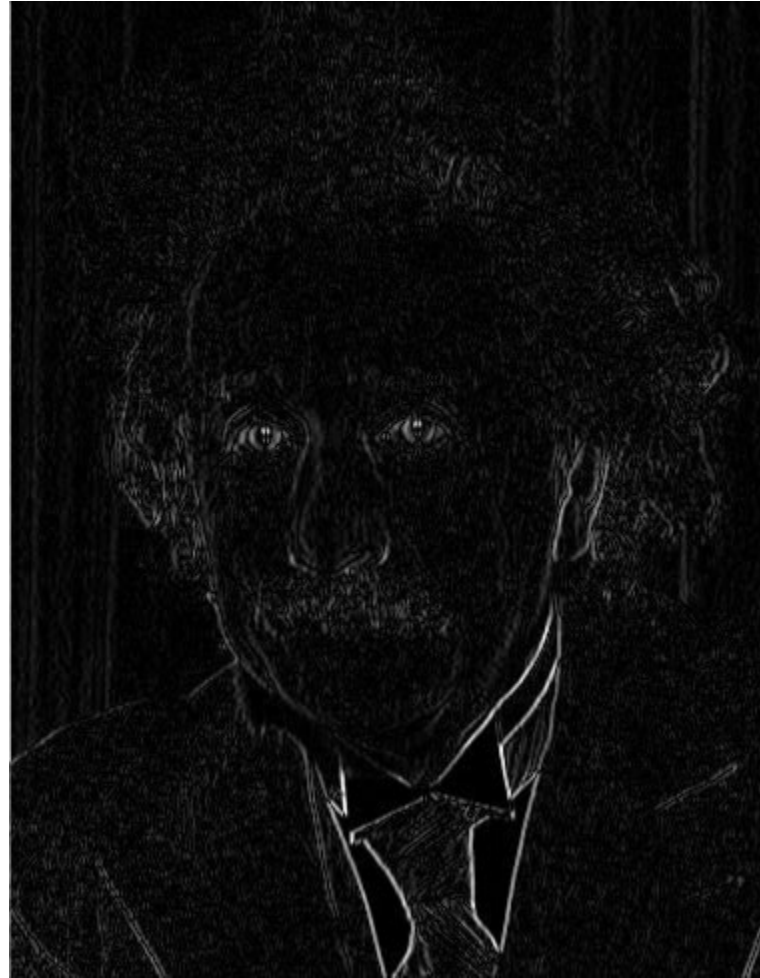
after

Other filters



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

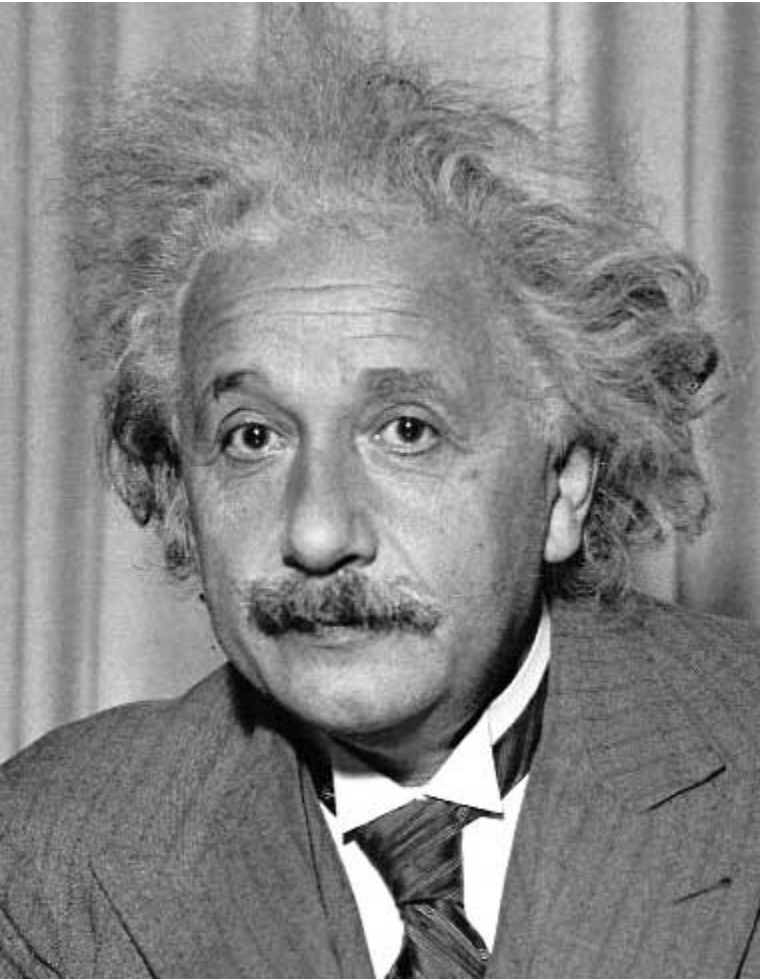
Sobel



Vertical Edge
(absolute value)

Other filters

Q?



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

Sobel



Horizontal Edge
(absolute value)

How could we synthesize motion blur?

```
theta = 30
len = 21
mid = (len-1)/2

fil = np.zeros((len,len))
fil[:,int(mid)] = 1/len
R = cv2.getRotationMatrix2D((mid,mid),theta,1)
fil = cv2.warpAffine(fil,R,(len,len))

im_fil = cv2.filter2D(im, -1, fil)
```

Correlation vs. Convolution

- 2d correlation

```
im_fil = cv2.filter2d(im, -1, fil)
```

$$im_fil[m,n] = \sum_{k,l} fil[k,l] im[m+k,n+l]$$

- 2d convolution

```
im_fil = scipy.signal.convolve2d(im, fil, [opts])
```

$$im_fil[m,n] = \sum_{k,l} fil[k,l] im[m-k,n-l]$$

- “convolve” mirrors the kernel, while “filter” doesn’t

```
cv2.filter2D(im, -1, cv2.flip(fil, -1)) same as  
signal.convolve2d(im, fil, mode='same', boundary='symm')
```


Key properties of linear filters

Linearity:

$$\text{filter}(f_1 + f_2) = \text{filter}(f_1) + \text{filter}(f_2)$$

Shift invariance: same behavior regardless of pixel location

$$\text{filter}(\text{shift}(f)) = \text{shift}(\text{filter}(f))$$

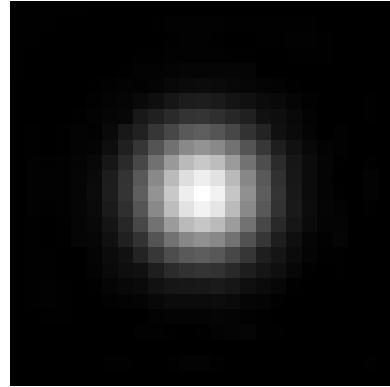
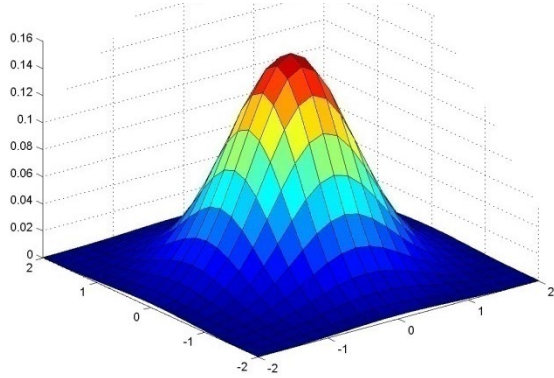
Any linear, shift-invariant operator can be represented as a convolution

More properties

- Commutative: $a * b = b * a$
 - Conceptually no difference between filter and signal (image)
- Associative: $a * (b * c) = (a * b) * c$
 - Often apply several filters one after another: $((a * b_1) * b_2) * b_3$
 - This is equivalent to applying one filter: $a * (b_1 * b_2 * b_3)$
- Distributes over addition: $a * (b + c) = (a * b) + (a * c)$
- Scalars factor out: $ka * b = a * kb = k(a * b)$
- Identity: unit impulse $e = [0, 0, 1, 0, 0]$,
 $a * e = a$

Important filter: Gaussian

- Weight contributions of neighboring pixels by nearness

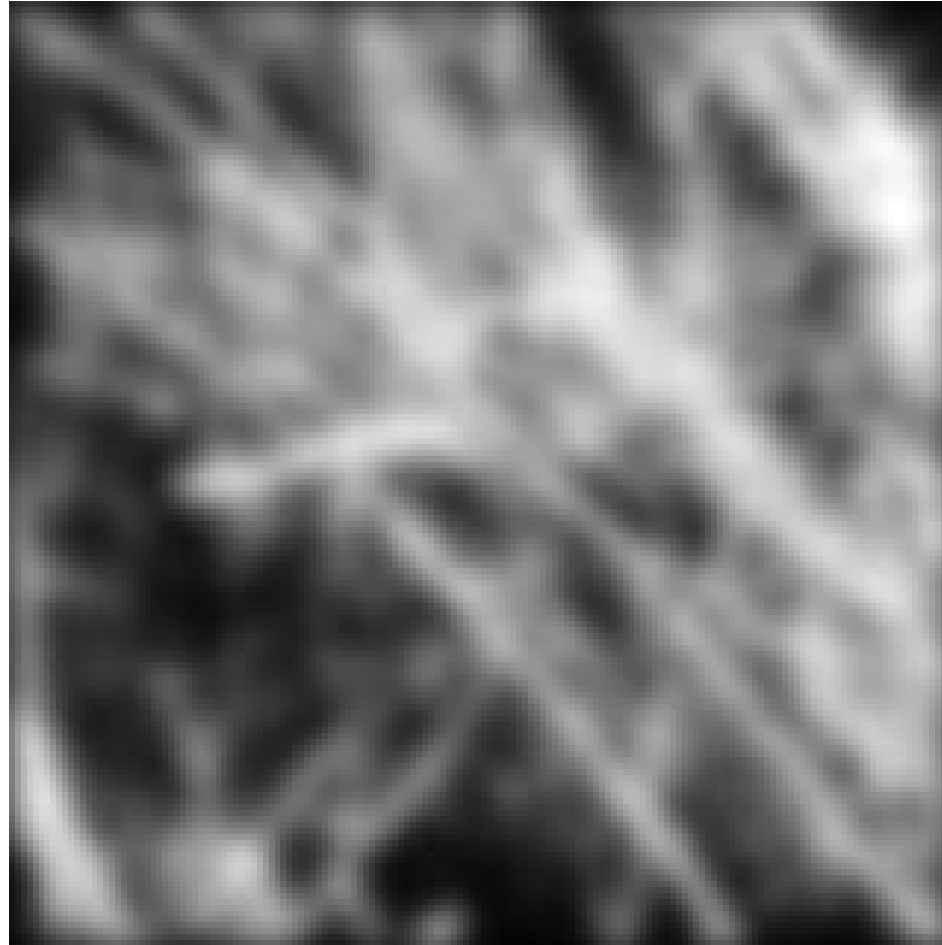


0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

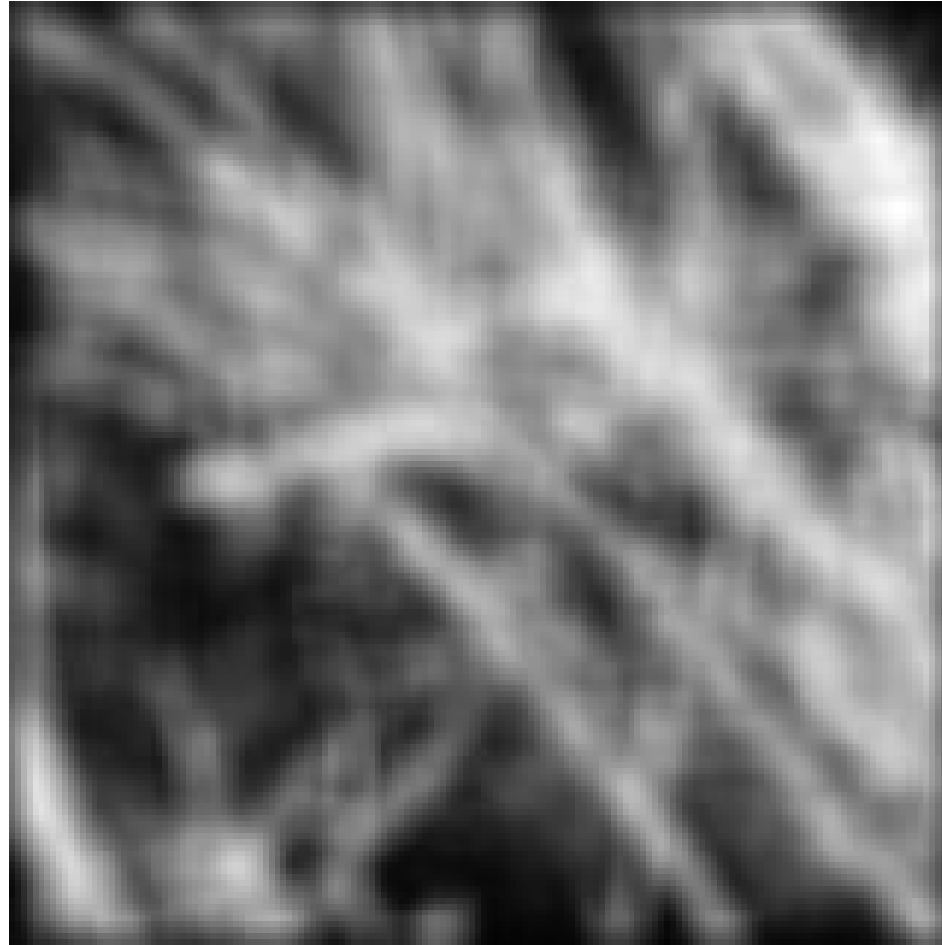
5 x 5, $\sigma = 1$

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

Smoothing with Gaussian filter



Smoothing with box filter



Gaussian filters

- Remove “high-frequency” components from the image (low-pass filter)
 - Images become more smooth
- Convolution with self is another Gaussian
 - So can smooth with small-width kernel, repeat, and get same result as larger-width kernel would have
 - Convoluting two times with Gaussian kernel of width σ is same as convoluting once with kernel of width $\sigma\sqrt{2}$
- *Separable* kernel
 - Factors into product of two 1D Gaussians

Separability of the Gaussian filter

$$\begin{aligned} G_{\sigma}(x, y) &= \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \\ &= \left(\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right)\right) \left(\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{y^2}{2\sigma^2}\right)\right) \end{aligned}$$

The 2D Gaussian can be expressed as the product of two functions, one a function of x and the other a function of y

In this case, the two functions are the (identical) 1D Gaussian

Separability example

2D filtering
(center location only)

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 3 & 3 \\ 3 & 5 & 5 \\ 4 & 4 & 6 \end{bmatrix}$$

The filter factors
into a product of 1D
filters:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

Perform filtering
along rows:

$$\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 3 & 3 \\ 3 & 5 & 5 \\ 4 & 4 & 6 \end{bmatrix} = \begin{bmatrix} & 11 & \\ & 18 & \\ & 18 & \end{bmatrix}$$

Followed by filtering
along the remaining column:

Separability

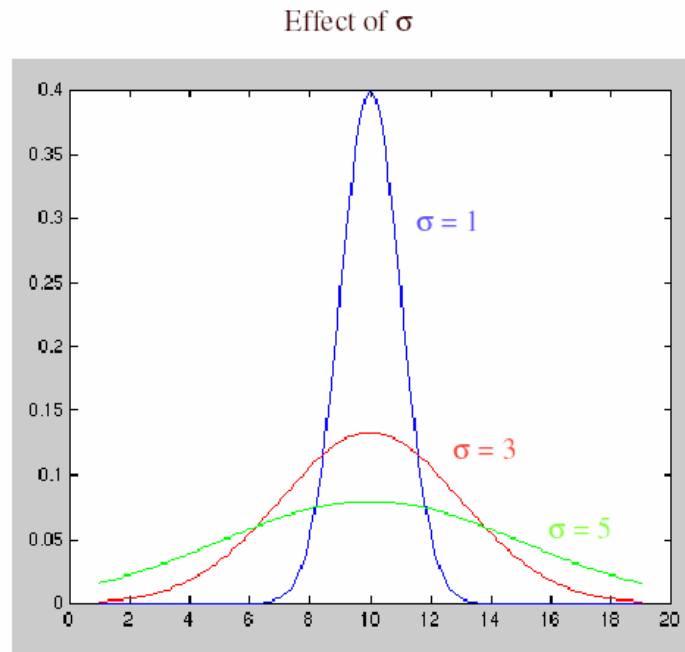
- Why is separability useful in practice?

Some practical matters

Practical matters

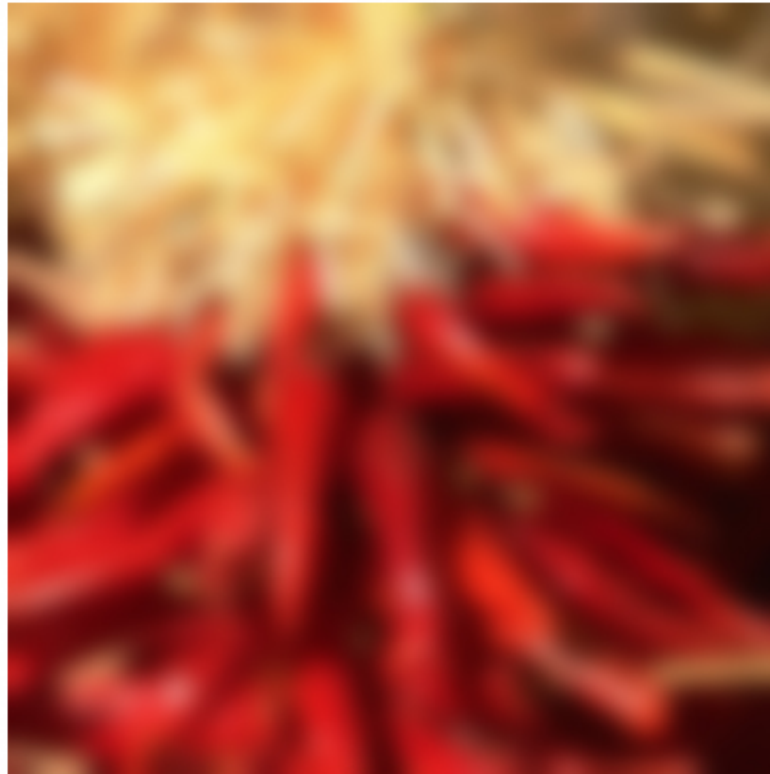
How big should the filter be?

- Values at edges should be near zero
- Rule of thumb for Gaussian: set kernel half-width to $\geq 3 \sigma$



Practical matters

- What about near the edge?
 - the filter window falls off the edge of the image
 - need to extrapolate
 - methods:
 - clip filter (black)
 - wrap around
 - copy edge
 - reflect across edge



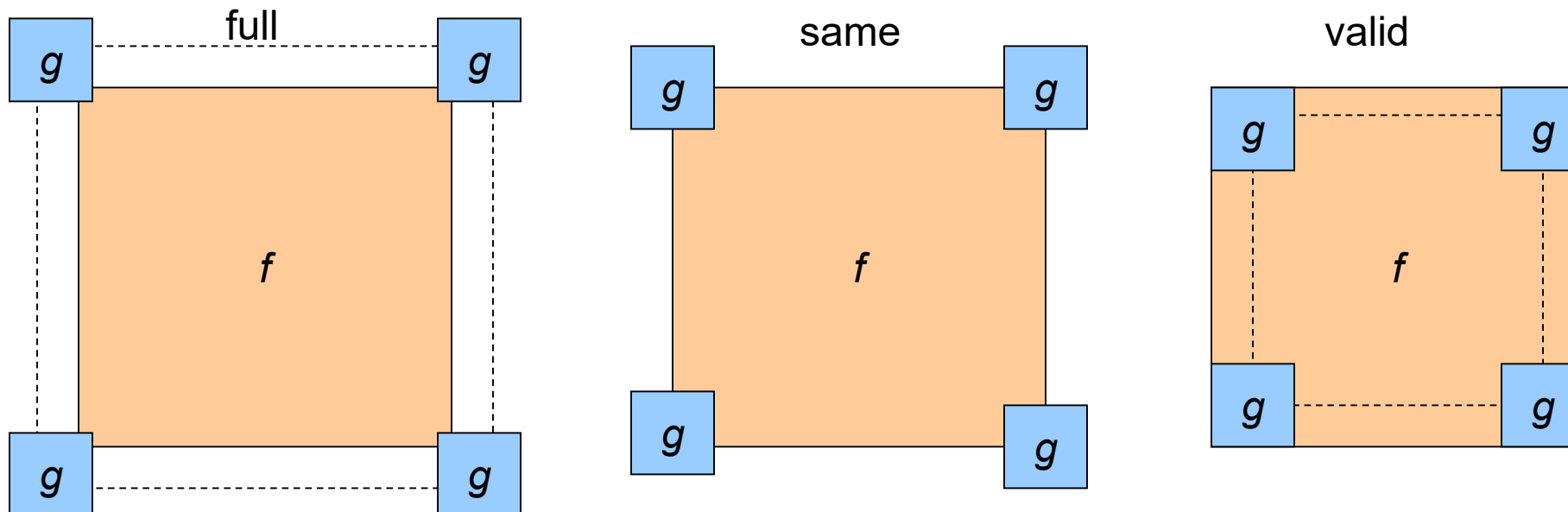
Practical matters

– methods (Python):

- clip filter (black): `convolve2d(f, g, boundary='fill', 0)`
- wrap around: `convolve2d(f, g, boundary='wrap')`
- reflect across edge: `convolve2d(f, g, boundary='symm')`

Practical matters

- What is the size of the output?
- Python: `convolve2d(g, f, mode)`
 - *mode* = 'full': output size is sum of sizes of *f* and *g*
 - *mode* = 'same': output size is same as *f*
 - *mode* = 'valid': output size is difference of sizes of *f* and *g*

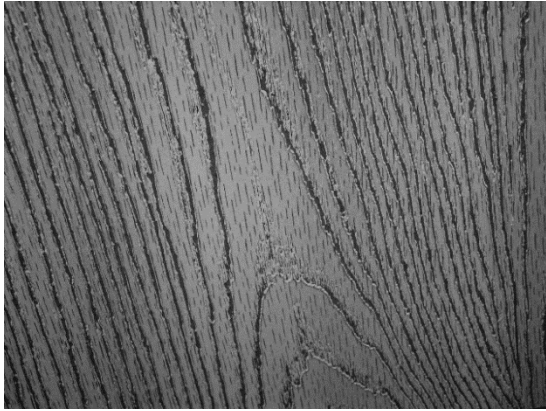


Application: Representing Texture

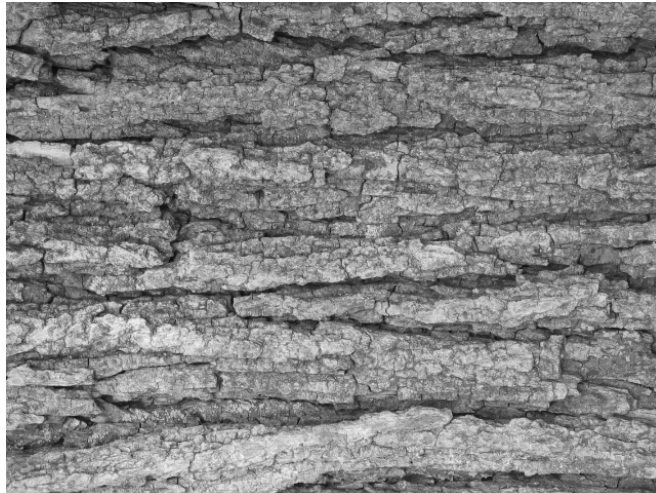


Source: Forsyth

Texture and Material



Texture and Orientation



Texture and Scale



http://www-cvr.ai.uiuc.edu/ponce_grp/data/texture_database/samples/

What is texture?

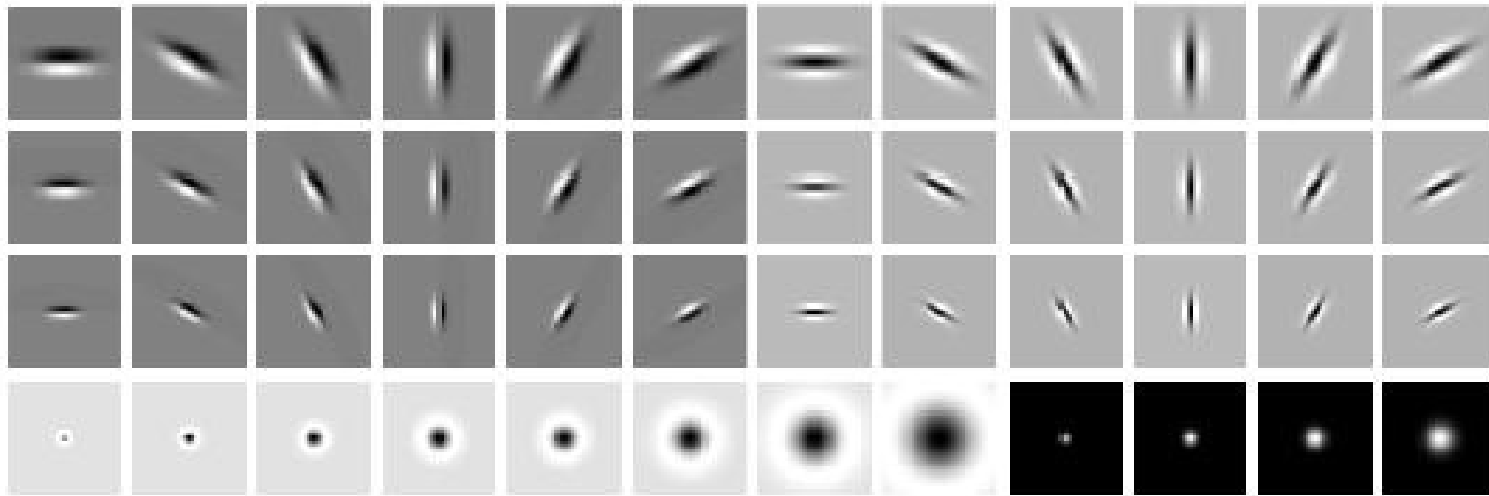
Regular or stochastic patterns caused by bumps, grooves, and/or markings

How can we represent texture?

- Compute responses of blobs and edges at various orientations and scales

Overcomplete representation: filter banks

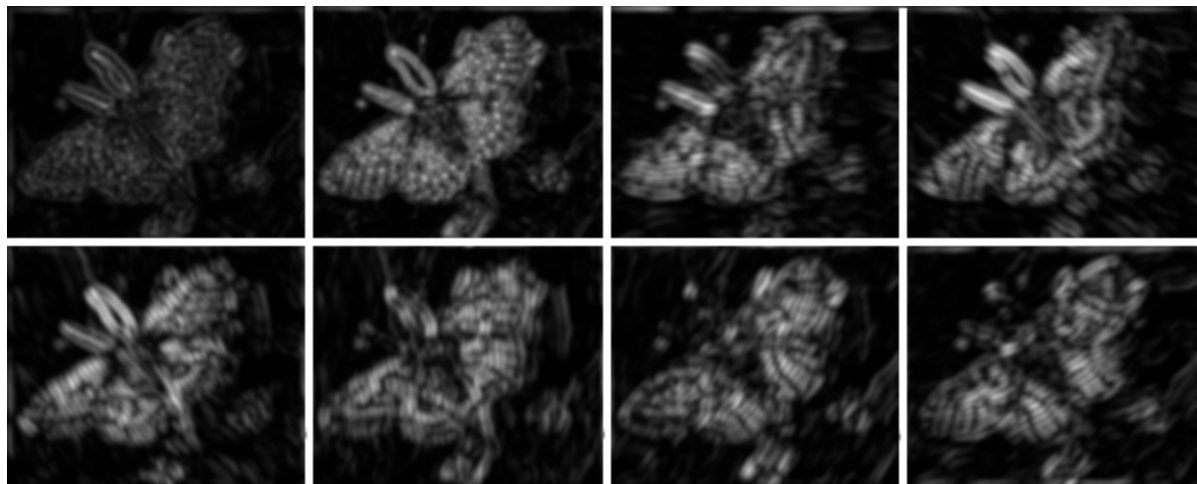
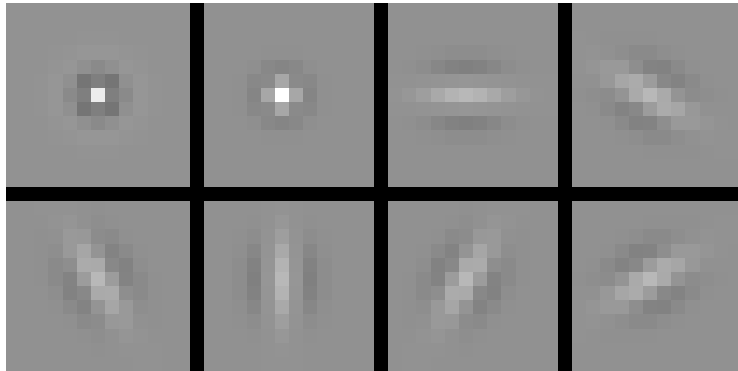
LM Filter Bank



Code for filter banks: www.robots.ox.ac.uk/~vgg/research/texclass/filters.html

Filter banks

- Process image with each filter and keep responses (or squared/abs responses)

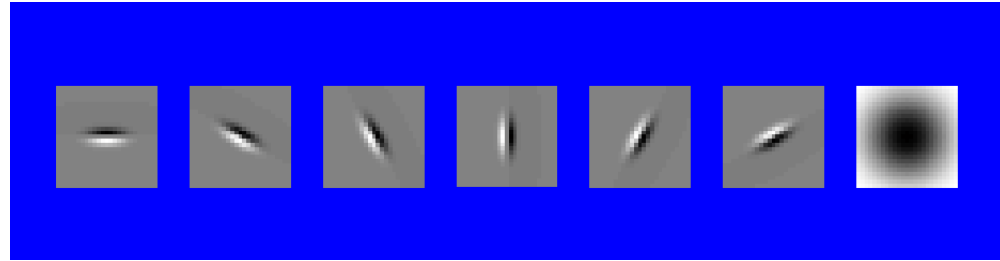


How can we represent texture?

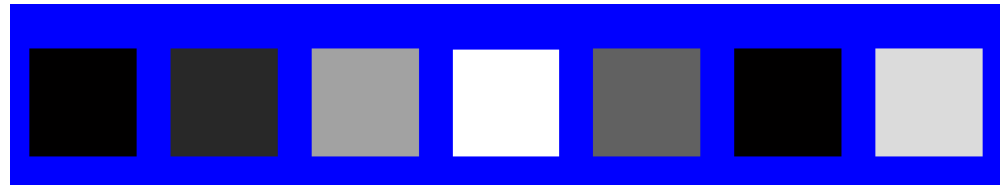
- Measure responses of blobs and edges at various orientations and scales
- Record simple statistics (e.g., mean, std.) of absolute filter responses

Can you match the texture to the response?

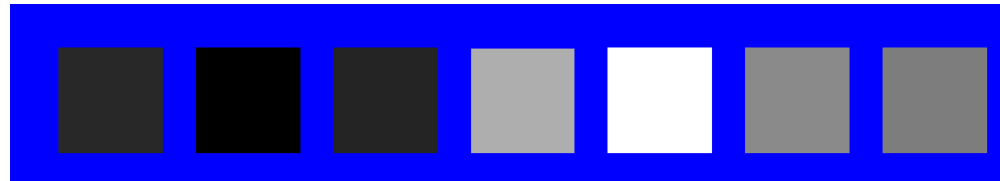
Filters



1



2

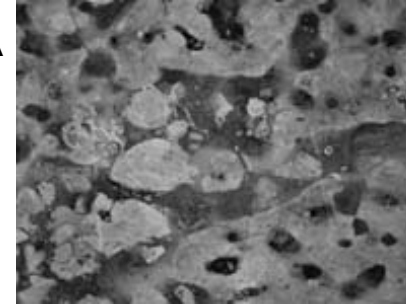


3



Mean abs responses

A



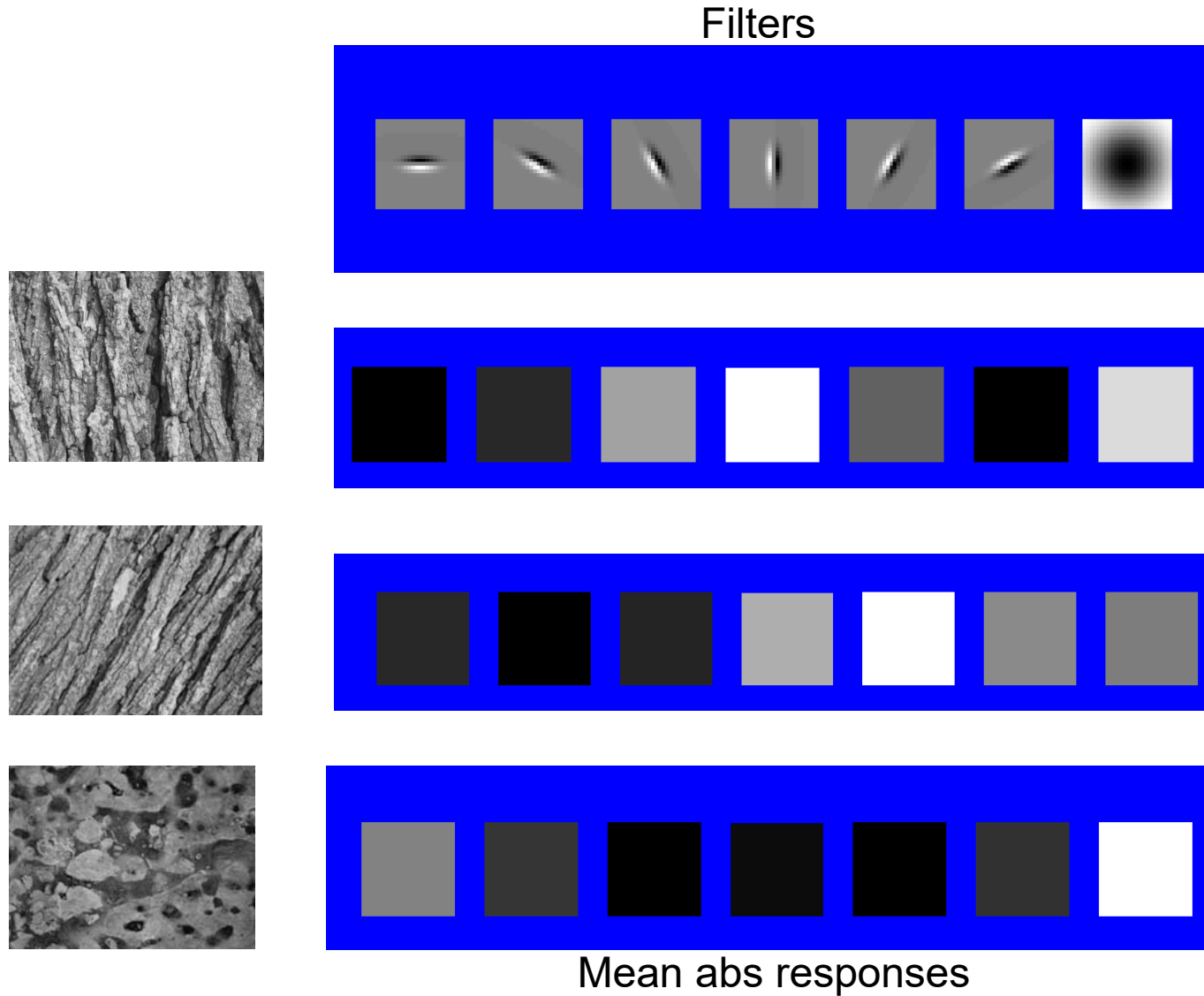
B



C



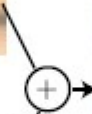
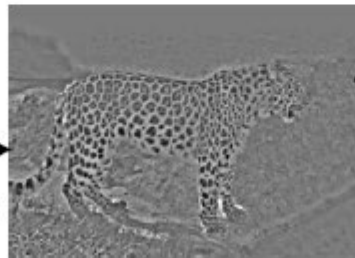
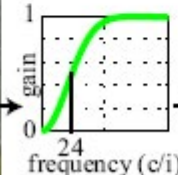
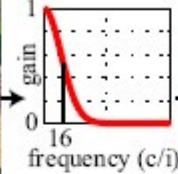
Representing texture by mean abs response



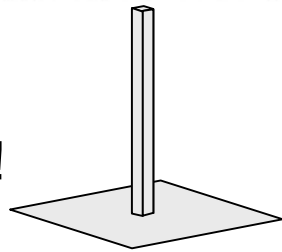
Project 1: Hybrid Images

A. Oliva, A. Torralba, P.G. Schyns,
["Hybrid Images,"](#) SIGGRAPH 2006

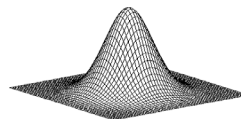
Gaussian Filter!



Laplacian Filter!

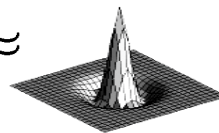


unit impulse



Gaussian

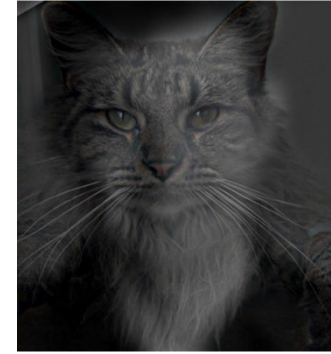
\approx



Laplacian of Gaussian

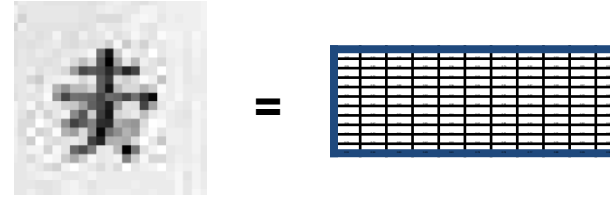
Project Instructions:

https://courses.engr.illinois.edu/cs445/fa2019/projects/hybrid/ComputationalPhotography_ProjectHybrid.html



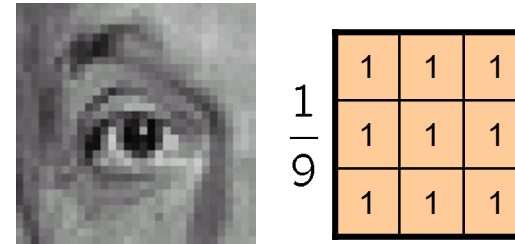
Take-home messages

- Image is a matrix of numbers



- Linear filtering is a dot product at each position

- Can smooth, sharpen, translate (among many other uses)



- Be aware of details for filter size, extrapolation, cropping



- Start thinking about project (read the paper, create a test project page)



Take-home questions

1. Write down a 3x3 filter that returns a positive value if the average value of the 4-adjacent neighbors is less than the center and a negative value otherwise
2. Write down a filter that will compute the gradient in the x-direction:

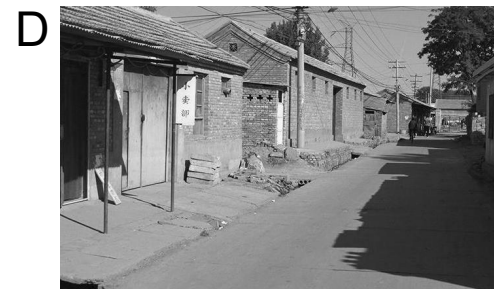
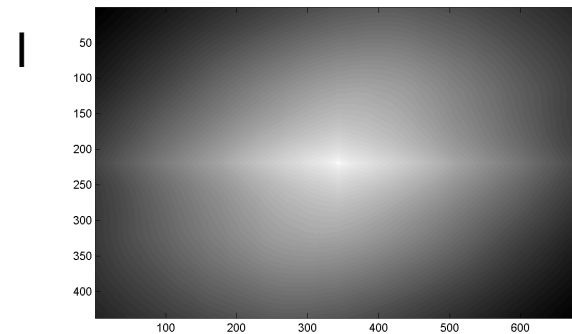
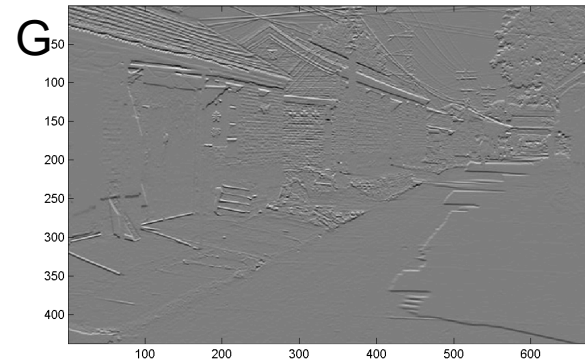
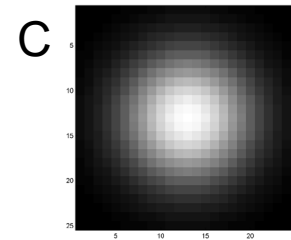
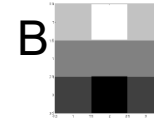
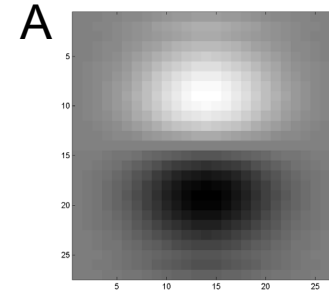
$$\text{grad}_x(y, x) = \text{im}(y, x+1) - \text{im}(y, x) \text{ for each } x, y$$

Take-home questions

Filtering Operator

3. Fill in the blanks:

- a) $_ = D * B$
 b) $A = _ * _$
 c) $F = D * _$
 d) $_ = D * D$



Next class: Thinking in Frequency

