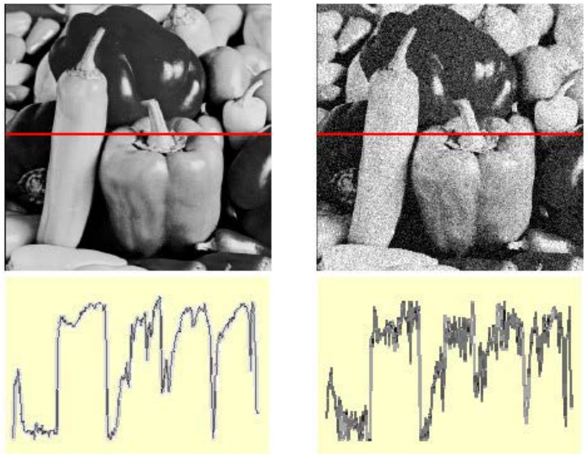
CS 4495 Computer Vision

Linear Filtering 1: Filters, Convolution, Smoothing

Aaron Bobick
School of Interactive Computing



Administrivia – Fall 2014

August 26:

There are a lot of you!

4495A – about 62 undergrads

4495GR – 40 grad students – mostly MS(CS)

7495 – 40 mostly PhD students

- Still working on the CS7495 model. We will **not** meet tonight – I'll be working on your web site instead. Still need a room.

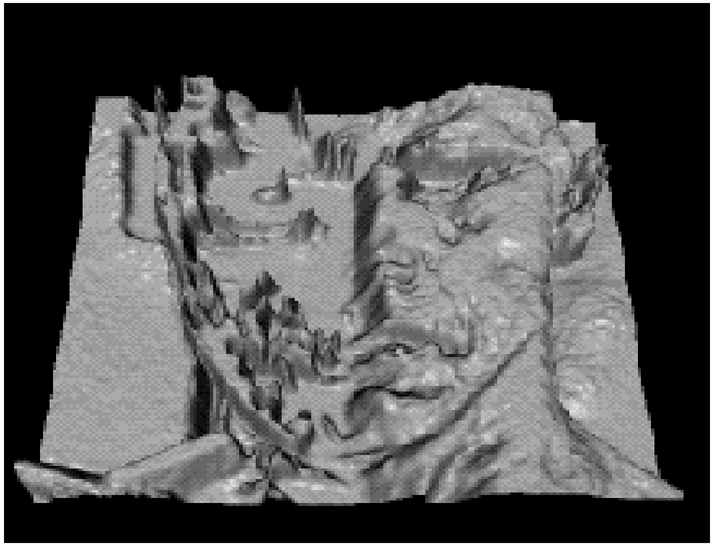
- 4495: PS0 can now be handed in.

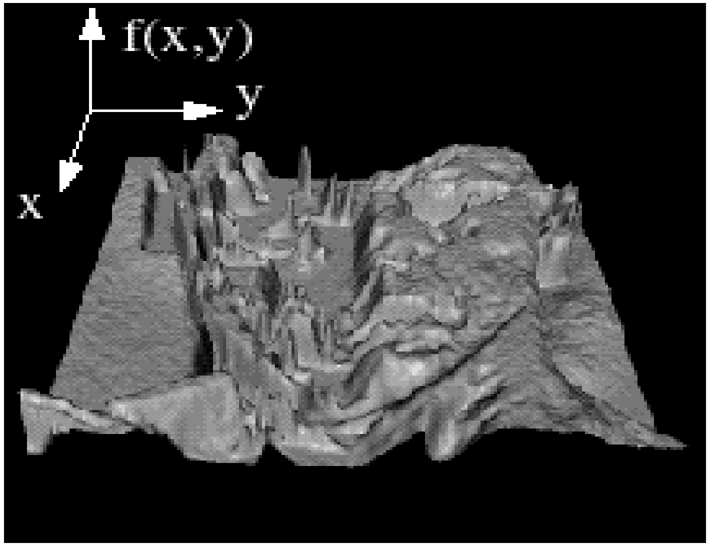
- 4495: PS1 will be out Thursday – due next Sunday (Sept 7) 11:55pm.

Linear outline *(hah!)*

- Images are really *functions* $\vec{I}(x, y)$ where the vector can be any dimension but typical are 1, 3, and 4. (When 4?) Or thought of as a multi-dimensional *signal* as a function of spatial location.
- ***Image processing*** is (mostly) computing new functions of image functions. Many involve ***linear*** *operators.*
- Very useful linear operator is *convolution /correlation* - what most people call filtering – because the new value is determined by local values.
- With convolution can do things like noise reduction, smoothing, and edge finding (last one is next time).

Images as functions





Source: S. Seitz

Images as functions

- We can think of an image as a function, f or I, from R^2 to R:

  $f(x, y)$ gives the *intensity* or value at position (x, y)

  Realistically, we expect the image only to be defined over a rectangle, with a finite range:

  $f: [a,b] \times [c,d] \rightarrow [0, 1.0]$ *(why sometimes 255???)*

- A color image is just three functions "pasted" together. We can write this as a "vector-valued" function:

$$f(x,y) = \begin{bmatrix} r(x,y) \\ g(x,y) \\ b(x,y) \end{bmatrix}$$

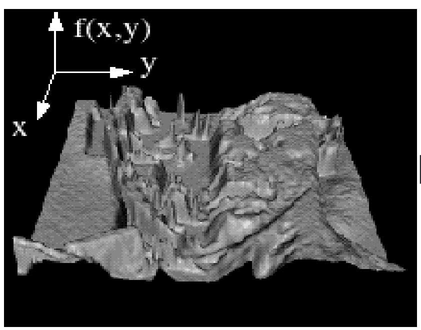
Source: S. Seitz

The real Arnold

```
>> arnold(40:60,30:40)

ans =

   152   122    99    83   122   120   154   150   123   141   112
   102   140   109   114   125   124    69   134   123   141   132
   138   160   135   109   104    89    91   145   128   102   154
   101   147   165    87    93    97   110   145   157   124   141
    58    68    96   115    80    98   137   160   145   168   166
    57   127    62    92   145   127    93   121   168   221   157
    69   108    74    71   156   119   106   140   156   161   158
   116   132   101    60   134   159   110   125   153   145   123
   109   119   130   113    80   176   121   108   111   152   133
   135    77   102   134   127   136   154   130   139   120   160
   175   127   112   145   153   125   160   126   103    94   166
   205   187   151    87   128   154   124   174    96   129   142
   206   211   207   171   153   146   173   194   125   129   164
   214   205   235   200   170   162   151   151   183   152   107
   225   199   211   203   125   145   154   181   201   184   137
   207   203   172   169   170   127   116    95   197   187   138
   171   208   150   157   184   153   109   119   148   182   138
   111   170   150   116   128   170   144   132   119   176   132
   101   172   168   130   112   131   116   136   129   137   121
   103   167   164   131   104   106    96   111   106   103   139
    92   136   146   138    92    63    73   101   120   126   134
```

Digital images

- In computer vision we typically operate on **digital** (**discrete**) images:
  - **Sample** the 2D space on a regular grid
  - **Quantize** each sample (round to "nearest integer")
- Image thus represented as a matrix of integer values.

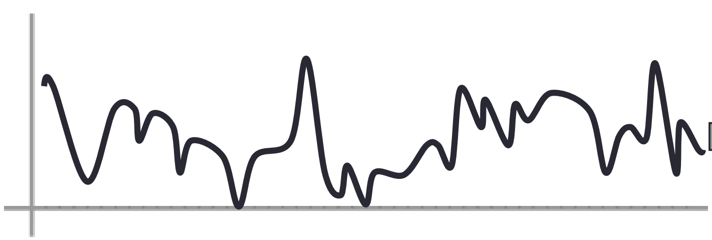


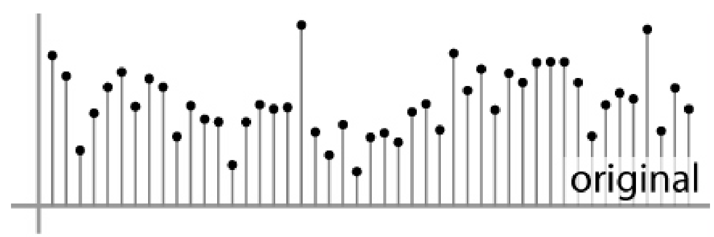
j

i

62	79	23	119	120	105	4	0
10	10	9	62	12	78	34	0
10	58	197	46	46	0	0	48
176	135	5	188	191	68	0	49
2	1	1	29	26	37	0	77
0	89	144	147	187	102	62	208
255	252	0	166	123	62	0	31
166	63	127	17	1	0	99	30

2D



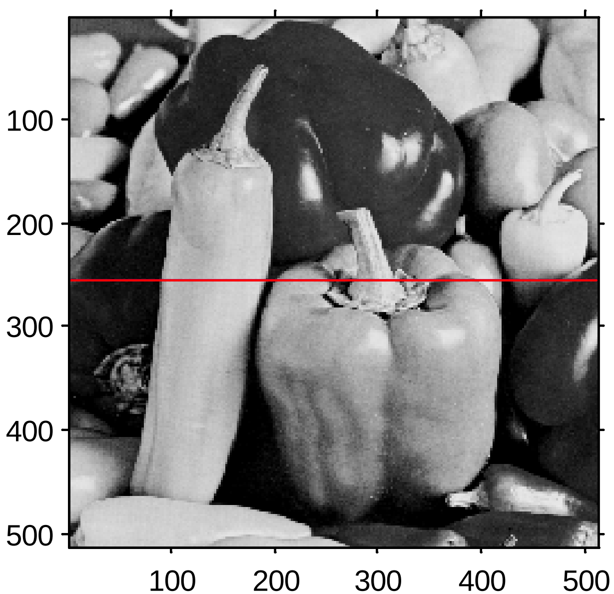


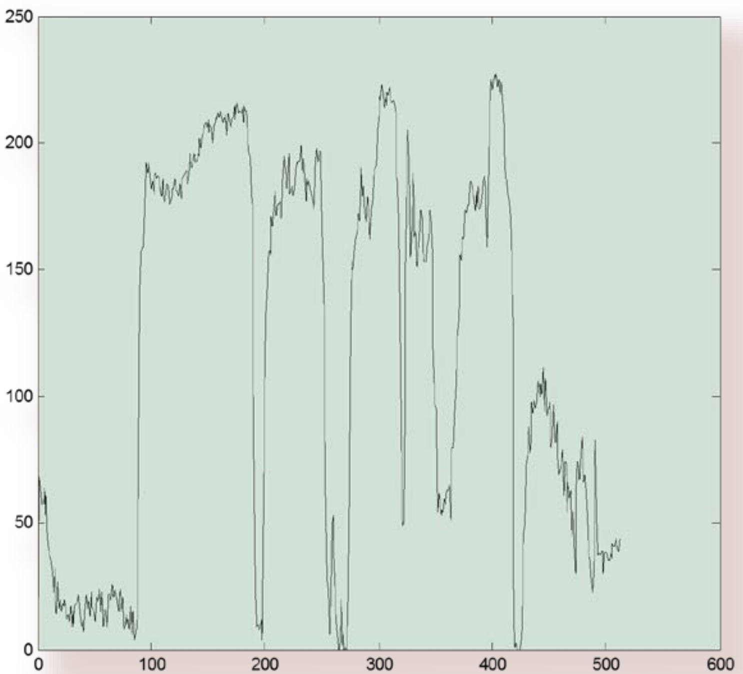
1D

Adapted from S. Seitz

Matlab – images are matrices

```
>> im = imread('peppers.png');  % semicolon or many numbers
>> imgreen = im(:,:,2);
>> imshow(imgreen)
>> line([1 512], [256 256],'color','r')
>> plot(imgreen(256,:));
```





Noise in images

- Noise as an example of images really being functions

- Noise is just another function that is combined with the original function to get a new – guess what – function

$$\vec{I}\,'(x,y) = \vec{I}(x,y) + \vec{\eta}(x,y)$$

- In images noise looks, well, noisy.

Common types of noise

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

Original

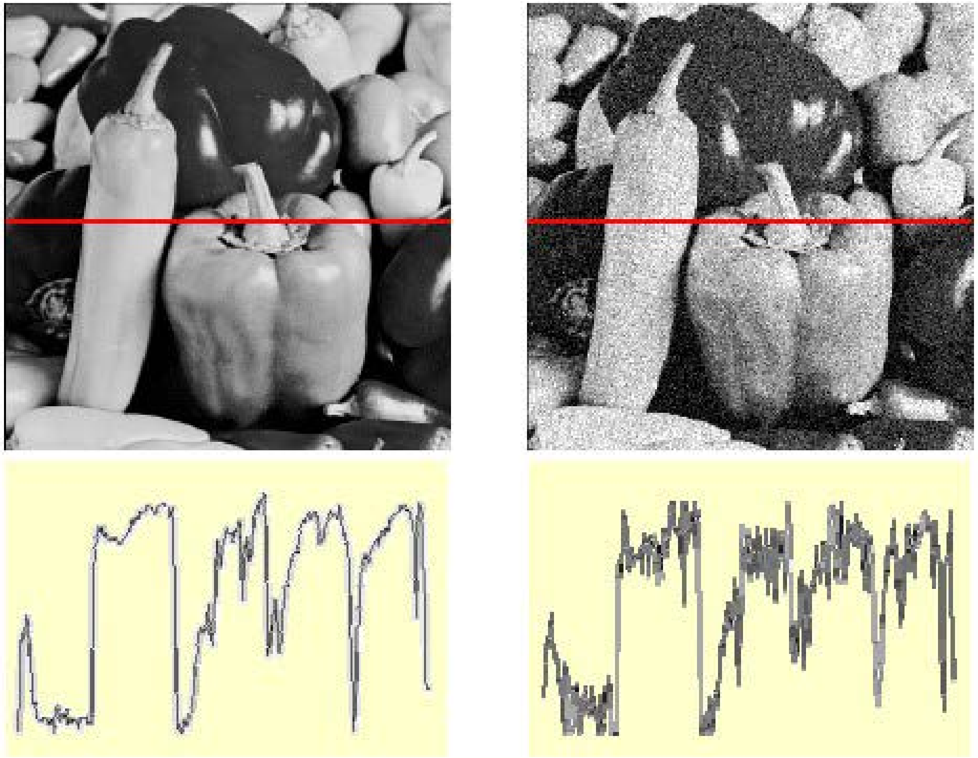
Salt and pepper noise

Impulse noise

Gaussian noise

Source: S. Seitz

Gaussian noise



$$f(x,y) = \overbrace{\bar{f}(x,y)}^{\text{Ideal Image}} + \overbrace{\eta(x,y)}^{\text{Noise process}}$$

Gaussian i.i.d. ("white") noise:
$\eta(x,y) \sim \mathcal{N}(\mu, \sigma)$

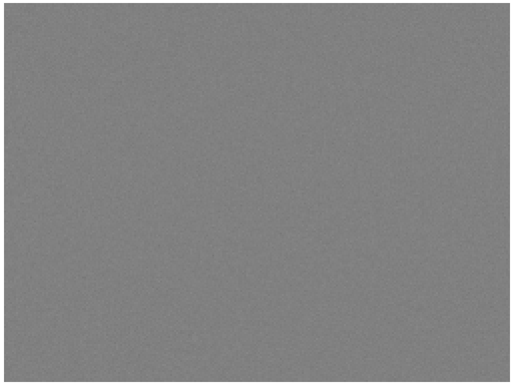
```
>> noise = randn(size(im)).*sigma;

>> output = im + noise;
```

Fig: M. Hebert

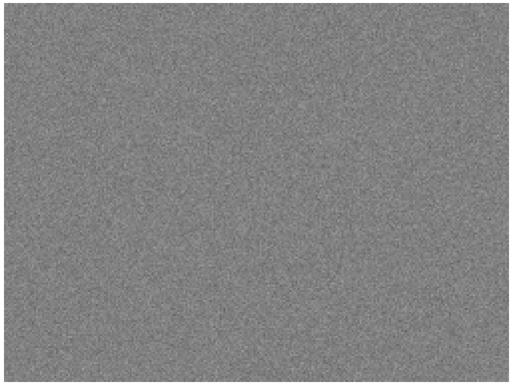
Effect of σ on Gaussian noise

Image shows the noise values themselves.

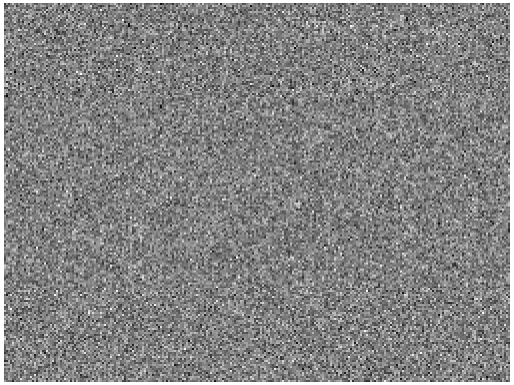
noise = randn(size(im)).*sigma;



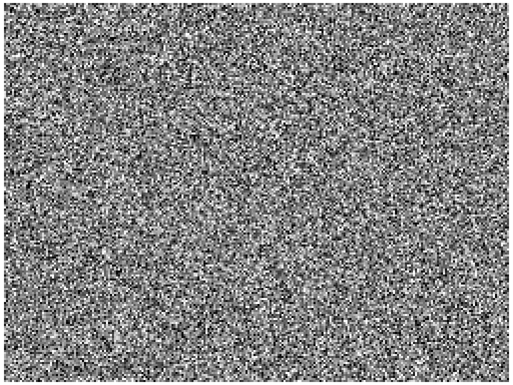
Sigma = 2



Sigma = 8



Sigma = 32



Sigma = 64

BE VERY CAREFUL!!!

- In previous slides, I did not say (at least wasn't supposed to say) what the range of the image was. A σ of 1.0 would be tiny if the range is [0 255] but huge if [0.0 1.0].

- Matlab can do either and you need to be very careful. If in doubt convert to double.

- Even more difficult can be displaying the image. Things like:
  - `imshow(I,[LOW HIGH])`

  display the image from [low high]

Don't worry – you'll get used to these hassles… see problem set PS0.

Back to our program…

Suppose want to remove the noise…

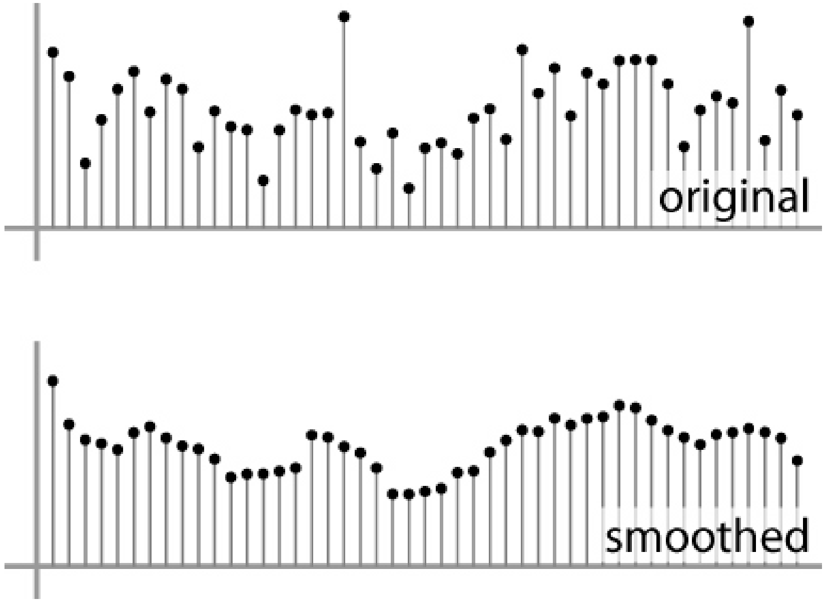
First attempt at a solution

- Suggestions?
- Let's replace each pixel with an average of all the values in its neighborhood
- Assumptions:
  - Expect pixels to be like their neighbors
  - Expect noise processes to be independent from pixel to pixel

K. Grauman

First attempt at a solution

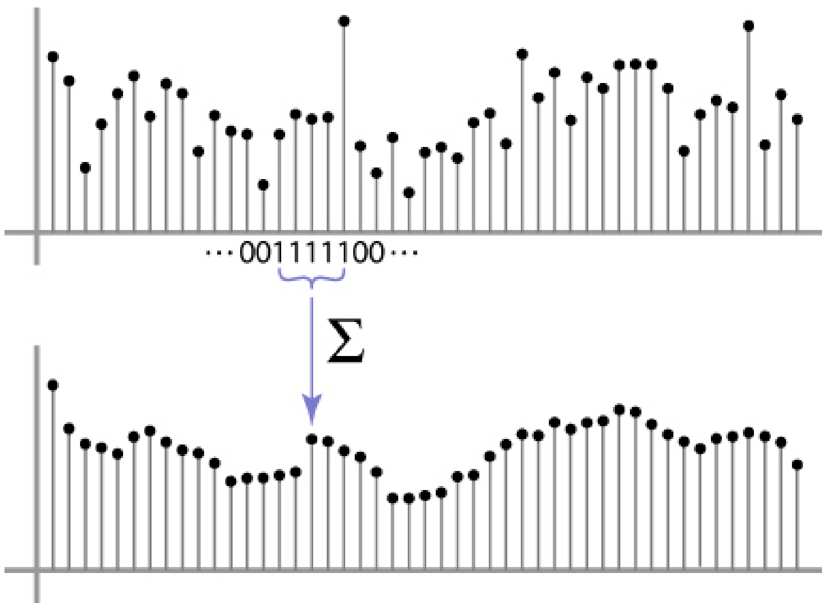
- Let's replace each pixel with an average of all the values in its neighborhood
- Moving average in 1D:



Source: S. Marschner

Weighted Moving Average

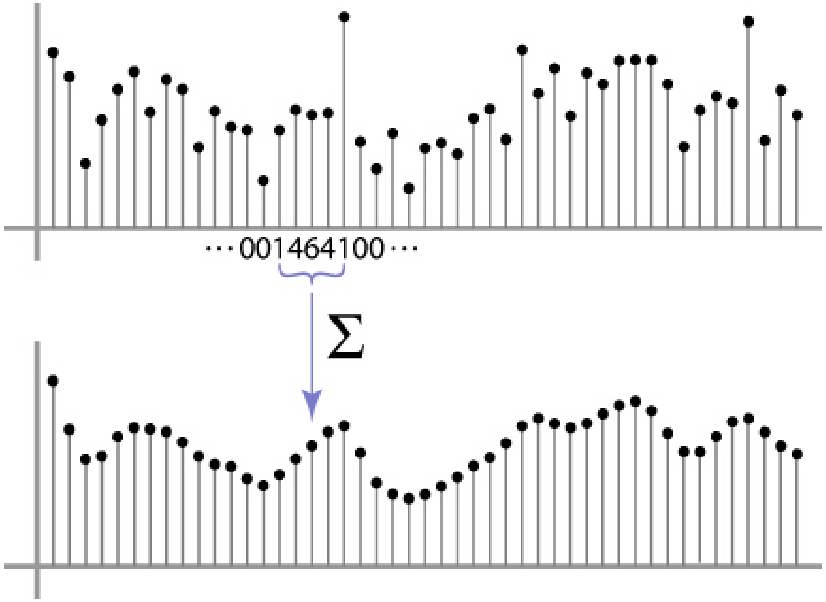
- Can add weights to our moving average
- *Weights* [1, 1, 1, 1, 1] / 5



Source: S. Marschner

Weighted Moving Average

- Non-uniform weights [1, 4, 6, 4, 1] / 16



Source: S. Marschner

Moving Average In 2D

Reference point

$F[x, y]$

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

$G[x, y]$

Source: S. Seitz

Moving Average In 2D

$F[x, y]$

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

$G[x, y]$

	0								

Source: S. Seitz

Moving Average In 2D

$$F[x, y]$$

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

$$G[x, y]$$

	0	10							

Source: S. Seitz

Moving Average In 2D

$$F[x, y]$$

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

$$G[x, y]$$

	0	10	20						

Source: S. Seitz

Moving Average In 2D

$F[x, y]$

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

$G[x, y]$

	0	10	20	30					

Source: S. Seitz

Moving Average In 2D

$F[x, y]$

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

$G[x, y]$

	0	10	20	30	30				

Source: S. Seitz

Moving Average In 2D

$$F[x, y]$$

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

$$G[x, y]$$

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

Source: S. Seitz

Correlation filtering

Say the averaging window size is 2k+1 x 2k+1:

$$G[i,j] = \underbrace{\frac{1}{(2k+1)^2}}_{\text{Attribute uniform weight to each pixel}} \underbrace{\sum_{u=-k}^{k} \sum_{v=-k}^{k} F[i+u, j+v]}_{\text{Loop over all pixels in neighborhood around image pixel } F[i,j]}$$

Now generalize to allow **different weights** depending on neighboring pixel's relative position:

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} \underbrace{H[u,v]}_{\text{Non-uniform weights}} F[i+u, j+v]$$

Correlation filtering

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u,j+v]$$

This is called **cross-correlation**, denoted $G = H \otimes F$

Filtering an image: replace each pixel with a linear combination of its neighbors.

The filter "**kernel**" or "**mask**" *H*[*u*,*v*] is the prescription for the weights in the linear combination.

Averaging filter

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

$F[x,y]$

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

$\otimes$

$H[u,v]$

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

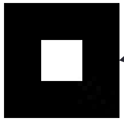
"box filter"

$G[x,y]$

	0	10	20	30	30				

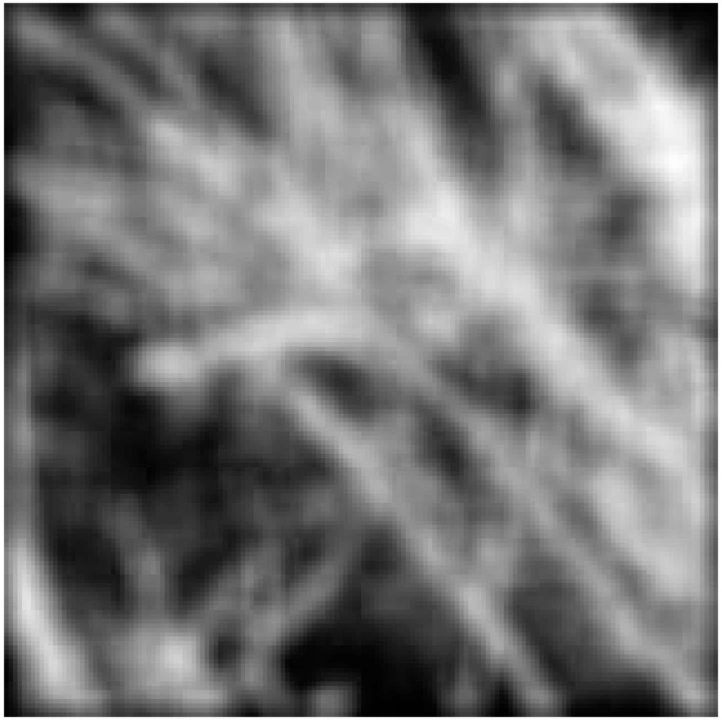
$$G = H \otimes F$$

Smoothing by averaging



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

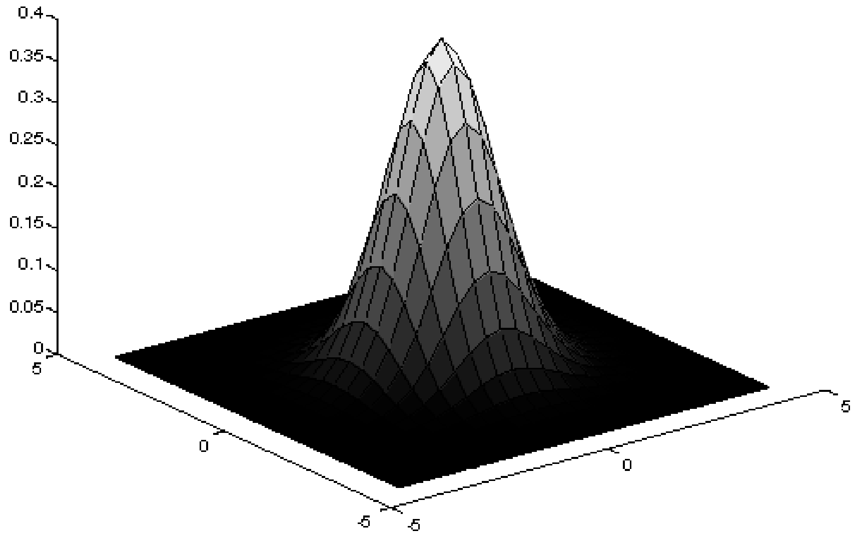
original



filtered

Squares aren't smooth…

- Smoothing with an average actually doesn't compare at all well with a defocussed lens
- Most obvious difference is that a single point of light viewed in a defocussed lens looks like a fuzzy blob; but the averaging process would give a little square.
- More about "impulse" responses later…



D. Forsyth

Gaussian filter

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

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

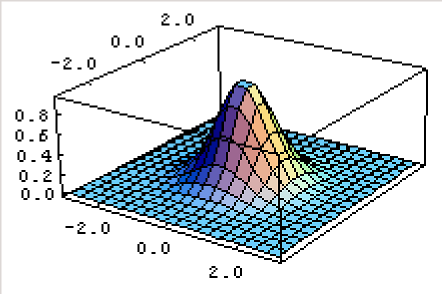
$F[x, y]$

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$H[u, v]$

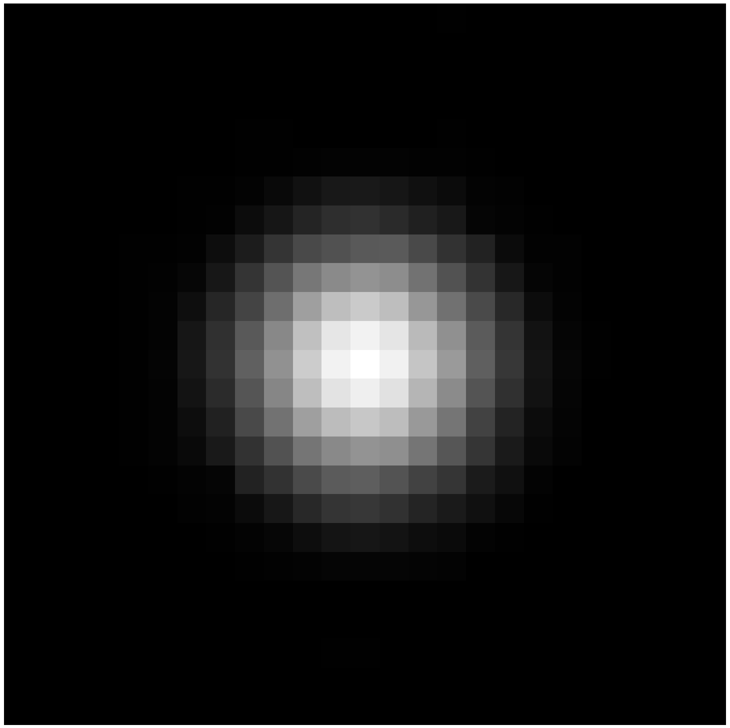
This kernel is an approximation of a Gaussian function:

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



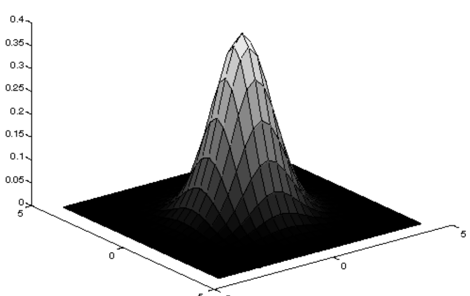
Source: S. Seitz

An Isotropic Gaussian



The picture shows a smoothing kernel proportional to

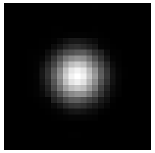
$$\exp\left(-\frac{(x^2 + x^2)}{2\sigma^2}\right)$$

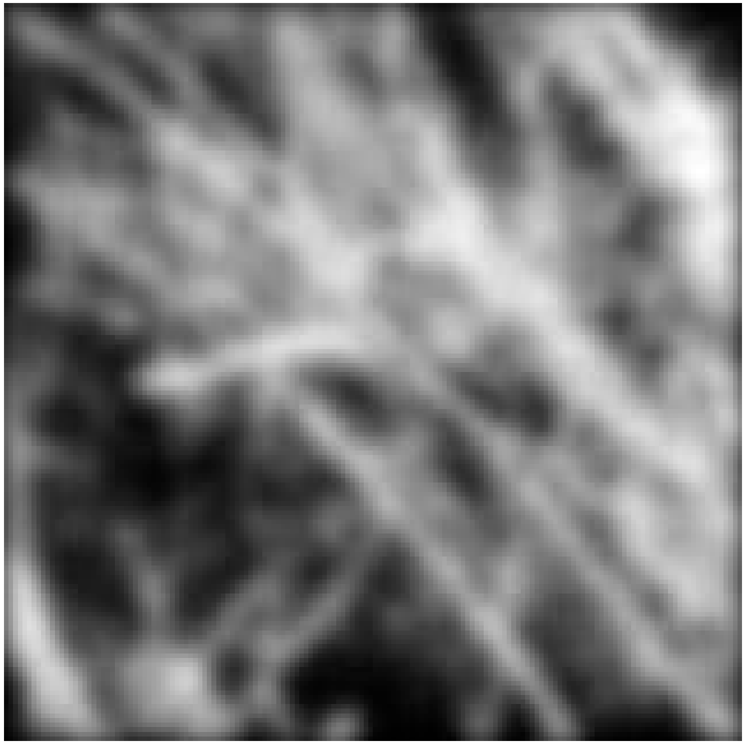


(which is a reasonable model of a circularly symmetric fuzzy blob)

D. Forsyth

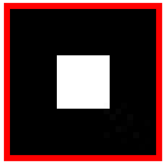
Smoothing with a Gaussian

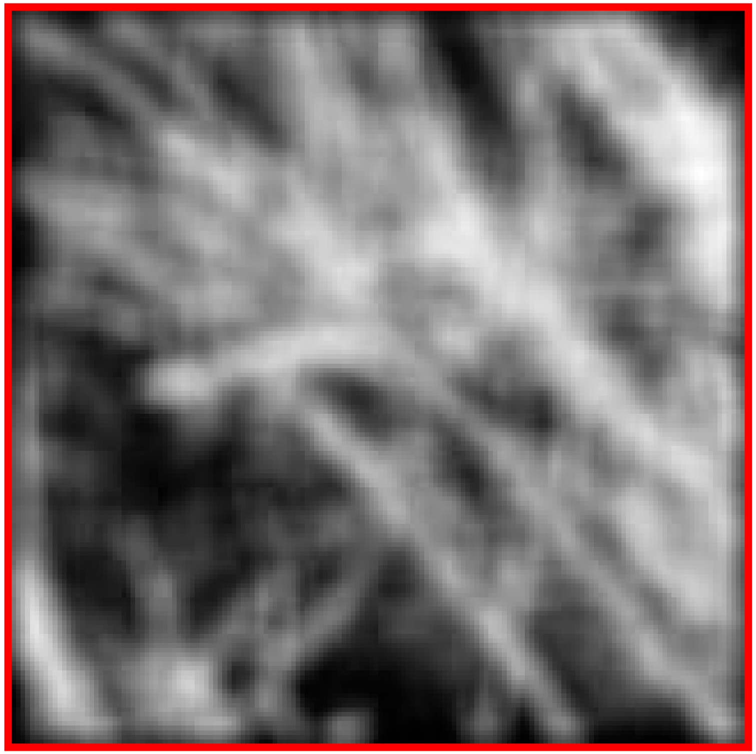




D. Forsyth

Smoothing with not a Gaussian

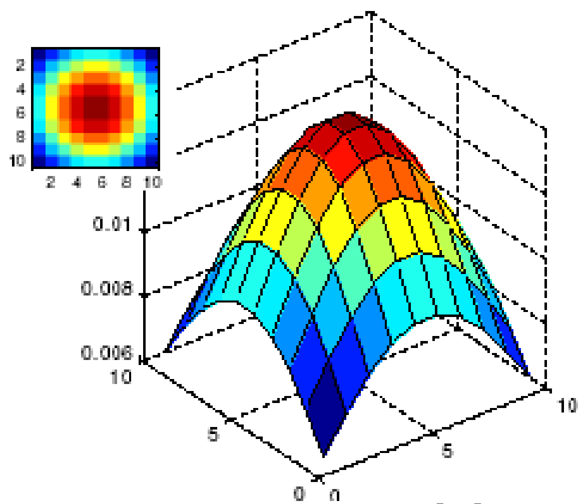




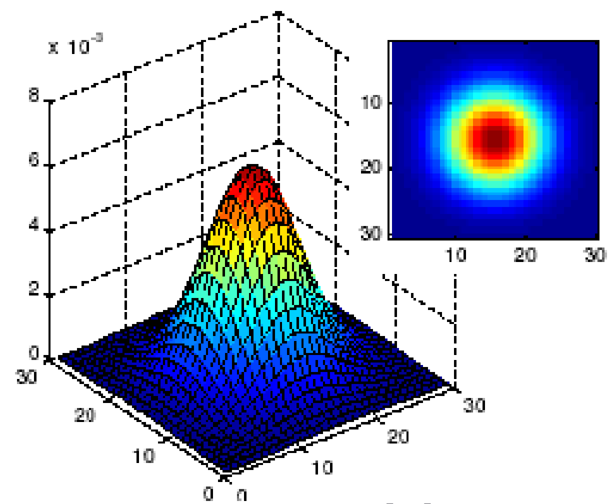
D. Forsyth

Gaussian filters

- What parameters matter here?
- **Size** of kernel or mask
  - Note, Gaussian function has infinite support, but discrete filters use finite kernels



σ = 5 with
10 x 10
kernel

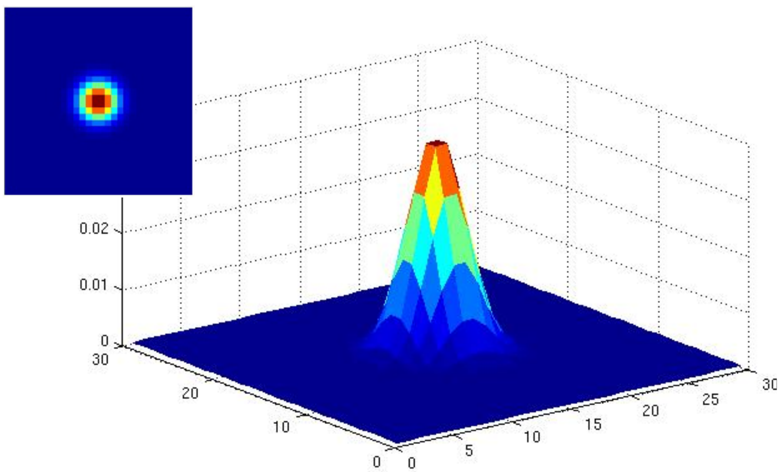


σ = 5 with
30 x 30
kernel

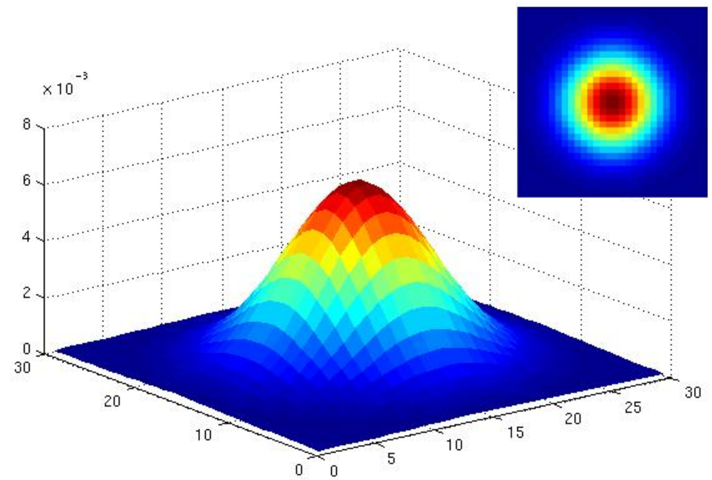
K. Grauman

Gaussian filters

- What parameters matter here?
- **Variance** of Gaussian: determines extent of smoothing



σ = 2 with
30 x 30
kernel



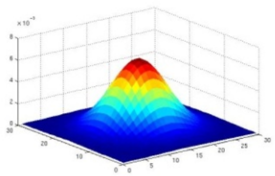
σ = 5 with
30 x 30
kernel

K. Grauman

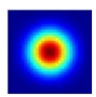
Matlab

```
>> hsize = 10;
>> sigma = 5;
>> h = fspecial('gaussian', hsize, sigma);
```

```
>> mesh(h);
```



```
>> imagesc(h);
```



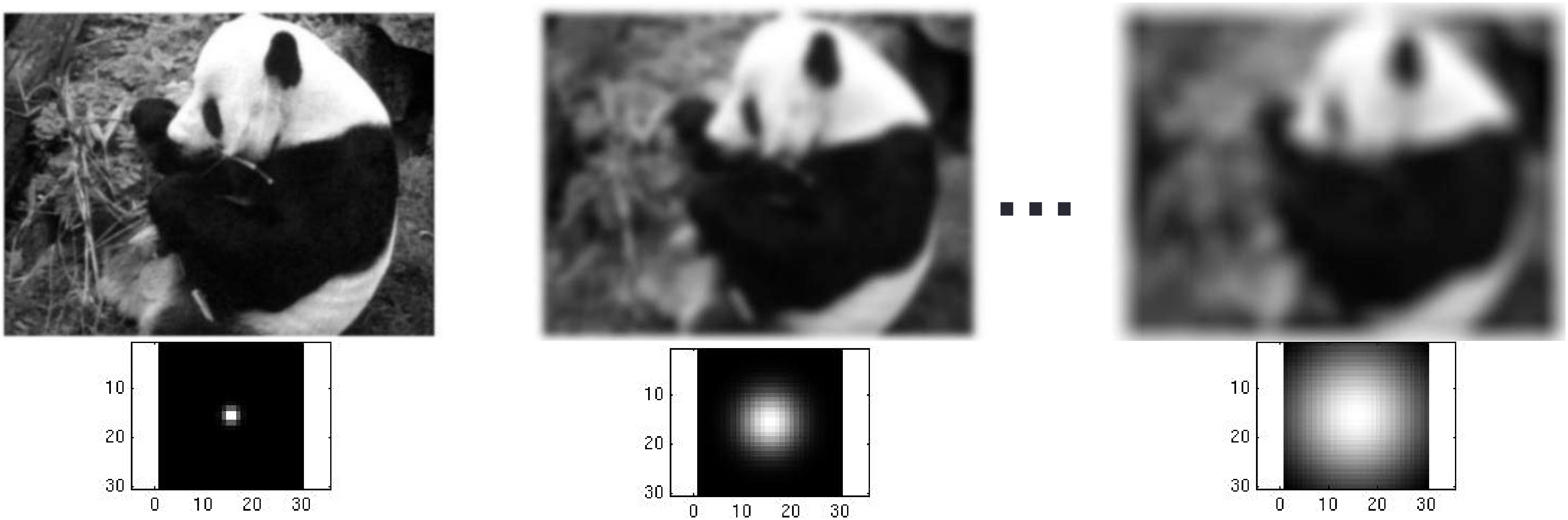
```
>> outim = imfilter(im, h);
>> imshow(outim);
```

outim

K. Grauman

Smoothing with a Gaussian

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

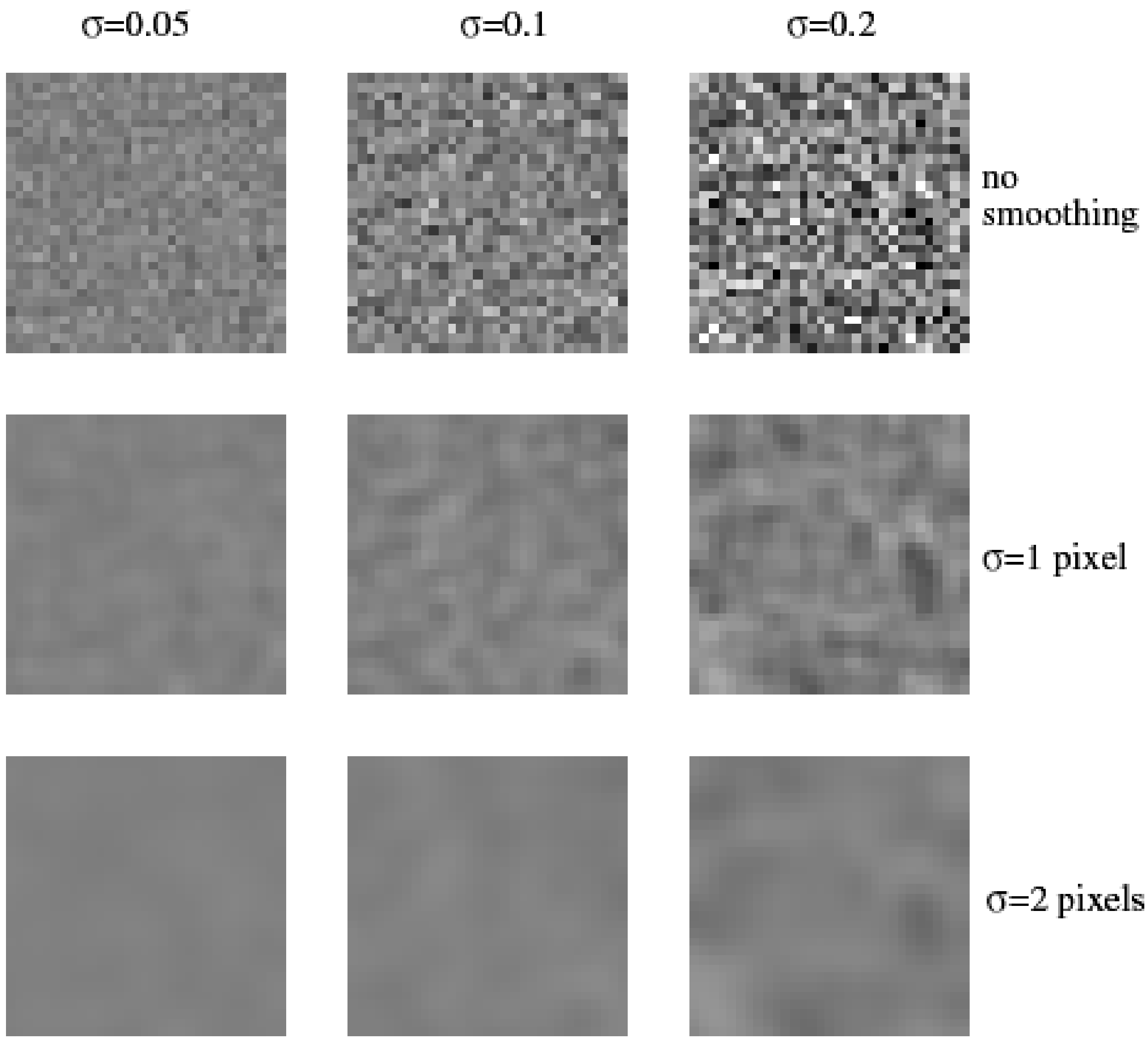



```
for sigma=1:3:10
  h = fspecial('gaussian‘, fsize, sigma);
  out = imfilter(im, h);
  imshow(out);
  pause;
end
```

K. Grauman

Keeping the two Gaussians straight…

More Gaussian noise (like earlier) σ →



Wider Gaussian smoothing kernel σ →

And now some linear intuition…

An operator H (or system) is *linear* if two properties hold ($f1$ and $f2$ are some functions, a is a constant):

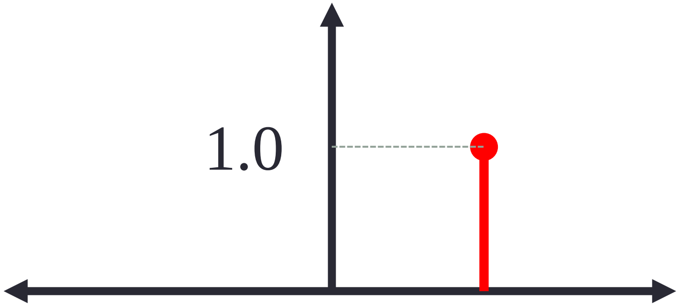
- Additivity (things sum) (superposition):
  $H(f1 + f2) = H(f1) + H(f2)$ *(looks like distributive law)*

- Multiplicative scaling (Homogeneity of degree 1)
  $H(a \cdot f1) = a \cdot H(f1)$

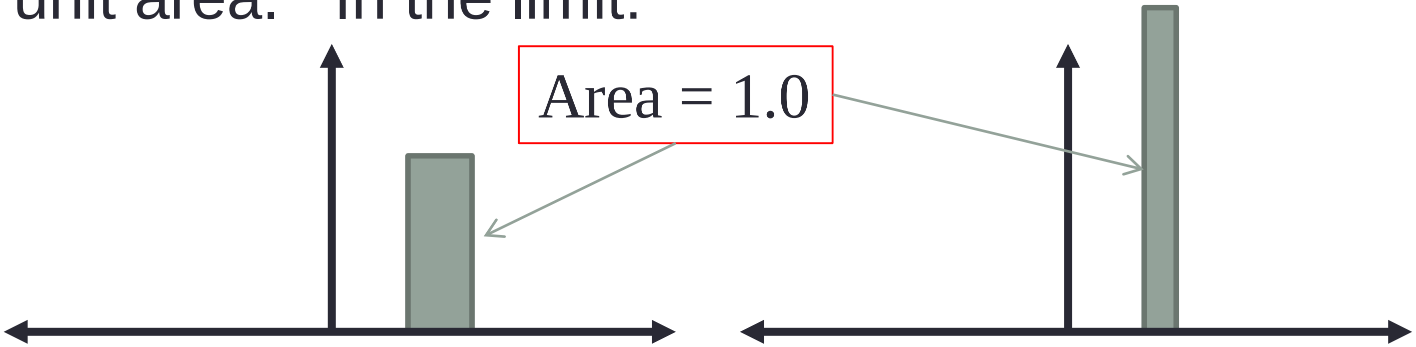
Because it is sums and multiplies, the "filtering" operation we were doing are linear.

An impulse function…

- In the discrete world, and *impulse* is a very easy signal to understand: it's just a value of 1 at a single location.



- In the continuous world, an *impulse* is an idealized function that is very narrow and very tall so that it has a unit area. In the limit:



An impulse response

- If I have an unknown system and I “put in” an impulse, the response is called the impulse response. (Duh?)

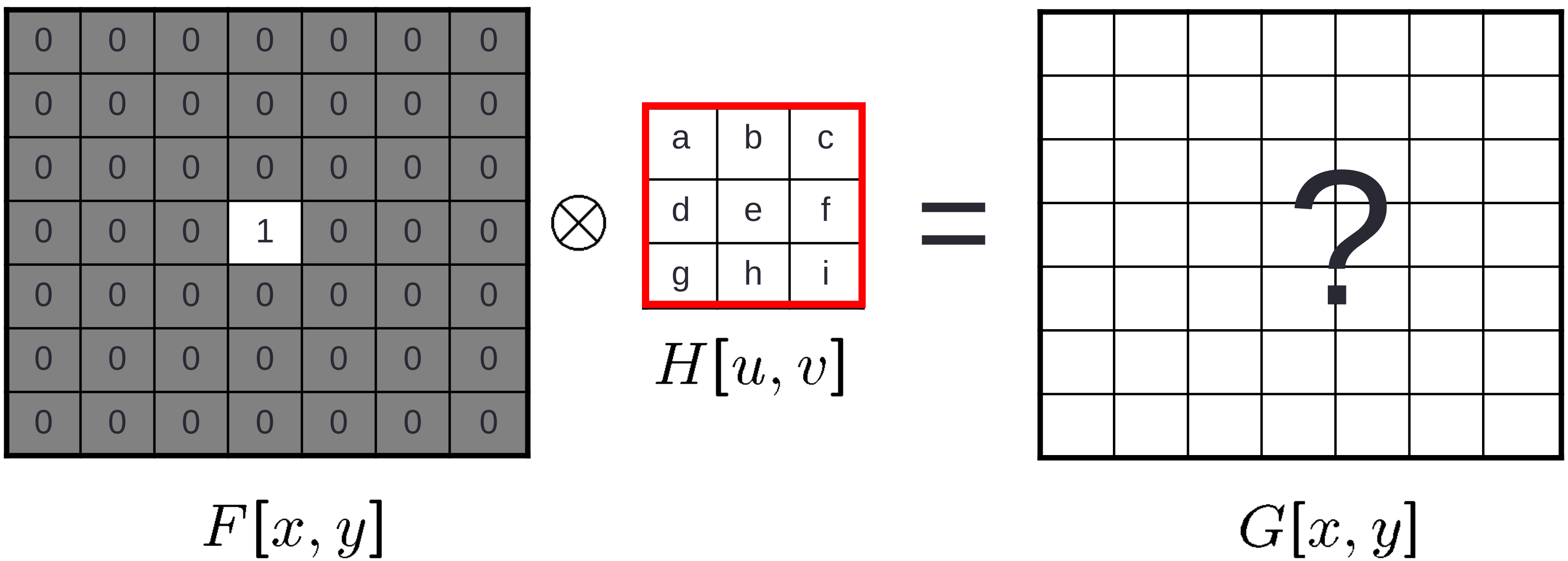
(or $h(t)$)

H

- So if the black box is linear you can describe H by $h(x)$. Why?

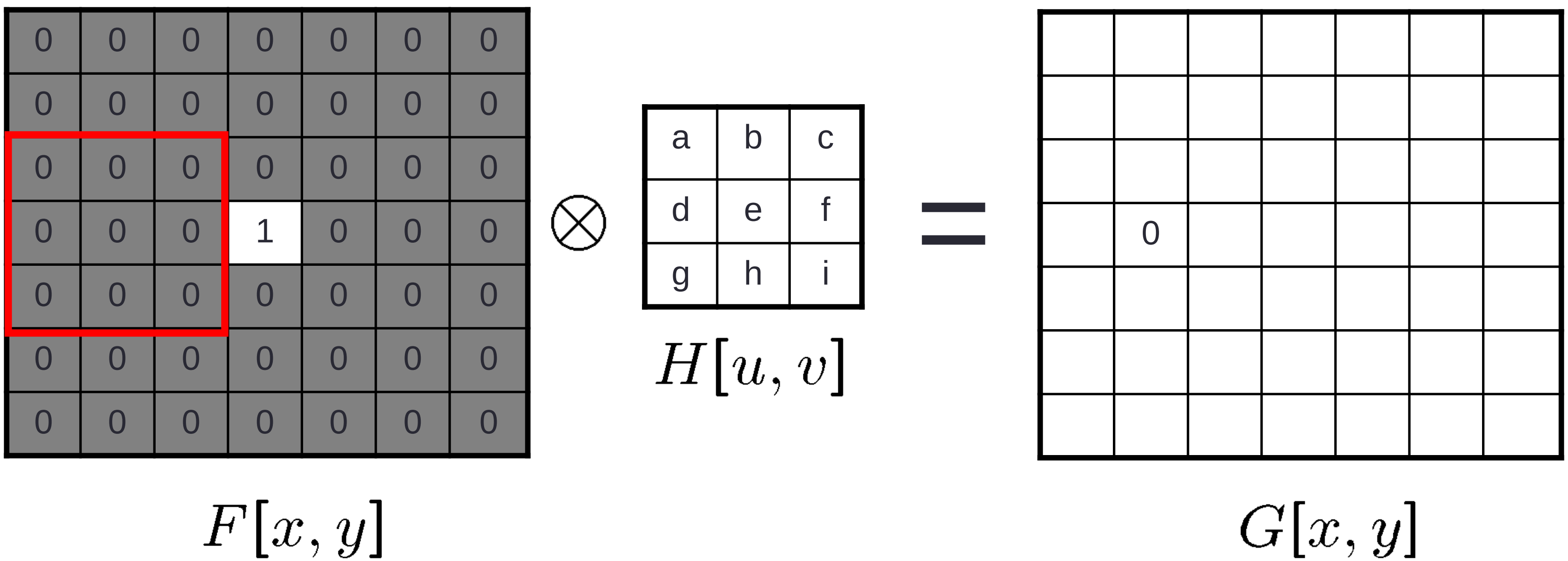
Filtering an impulse signal

What is the result of filtering the impulse signal (image) *F* with the arbitrary kernel *H*?

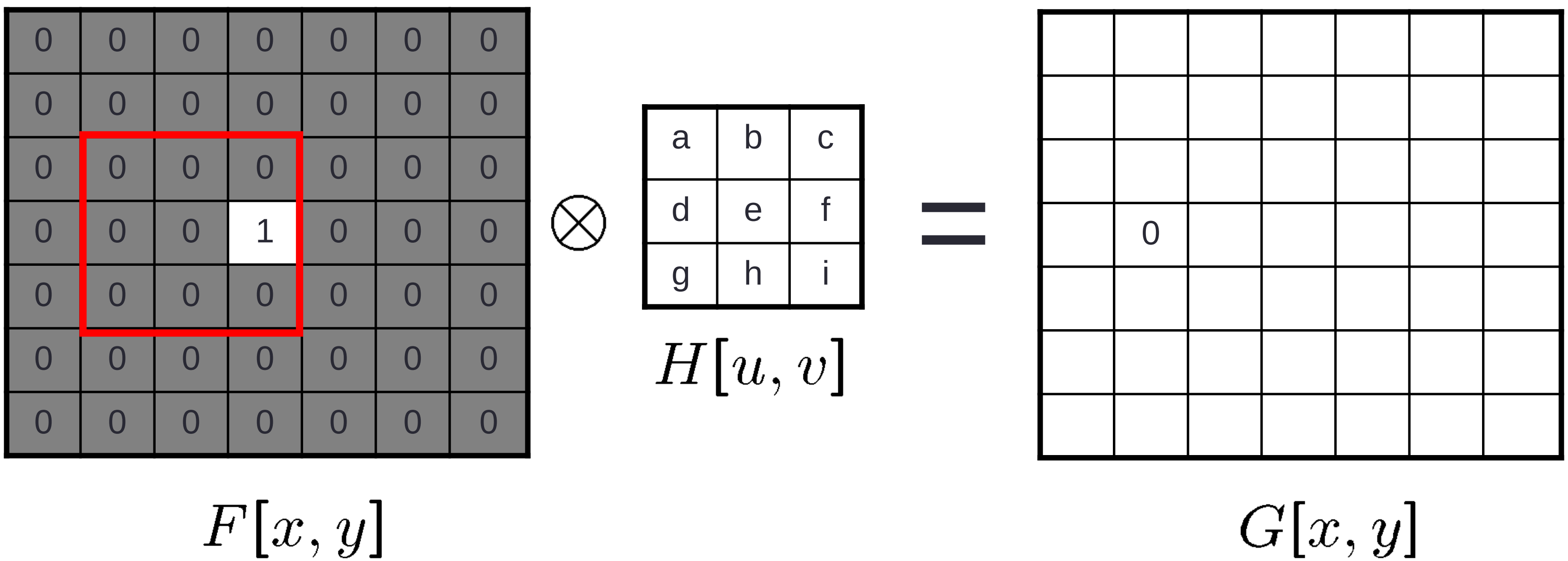


K. Grauman

Filtering an impulse signal



Filtering an impulse signal



Filtering an impulse signal

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$F[x, y]$

$\otimes$

a	b	c
d	e	f
g	h	i

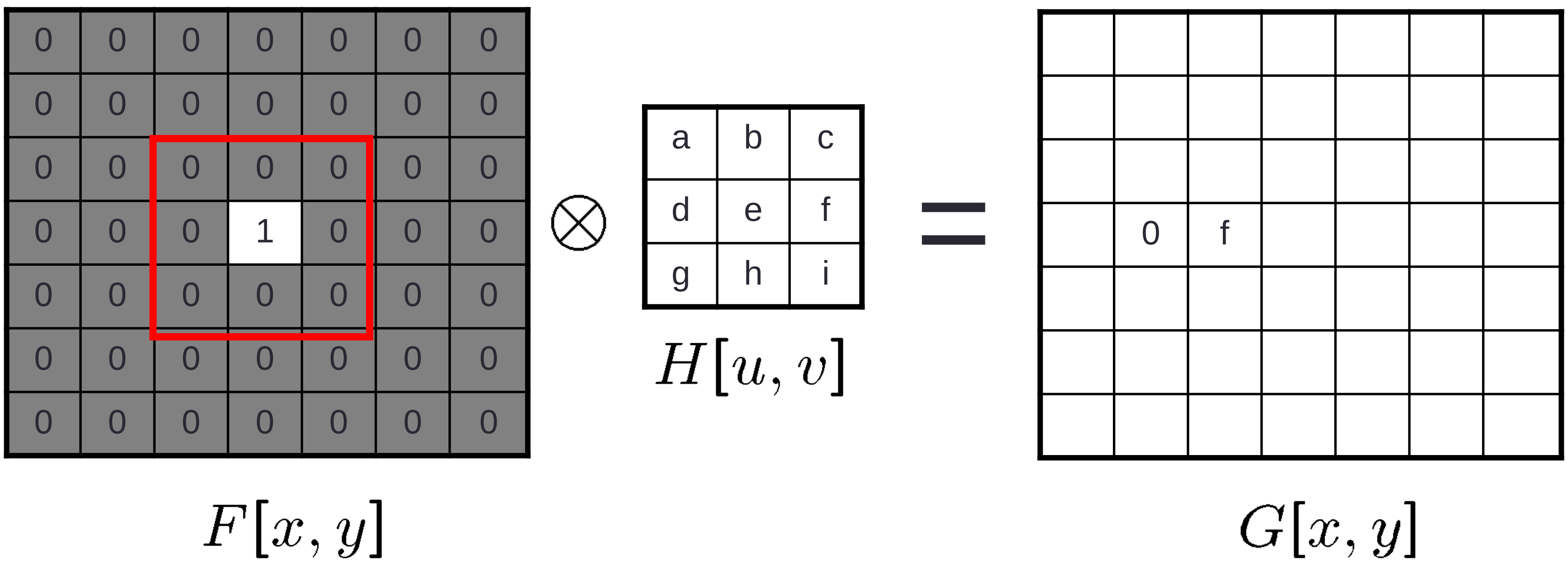
$H[u, v]$

$=$

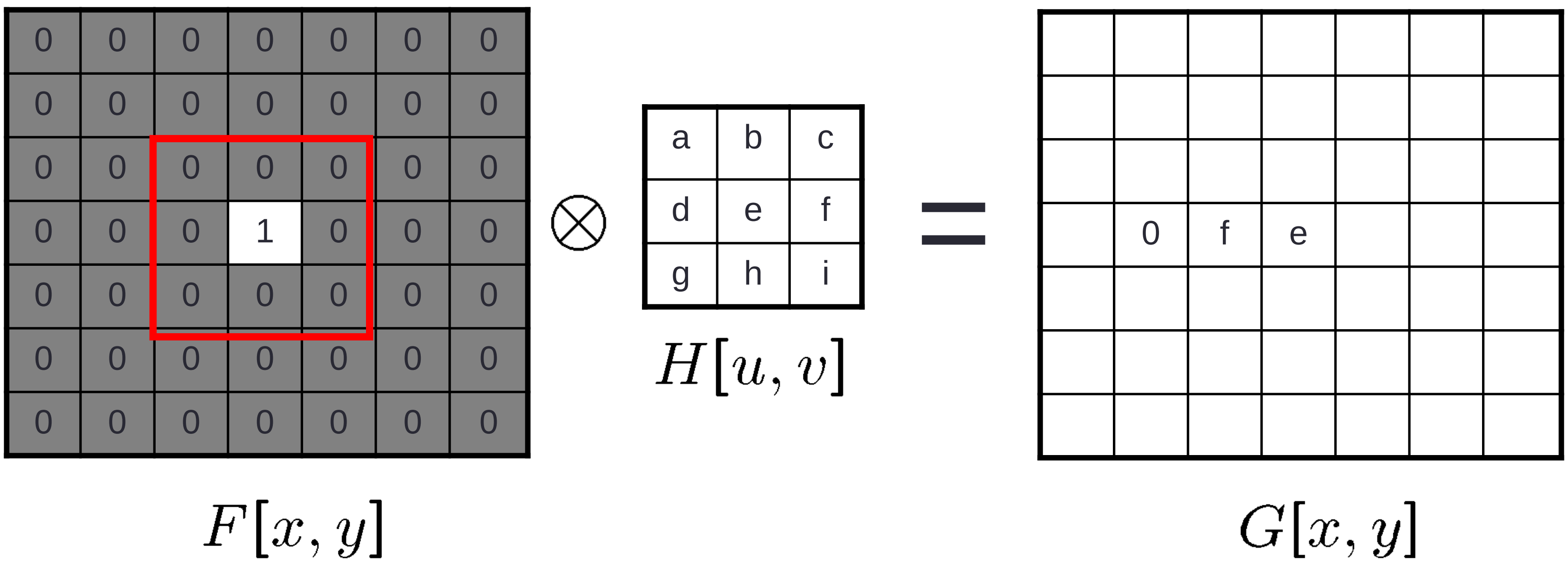
	0	f				

$G[x, y]$

Filtering an impulse signal



Filtering an impulse signal



Filtering an impulse signal

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$F[x, y]$

$\otimes$

a	b	c
d	e	f
g	h	i

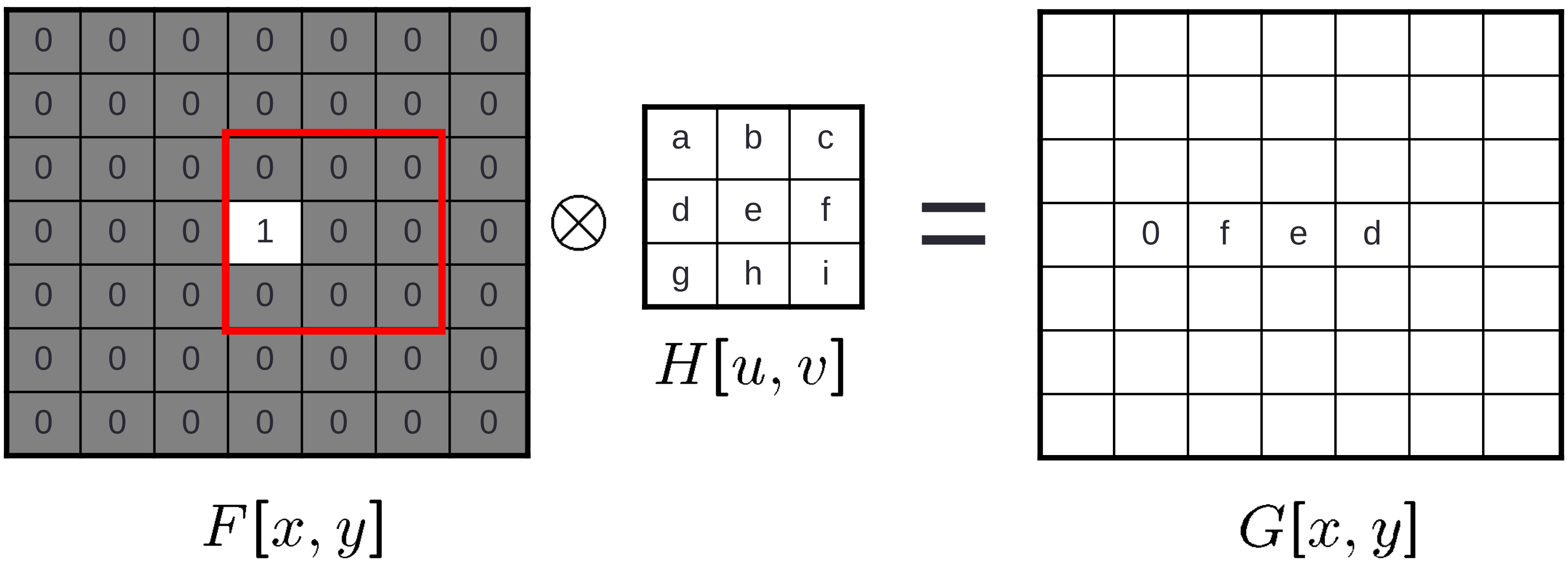
$H[u, v]$

$=$

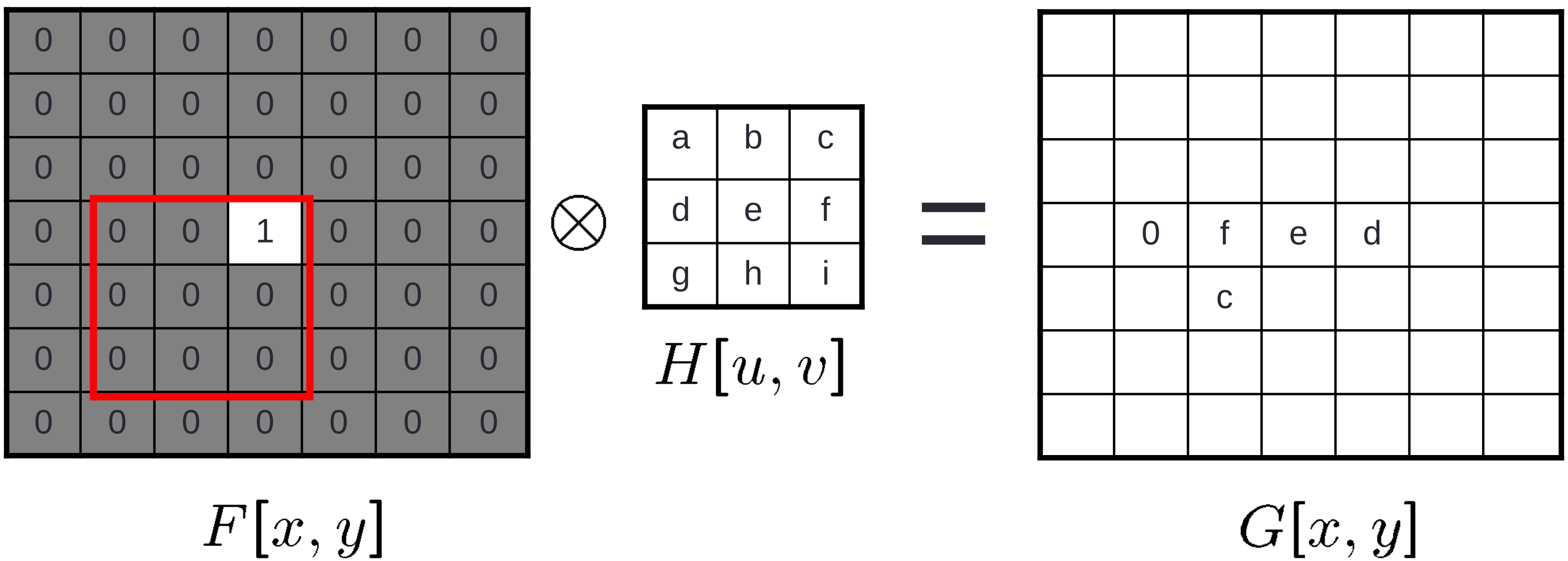
	0	f	e	d		

$G[x, y]$

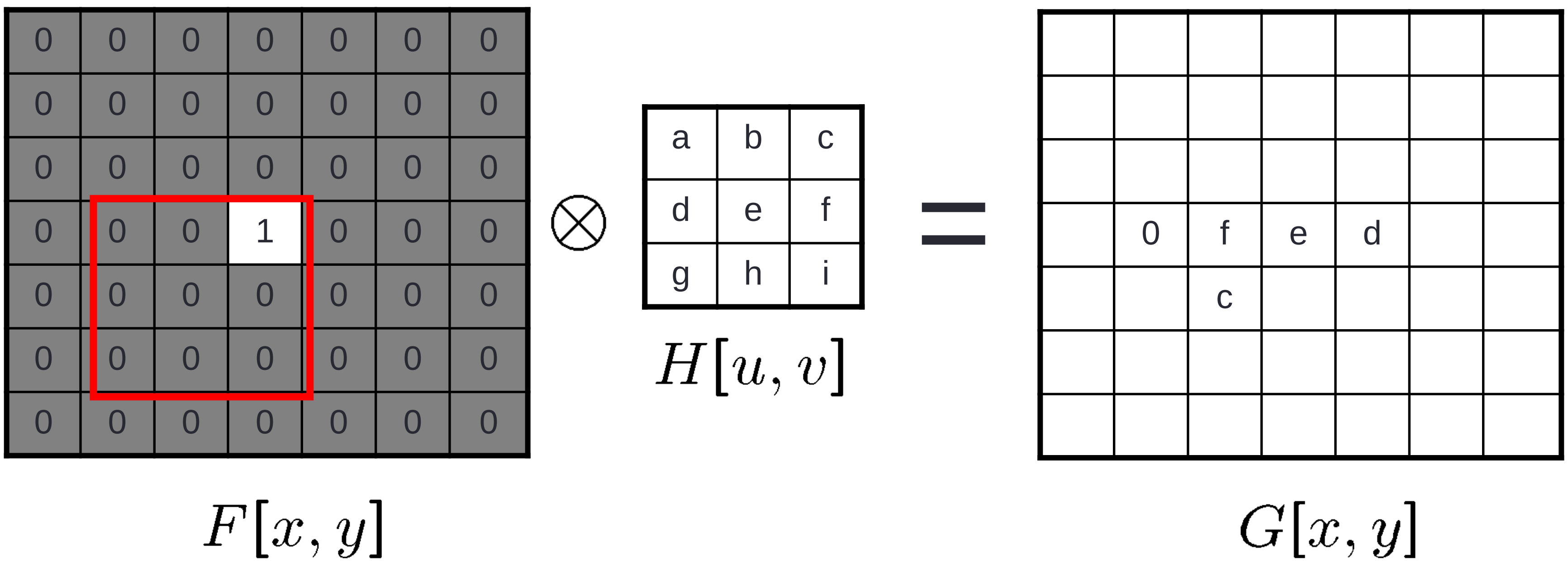
Filtering an impulse signal



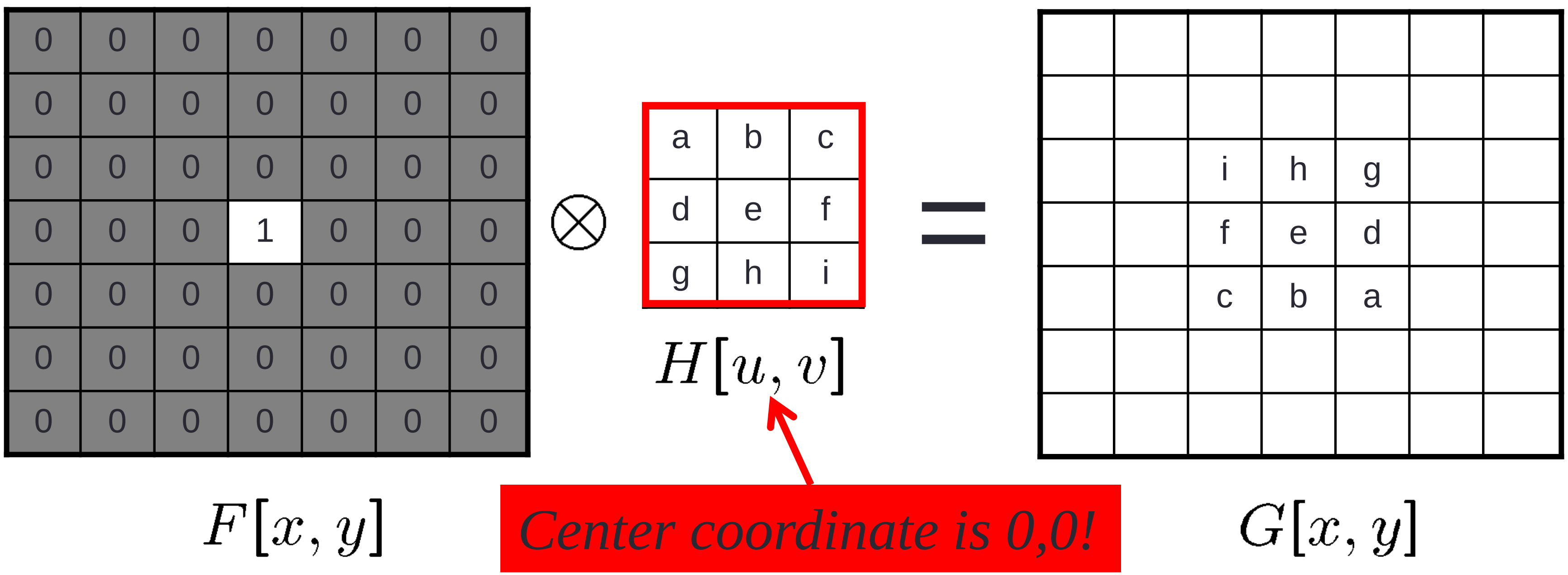
Filtering an impulse signal



Filtering an impulse signal



"Filtering" an impulse signal



If you just "filter" meaning slide the kernel over the image you get a ***reversed*** response.

Convolution

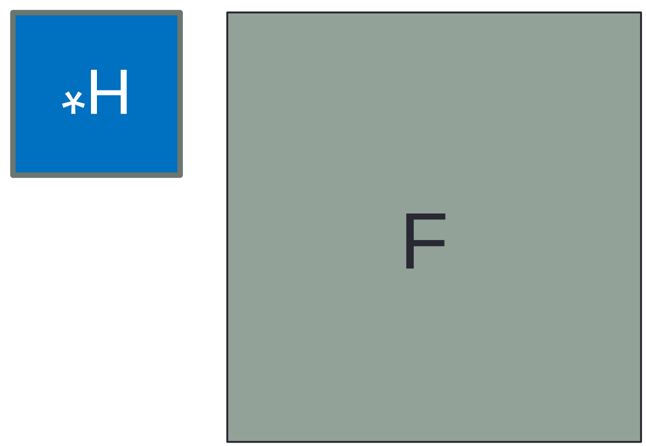
- Convolution:
  - Flip where the filter is applied in both dimensions (bottom to top, right to left)
  - Then apply cross-correlation

Centered at zero!

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v] F[i-u, j-v]$$

$$G = H * F$$

Notation for convolution operator



K. Grauman

One more thing…

- **Shift invariant:**
  - Operator behaves the same everywhere, i.e. the value of the output depends on the pattern in the image neighborhood, not the position of the neighborhood.

Properties of convolution

- Linear & shift invariant
- Commutative:

  f * g = g * f
- Associative

  (f * g) * h = f * (g * h)
- Identity:

  unit impulse e = […, 0, 0, 1, 0, 0, …]. f * e = f
- Differentiation:

$$\frac{\partial}{\partial x}(f * g) = \frac{\partial f}{\partial x} * g$$

We'll use this later!

Convolution vs. correlation

Convolution

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u,j-v]$$

$$G = H \star F$$

Cross-correlation

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u,j+v]$$

$$G = H \otimes F$$

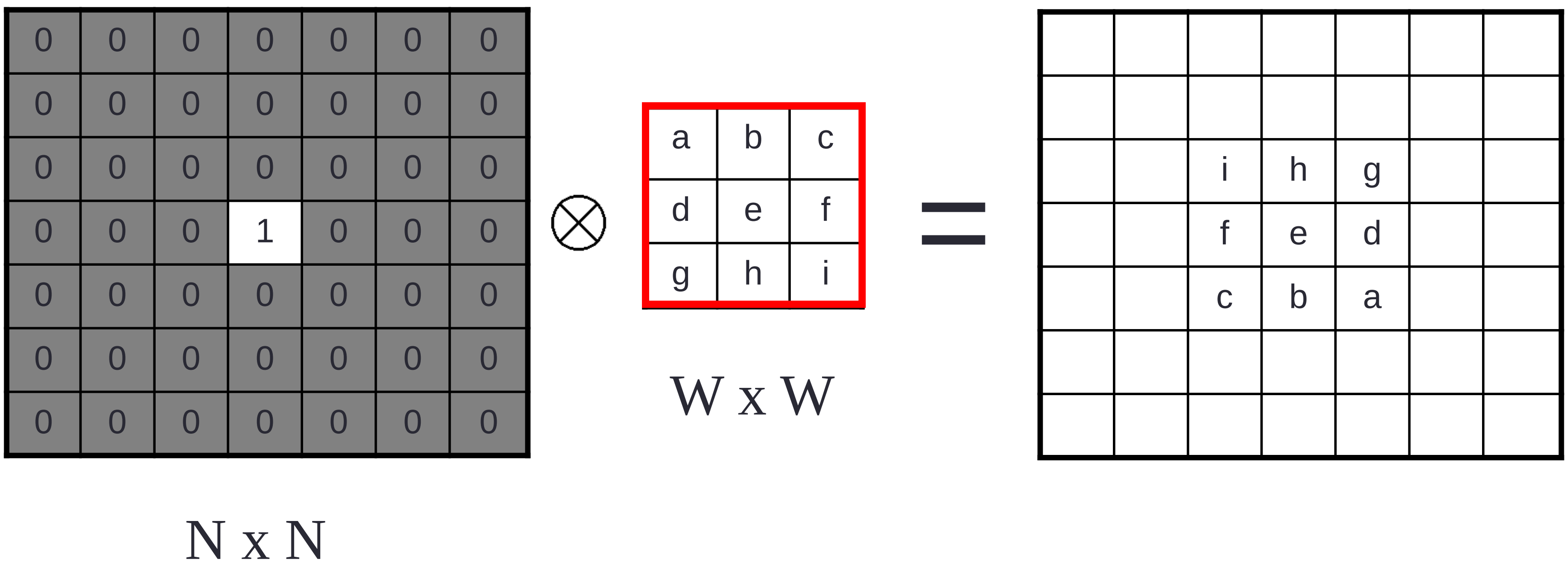
For a Gaussian or box filter, how will the outputs differ?

If the input is an impulse signal, how will the outputs differ?

K. Grauman

Computational Complexity

- If an image is *NxN* and a kernel (filter) is WxW, how many multiplies do you need to compute a convolution?



- You need N*N*W*W = N^2W^2
  - which can get big (ish)

Separability

- Now we're going to take advantage of the associative property of convolution.
- In some cases, filter is separable, meaning you can get the square kernel H by convolving a single column vector by some row vector:

$$c\;\begin{bmatrix}1\\2\\1\end{bmatrix} \times \overset{r}{\begin{bmatrix}1 & 2 & 1\end{bmatrix}} = \begin{bmatrix}1 & 2 & 1\\2 & 4 & 2\\1 & 2 & 1\end{bmatrix}\;H$$

Separability

r

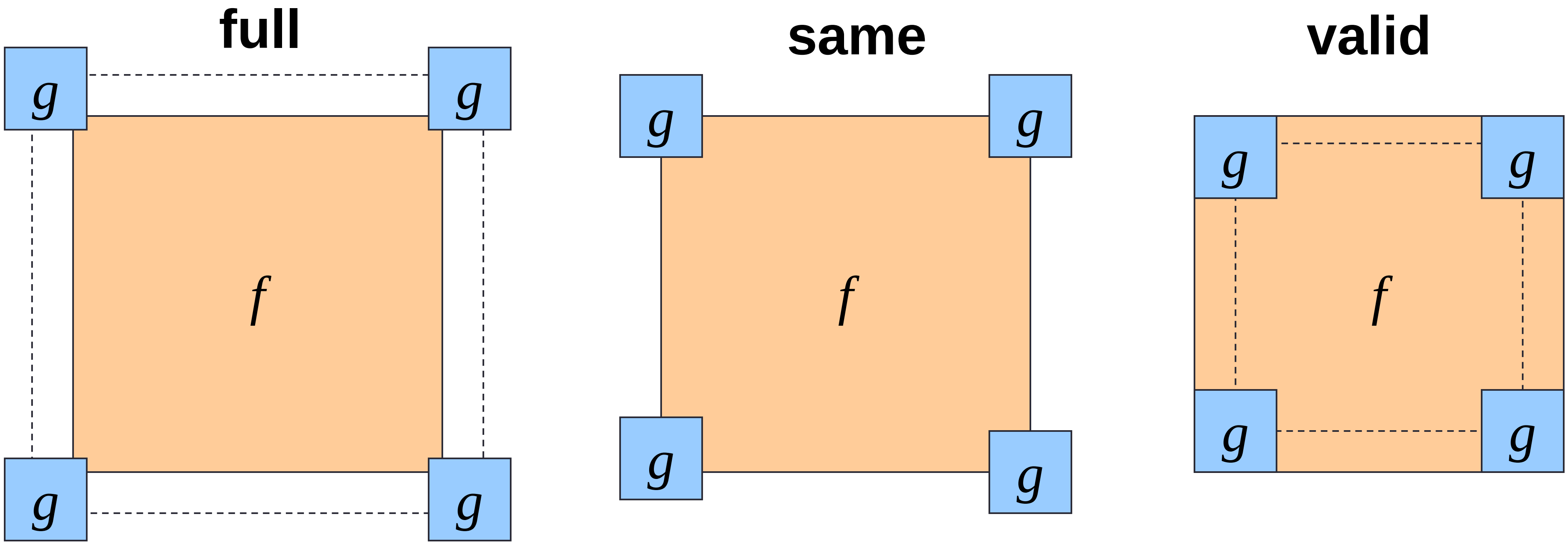
$$\mathbf{c} \;\; \begin{bmatrix}1\\2\\1\end{bmatrix} \times \begin{bmatrix}1 & 2 & 1\end{bmatrix} = \begin{bmatrix}1 & 2 & 1\\2 & 4 & 2\\1 & 2 & 1\end{bmatrix} \;\; \mathbf{H}$$

$$G = H * F = (C * R) * F = C * (R * F)$$

- So we do two convolutions but each is W*N*N. So this is useful if W is big enough such that $2WN^2 \ll W^2N^2$
- Used to be ***very*** important. Still, if W=31, save a factor of 15.

Boundary issues

- What is the size of the output?
- Old MATLAB: filter2(g, f, *shape*)
    - *shape* = 'full': output size is sum of sizes of f and g
    - *shape* = 'same': output size is same as f
    - *shape* = 'valid': output size is difference of sizes of f and g



Source: S. Lazebnik

Boundary issues

- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:
    - **clip filter (black)**

Source: S. Marschner

Boundary issues

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

Source: S. Marschner

Boundary issues

- 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**

Source: S. Marschner

Boundary issues

- 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**

Source: S. Marschner

Boundary issues

- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods (new MATLAB):
    - clip filter (black): imfilter(f, g, 0)
    - wrap around: imfilter(f, g, ‘circular’)
    - copy edge: imfilter(f, g, ‘replicate’)
    - reflect across edge: imfilter(f, g, ‘symmetric’)

Source: S. Marschner

Predict the filtered outputs

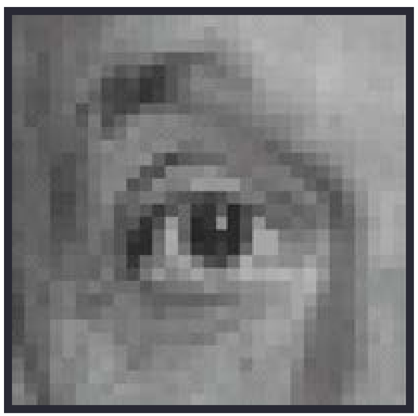
$$* \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = ?$$

$$* \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} = ?$$

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

K. Grauman

Practice with linear filters



Original

0	0	0
0	1	0
0	0	0

?

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	1	0
0	0	0

Filtered (no change)

Source: D. Lowe

Practice with linear filters

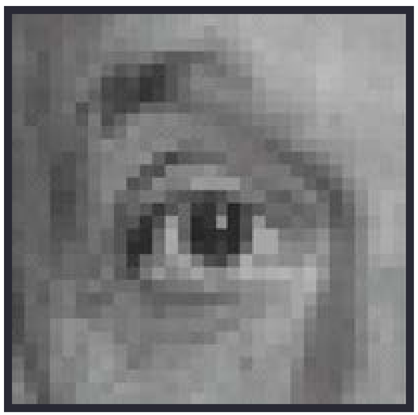
Original

0	0	0
0	0	1
0	0	0

?

Source: D. Lowe

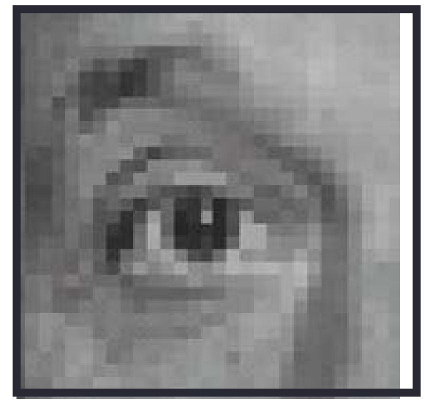
Practice with linear filters



Original

0	0	0
0	0	1
0	0	0

Center coordinate is 0,0!



Shifted left by 1 pixel with correlation

Source: D. Lowe

Practice with linear filters

Original

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

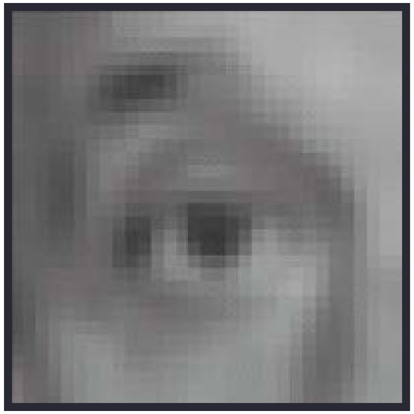
?

Source: D. Lowe

Practice with linear filters

Original

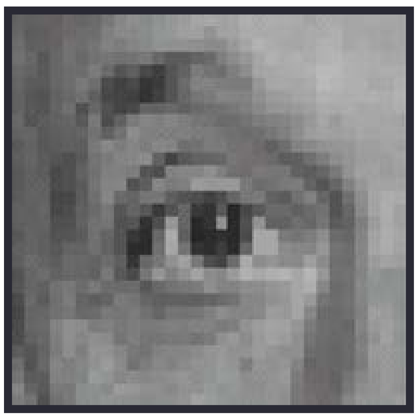
$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$



Blur (with a box filter)

Source: D. Lowe

Practice with linear filters



Original

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

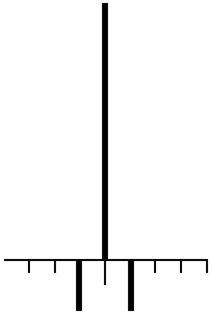
?

Source: D. Lowe

Practice with linear filters

Original

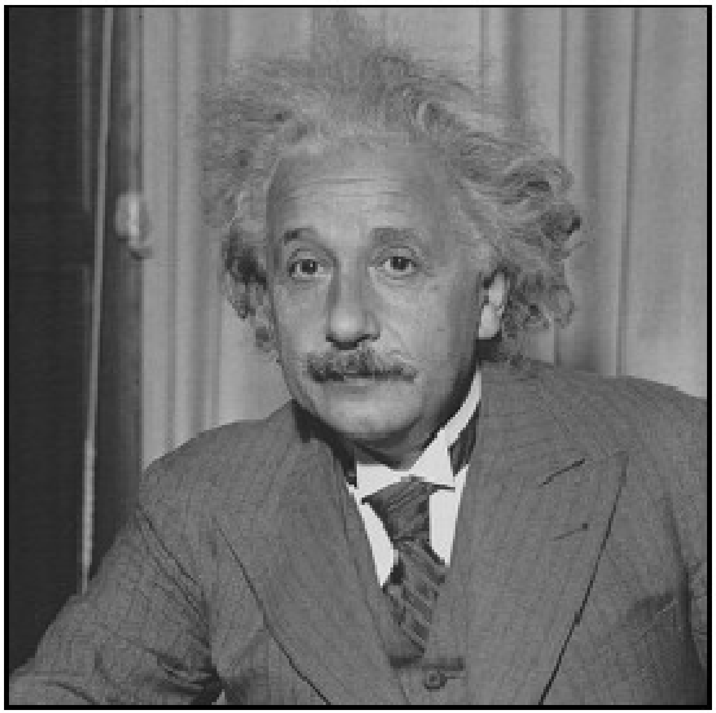
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



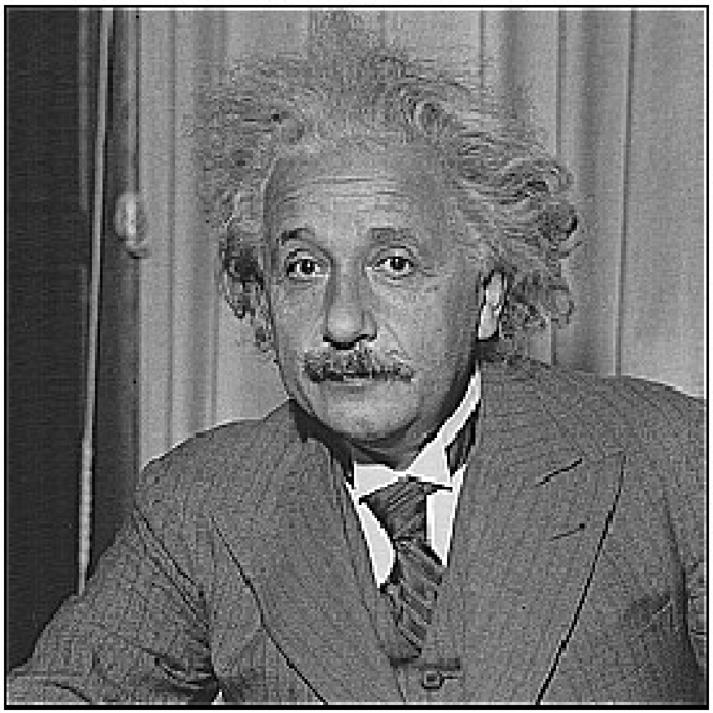
Sharpening filter
- Accentuates differences with local average

Source: D. Lowe

Filtering examples: sharpening



before



after

K. Grauman

Effect of smoothing filters

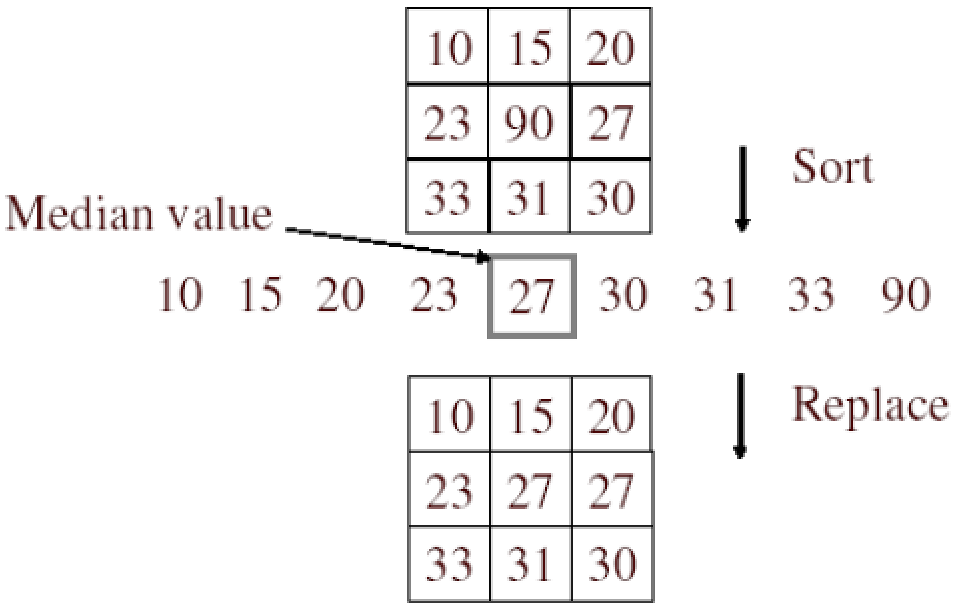
5x5

Additive Gaussian noise

Salt and pepper noise

K. Grauman

Median filter

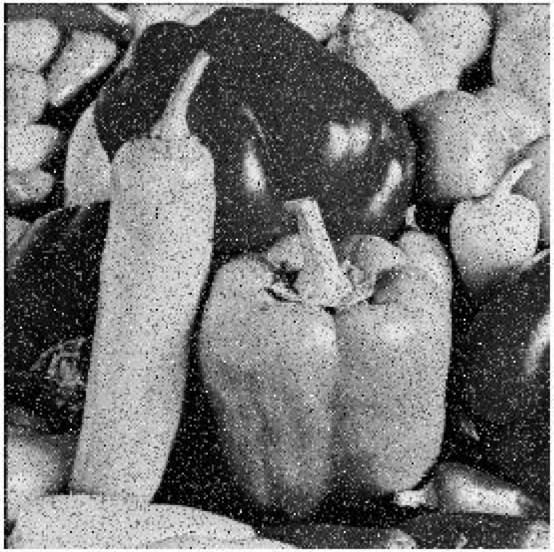


- No new pixel values introduced
- Removes spikes: good for impulse, salt & pepper noise
- Linear?

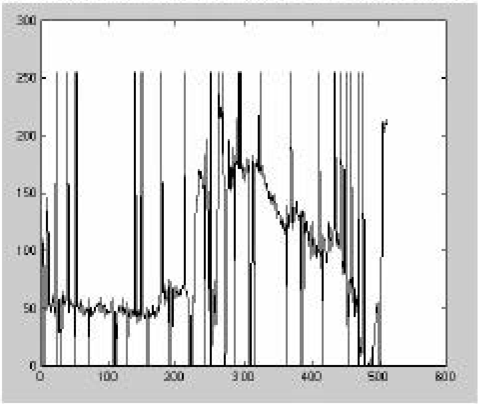
K. Grauman

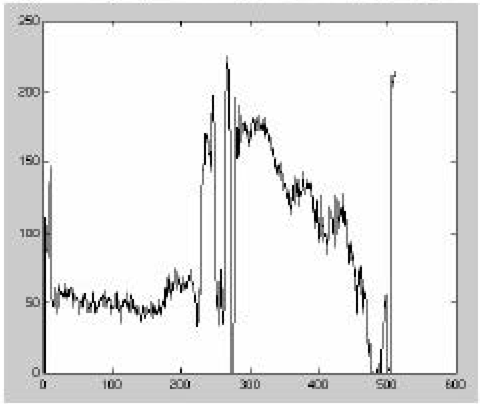
Median filter

Salt and pepper noise



Median filtered



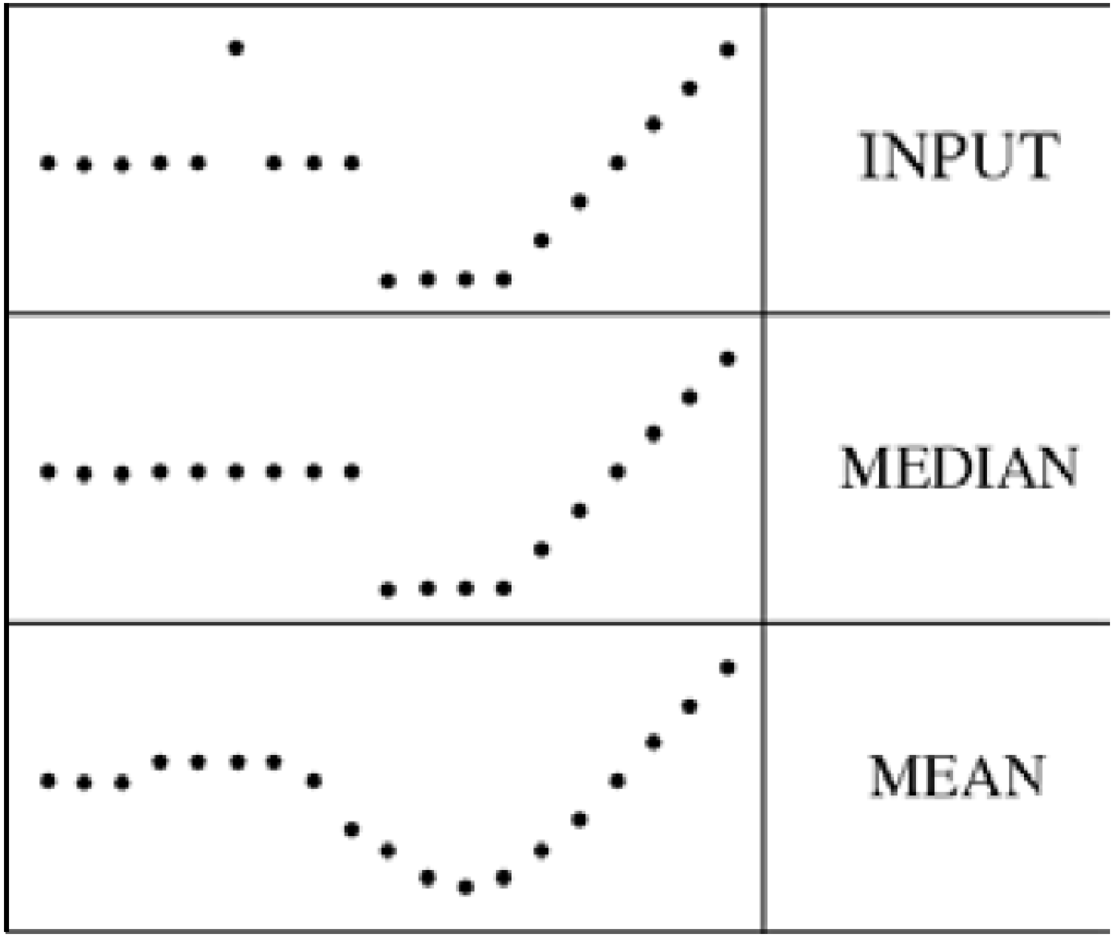


Plots of a row of the image

Source: M. Hebert

Median filter

- Median filter is edge preserving



K. Grauman

To do:

- Problem set 0 available; due 11:59pm Thurs Aug 29th
- Problem set 1 – Filtering, Edges, Hough – will be handed out Aug 28th (Thurs) and is due Sun Sept 7, 11:59pm.