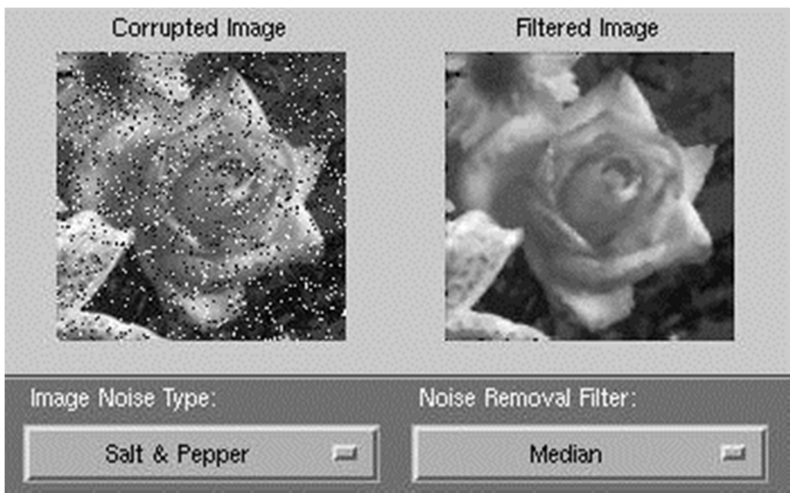
Image Processing



Edge detection

Filtering: Noise suppresion

Image filtering

Linear filtering = Applying a local function to the image using a sum of weighted neighboring pixels.

Such as blurring:

Input image

0	0	0	0	0	0
0	0	0	0	0	0
0	0	200	0	0	0
0	0	0	0	0	0
0	0	0	100	0	0
0	0	0	0	0	0

*

Kernel

0	0	0	0	0
0	0.11	0.11	0.11	0
0	0.11	0.11	0.11	0
0	0.11	0.11	0.11	0
0	0	0	0	0

=

Output image

0	0	0	0	0	0
0	22	22	22	0	0
0	22	22	22	0	0
0	22	33	33	11	0
0	0	11	11	11	0
0	0	11	11	11	0

Image filtering

$$g(x,y) = \sum_{x'} \sum_{y'} f(x+x', y+y')h(x', y')$$

0	0	0	0	0	0
0	0	0	0	0	0
0	0	200	0	0	0
0	0	0	0	0	0
0	0	0	100	0	0
0	0	0	0	0	0

Input image f

$*$

Mean filter

0	0	0	0	0
0	0.11	0.11	0.11	0
0	0.11	0.11	0.11	0
0	0.11	0.11	0.11	0
0	0	0	0	0

Filter h

$=$

0	0	0	0	0	0
0	22	22	22	0	0
0	22	22	22	0	0
0	22	33	33	11	0
0	0	11	11	11	0
0	0	11	11	11	0

Output image g

Image filtering

- Linear filters can have arbitrary weights.
- Typically they sum to 0 or 1, but not always.
- Weights may be positive or negative.
- Many filters aren't linear (median filter.)

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

What does this filter do?

Animation of Convolution

To accomplish convolution of the whole image, we just ***Slide the mask***

$p_{1,1}$	$p_{1,2}$	$p_{1,3}$	$p_{1,4}$	$p_{1,5}$	$p_{1,6}$
$p_{2,1}$	$p_{2,2}$	$p_{2,3}$	$p_{2,4}$	$p_{2,5}$	$p_{2,6}$
$p_{3,1}$	$p_{3,2}$	$p_{3,3}$	$p_{3,4}$	$p_{3,5}$	$p_{3,6}$
$p_{4,1}$	$p_{4,2}$	$p_{4,3}$	$p_{4,4}$	$p_{4,5}$	$p_{4,6}$
$p_{5,1}$	$p_{5,2}$	$p_{5,3}$	$p_{5,4}$	$p_{5,5}$	$p_{5,6}$
$p_{6,1}$	$p_{6,2}$	$p_{6,3}$	$p_{6,4}$	$p_{6,5}$	$p_{6,6}$

Original Image

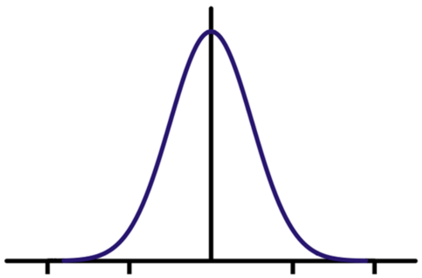
m1,1	m1,2	m1,3
m2,1	m2,2	m2,3
m3,1	m3,2	m3,3

Mask

	$C_{4,2}$	$C_{4,3}$	$C_{4,4}$		

Image after convolution

Gaussian filter

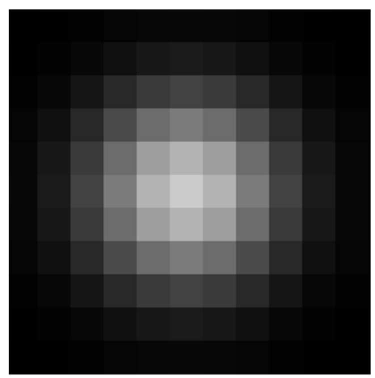


$$\mathcal{G}_\sigma(x,y) = \frac{1}{Z} e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

Compute empirically

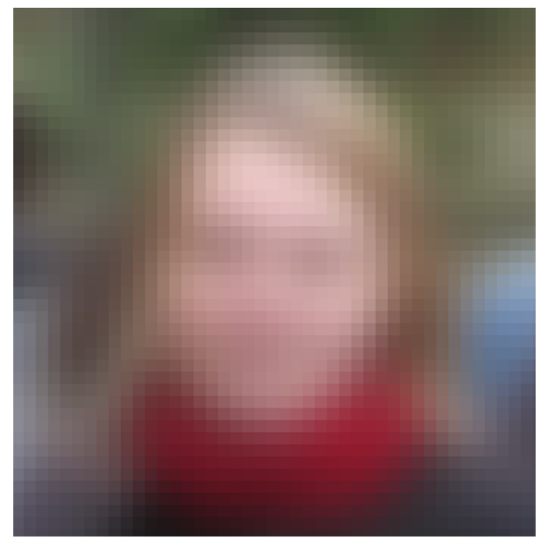
Input image f

$*$



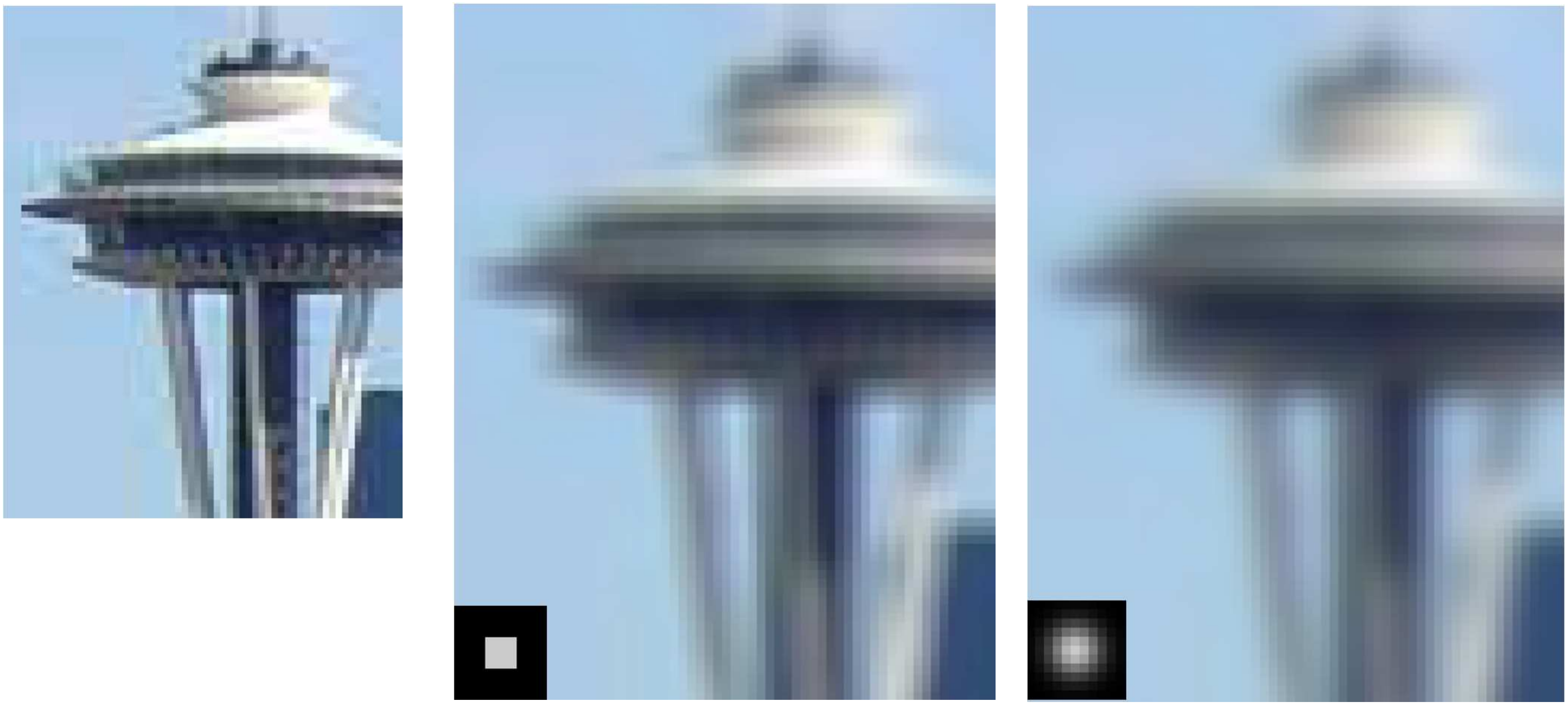
Filter h

$=$



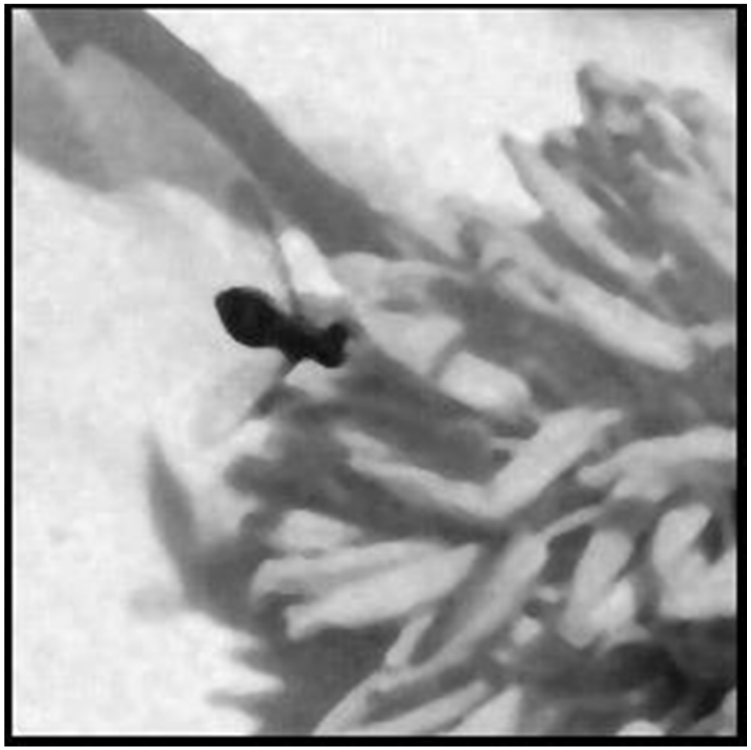
Output image g

Gaussian vs. mean filters



What does real blur look like?

Convolution for Noise Elimination



--Noise Elimination

The noise is eliminated but the operation causes loss of sharp edge definition.

In other words, the image becomes *blurred*

Convolution with a non-linear mask
Median filtering

There are many masks used in *Noise Elimination*

Median Mask is a typical one

The principle of Median Mask is to mask some sub-image, use the median of the values of the sub-image as its value in new image

	J=1	2	3
I=1	23	65	64
2	120	187	90
3	47	209	72

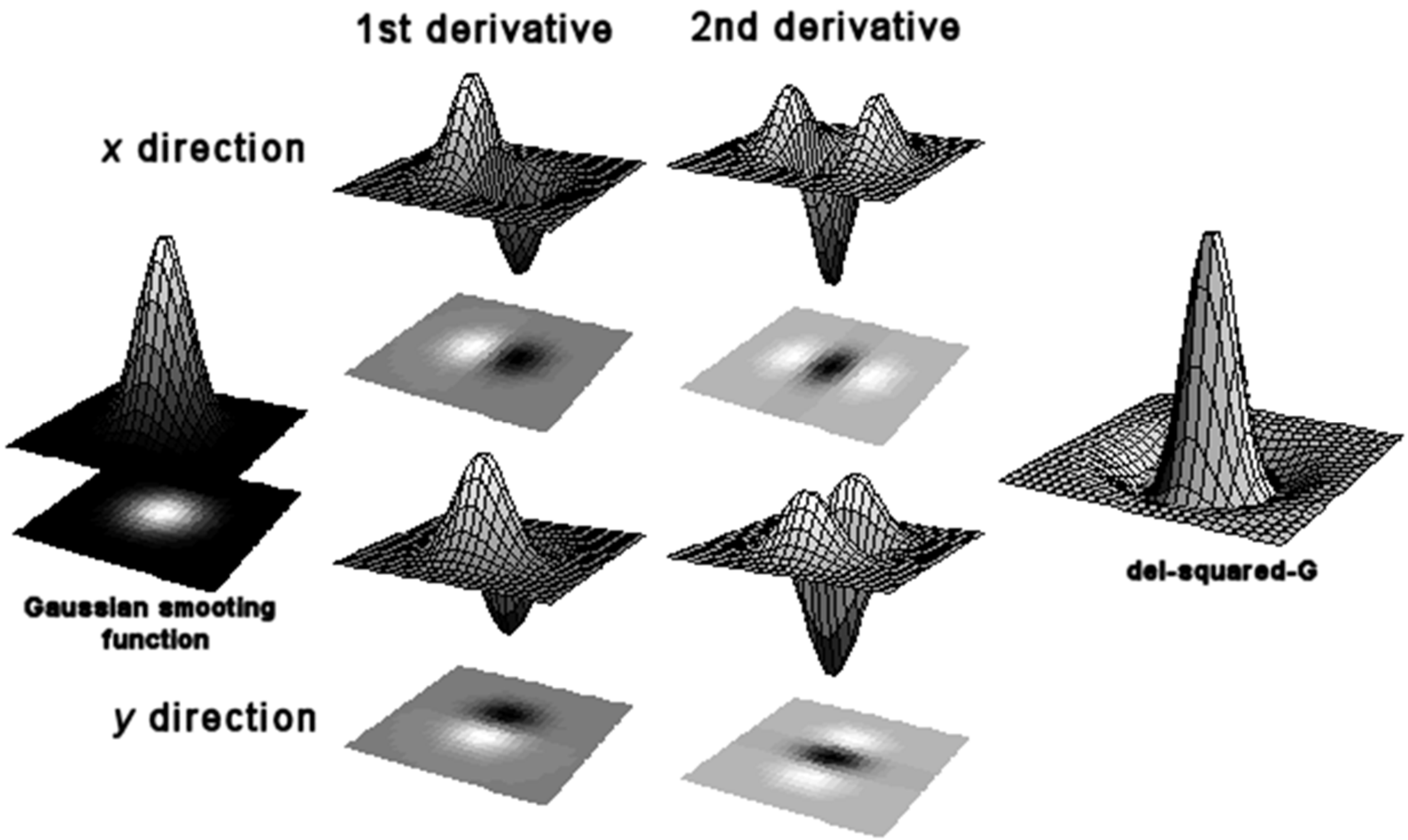
Masked Original Image

Rank: 23, 47, 64, 65, **72**, 90, 120, 187, 209

median

Median Filtering

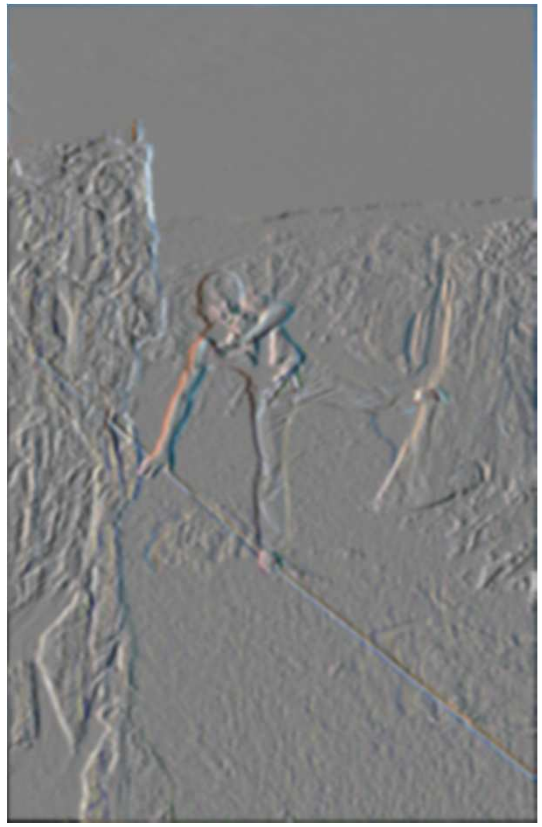
Back to Linear filters:
First and second derivatives



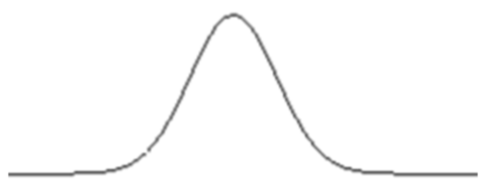
First and second derivatives

What are these good for?

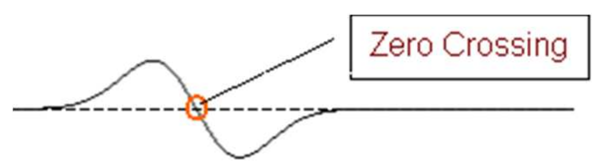
Original



First Derivative x



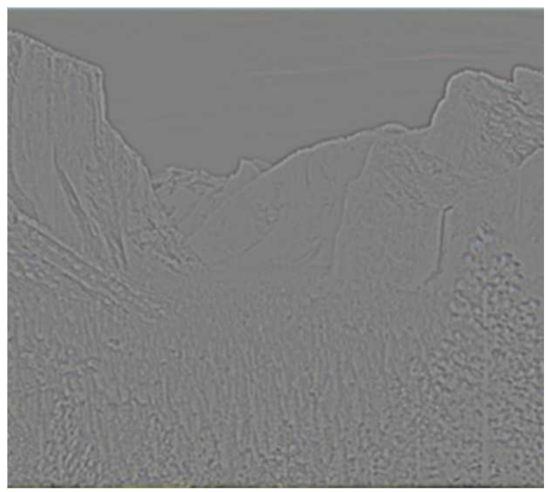
Second Derivative x, y



Subtracting filters

$$Sharpen(x,y) = f(x,y) - \alpha(f * \nabla^2 \mathcal{G}_\sigma(x,y))$$

Original



Second Derivative

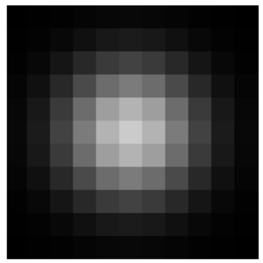
Sharpened

Combining filters

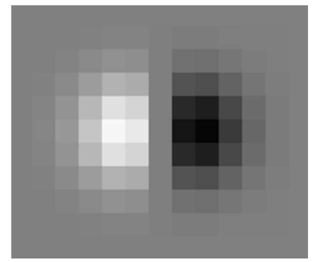
$$f * g * g' = f * h \text{ for some } h$$

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

$*$

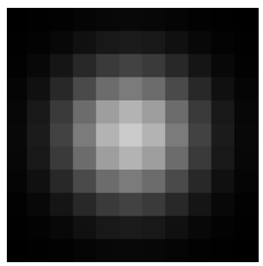


$=$

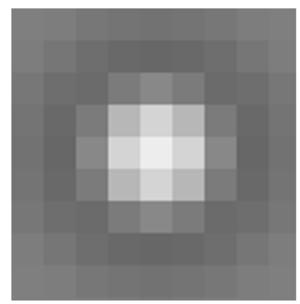


0	0	0	0	0
0	0	-1	0	0
0	-1	4	-1	0
0	0	-1	0	0
0	0	0	0	0

$*$



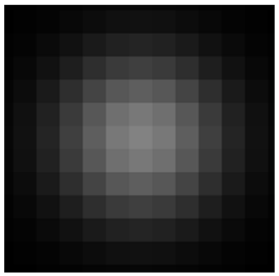
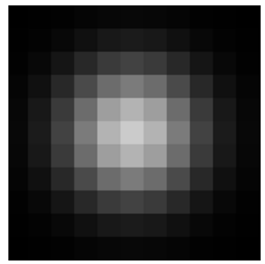
$=$



It's also true: $f * (g * h) = (f * g) * h$

$$f * g = g * f$$

Combining Gaussian filters

 *  = ?

$$f * \mathcal{G}_{\sigma} * \mathcal{G}_{\sigma'} = f * \mathcal{G}_{\sigma''}$$

$$\sigma'' = \sqrt{\sigma^2 + \sigma'^2}$$

More blur than either individually (but less than $\sigma'' = \sigma + \sigma'$)

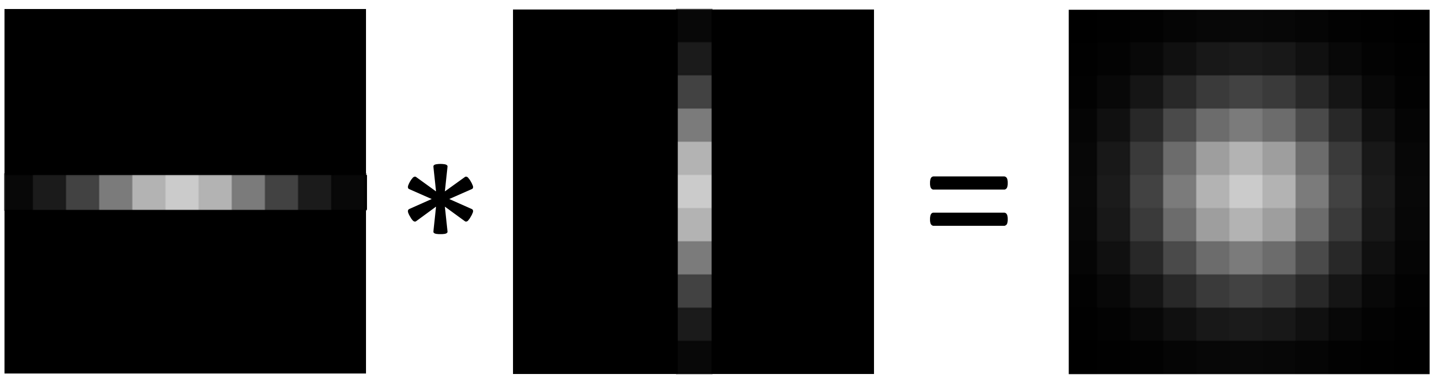
Separable filters

$$\mathcal{G}_\sigma = \mathcal{G}_\sigma^x * \mathcal{G}_\sigma^y$$

$$\mathcal{G}_\sigma^x(x, y) = \frac{1}{Z} e^{\frac{-(x^2)}{2\sigma^2}}$$

$$\mathcal{G}_\sigma^y(x, y) = \frac{1}{Z} e^{\frac{-(y^2)}{2\sigma^2}}$$

Compute Gaussian in horizontal direction, followed by the vertical direction. **Much faster!**



Not all filters are separable.

Freeman and Adelson, 1991

Sums of rectangular regions

How do we compute the sum of the pixels in the red box?

After some pre-computation, this can be done in constant time for any box.

This “trick” is commonly used for computing Haar wavelets (a fundemental building block of many object recognition approaches.)

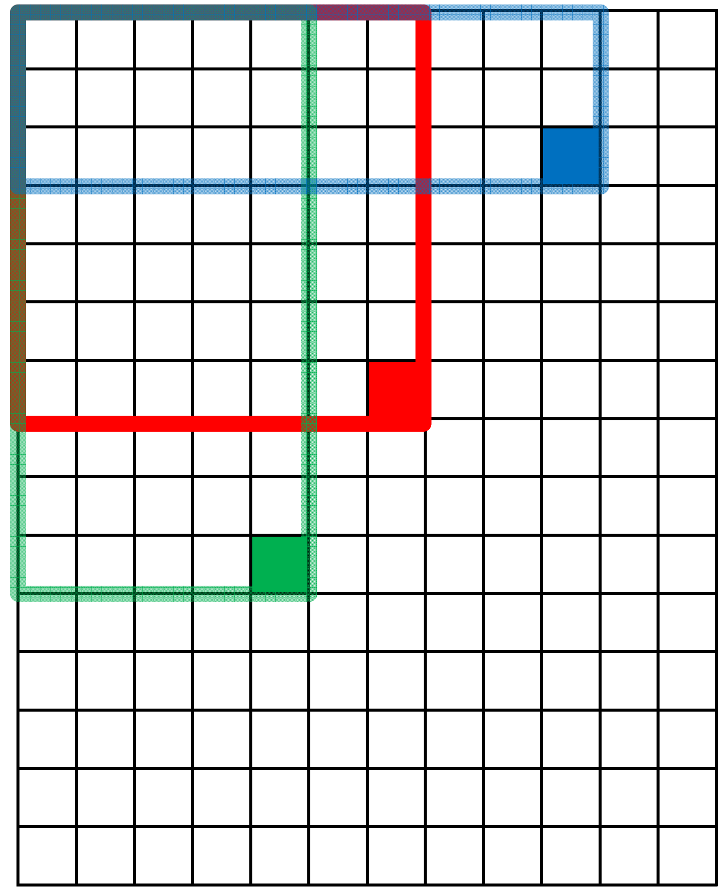
243	239	240	225	206	185	188	218	211	206	216	225
242	239	218	110	67	31	34	152	213	206	208	221
243	242	123	58	94	82	132	77	108	208	208	215
235	217	115	212	243	236	247	139	91	209	208	211
233	208	131	222	219	226	196	114	74	208	213	214
232	217	131	116	77	150	69	56	52	201	228	223
232	232	182	186	184	179	159	123	93	232	235	235
232	236	201	154	216	133	129	81	175	252	241	240
235	238	230	128	172	138	65	63	234	249	241	245
237	236	247	143	59	78	10	94	255	248	247	251
234	237	245	193	55	33	115	144	213	255	253	251
248	245	161	128	149	109	138	65	47	156	239	255
190	107	39	102	94	73	114	58	17	7	51	137
23	32	33	148	168	203	179	43	27	17	12	8
17	26	12	160	255	255	109	22	26	19	35	24

Sums of rectangular regions

The trick is to compute an "integral image." Every pixel is the sum of its neighbors to the upper left.

Sequentially compute using:

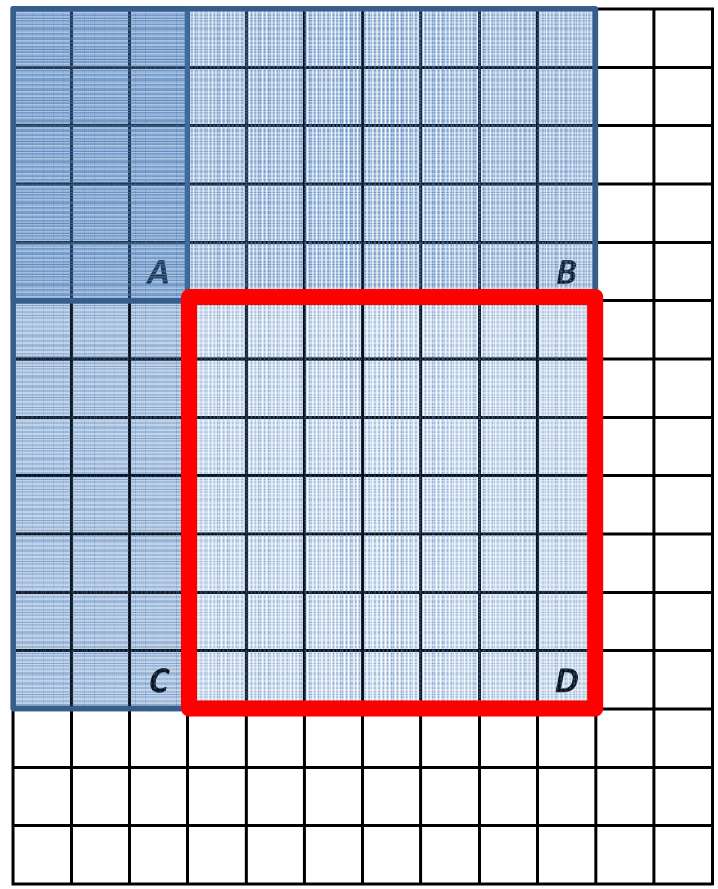
$$I(x,y) = I(x,y) + I(x-1,y) + I(x,y-1) - I(x-1,y-1)$$



Sums of rectangular regions

Solution is found using:

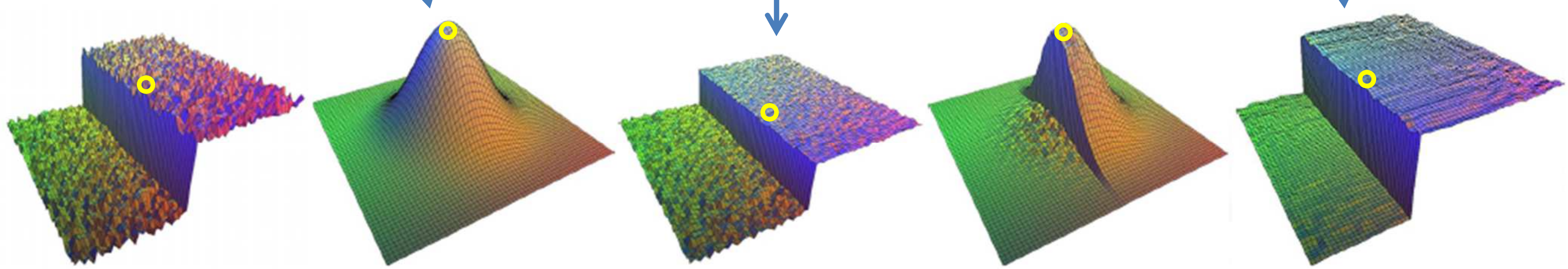
$$A + D - B - C$$



Spatially varying filters

Some filters vary spatially.

$$\sum_{x'}\sum_{y'}\mathcal{G}_{\sigma}(x',y')\mathcal{G}_{\sigma'}(f(x,y)-f(x+x',y+y'))=g(x,y)$$

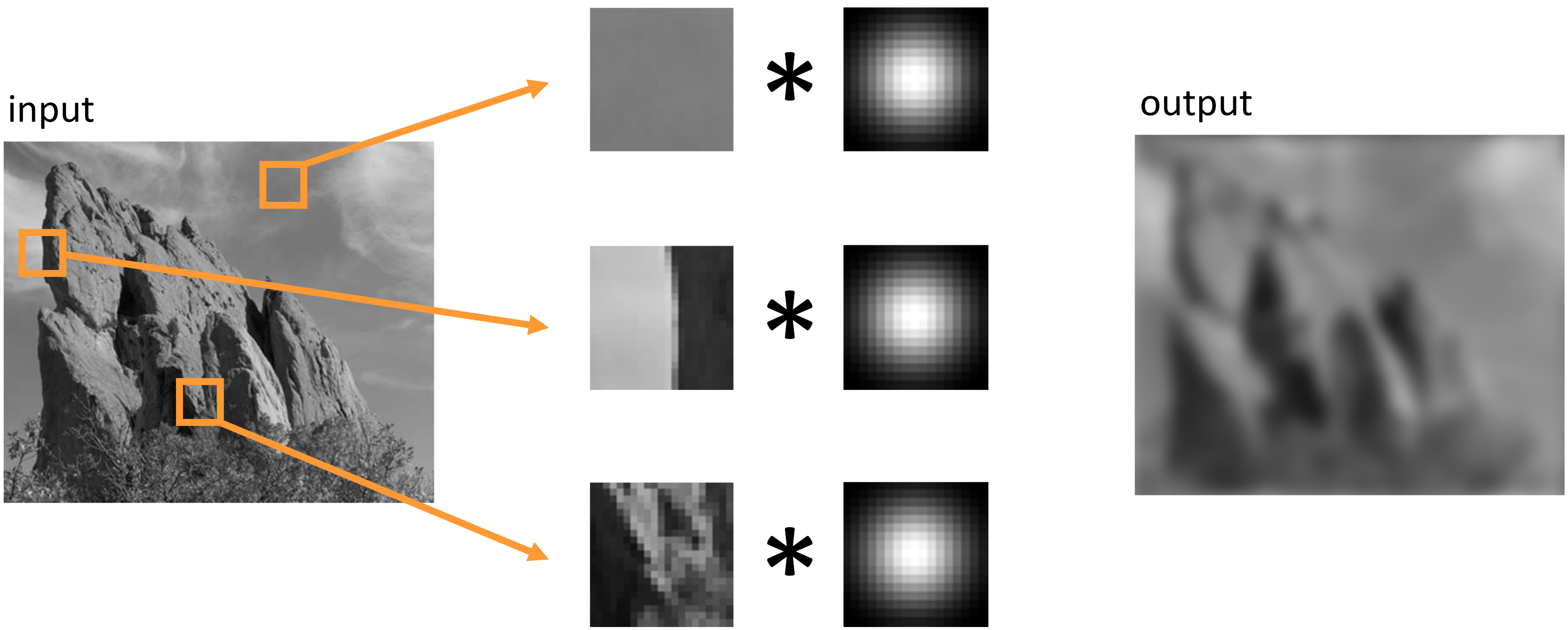


input | spatial kernel | influence in the intensity domain for the central pixel | weight for the central pixel | output

Durand, 02

Useful for deblurring.

Constant blur

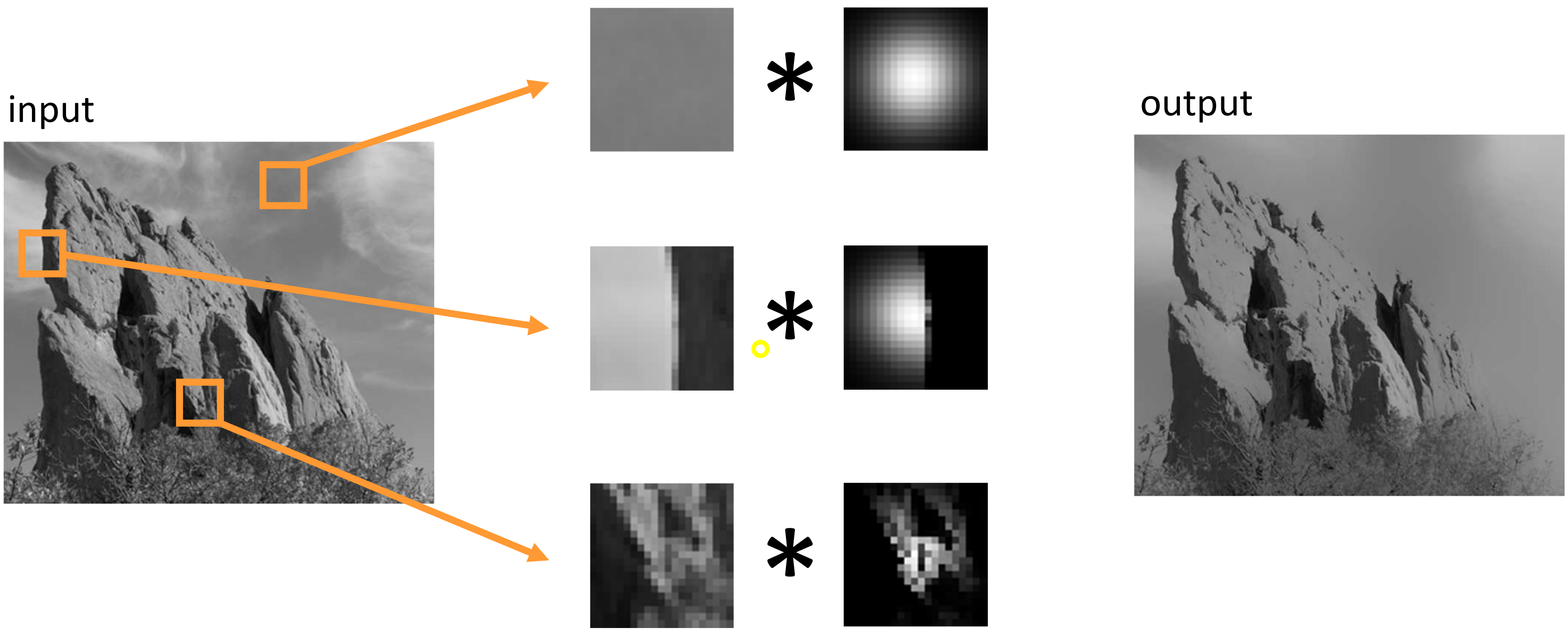


Same Gaussian kernel everywhere.

Slides courtesy of Sylvian Paris

Bilateral filter

Maintains edges when blurring!



The kernel shape depends on the image content.

Slides courtesy of Sylvian Paris

Borders

What to do about image borders:


black


fixed


periodic

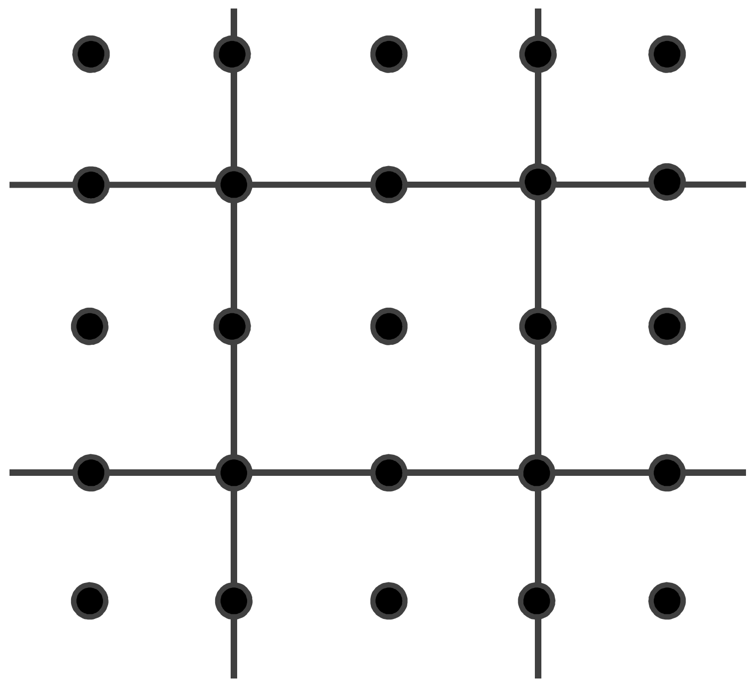

reflected

Sampling

Larry Zitnick

Up-sampling

How do we compute the values of pixels at fractional positions?



Up-Sampling as a Convolution Application Example

$$\begin{pmatrix}1&3&2\\4&5&6\end{pmatrix} \rightarrow \boxed{\text{Zero interlace}} \rightarrow \begin{pmatrix}1&0&3&0&2&0\\0&0&0&0&0&0\\4&0&5&0&6&0\\0&0&0&0&0&0\end{pmatrix} \rightarrow \boxed{\text{Convolve} \begin{pmatrix}1&1&1\\1&1&1\\1&1&1\end{pmatrix}}$$

Original Image

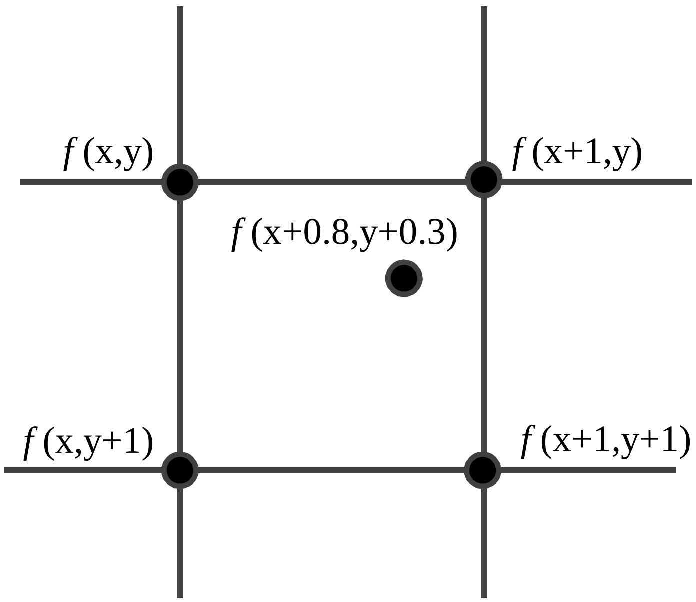
$$(1+0+3+0+0+0+4+0+5) \div (1+1+1+1+1+1+1+1+1) = 13/9$$

The value of a pixel in the enlarged image is the average of the value of around pixels. The difference between insert 0 and original value of pixels is **“*smoothed*”** by convolution

Up-Sampling as a Convolution Application Example

Up-sampling: General solutions

How do we compute the values of pixels at fractional positions?



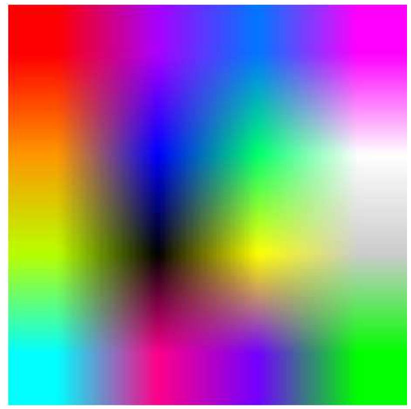
Bilinear sampling:

$$
\begin{aligned}
f(x + a, y + b) = \; & (1 - a)(1 - b)\, f(x, y) + \\
& a(1 - b)\, f(x + 1, y) + \\
& (1 - a)b\, f(x, y + 1) + \\
& ab\, f(x + 1, y + 1)
\end{aligned}
$$

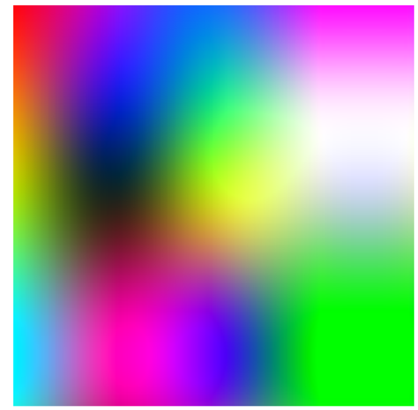
Bicubic sampling fits a higher order function using a larger area of support.

Up-sampling

Nearest neighbor



Bilinear



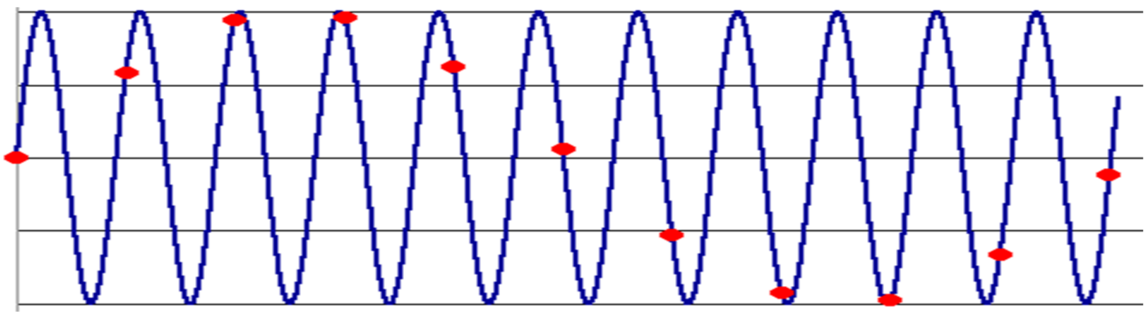
Bicubic

Down-sampling

If you do it incorrectly your images could look like this:

Check out Moire patterns on the web.

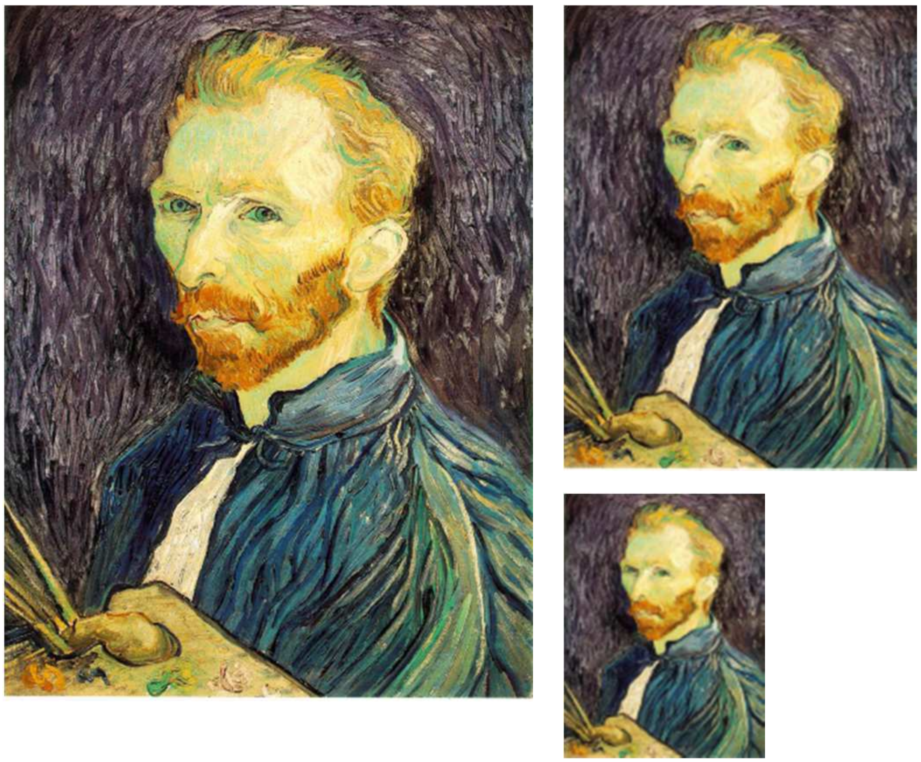
Down-sampling



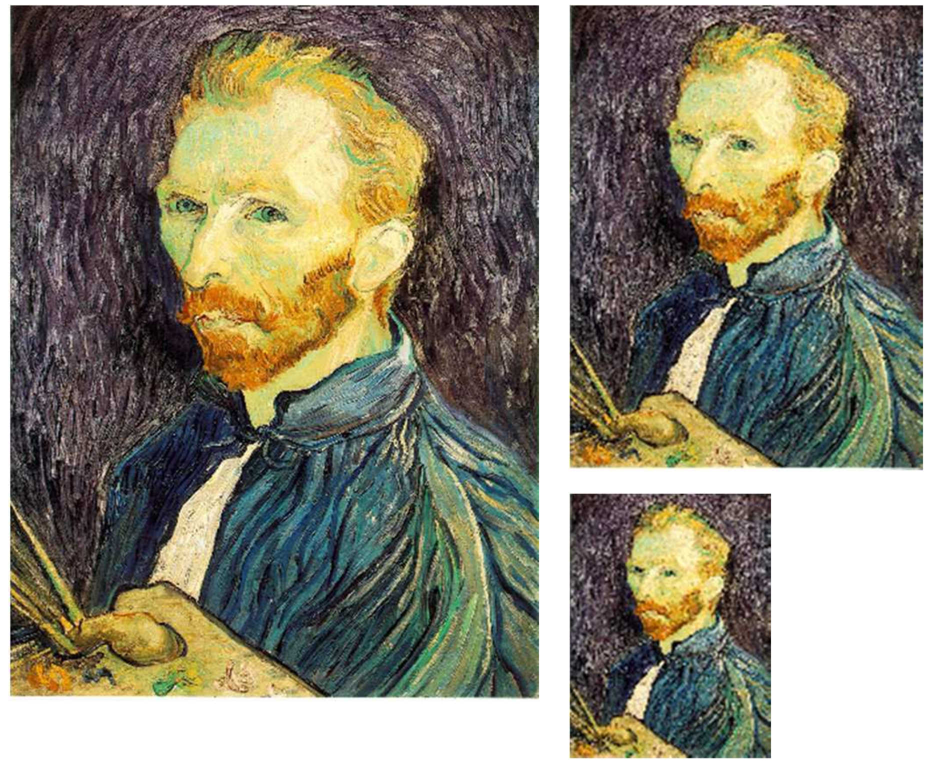
- **Aliasing** can arise when you sample a continuous signal or image
  - occurs when your sampling rate is not high enough to capture the amount of detail in your image
  - Can give you the wrong signal/image—an *alias*
  - formally, the image contains structure at different scales
    - called “frequencies” in the Fourier domain
  - the sampling rate must be high enough to capture the highest frequency in the image

Solution

Filter before sampling, i.e. blur the image first.



With blur

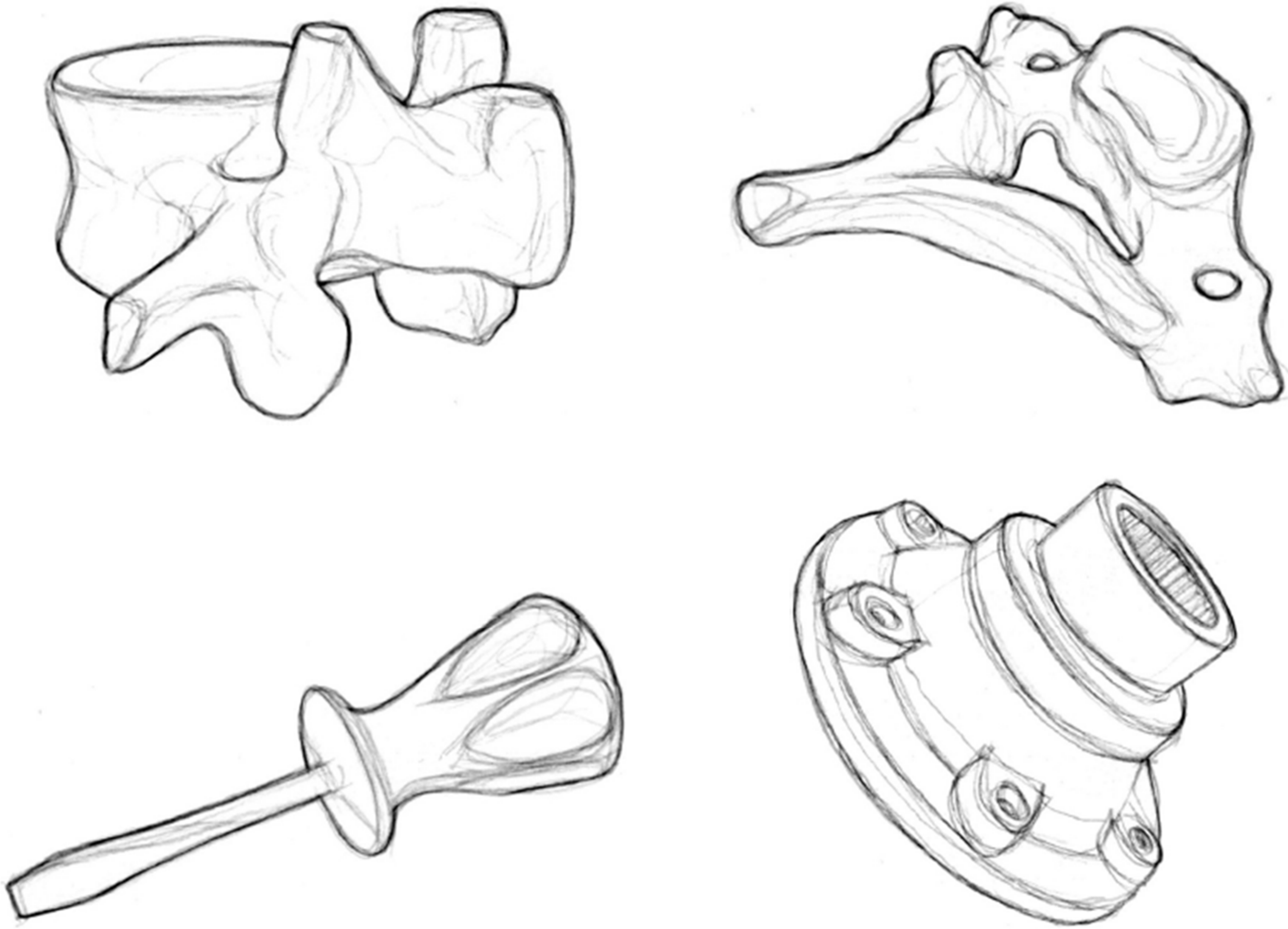


Without blur

Edge Detection

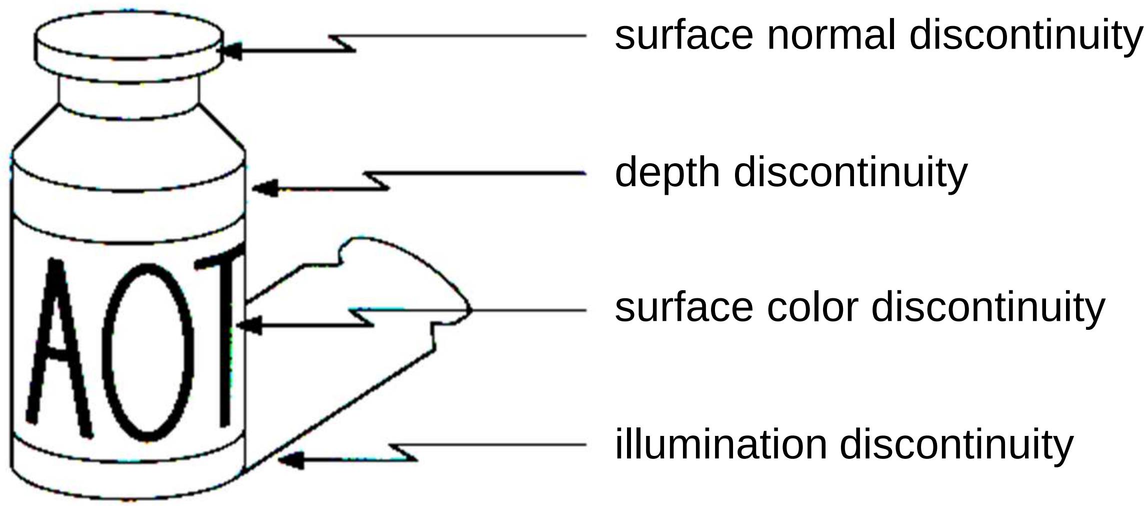
Larry Zitnick

What is an edge?



Cole et al. Siggraph 2008, results of 107 humans.

Origin of edges



Edges are caused by a variety of factors

Illusory contours

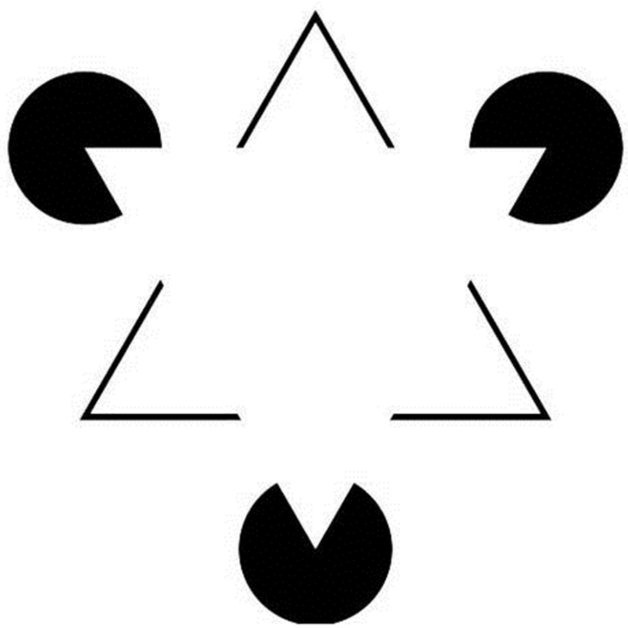
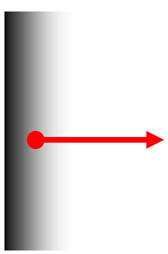


Image gradient

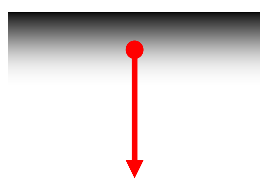
- The gradient of an image:

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

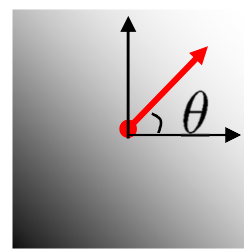
- The gradient points in the direction of most rapid change in intensity



$$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$$

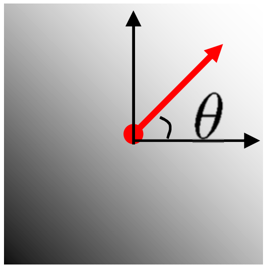


$$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$$



$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

Image gradient



$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

$$\frac{\partial f}{\partial x} = f(x+1, y) - f(x, y)$$

How would you implement this as a filter?

The gradient direction is given by:

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

How does this relate to the direction of the edge?

The *edge strength* is given by the gradient magnitude

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Sobel operator

In practice, it is common to use:

$$g_x = \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \qquad g_y = \begin{array}{|c|c|c|}\hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

Magnitude:

$$g = \sqrt{g_x^2 + g_y^2}$$

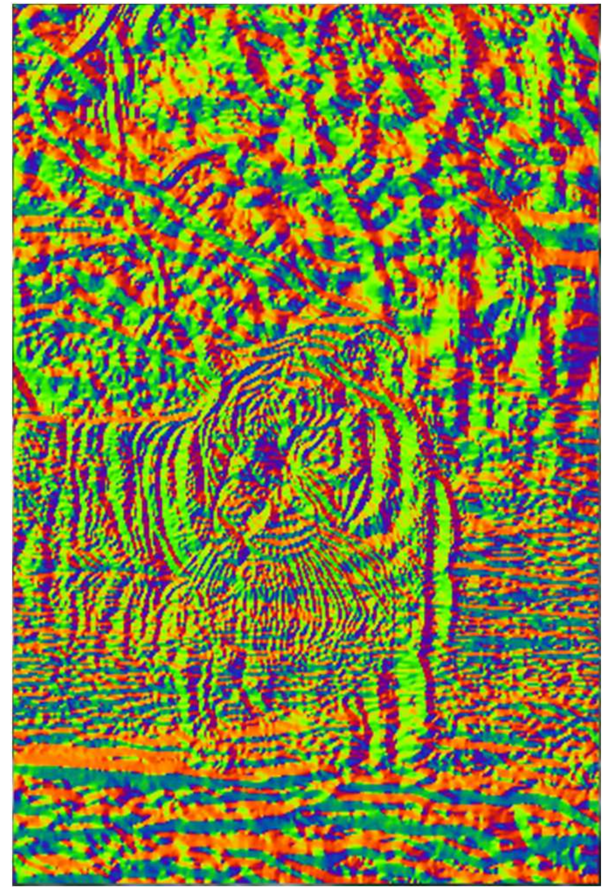
Orientation:

$$\Theta = \tan^{-1}\left(\frac{g_y}{g_x}\right)$$

Sobel operator

Original

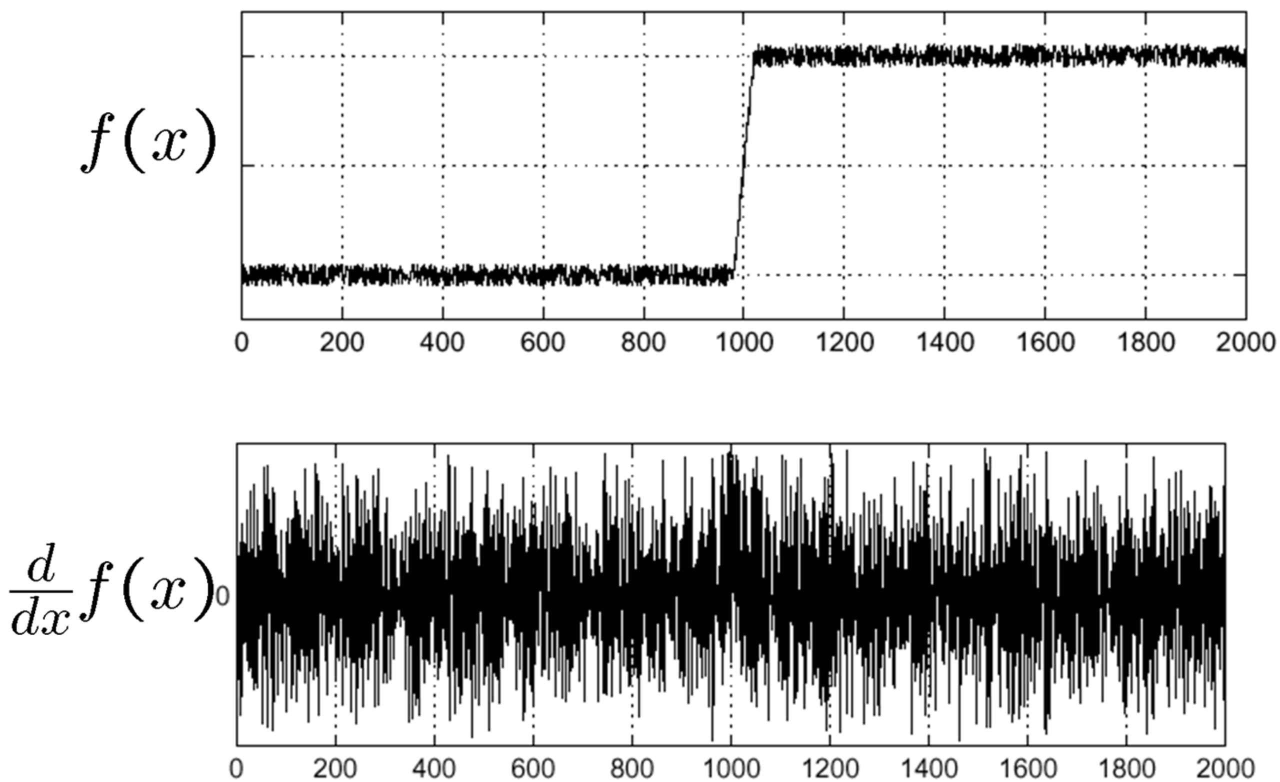
Magnitude



Orientation

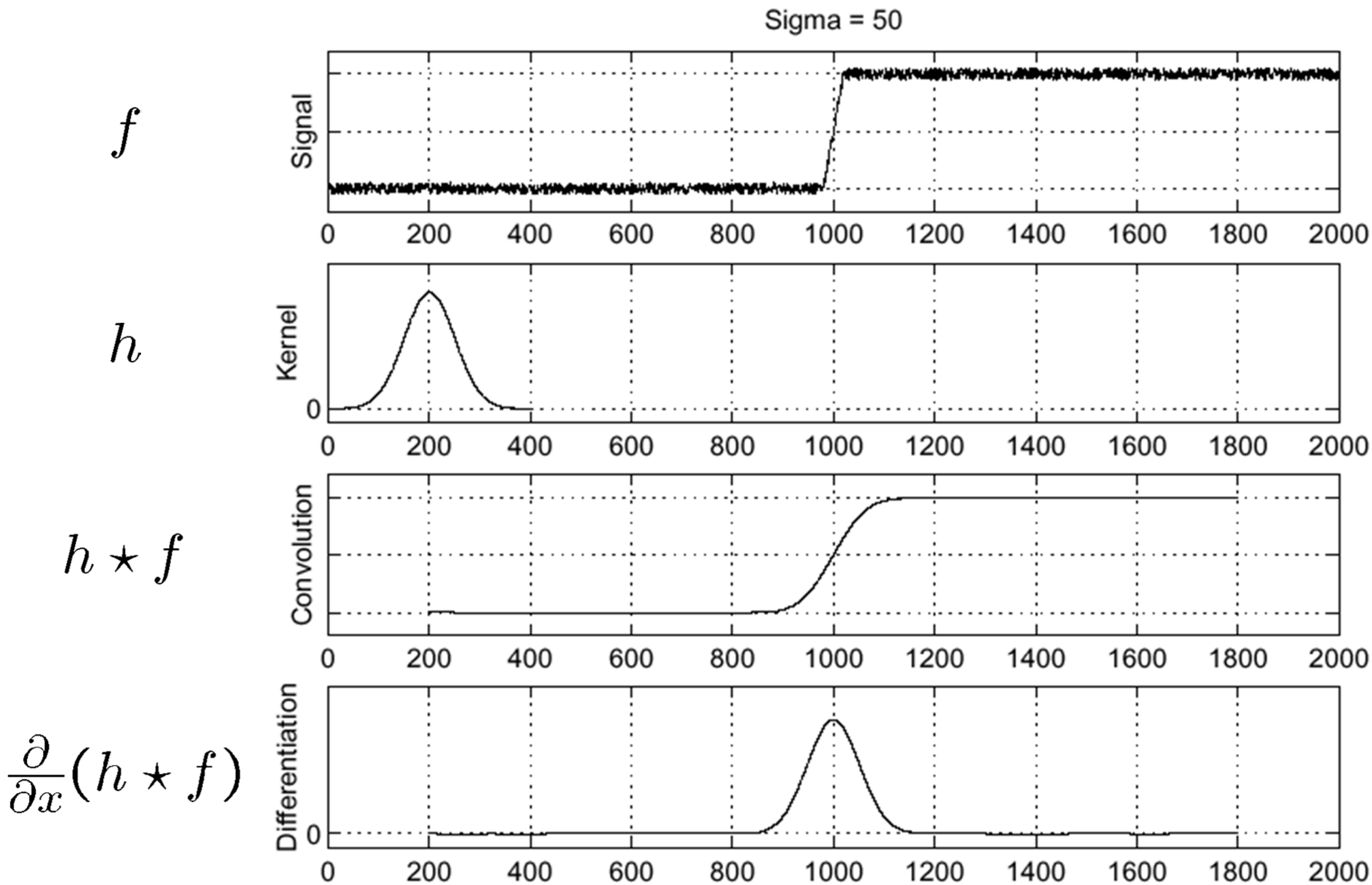
Effects of noise

- Consider a single row or column of the image
  - Plotting intensity as a function of position gives a signal



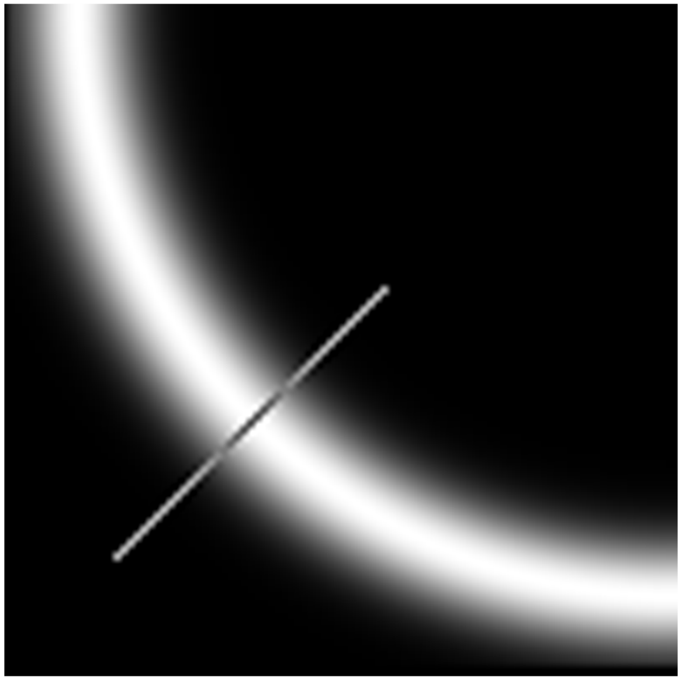
Where is the edge?

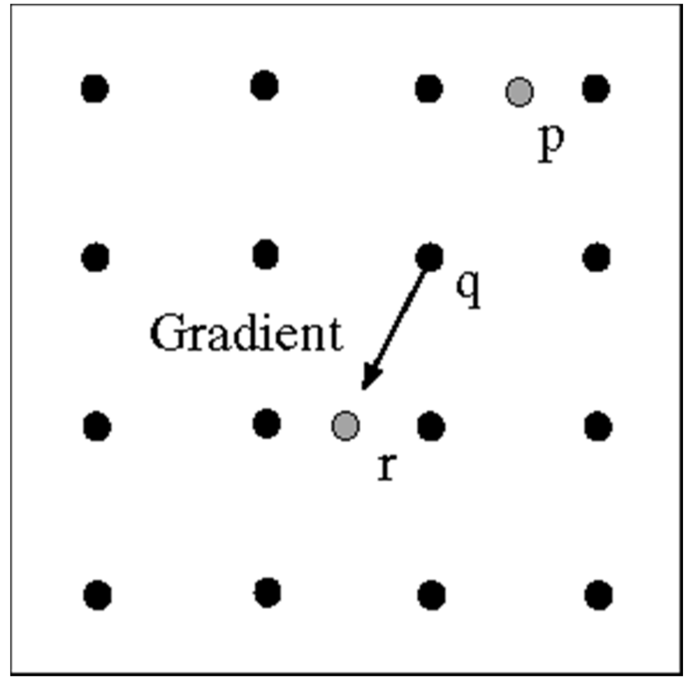
Solution: smooth first



Where is the edge? Look for peaks in $\frac{\partial}{\partial x}(h \star f)$

Non-maximum suppression





- Check if pixel is local maximum along gradient direction
  – requires checking interpolated pixels p and r

Effect of σ (Gaussian kernel spread/size)

original | Canny with $\sigma = 1$ | Canny with $\sigma = 2$

The choice of σ depends on desired behavior

- large σ detects large scale edges
- small σ detects fine features

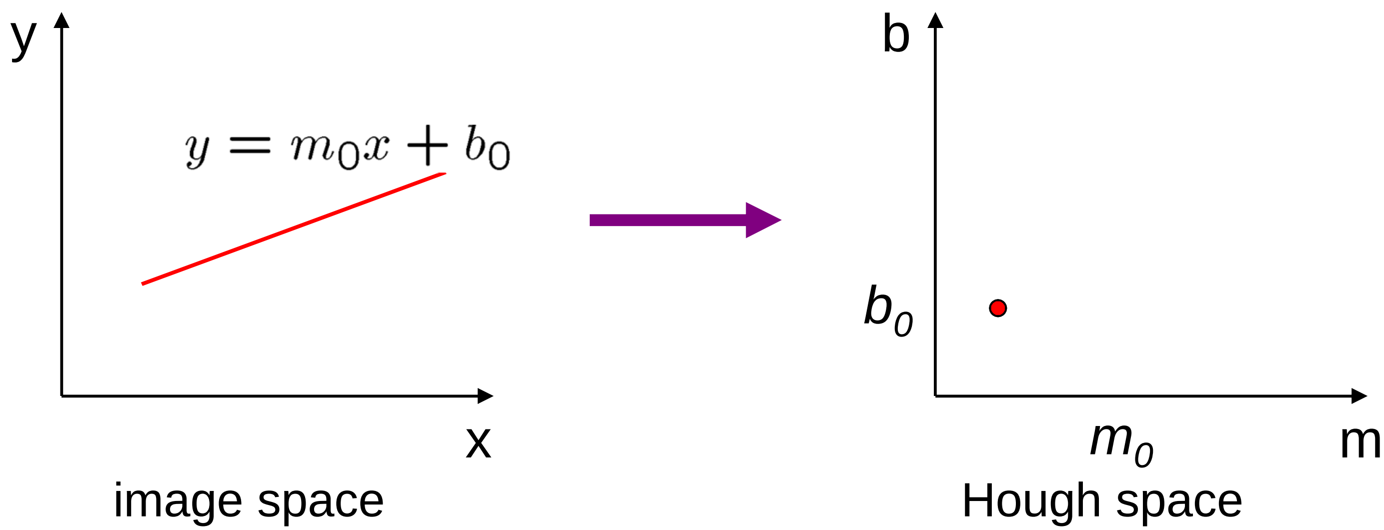
An edge is not a line...

How can we detect ***lines*** ?

Finding lines in an image

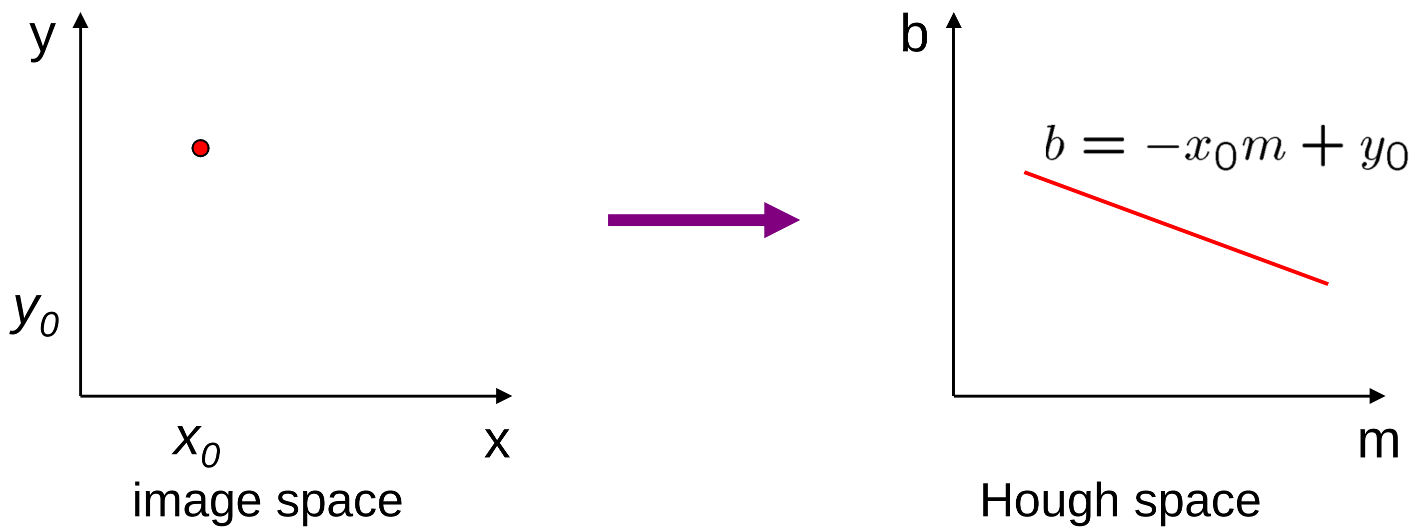
- Option 1:
  - Search for the line at every possible position/orientation
  - What is the cost of this operation?
- Option 2:
  - Use a voting scheme: Hough transform

Finding lines in an image



- Connection between image (x,y) and Hough (m,b) spaces
  - A line in the image corresponds to a point in Hough space
  - To go from image space to Hough space:
    - given a set of points (x,y), find all (m,b) such that y = mx + b

Finding lines in an image



- Connection between image (x,y) and Hough (m,b) spaces
  - A line in the image corresponds to a point in Hough space
  - To go from image space to Hough space:
    - given a set of points (x,y), find all (m,b) such that y = mx + b
  - What does a point (x_0, y_0) in the image space map to?

    - A: the solutions of $b = -x_0 m + y_0$
    - this is a line in Hough space

Hough transform algorithm

- Typically use a different parameterization

$$d = x cos\theta + y sin\theta$$

  - d is the perpendicular distance from the line to the origin
  - θ is the angle
  - Why?

Hough transform algorithm

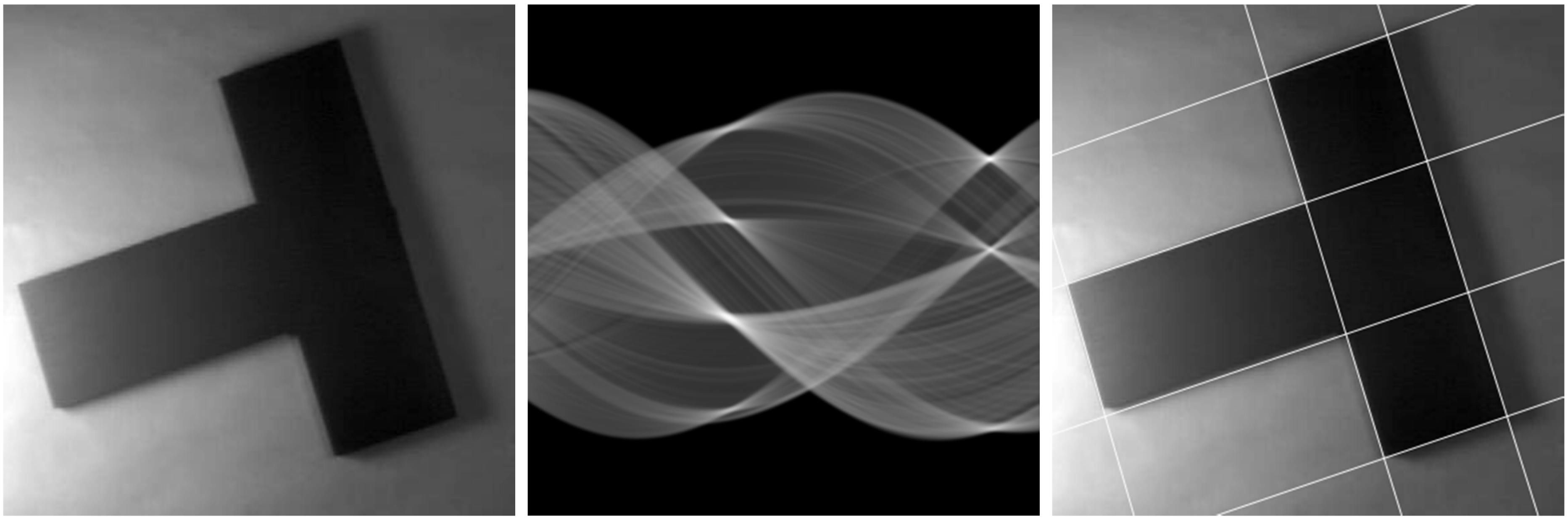
- Basic Hough transform algorithm
  1. Initialize H[d, θ]=0
  2. for each edge point I[x,y] in the image

     for θ = 0 to 180

     $$d = x cos\theta + y sin\theta$$

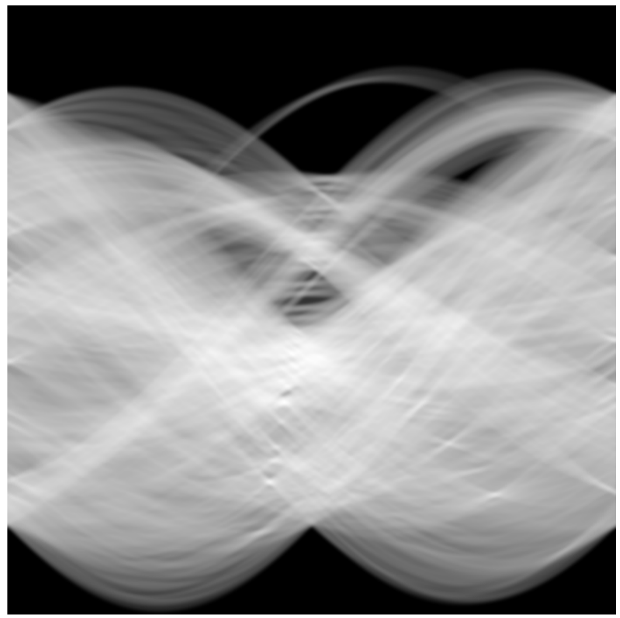
     H[d, θ] += 1
  3. Find the value(s) of (d, θ) where H[d, θ] is maximum
  4. The detected line in the image is given by $d = x cos\theta + y sin\theta$
- What's the running time (measured in # votes)?

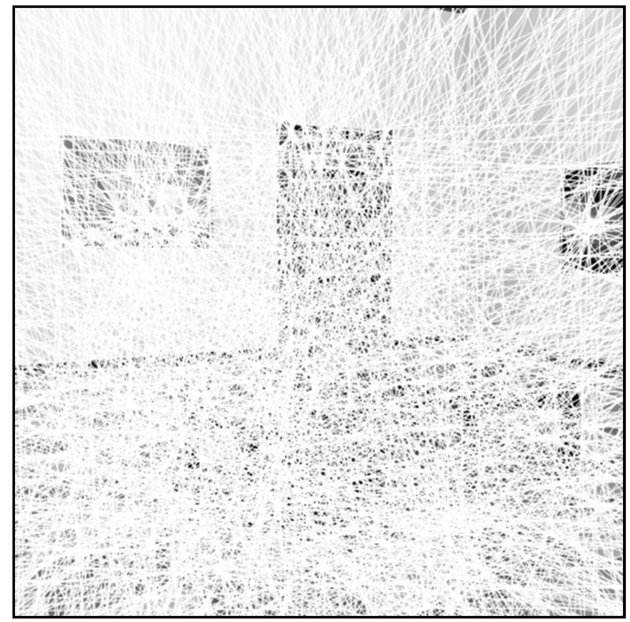
Hough transform algorithm



http://www.cs.utah.edu/~vpegorar/courses/cs7966/

Hough transform algorithm





http://www.cs.utah.edu/~vpegorar/courses/cs7966/

Extensions

- Extension 1: Use the image gradient
  1. same
  2. for each edge point I[x,y] in the image

     compute unique (d, θ) based on image gradient at (x,y)

     H[d, θ] += 1
  3. same
  4. same
- What's the running time measured in votes?

- Extension 2
  – give more votes for stronger edges
- Extension 3
  – change the sampling of (d, θ) to give more/less resolution
- Extension 4
  – The same procedure can be used with circles, squares, or any other shape

Sources

- Zitnick course. Has filtering etc
- http://www.cs.washington.edu/education/courses/csep576/11sp/schedule.htm
- http://sern.ucalgary.ca/courses/CPSC/533/W99/presentations/L2_24A_Lee_Wang/
  http://sern.ucalgary.ca/courses/CPSC/533/W99/presentations/L1_24A_Kaasten_Steller_Hoang/main.htm
  http://sern.ucalgary.ca/courses/CPSC/533/W99/presentations/L1_24_Schebywolok/index.html
  http://sern.ucalgary.ca/courses/CPSC/533/W99/presentations/L2_24B_Doering_Grenier/
- http://www.geocities.com/SoHo/Museum/3828/optical.html