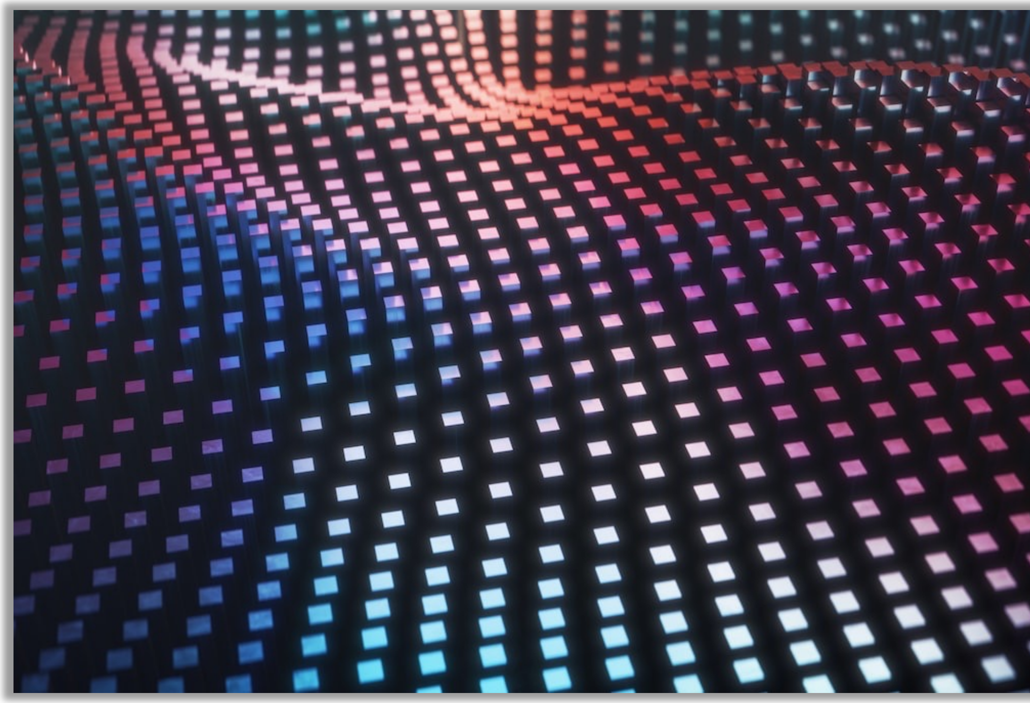
Edges

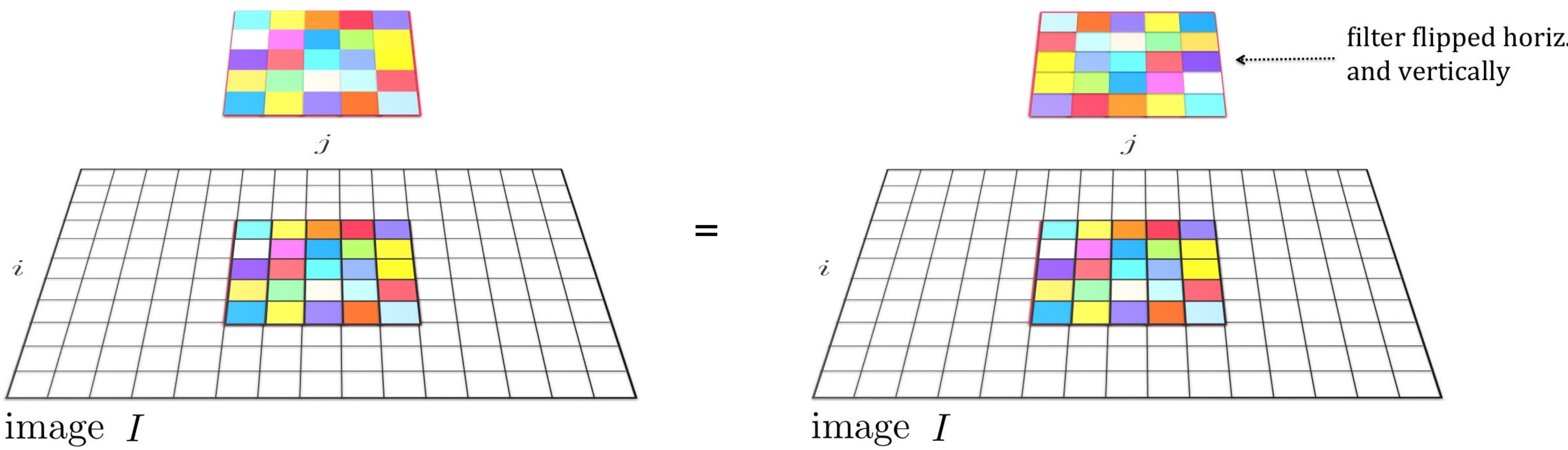
Review of Fourier Transform, Edge Detection



CSC420
David Lindell
University of Toronto
cs.toronto.edu/~lindell/teaching/420
Slide credit: Babak Taati ←Ahmed Ashraf ←Sanja Fidler

Wrap up lecture 1...

Correlation vs Convolution



Correlation

Convolution

Separable Filters: Speed-up Trick

- The process of performing a convolution requires K^2 operations per pixel, where K is the size (width or height) of the convolution filter

[Source: R. Urtasun]

Separable Filters: Speed-up Trick

- The process of performing a convolution requires K^2 operations per pixel, where K is the size (width or height) of the convolution filter
- Can we do faster?

[Source: R. Urtasun]

Separable Filters: Speed-up Trick

- The process of performing a convolution requires K^2 operations per pixel, where K is the size (width or height) of the convolution filter
- Can we do faster?
- In many cases (**not all!)**, this operation can be sped up by first performing a 1D horizontal convolution followed by a 1D vertical convolution, **requiring only 2K operations**

[Source: R. Urtasun]

Separable Filters: Speed-up Trick

- The process of performing a convolution requires K^2 operations per pixel, where K is the size (width or height) of the convolution filter
- Can we do faster?
- In many cases (**not all!**), this operation can be sped up by first performing a 1D horizontal convolution followed by a 1D vertical convolution, **requiring only 2K operations**
- If this is possible, then the convolutional filter is called **separable**

[Source: R. Urtasun]

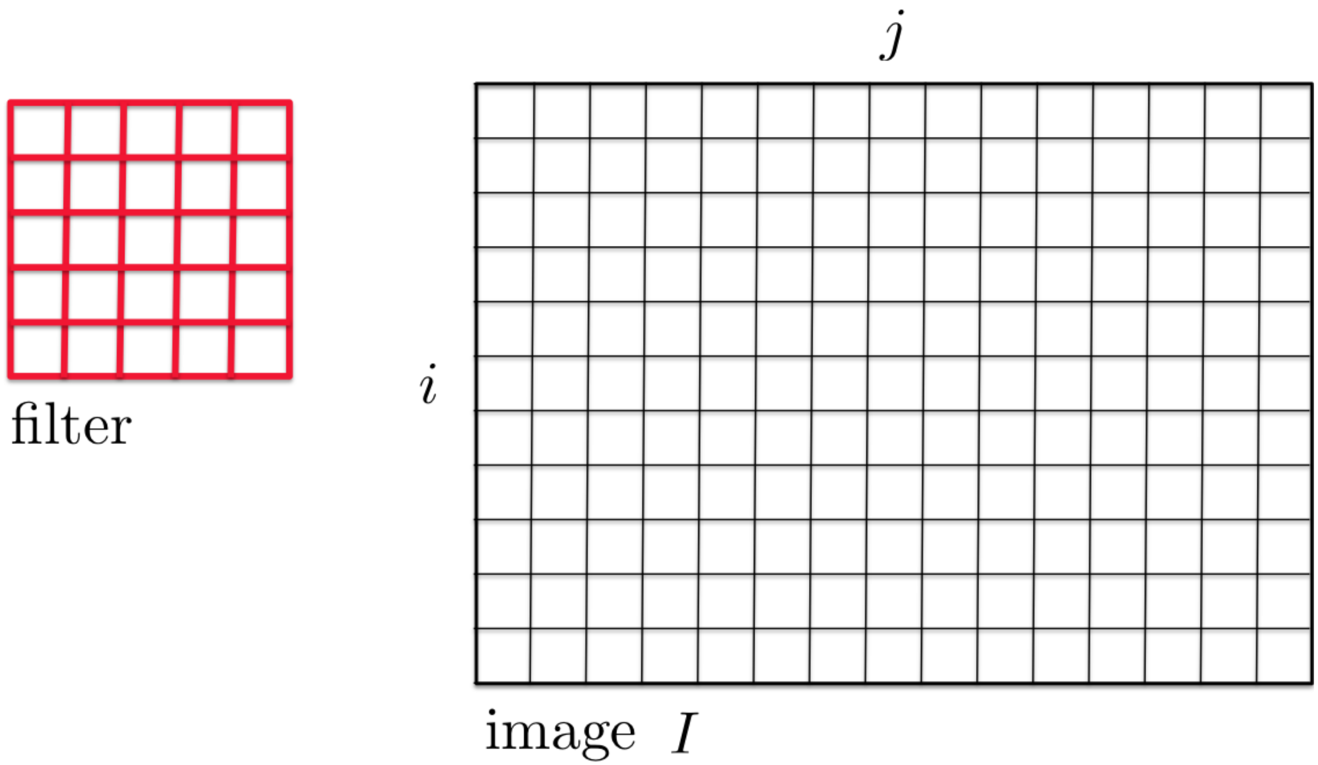
Separable Filters: Speed-up Trick

- The process of performing a convolution requires K^2 operations per pixel, where K is the size (width or height) of the convolution filter
- Can we do faster?
- In many cases (**not all!**), this operation can be sped up by first performing a 1D horizontal convolution followed by a 1D vertical convolution, **requiring only 2K operations**
- If this is possible, then the convolutional filter is called **separable**
- And it is the outer product of two filters:

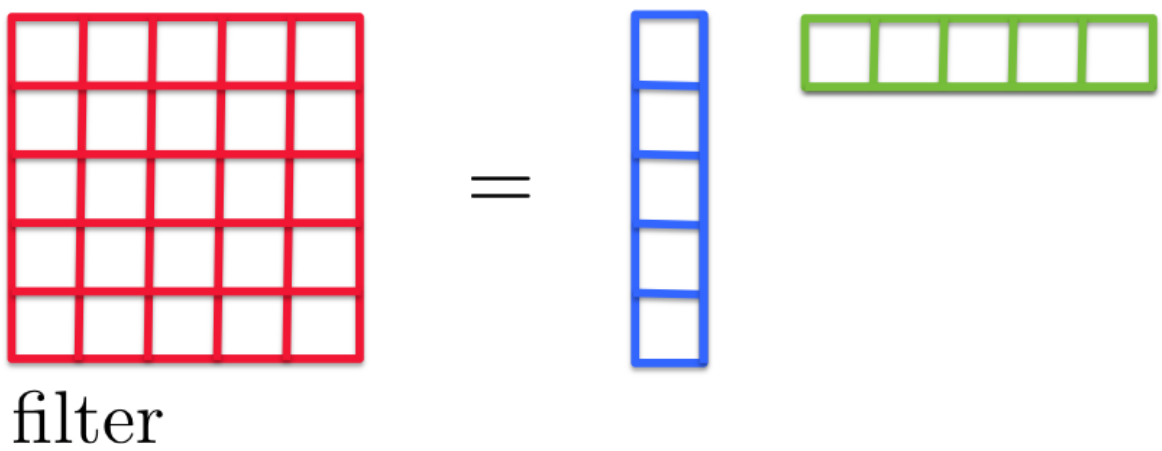
$$\mathbf{F} = \mathbf{v}\mathbf{h}^T$$

[Source: R. Urtasun]

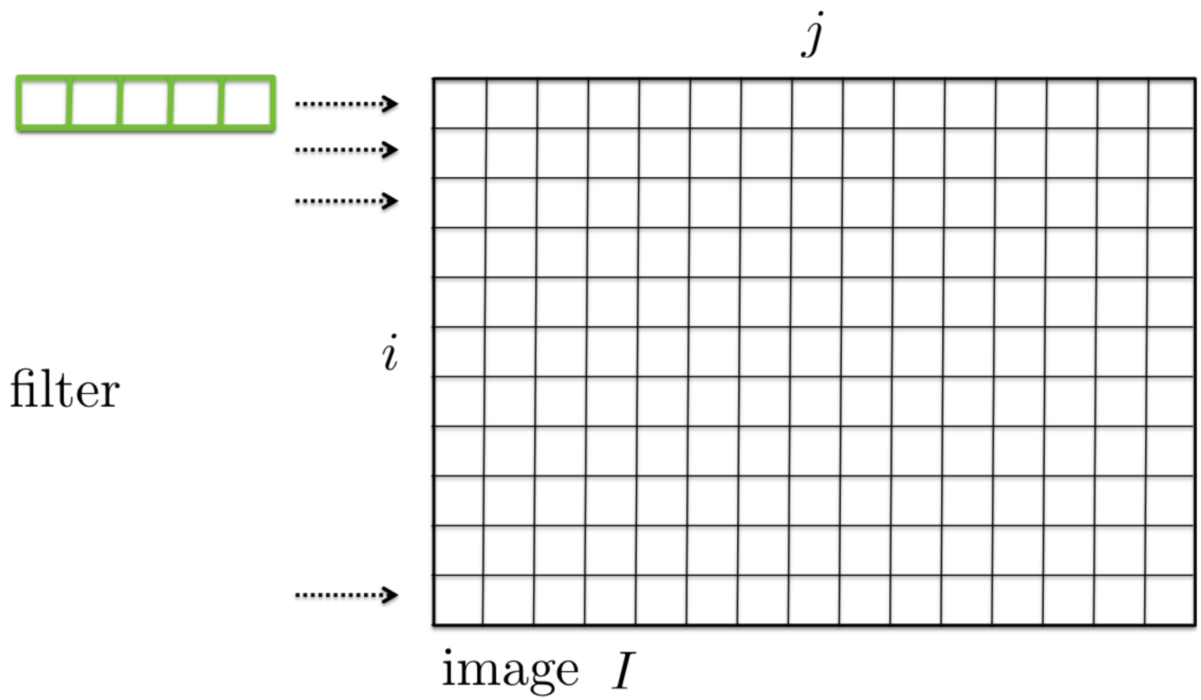
How it works



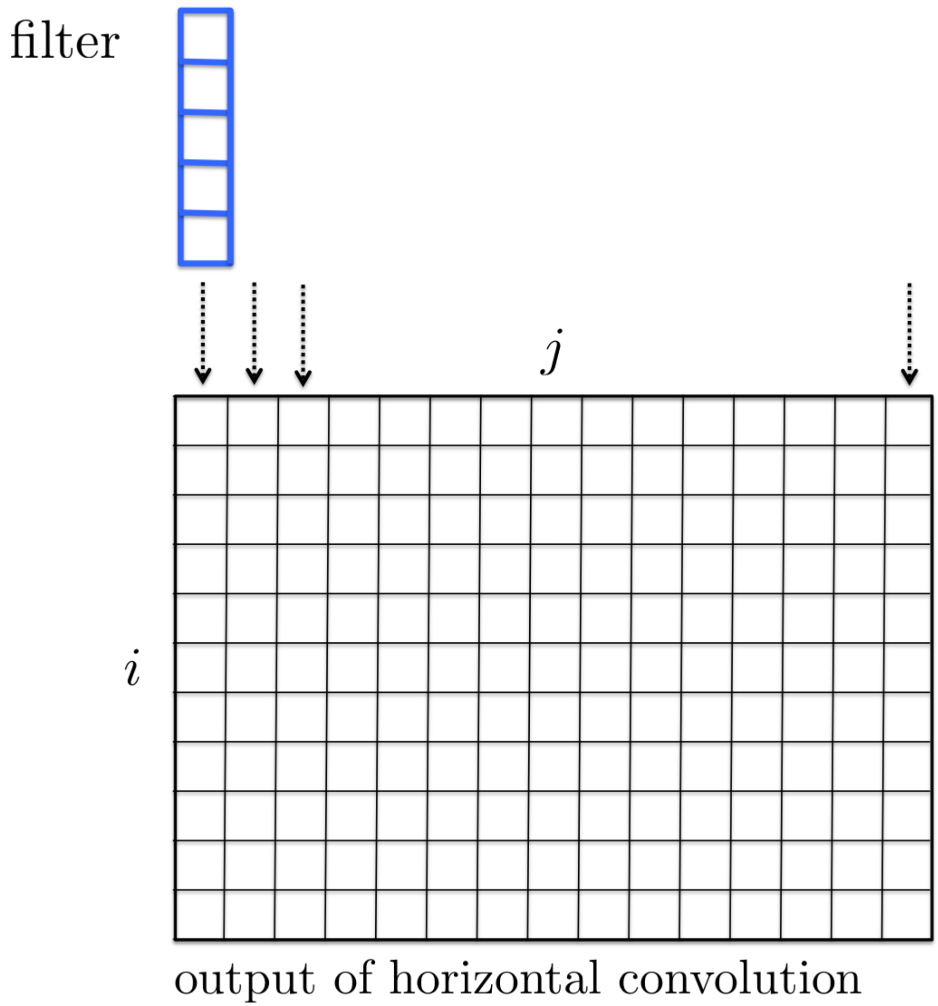
How it works



How it works



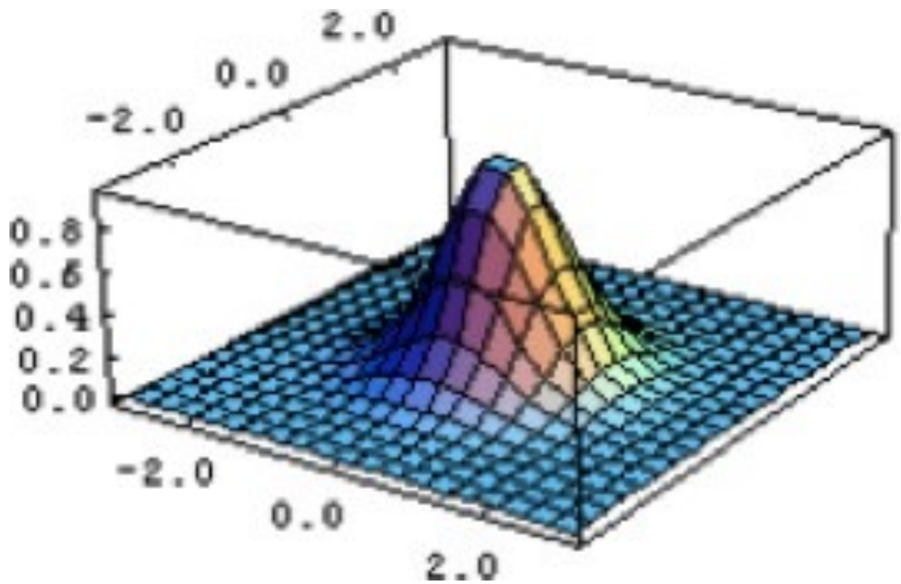
How it works



How it works

One famous separable filter we already know:

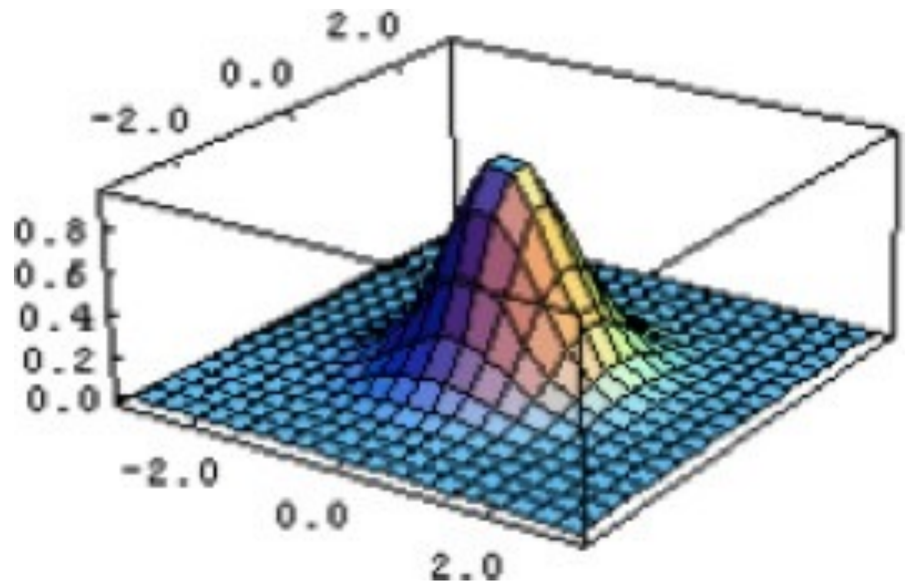
Gaussian: $f(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{\sigma^2}\right)$



How it works

One famous separable filter we already know:

Gaussian: $f(x,y) = \left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{\sigma^2}}\right) \cdot \left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{\sigma^2}}\right)$



How it works

Is this separable? If yes, what's the separable version?

$$\frac{1}{K^2}\begin{array}{|c|c|c|c|}\hline 1 & 1 & \cdots & 1 \\ \hline 1 & 1 & \cdots & 1 \\ \hline \vdots & \vdots & 1 & \vdots \\ \hline 1 & 1 & \cdots & 1 \\ \hline \end{array}$$

[Source: R. Urtasun]

How it works

Is this separable? If yes, what's the separable version?

$$\frac{1}{K^2}\begin{array}{|c|c|c|c|}\hline 1 & 1 & \cdots & 1 \\ \hline 1 & 1 & \cdots & 1 \\ \hline \vdots & \vdots & 1 & \vdots \\ \hline 1 & 1 & \cdots & 1 \\ \hline \end{array} \qquad \frac{1}{K}\begin{array}{|c|c|c|c|}\hline 1 & 1 & \cdots & 1 \\ \hline \end{array}$$

What does this filter do?

[Source: R. Urtasun]

How it works

Is this separable? If yes, what's the separable version?

$$\frac{1}{16}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

[Source: R. Urtasun]

How it works

Is this separable? If yes, what's the separable version?

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

What does this filter do?

[Source: R. Urtasun]

How it works

Is this separable? If yes, what's the separable version?

$$\frac{1}{8}\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$$

[Source: R. Urtasun]

How it works

Is this separable? If yes, what's the separable version?

$$\frac{1}{8}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \frac{1}{8}\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

What does this filter do?

[Source: R. Urtasun]

How can we tell if a given filter F is indeed separable?

- Inspection... this is what we were doing.

[Source: R. Urtasun]

How can we tell if a given filter F is indeed separable?

- Inspection... this is what we were doing
- Look at the singular value decomposition (SVD), and if only one singular value is non-zero, then it is separable

$$F = \mathbf{U}\Sigma\mathbf{V}^T = \sum_i \sigma_i u_i v_i^T$$

with $\Sigma = \mathrm{diag}(\sigma_i)$

[Source: R. Urtasun]

How can we tell if a given filter F is indeed separable?

- Inspection... this is what we were doing
- Look at the singular value decomposition (SVD), and if only one singular value is non-zero, then it is separable

$$F = \mathbf{U}\Sigma\mathbf{V}^T = \sum_i \sigma_i u_i v_i^T$$

with $\Sigma = \mathrm{diag}(\sigma_i)$

- Python: `np.linalg.svd`

[Source: R. Urtasun]

How can we tell if a given filter F is indeed separable?

- Inspection... this is what we were doing
- Look at the singular value decomposition (SVD), and if only one singular value is non-zero, then it is separable

$$F = \mathbf{U}\Sigma\mathbf{V}^T = \sum_i \sigma_i u_i v_i^T$$

with $\Sigma = \mathrm{diag}(\sigma_i)$

- Python: `np.linalg.svd`
- $\sqrt{\sigma_1}\mathbf{u}_1$ and $\sqrt{\sigma_1}\mathbf{v}_1$ are the vertical and horizontal filters

[Source: R. Urtasun]

Summary – Stuff You Should Know

- **Correlation:** Slide a filter across image and compare (via dot product)
- **Convolution:** Flip the filter to the right and down and do correlation
- **Smooth** image with a Gaussian kernel: bigger σ means more blurring
- **Some** filters (like Gaussian) are separable: you can filter faster. First apply 1D convolution to each row, followed by another 1D conv. to each column

OpenCV:

- `Filter2D` (or `sepFilter2D`): can do both correlation and convolution
- `GaussianBlur`: create a Gaussian kernel
- `medianBlur, medianBlur, bilateralFilter`

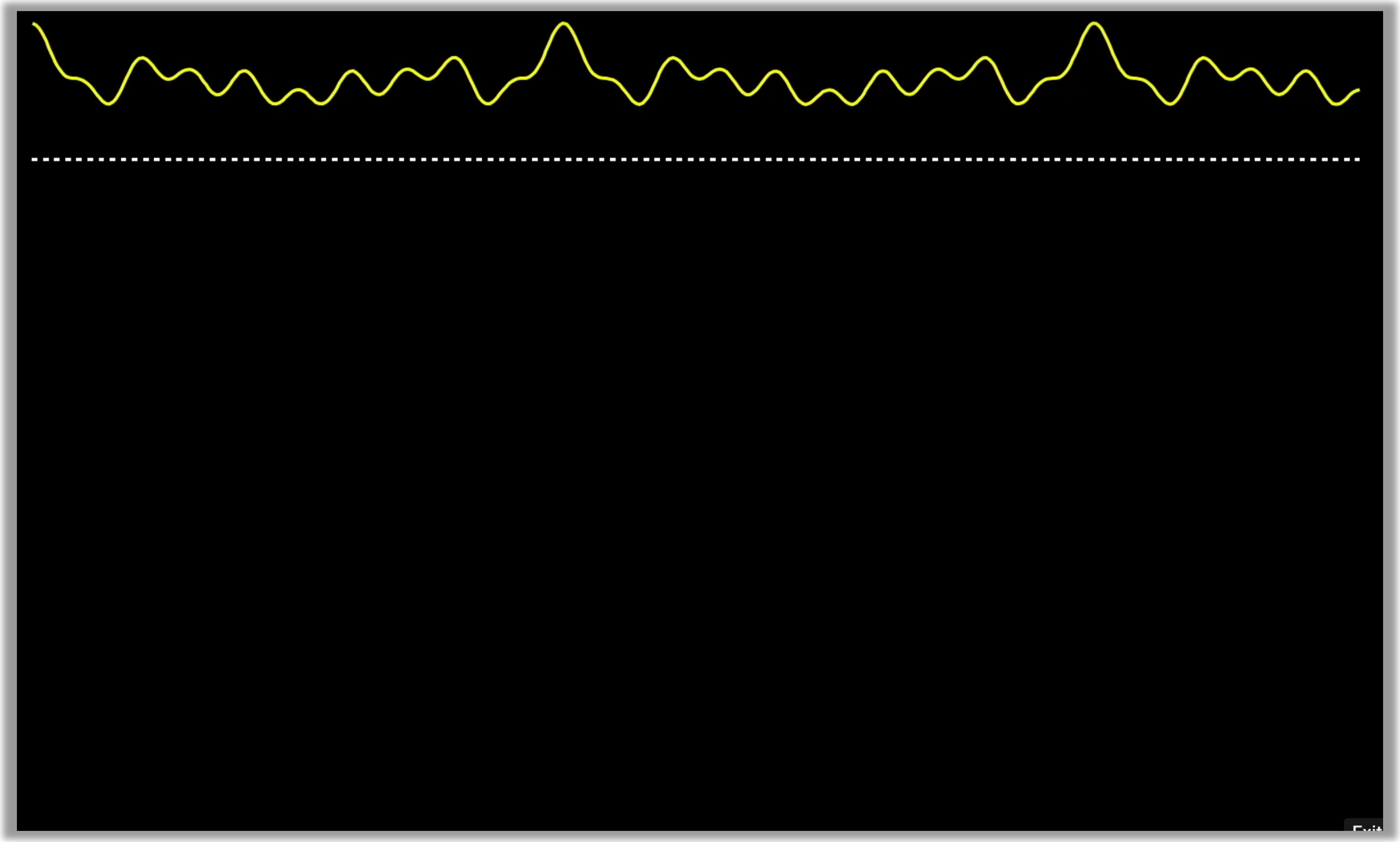
Edges

- What does blurring take away?

[Source: S. Lazebnik]

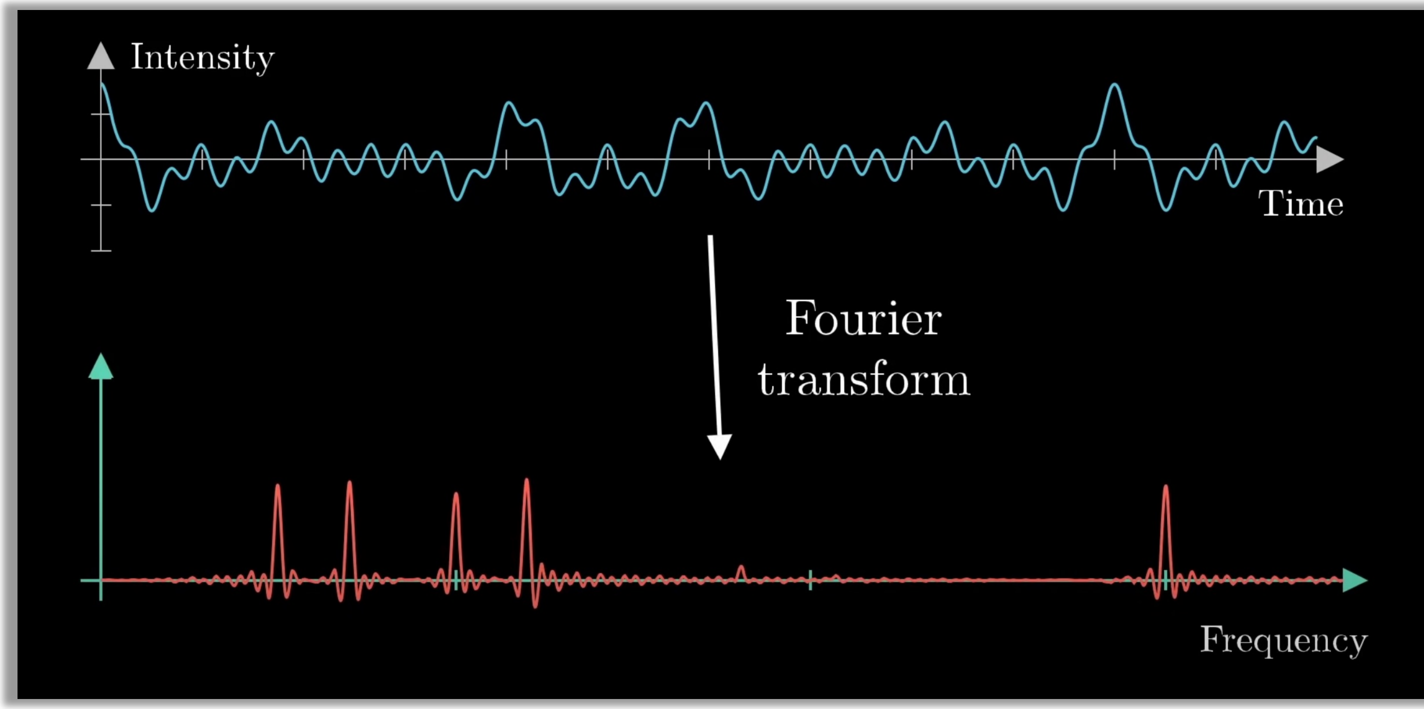
Review of Fourier Transform

1D Fourier Transform



[Source: 3B1B]

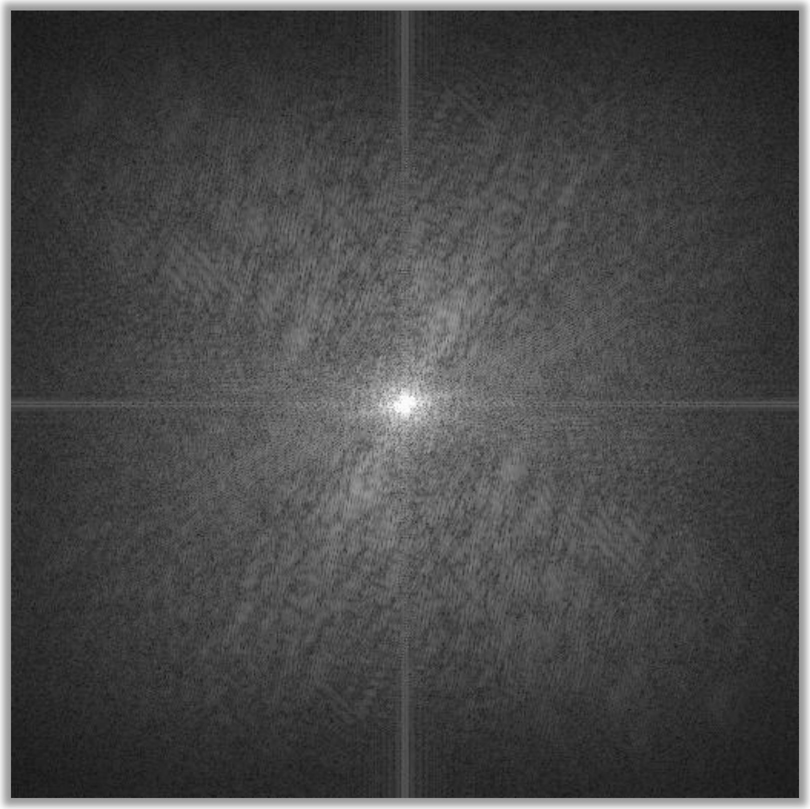
1D Fourier Transform



[Source: 3B1B]

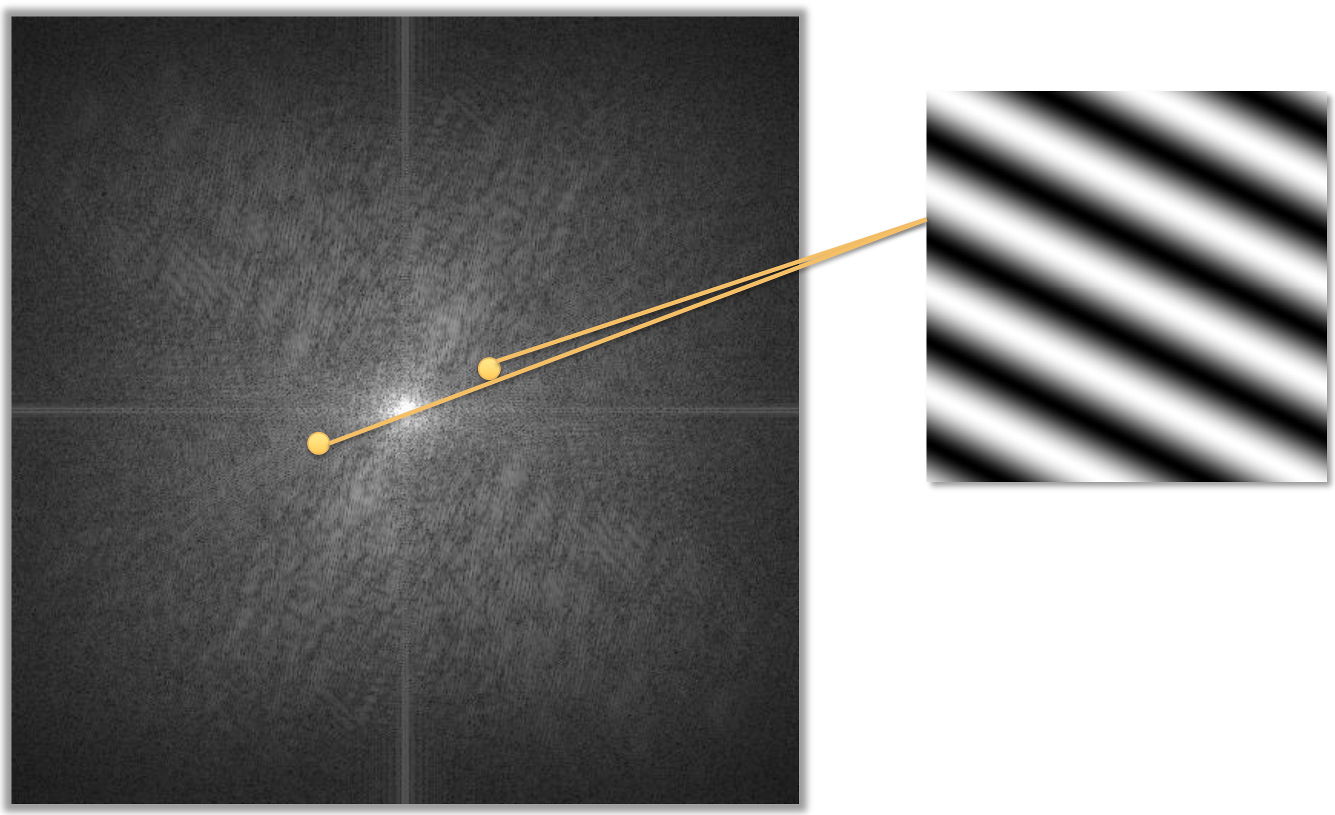
Fourier Transform

- What is this?



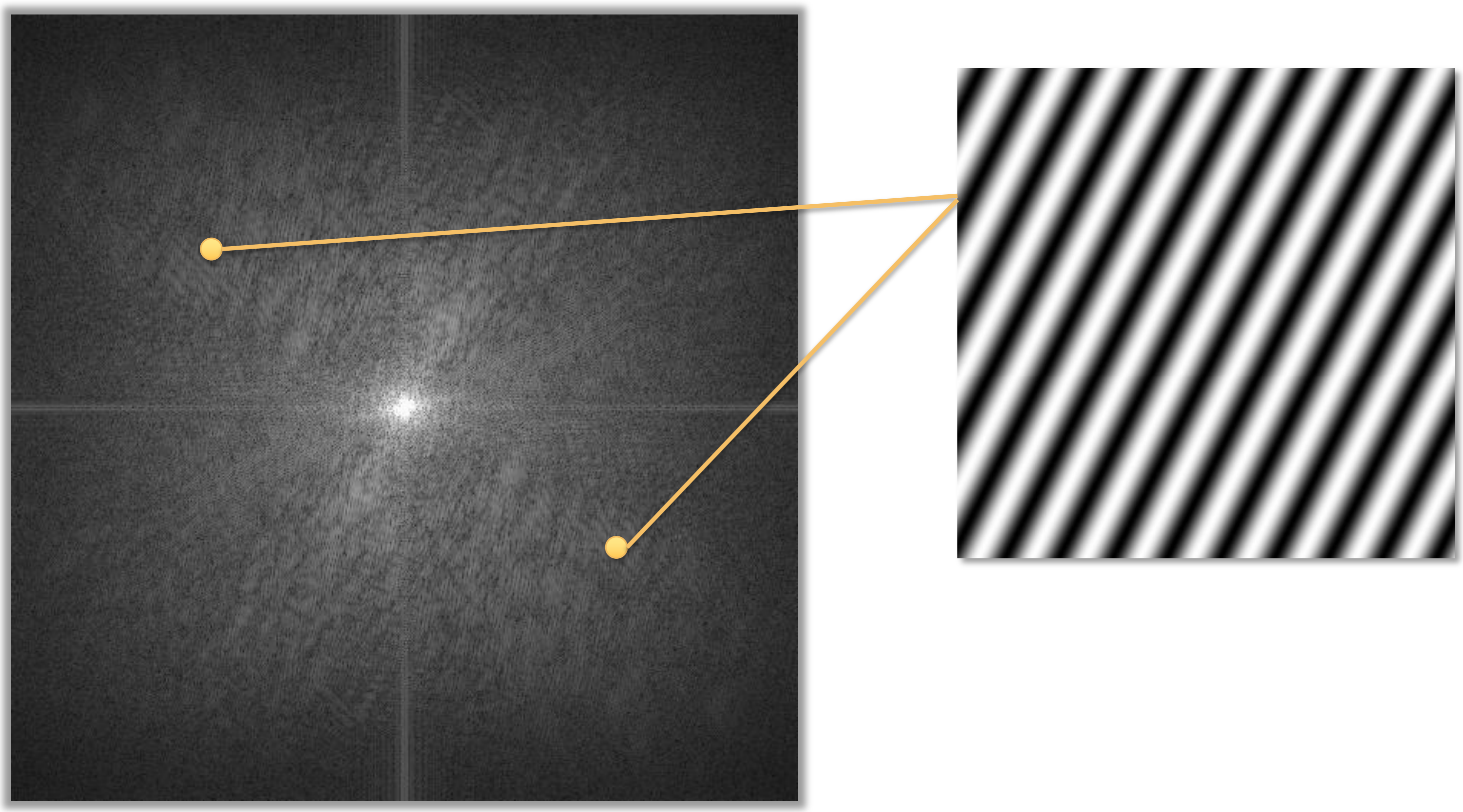
Fourier Transform

- What is this?



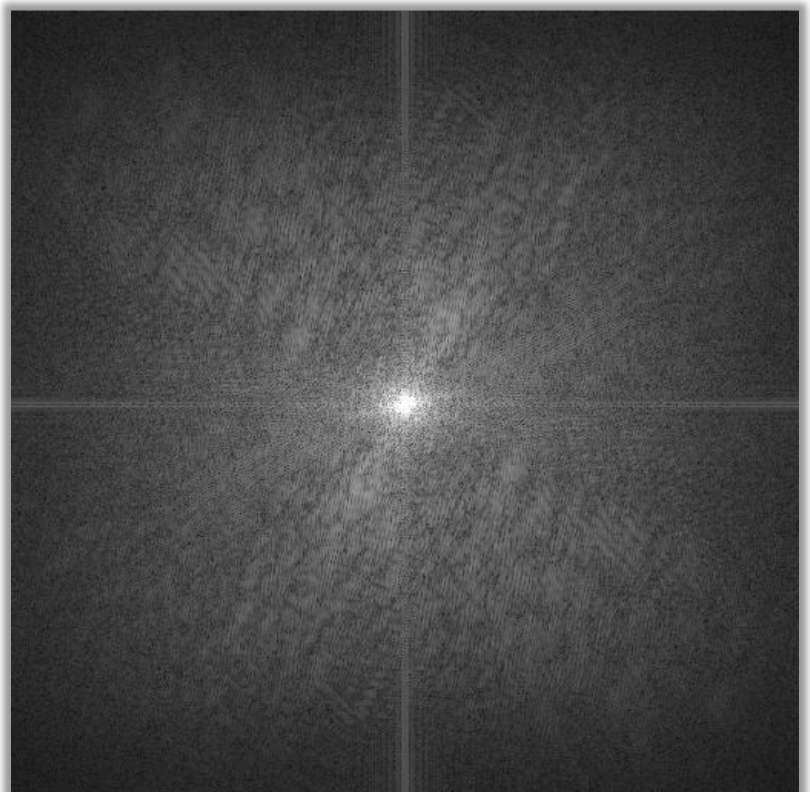
Fourier Transform

- What is this?

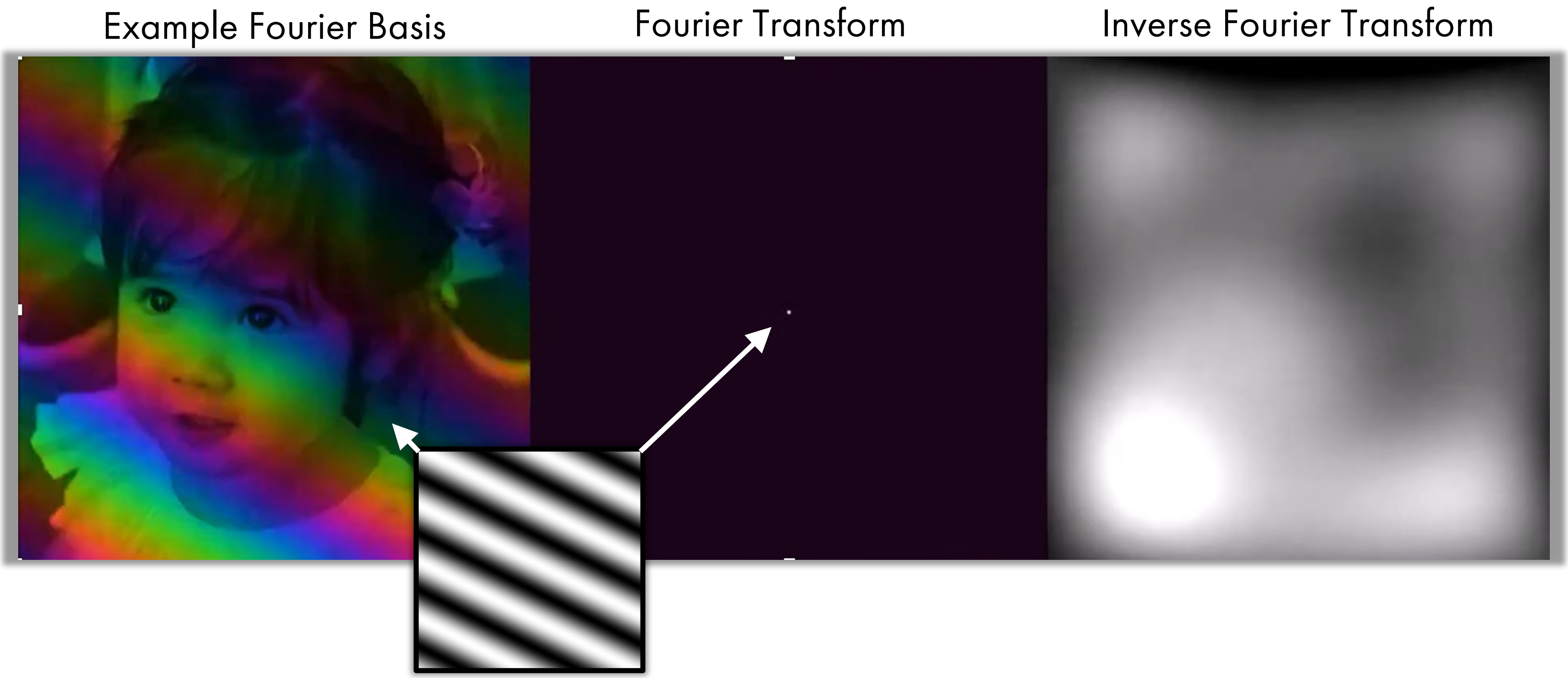


Fourier Transform

- What is this?



2D Fourier Transform



[Source: Youtube, Tyler Moore]

2D Fourier Transform

Example Fourier Basis | Fourier Transform | Inverse Fourier Transform

[Source: Youtube, Tyler Moore]

2D Fourier Transform

Example Fourier Basis | Fourier Transform | Inverse Fourier Transform

[Source: Youtube, Tyler Moore]

Fourier Transform

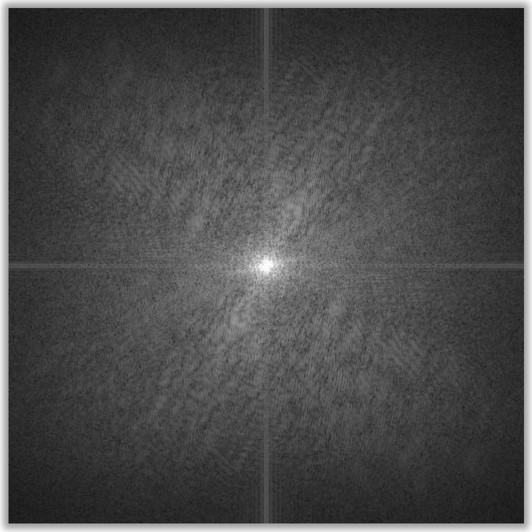
- any continuous, integrable function can be represented as an infinite sum of sines and cosines:

$$f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi) e^{2\pi i \xi x} \, d\xi \quad \longleftrightarrow \quad \hat{f}(\xi) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i \xi x} \, dx$$

Synthesize

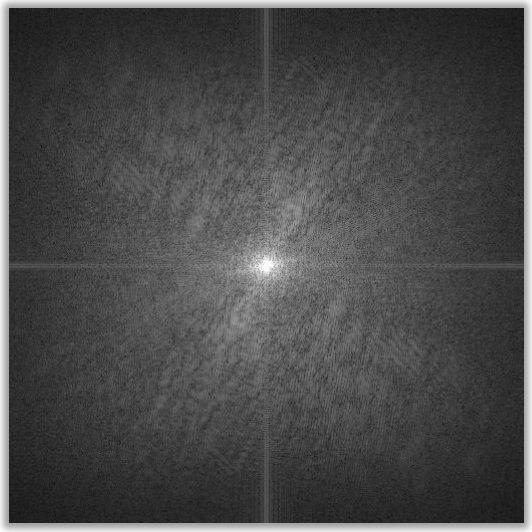
Decompose

Fourier Transform



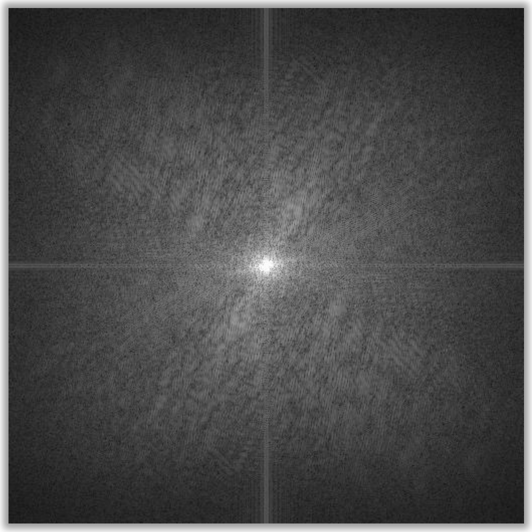
$$f(x,y) = \int_{-\infty}^{\infty} F(k_x, k_y) e^{2\pi i (k_x x + k_y y)} \mathrm{d}k_x \mathrm{d}k_y$$

Fourier Transform



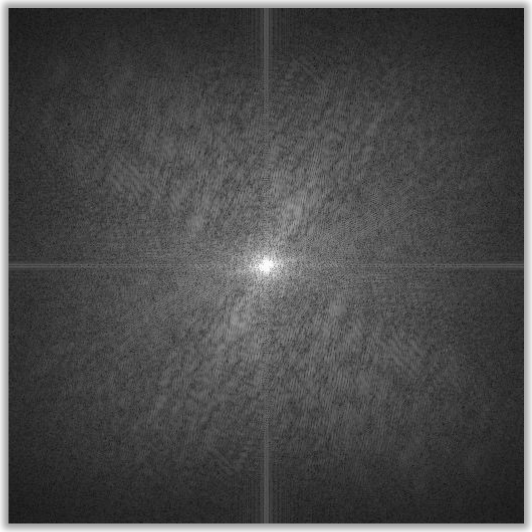
$$f(x,y) = \int_{-\infty}^{\infty} F(k_x, k_y) e^{2\pi i (k_x x + k_y y)} \mathrm{d}k_x \mathrm{d}k_y$$

$$\cos(2\pi[k_x x + k_y y]) + j \sin(2\pi[k_x x + k_y y])$$



Fourier Transform

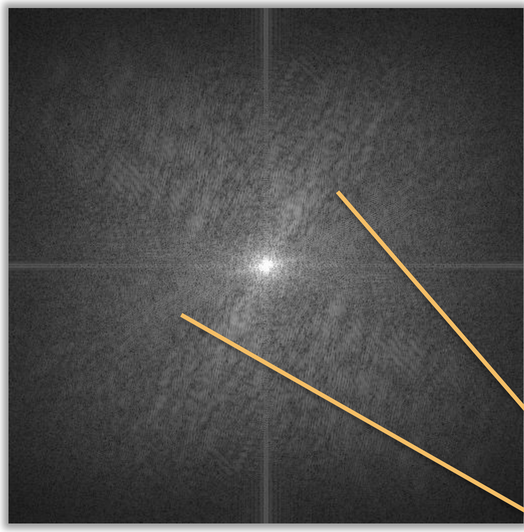
$$f(x,y) = \int_{-\infty}^{\infty} \underbrace{F(k_x, k_y)}_{Ae^{j\phi}} e^{2\pi i(k_x x + k_y y)} \mathrm{d}k_x \mathrm{d}k_y$$



Fourier Transform

$$f(x,y) = \int_{-\infty}^{\infty} \underbrace{F(k_x, k_y) e^{2\pi i (k_x x + k_y y)}} \mathrm{d}k_x \mathrm{d}k_y$$

$$A\cos(2\pi[k_x x + k_y y] + \phi) + jA\sin(2\pi[k_x x + k_y y] + \phi)$$



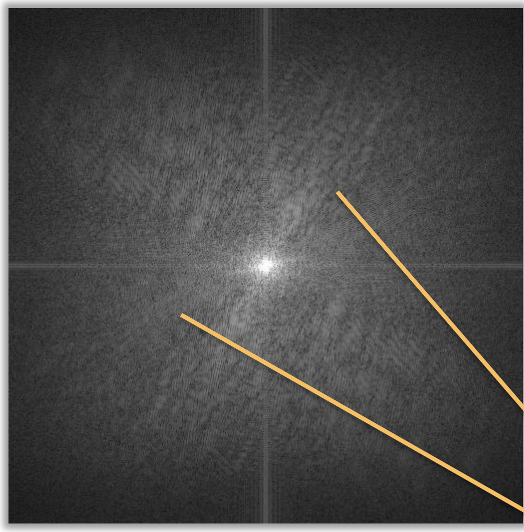
Fourier Transform

$$f(x,y) = \int_{-\infty}^{\infty} \underbrace{F(k_x, k_y) e^{2\pi i (k_x x + k_y y)}} \mathrm{d}k_x \mathrm{d}k_y$$

$$A\cos(2\pi[k_x x + k_y y] + \phi) + jA\sin(2\pi[k_x x + k_y y] + \phi)$$

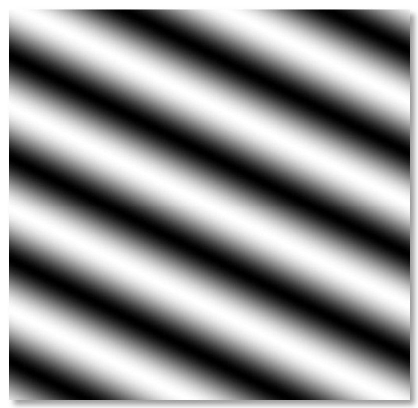
Fourier coefficients of real signals are conjugate symmetric

Fourier Transform



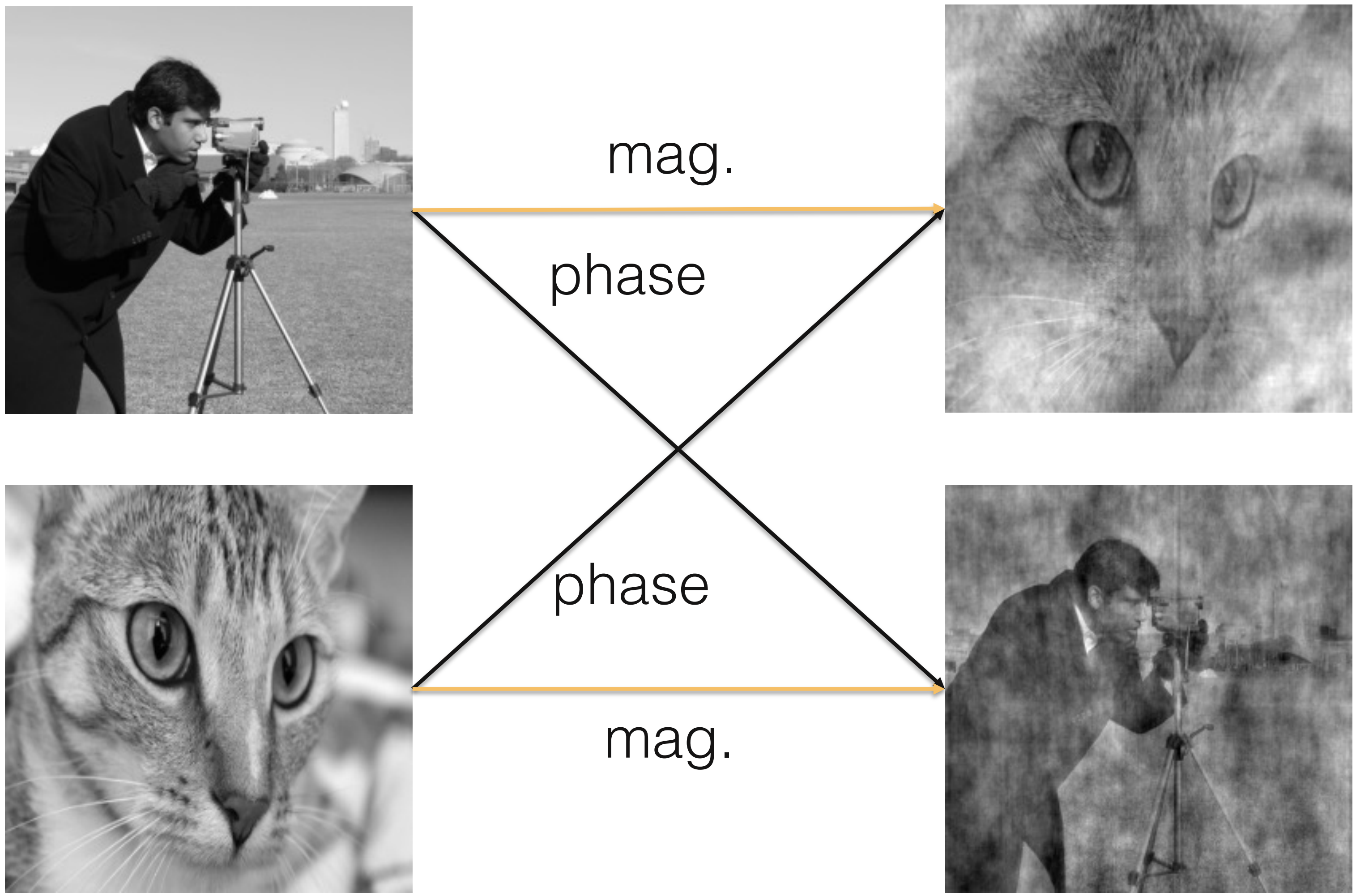
$$f(x, y) = \int_{-\infty}^{\infty} \underbrace{F(k_x, k_y) e^{2\pi i (k_x x + k_y y)}}_{} \mathrm{d}k_x \mathrm{d}k_y$$

$$A\cos(2\pi[k_x x + k_y y] + \phi) + jA\sin(2\pi[k_x x + k_y y] + \phi)$$



Images are sums of cosines at different amplitudes, phases, spatial frequencies

Magnitude vs Phase



Fourier Transform

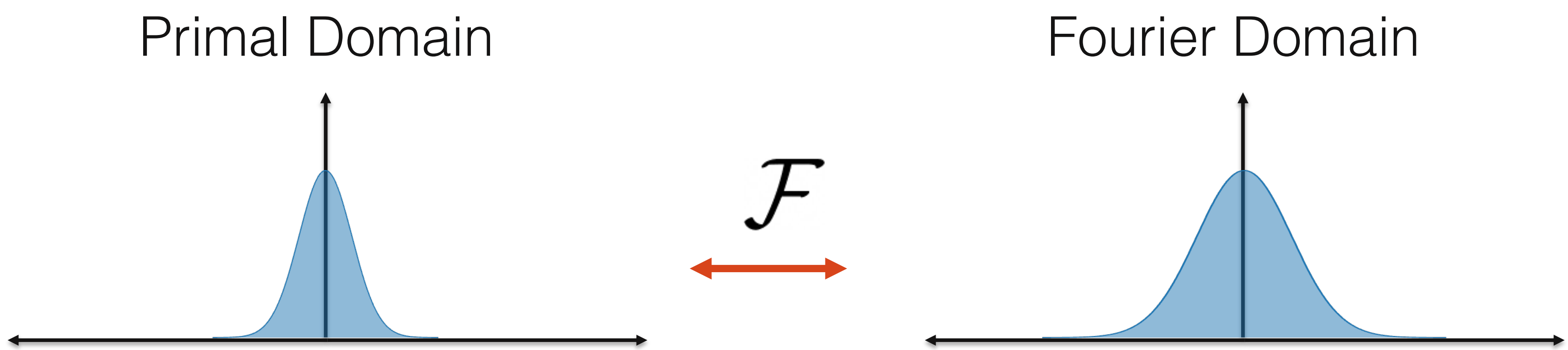
- any continuous, integrable, periodic function can be represented as an infinite sum of sines and cosines:

$$f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi) e^{2\pi i \xi x} \, d\xi \quad \longleftrightarrow \quad \hat{f}(\xi) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i \xi x} \, dx$$

- convolution theorem (critical):

$$x * g = F^{-1}\left\{F\left\{x\right\} \cdot F\left\{g\right\}\right\}$$

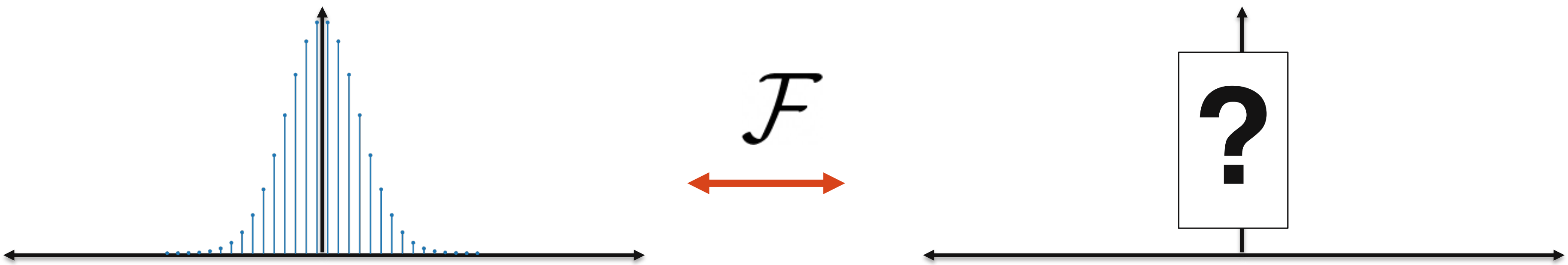
Discrete vs Continuous Fourier Transform



Sampling

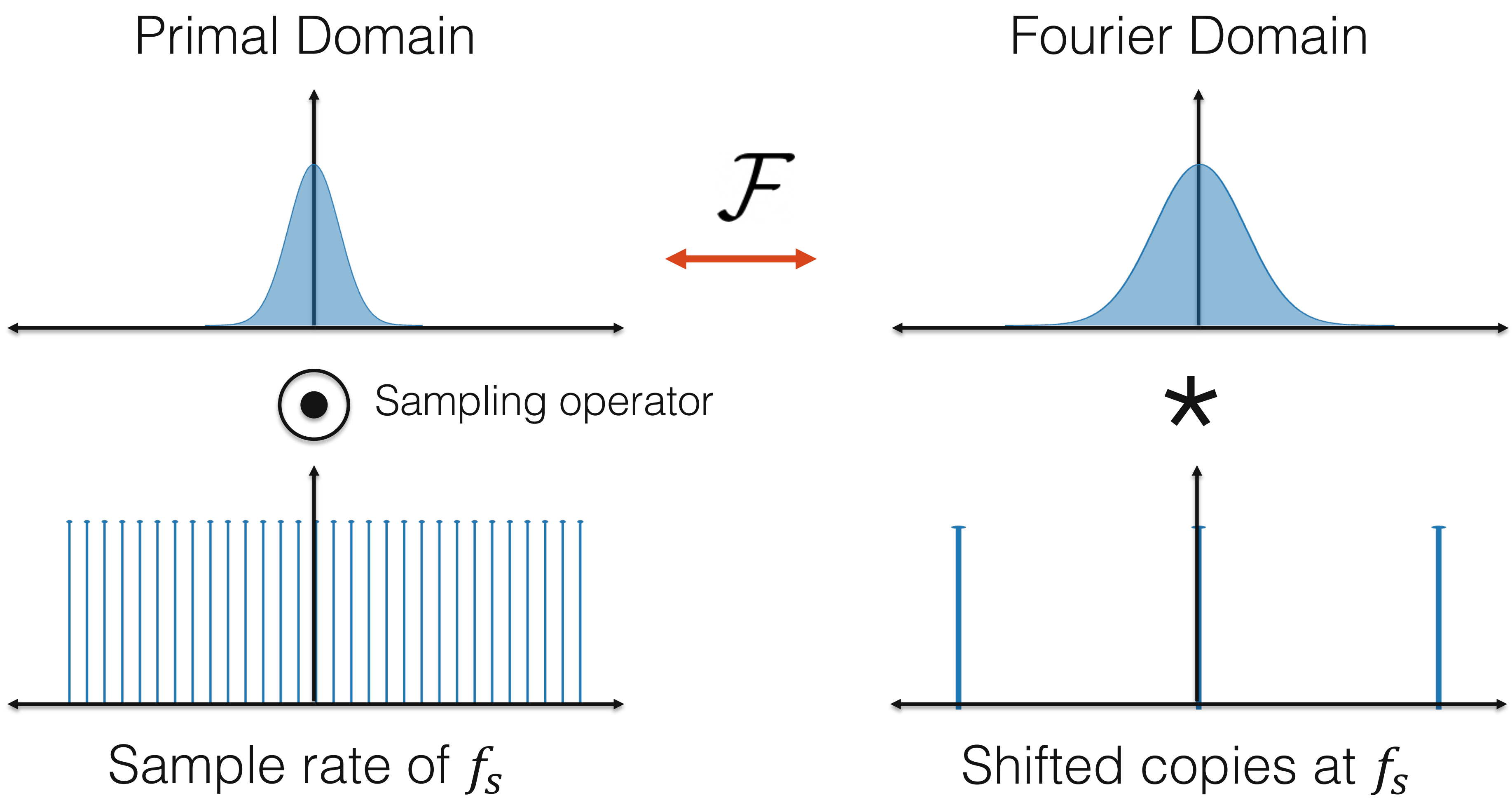
Primal Domain

Fourier Domain

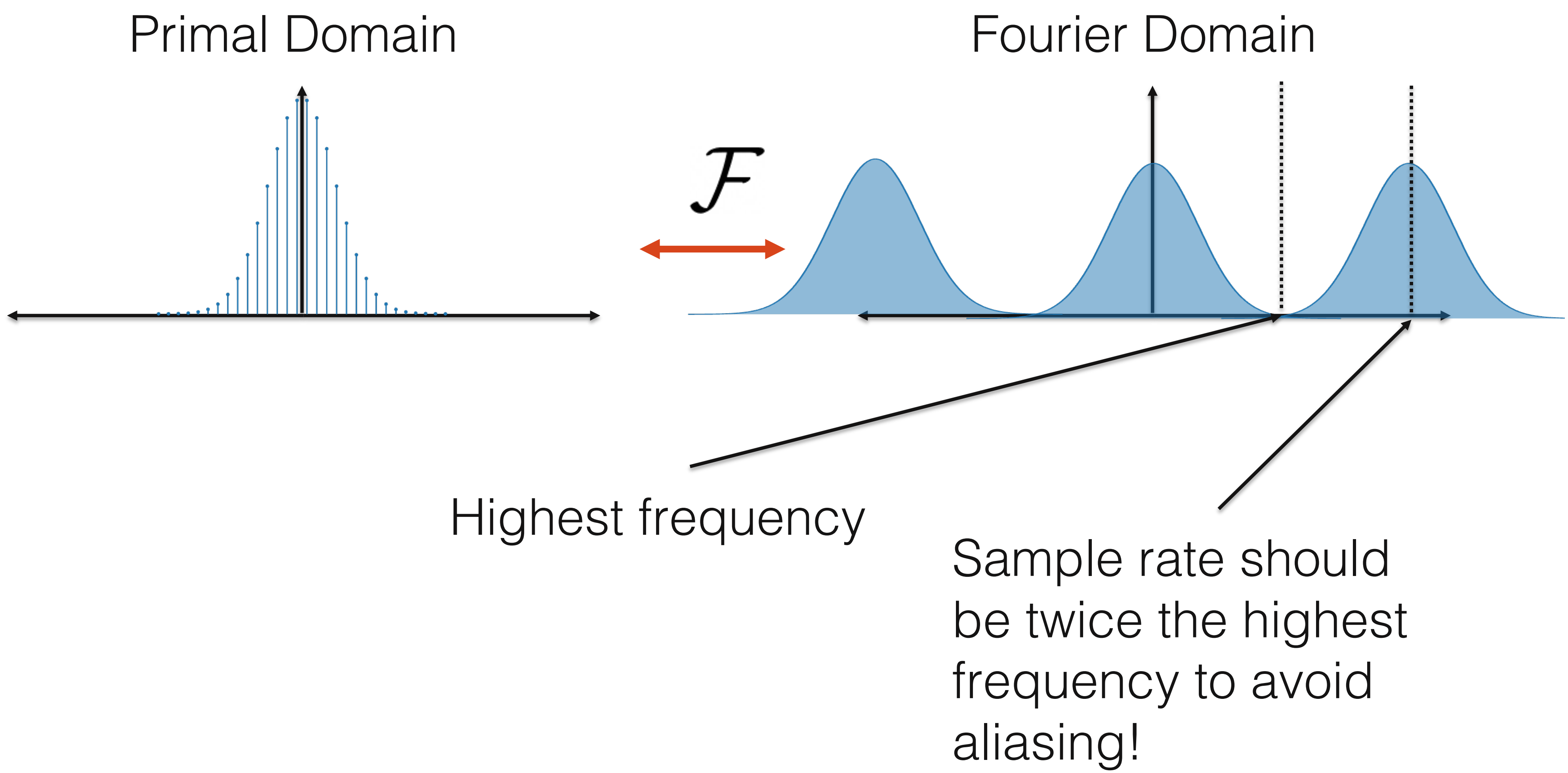


discrete sampled signal

Sampling

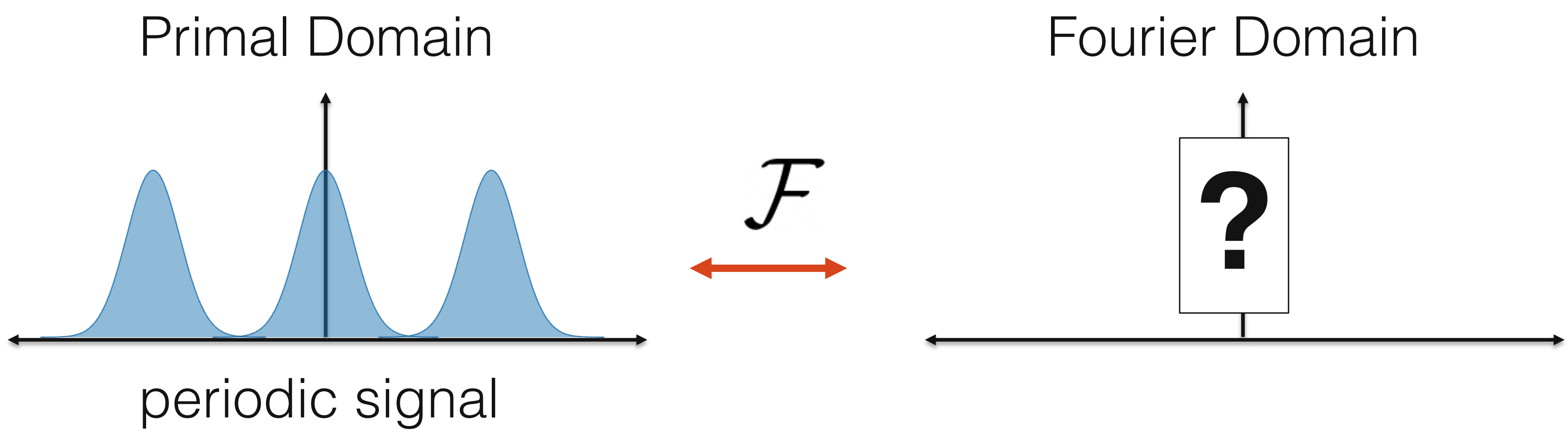

Primal Domain
Fourier Domain
$\mathcal{F}$
Sampling operator
*
Sample rate of f_s
Shifted copies at f_s

Sampling

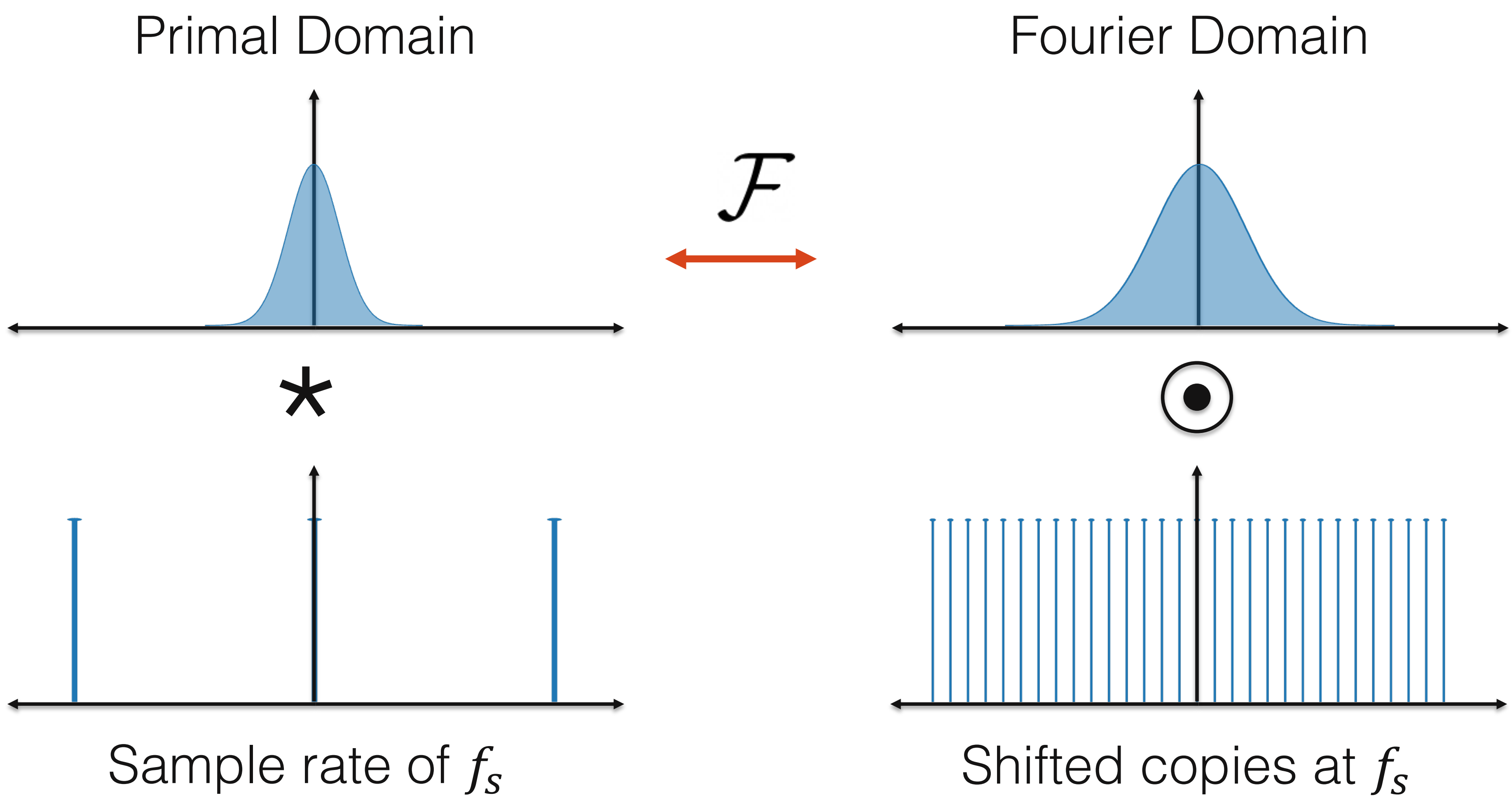


Primal Domain
Fourier Domain
Highest frequency
Sample rate should be twice the highest frequency to avoid aliasing!

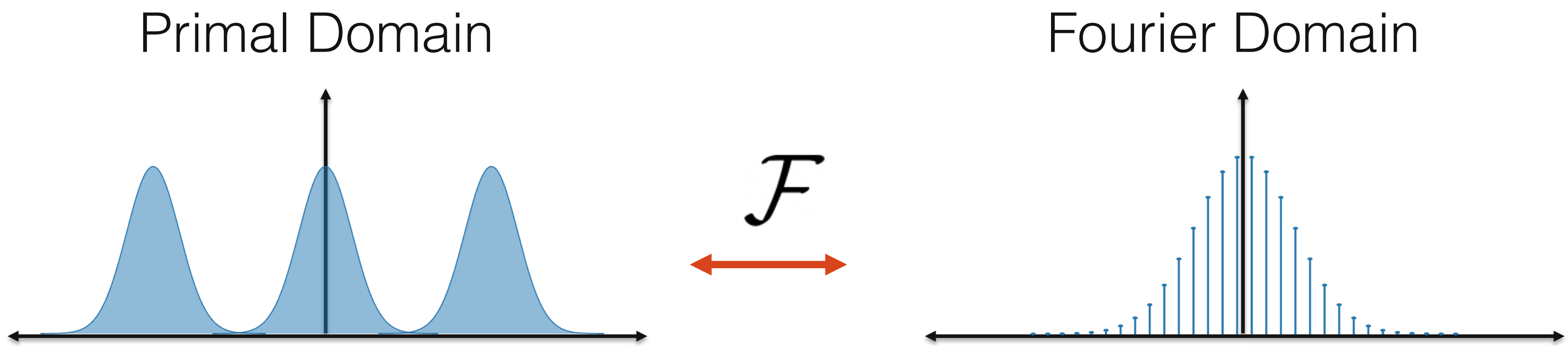
Periodicity



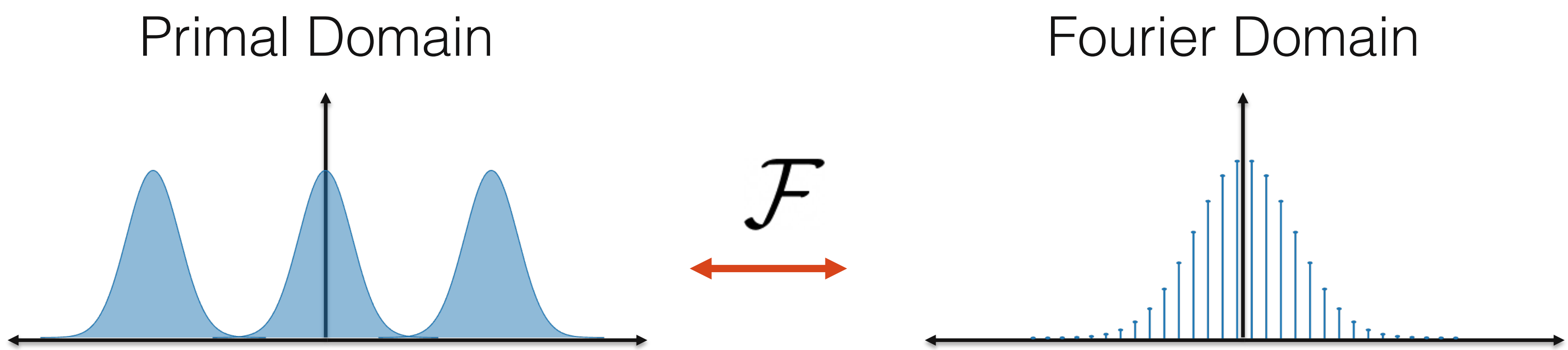
Periodicity



Periodicity



Periodicity



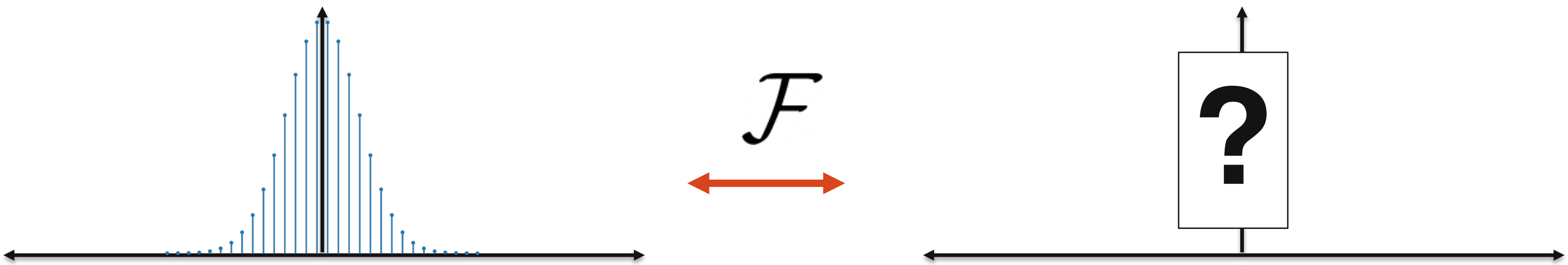
A periodic signal can be represented by a discrete set of Fourier coefficients

- These are called the “Fourier series coefficients”

Discrete Fourier Transform

Primal Domain

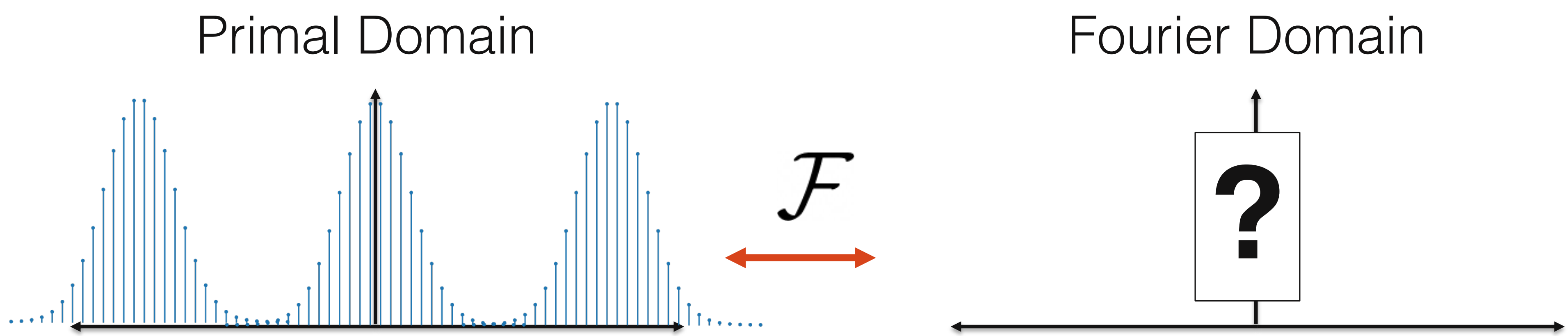
Fourier Domain



In practice, we wish to take the Fourier transform of discrete signals.

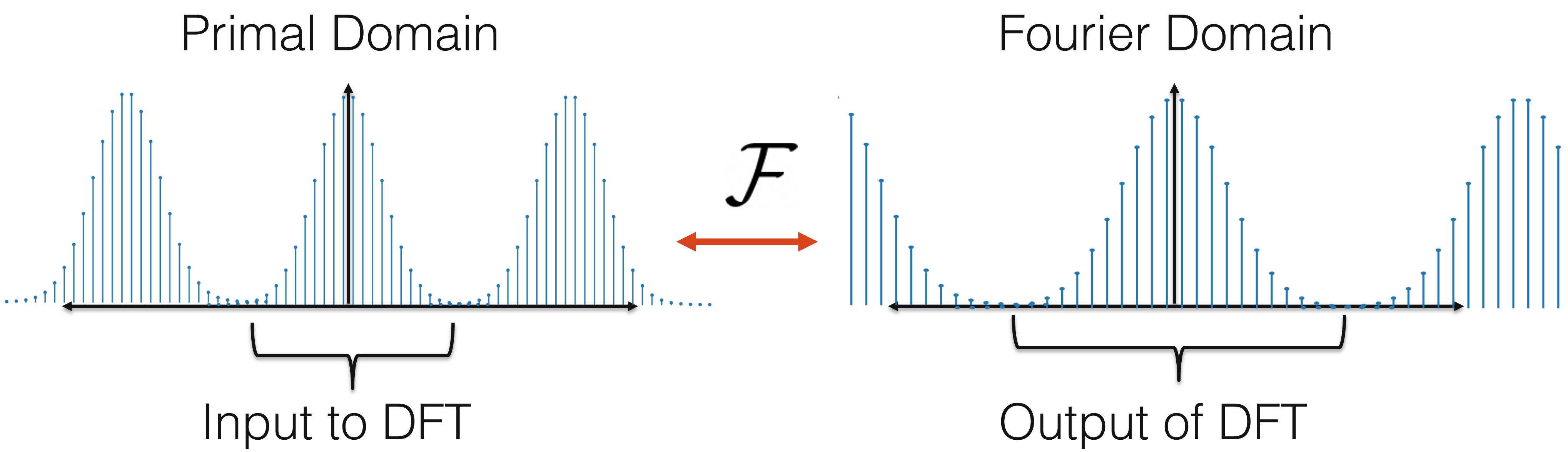
But we need to represent the Fourier domain with discrete values, too!

Discrete Fourier Transform



Assume the primal domain signal is periodic

Discrete Fourier Transform



Assume the primal domain signal is periodic

Discrete Fourier Transform

- most important for us: discrete Fourier transform

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} \hat{x}[k] e^{2\pi ikn/N} \quad \longleftrightarrow \quad \hat{x}[k] = \sum_{n=0}^{N-1} x[n] e^{-2\pi ikn/N}$$

Discrete Fourier Transform

An Algorithm for the Machine Calculation of Complex Fourier Series

By James W. Cooley and John W. Tukey

An efficient method for the calculation of the interactions of a 2^m factorial experiment was introduced by Yates and is widely known by his name. The generalization to 3^m was given by Box et al. [1]. Good [2] generalized these methods and gave elegant algorithms for which one class of applications is the calculation of Fourier series. In their full generality, Good's methods are applicable to certain problems in which one must multiply an N-vector by an $N \times N$ matrix which can be factored into m sparse matrices, where m is proportional to $\log N$. This results in a procedure requiring a number of operations proportional to $N \log N$ rather than N^2. These methods are applied here to the calculation of complex Fourier series. They are useful in situations where the number of data points is, or can be chosen to be, a highly composite number. The algorithm is here derived and presented in a rather different form. Attention is given to the choice of N. It is also shown how special advantage can be obtained in the use of a binary computer with $N = 2^m$ and how the entire calculation can be performed within the array of N data storage locations used for the given Fourier coefficients.

Fast Fourier Transform: Cooley & Tukey 1965

Discrete Fourier Transform

An Algorithm for the Machine Calculation of Complex Fourier Series

By James W. Cooley and John W. Tukey

An efficient method for the calculation of the interactions of a 2^m factorial experiment was introduced by Yates and is widely known by his name. The generalization to 3^m was given by Box et al. [1]. Good [2] generalized these methods and gave elegant algorithms for which one class of applications is the calculation of Fourier series. In their full generality, Good's methods are applicable to certain problems in which one mu n be factored into m sparse n a procedure requiring a n an N^2. These methods are applied here to the calculation of complex Fourier series. They are useful in situations where the number of data points is, or can be chosen to be, a highly composite number. The algorithm is here derived and presented in a rather different form. Attention is given to the choice of N. It is also shown how special advantage can be obtained in the use of a binary computer with $N = 2^m$ and how the entire calculation can be performed within the array of N data storage locations used for the given Fourier coefficients.

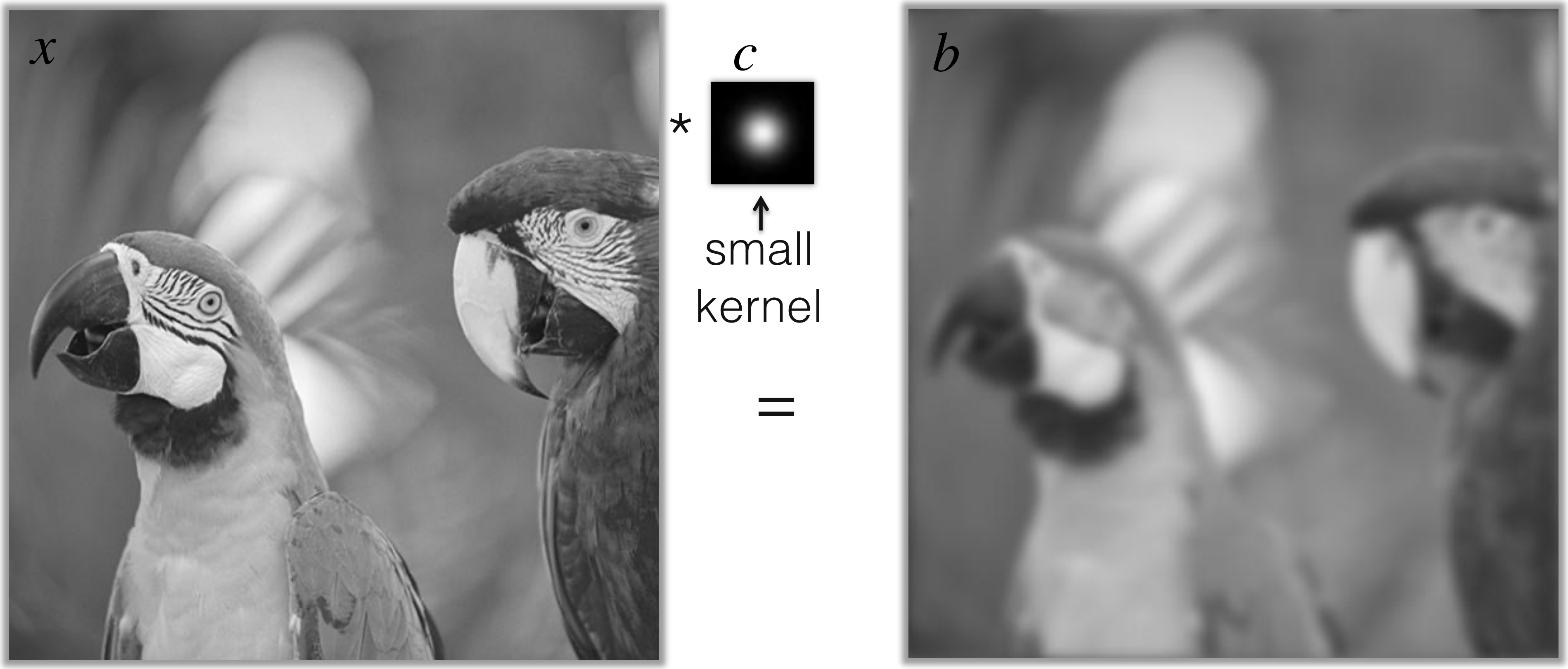
$O(N^2) \rightarrow O(N \log N)$

Fast Fourier Transform: Cooley & Tukey 1965

Filter Examples

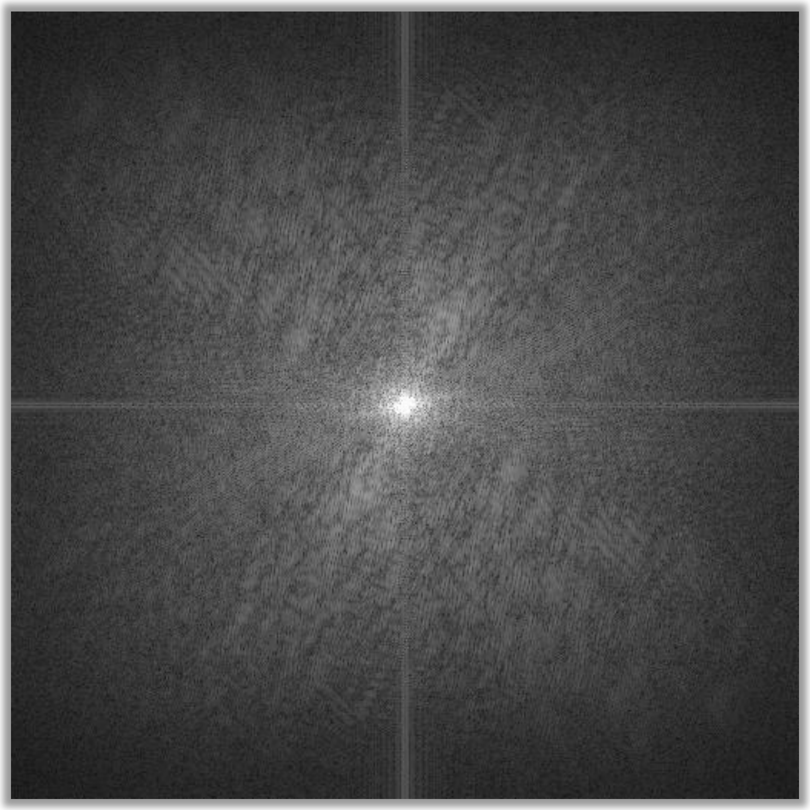
Filtering – Low-pass Filter

- low-pass filter: convolution in primal domain $b = x * c$
- convolution kernel c is also known as point spread function (PSF)

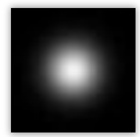


Filtering – Low-pass Filter

- low-pass filter: multiplication in frequency domain $F\{b\} = F\{x\} \cdot F\{c\}$

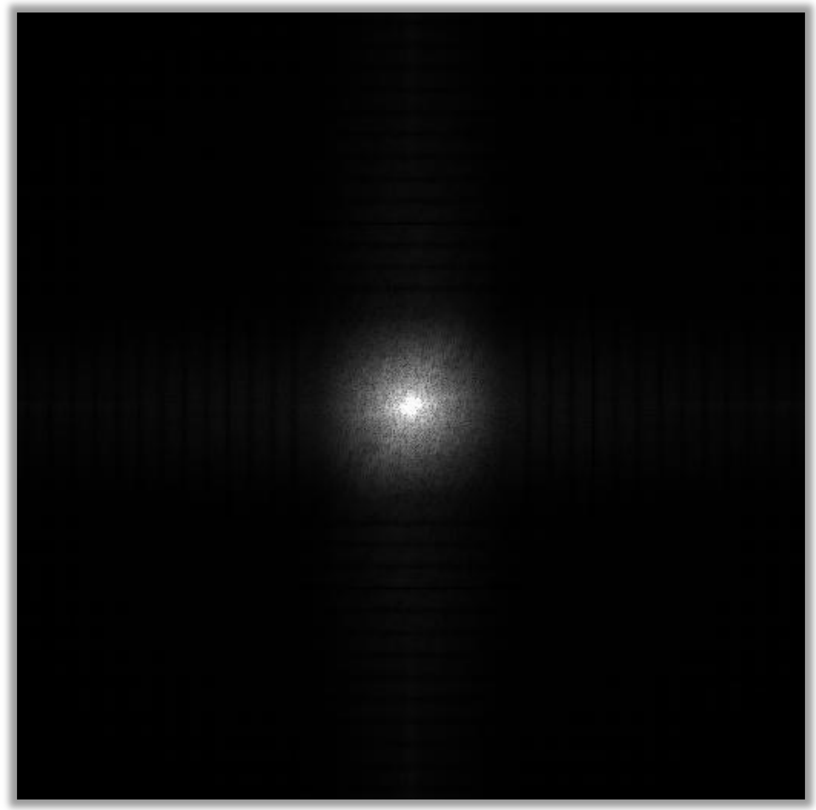


$\cdot$



↑ big

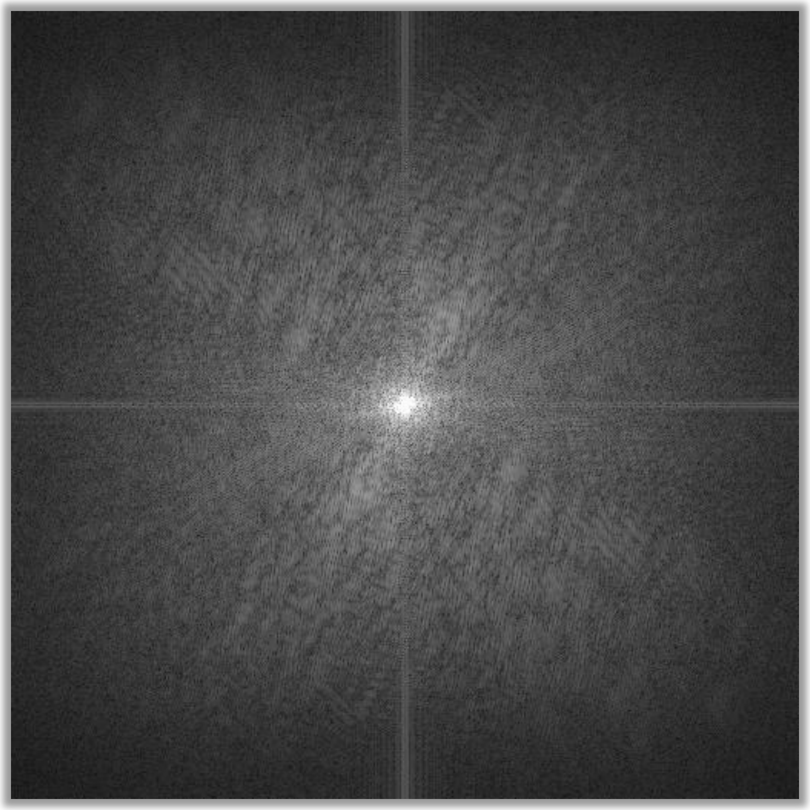
=



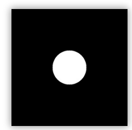
Filtering – Low-pass Filter

$$F\{b\} = F\{x\} \cdot F\{c\}$$

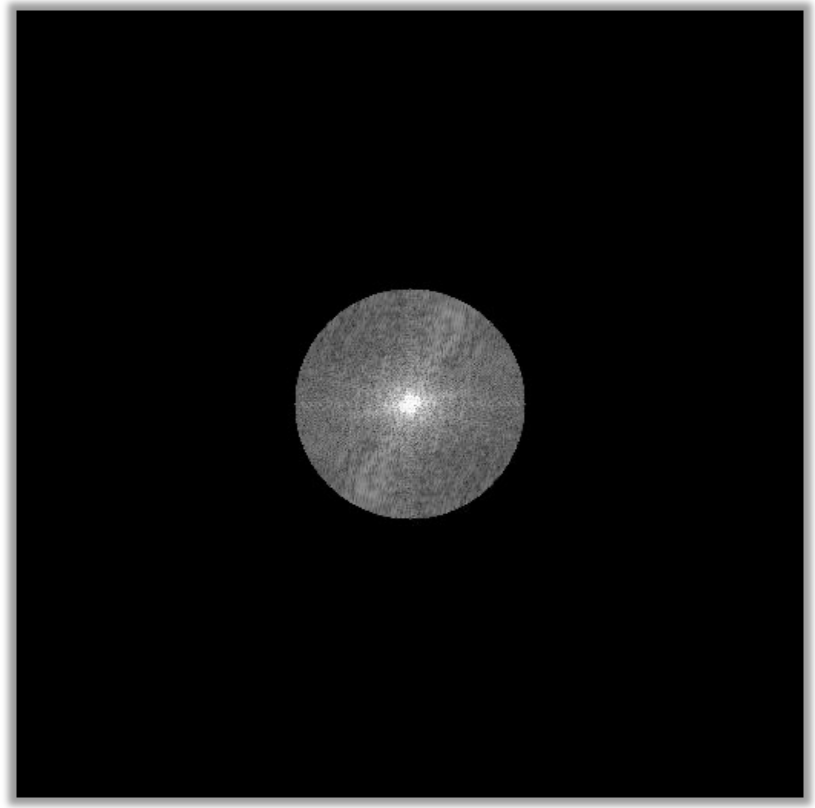
- low-pass filter: hard cutoff



$\cdot$

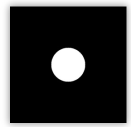


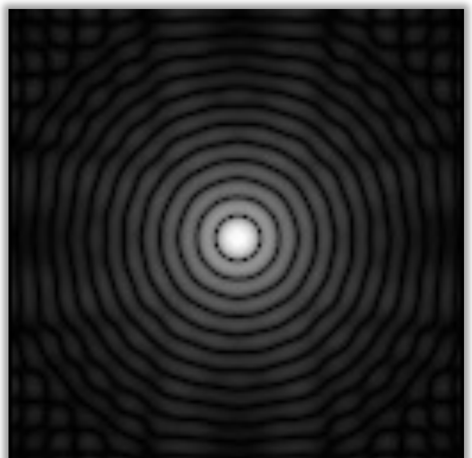
=



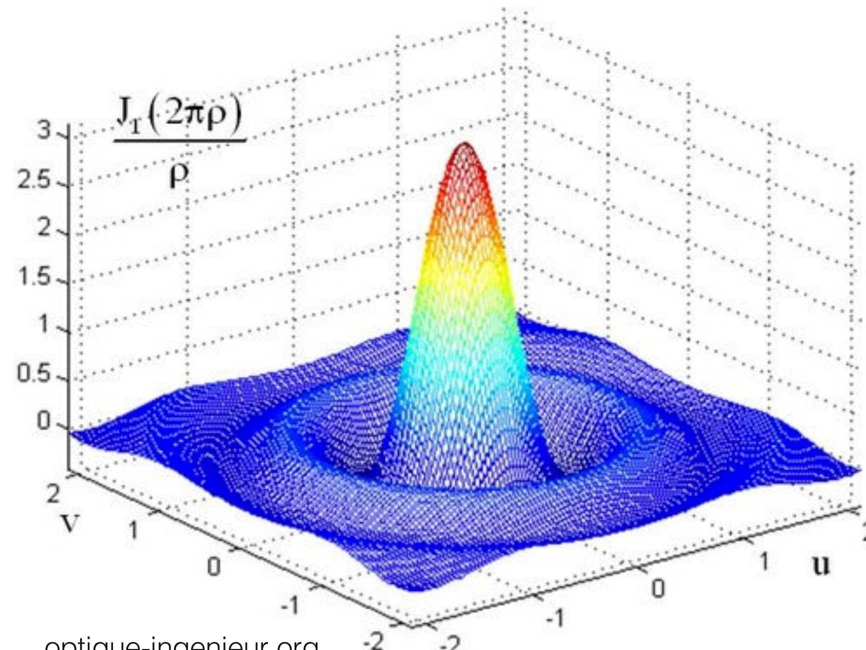
Filtering – Low-pass Filter

- Bessel function of the first kind or "jinc"

$F^{-1}\{$  $\}$



imagemagick.org

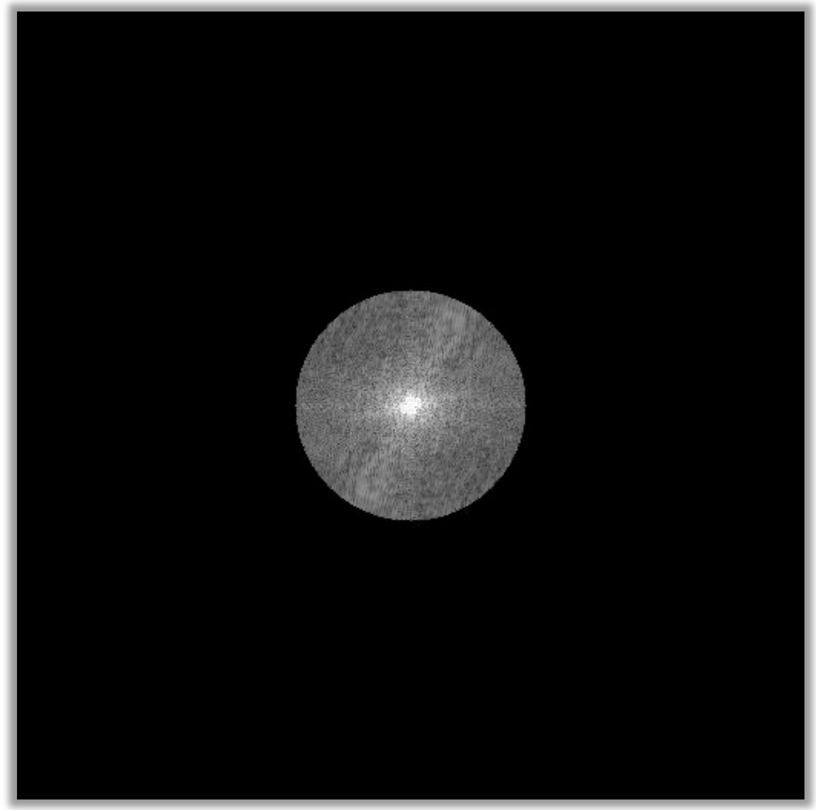


optique-ingenieur.org

Filtering – Low-pass Filter

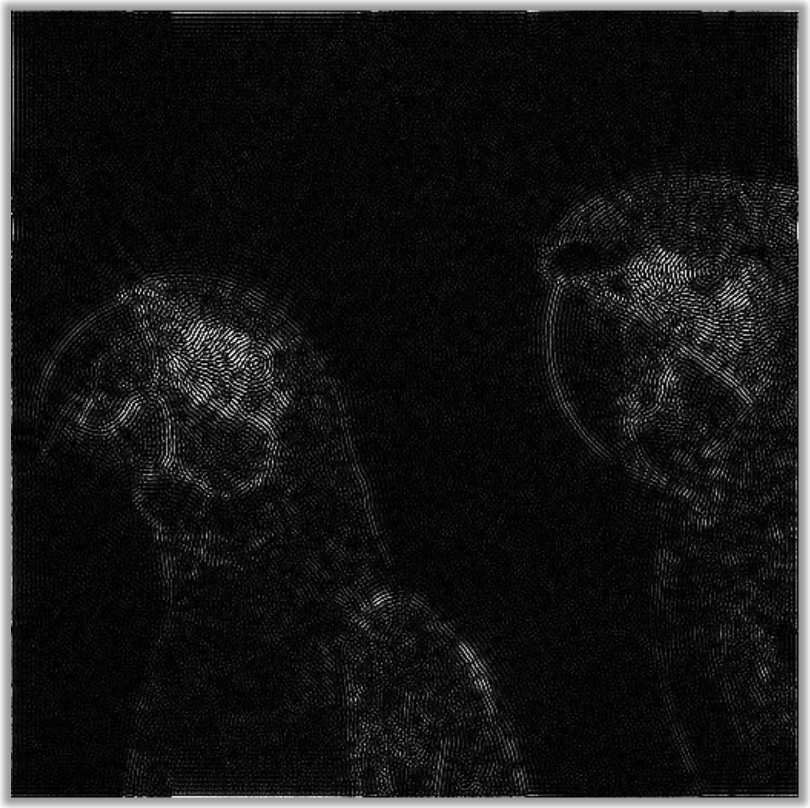
- hard frequency filters often introduce ringing

←

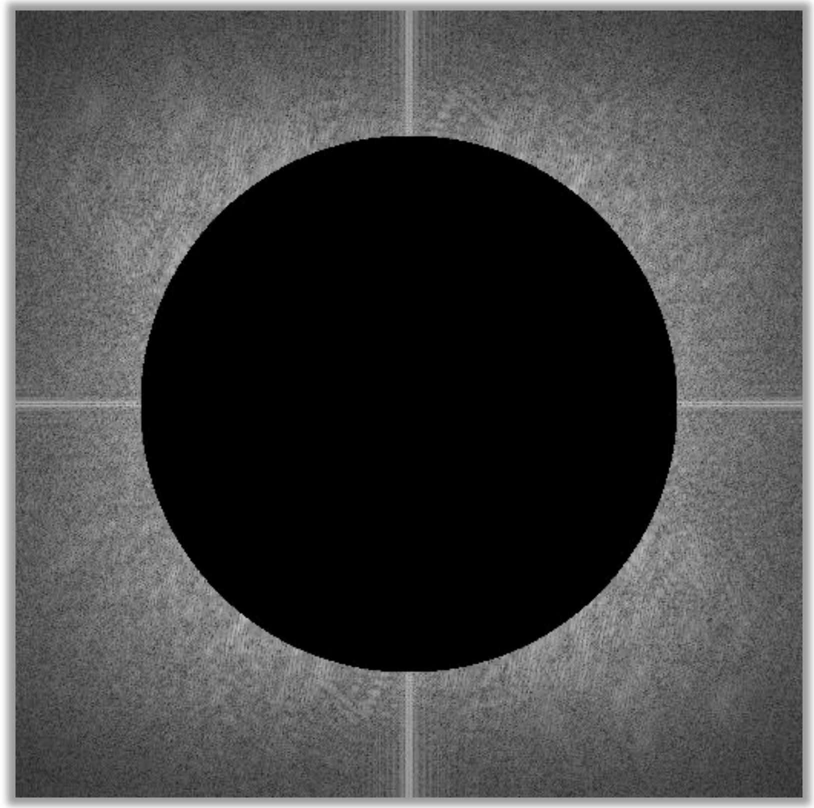


Filtering – High-pass Filter

- sharpening (possibly with ringing)



←



Filtering – Unsharp Masking

- sharpening (without ringing): unsharp masking, e.g. in Photoshop

$$b = x * (\delta - c_{lowpass\_gauss}) = x - x * c_{lowpass\_gauss}$$

or

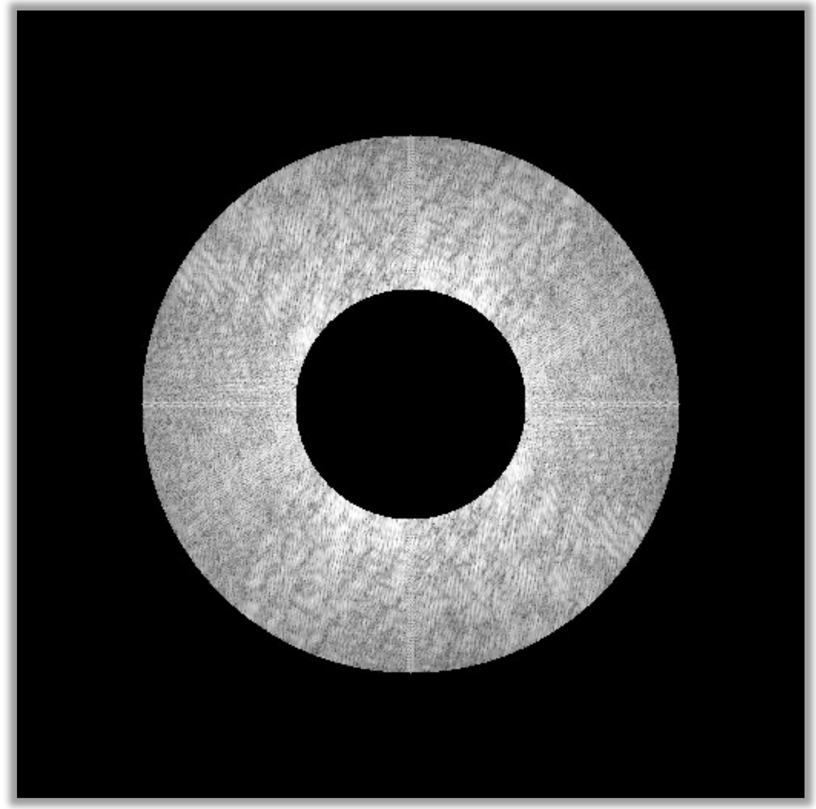
$$b = x * (\delta + c_{highpass}) = x + x * c_{highpass}$$

Filtering – Unsharp Masking

- sharpening (without ringing): unsharp masking, e.g. in Photoshop

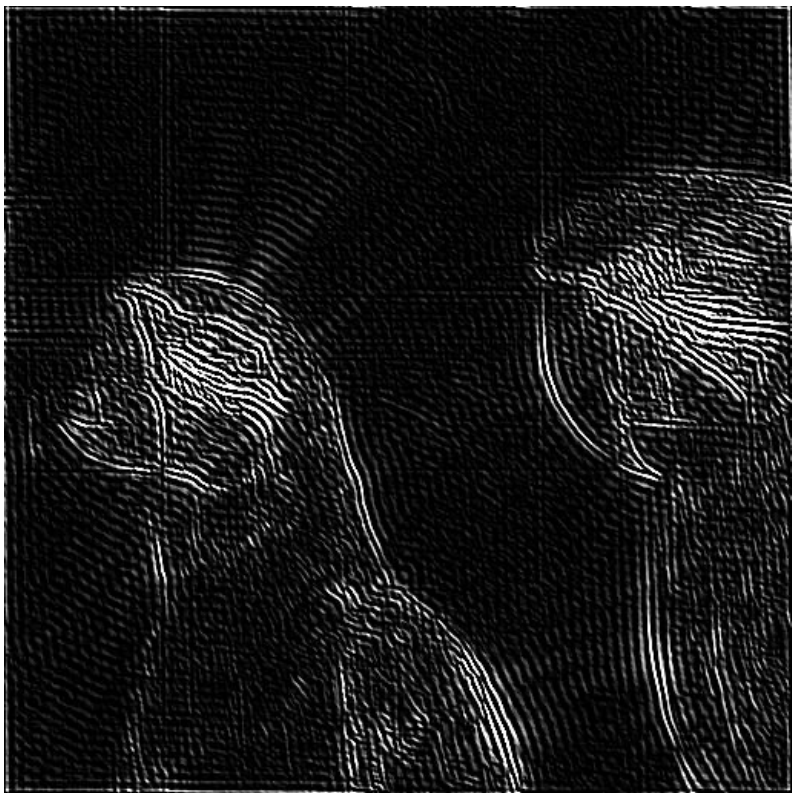
Filtering – Band-pass Filter

←

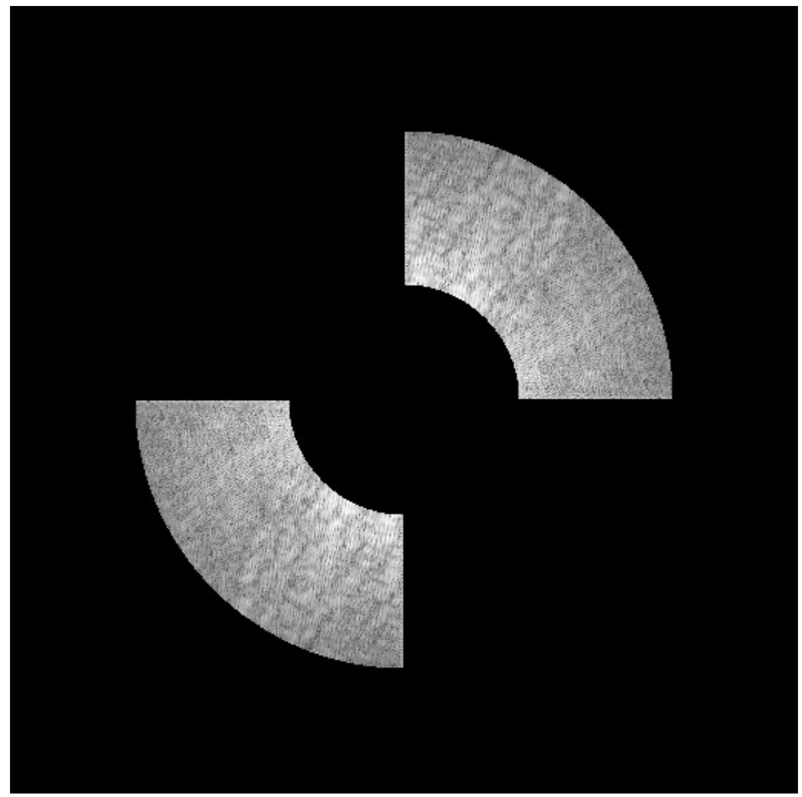


Filtering – Oriented Band-pass Filter

- edges with specific orientation (e.g., hat) are gone!



←



Edge Detection

Finding Waldo

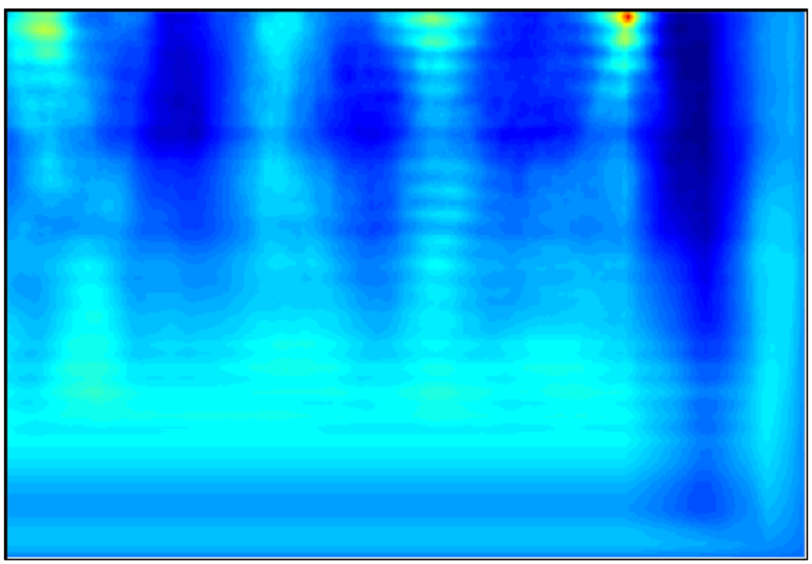
- Let's revisit the problem of finding Waldo
- And let's take a simple example

image

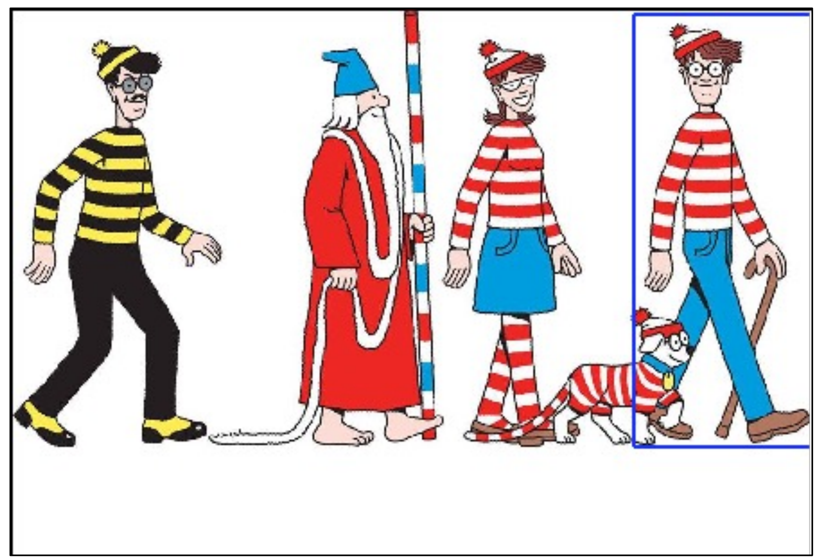
Template(filter)

Finding Waldo

- Let's revisit the problem of finding Waldo
- And let's take a simple example



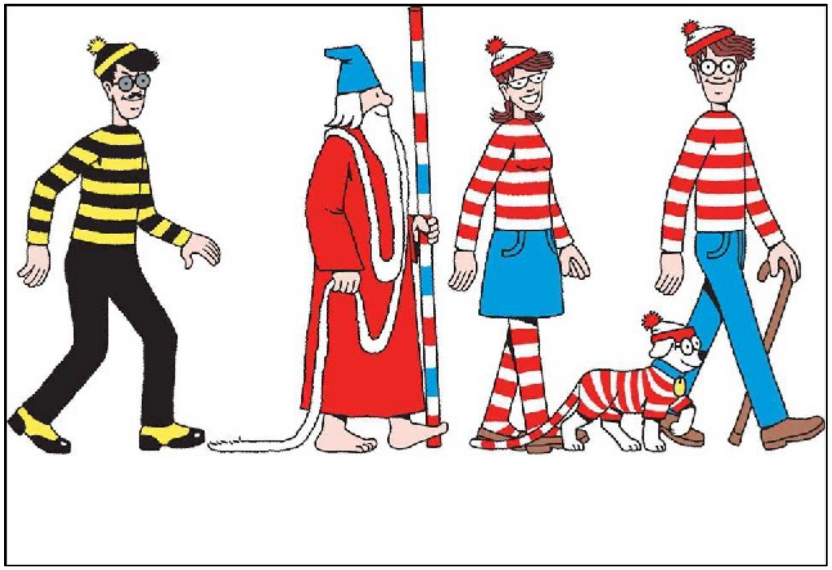
normalized cross-correlation



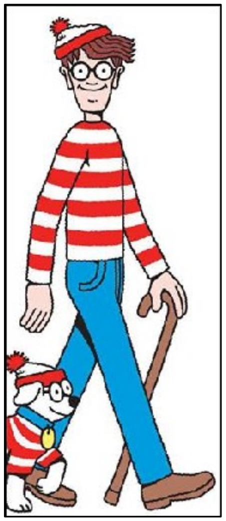
Waldo detection
(putting box around max response)

Finding Waldo

- Let's revisit the problem of finding Waldo
- And let's take a simple example



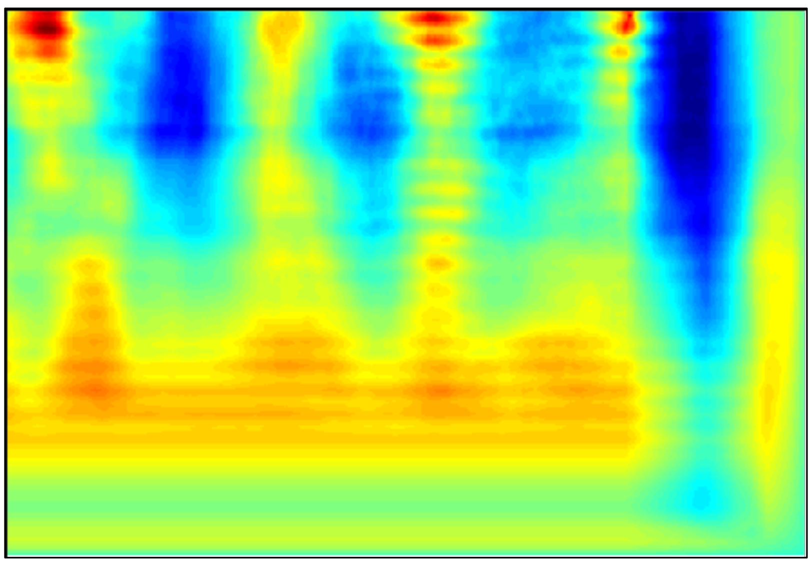
image



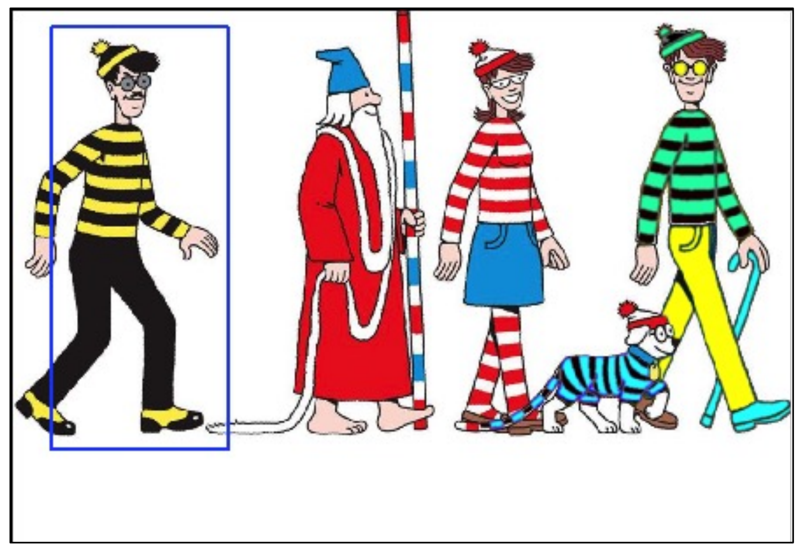
Template(filter)

Finding Waldo

- Now imagine Waldo goes shopping (and the dog too)
- ... but our filter doesn't know that



normalized cross-correlation



Waldo detection
(putting box around max response)

Finding Waldo (again)

- What can we do to find Waldo again?

Finding Waldo (again)

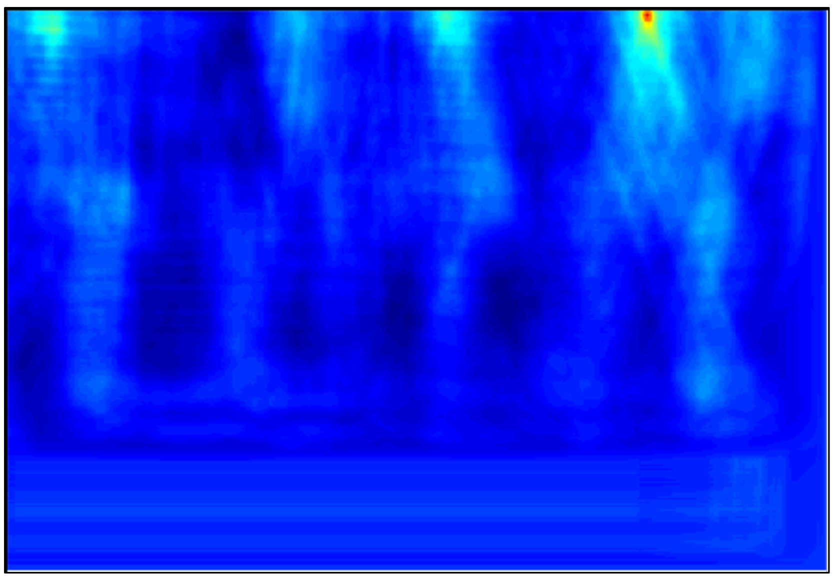
- What can we do to find Waldo again?
- Edges!!!

image

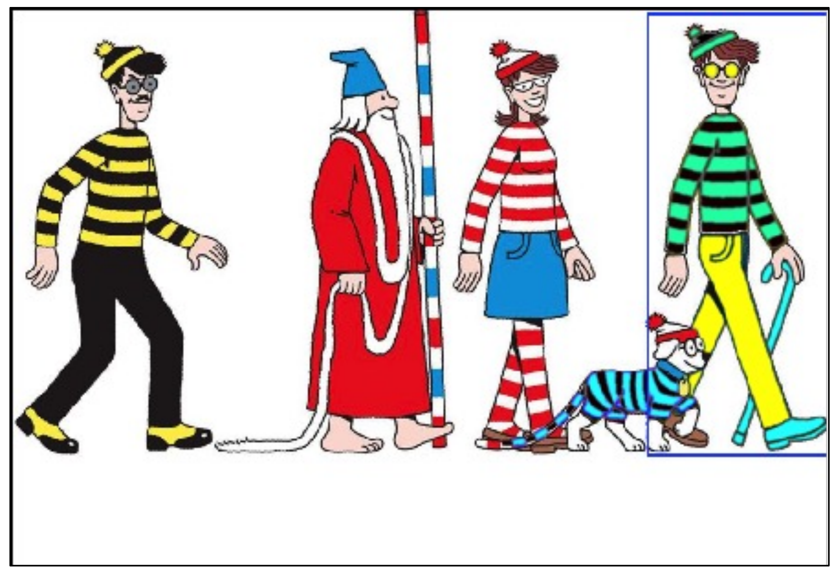
Template(filter)

Finding Waldo (again)

- What can we do to find Waldo again?
- Edges!!!

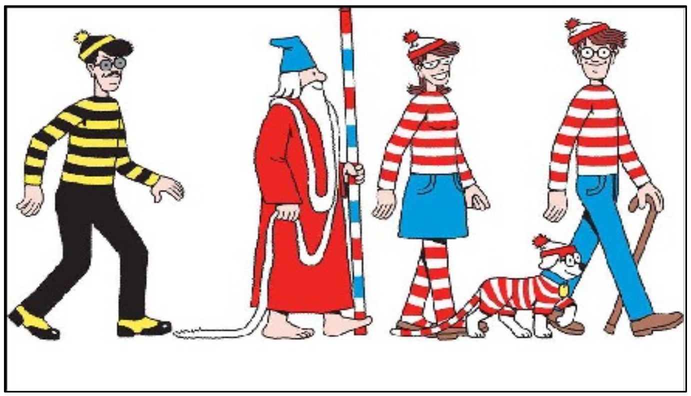


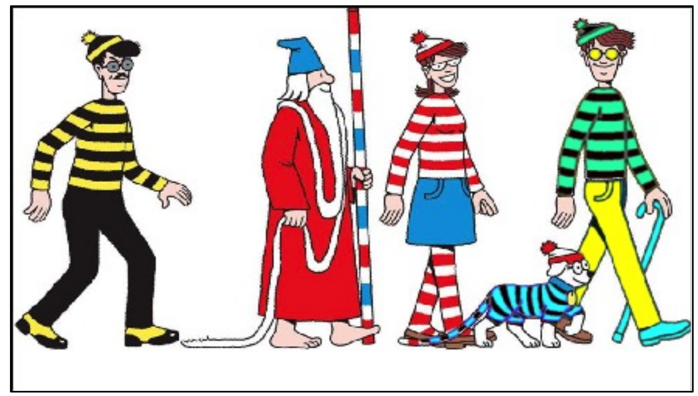
normalized cross-correlation
(using the edge maps)



Waldo detection
(putting box around max response)

Waldo and Edges





Edge detection

- Map image to a set of curves or line segments or contours.
- More compact than pixels.
- Edges are invariant to changes in illumination
- Important for recognition

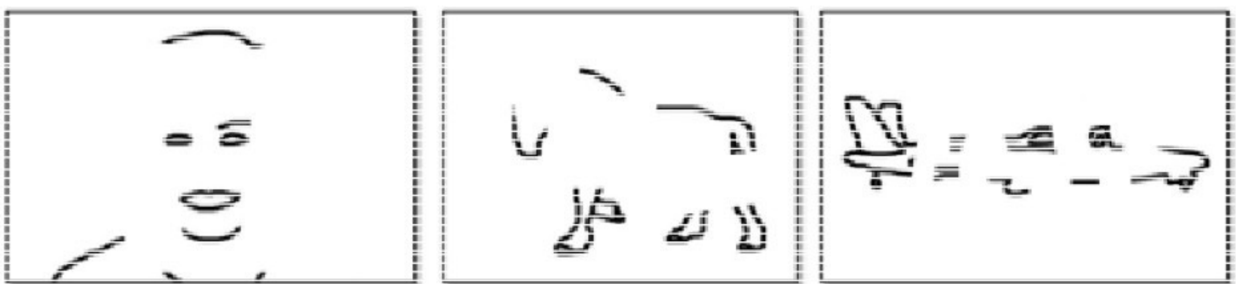


Figure: [Shotton et al. PAMI, 07]

[Source: K. Grauman]

Edge detection

- Map image to a set of curves or line segments or contours.
- More compact than pixels.
- Edges are invariant to changes in illumination
- Important for recognition
- Important for various applications

Figure: [Shotton et al. PAMI, 07]

Edge detection

- Map image to a set of curves or line segments or contours.
- More compact than pixels.
- Edges are invariant to changes in illumination
- Important for recognition
- Important for various applications

Figure: How can a robot pick up or grasp objects?

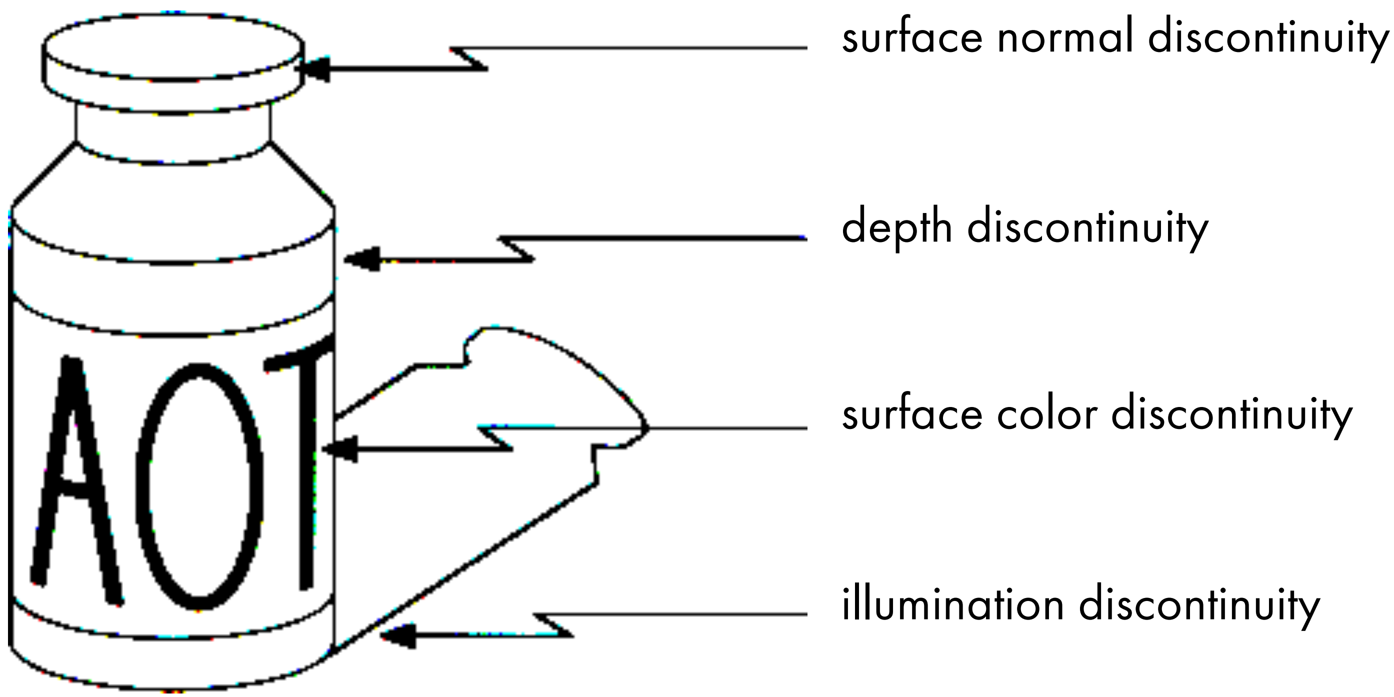
Edge detection

- Map image to a set of curves or line segments or contours.
- More compact than pixels.
- Edges are invariant to changes in illumination
- Important for recognition
- Important for various applications

Figure: How can a robot pick up or grasp objects?

Origin of Edges

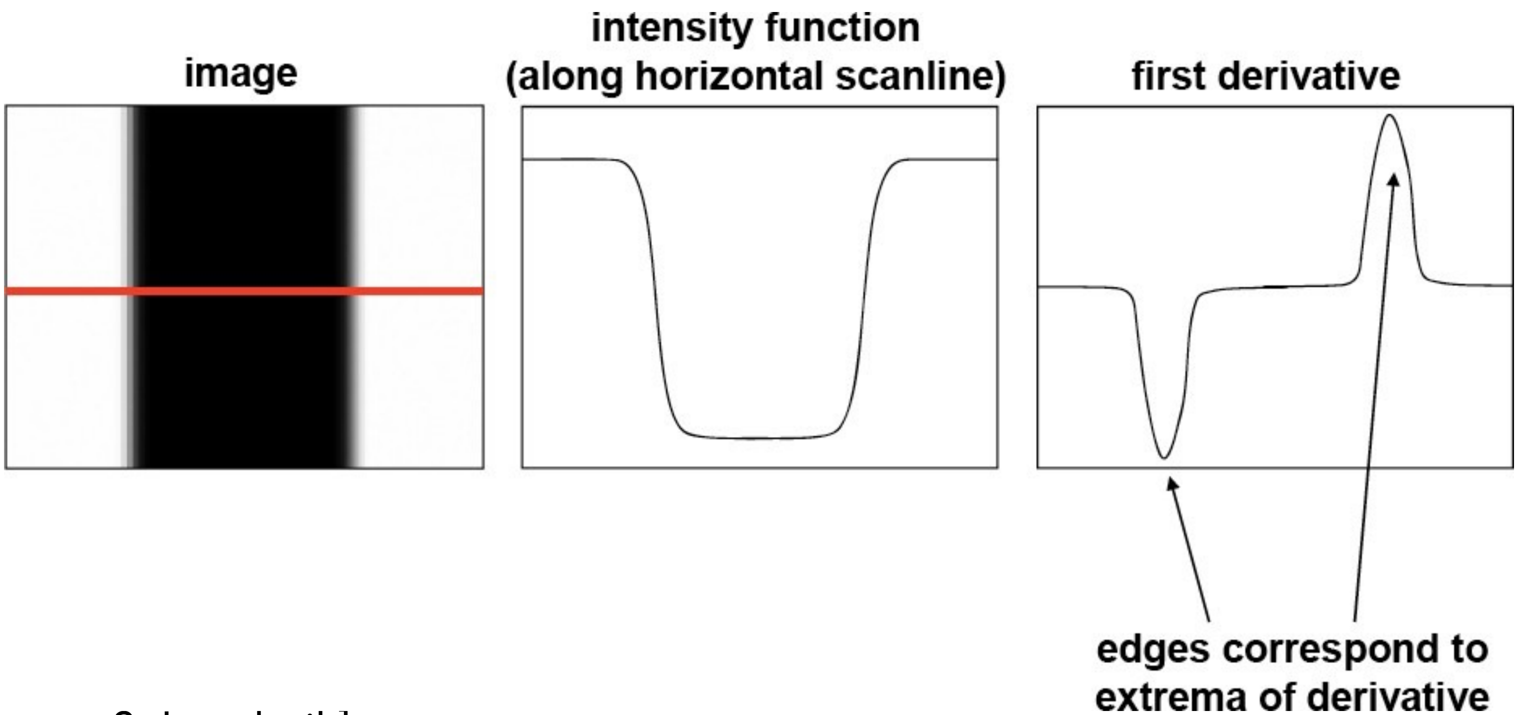
- Edges are caused by a variety of factors



[Source: N. Snavely]

Characterizing Edges

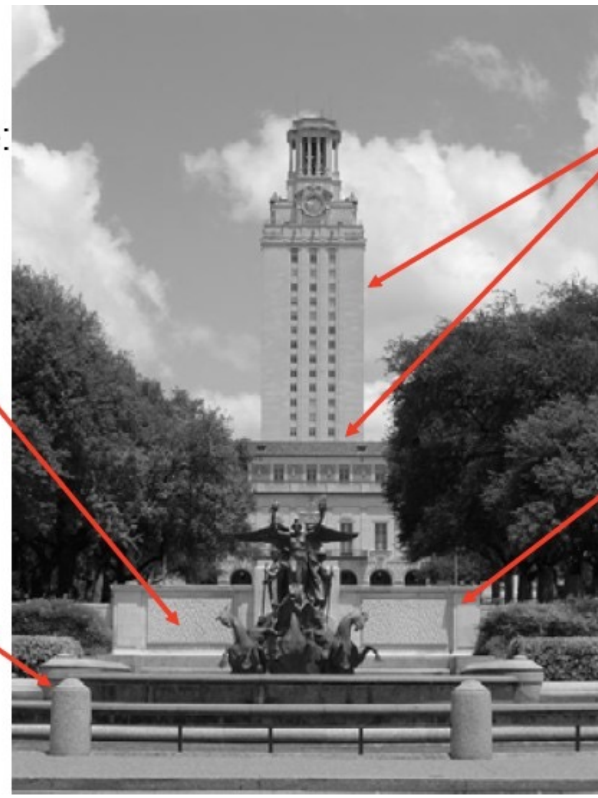
- An edge is a place of rapid change in the image intensity function.



[Source: S. Lazebnik]

What Causes an Edge?

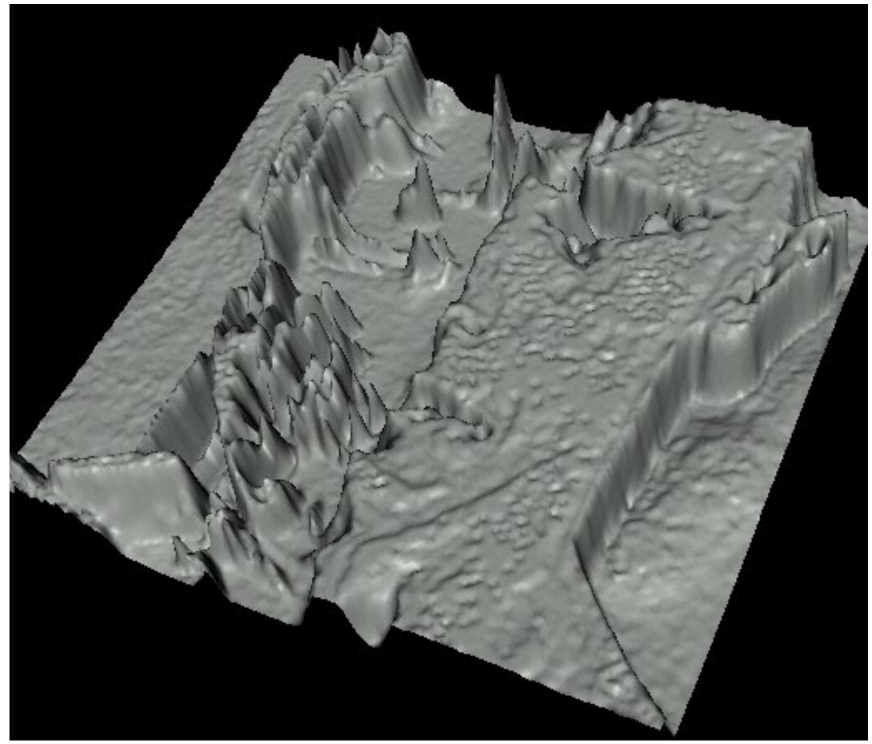
- An edge is a place of rapid change in the image intensity function.



[Source: K. Grauman]

Images as Functions

• Edges look like steep cliffs



[Source: N. Snavely]

How to Implement Derivatives with Convolution

- How can we differentiate a digital image *f*[*x*, *y*]?
  - If image *f* was continuous, then compute the partial derivative as
    - $$\frac{\partial f(x,y)}{\partial x} = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon,y)-f(x,y)}{\varepsilon}$$

How to Implement Derivatives with Convolution

- How can we differentiate a digital image $f[x, y]$?
  - If image f was continuous, then compute the partial derivative as
  - $\frac{\partial f(x,y)}{\partial x} = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon, y) - f(x,y)}{\varepsilon}$
  - Since it's discrete, take first-order forward discrete derivative (finite difference)
  - $\frac{\partial f(x,y)}{\partial x} \approx \lim_{\varepsilon \to 0} \frac{f(x+1, y) - f(x,y)}{1}$

How to Implement Derivatives with Convolution

- How can we differentiate a digital image $f[x, y]$?
  - If image f was continuous, then compute the partial derivative as
    - $$\frac{\partial f(x,y)}{\partial x} = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon,y) - f(x,y)}{\varepsilon}$$
  - Since it's discrete, take first-order forward discrete derivative (finite difference)
    - $$\frac{\partial f(x,y)}{\partial x} \approx \lim_{\varepsilon \to 0} \frac{f(x+1,y) - f(x,y)}{1}$$
  - What would be the filter to implement this using correlation/convolution?

How to Implement Derivatives with Convolution

- How can we differentiate a digital image $f[x, y]$?
  - If image f was continuous, then compute the partial derivative as
    - $\frac{\partial f(x,y)}{\partial x} = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon,y)-f(x,y)}{\varepsilon}$
  - Since it's discrete, take first-order forward discrete derivative (finite difference)
    - $\frac{\partial f(x,y)}{\partial x} \approx \lim_{\varepsilon \to 0} \frac{f(x+1,y)-f(x,y)}{1}$
  - What would be the filter to implement this using correlation/convolution?

$\frac{\partial f}{\partial x}$:

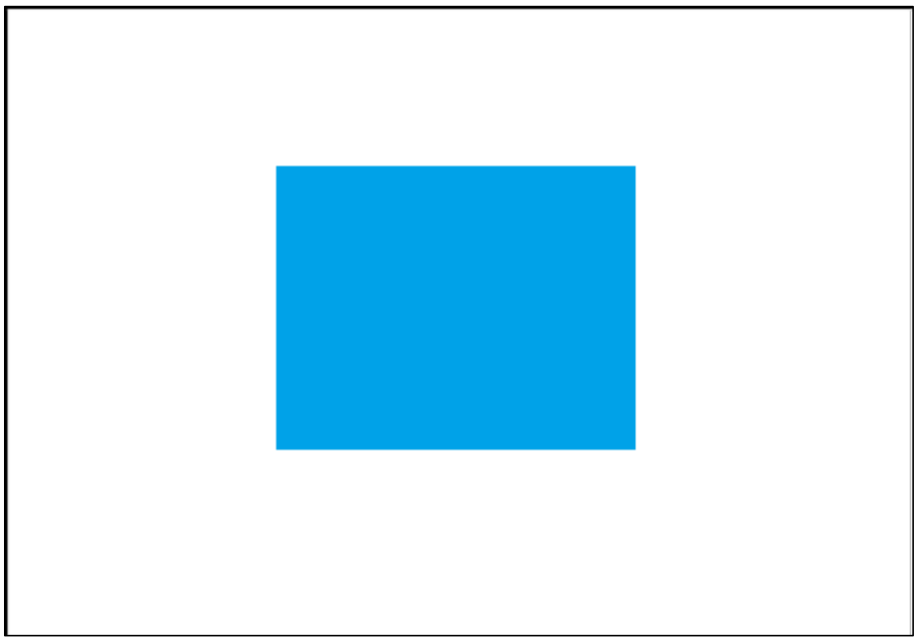
H_x

$\frac{\partial f}{\partial y}$:

H_y

Examples: Partial Derivatives of an Image

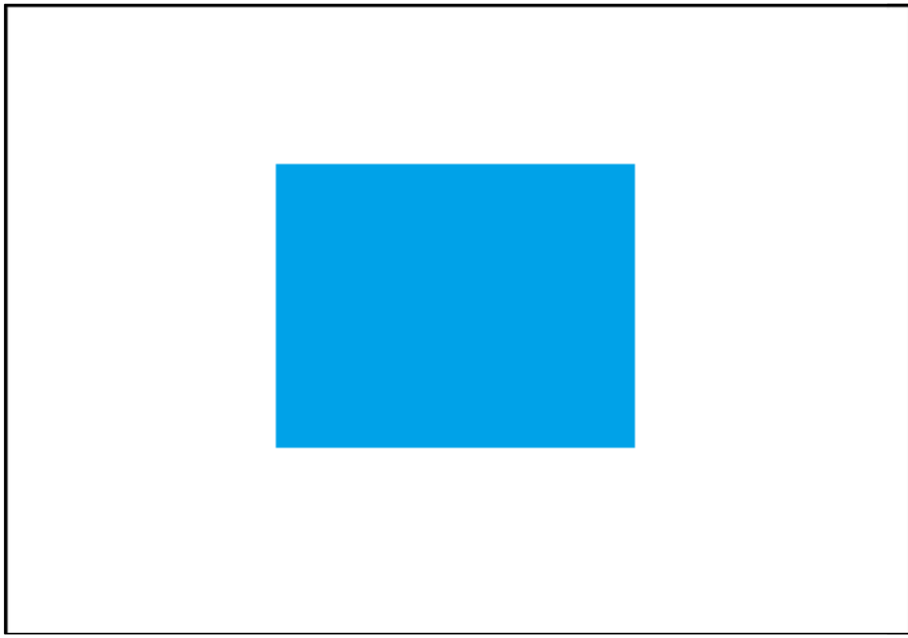
- How does the horizontal derivative using the filter $[-1, 1]$ look like?



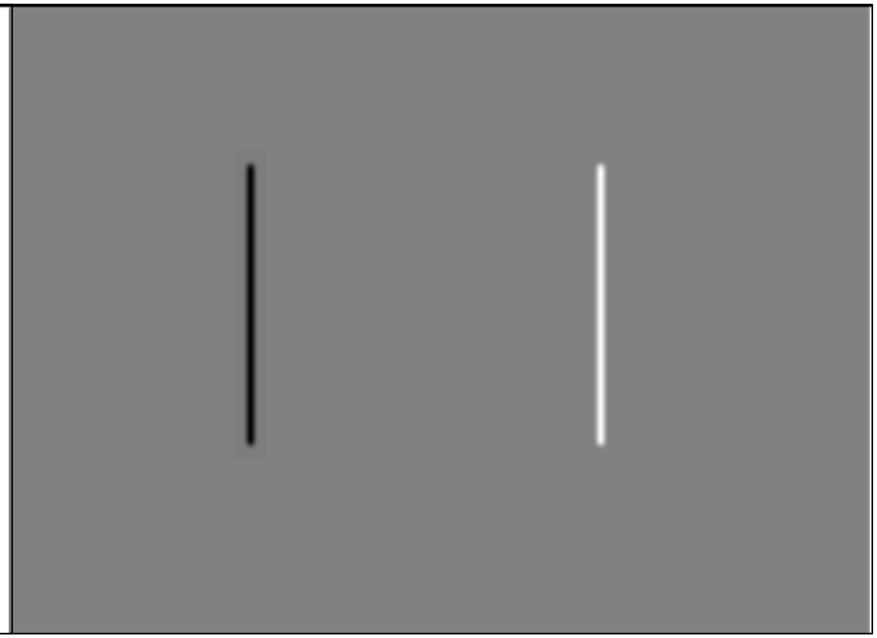
Image

Examples: Partial Derivatives of an Image

- How does the horizontal derivative using the filter [−1, 1] look like?



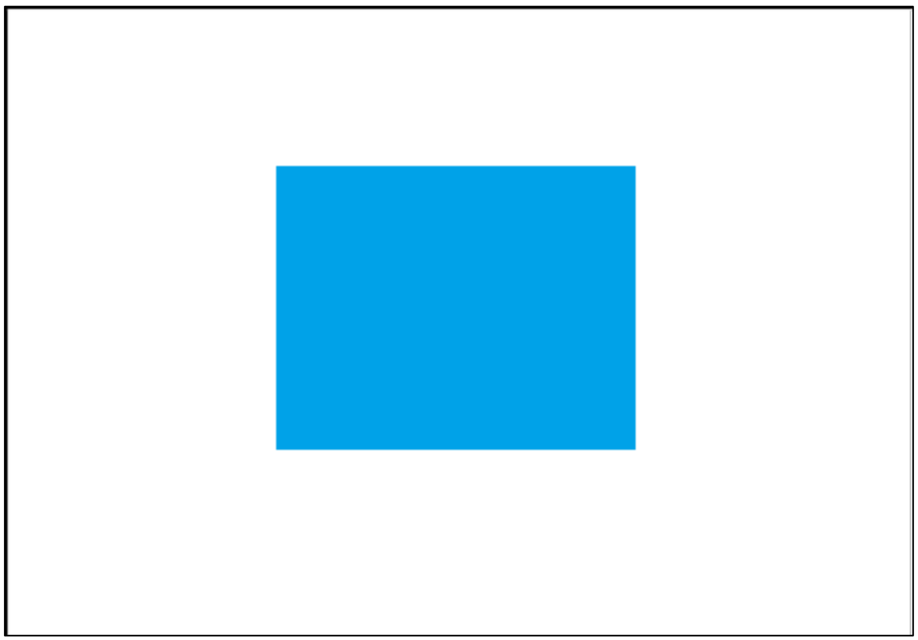
Image



$\frac{\partial f(x,y)}{\partial x}$ with [−1, 1] and correlation

Examples: Partial Derivatives of an Image

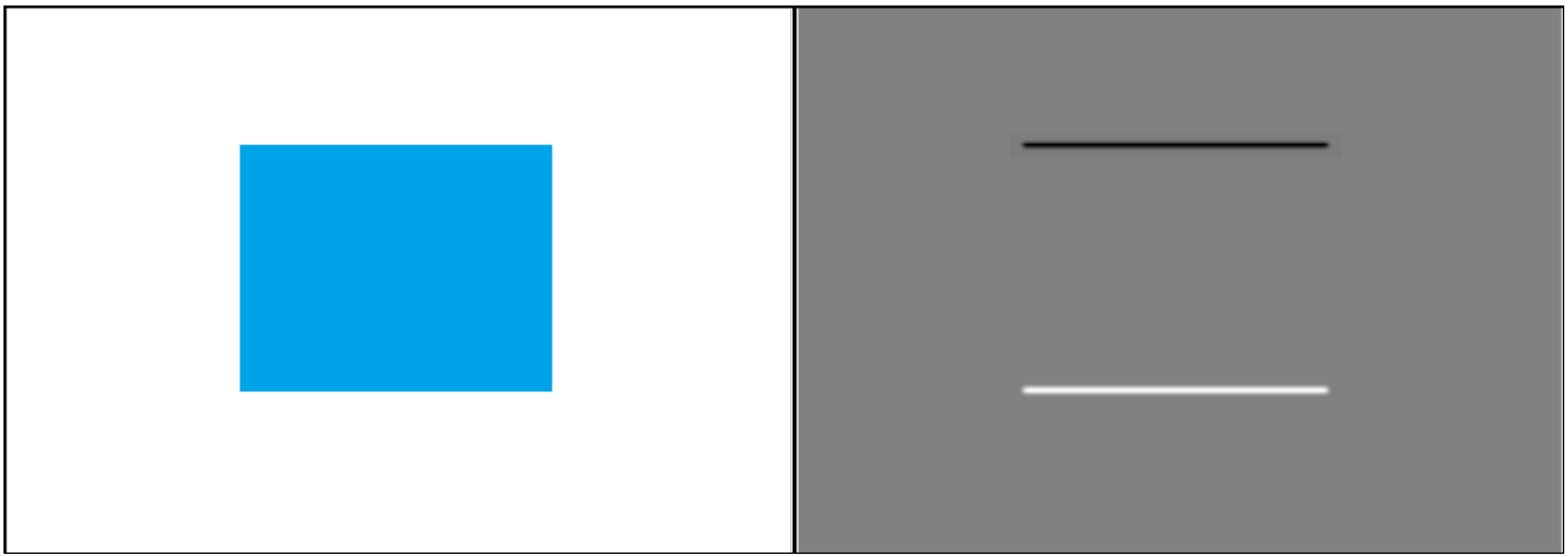
- How about the vertical derivative using filter $[-1, 1]^T$?



Image

Examples: Partial Derivatives of an Image

- How about the vertical derivative using filter [−1, 1]T ?

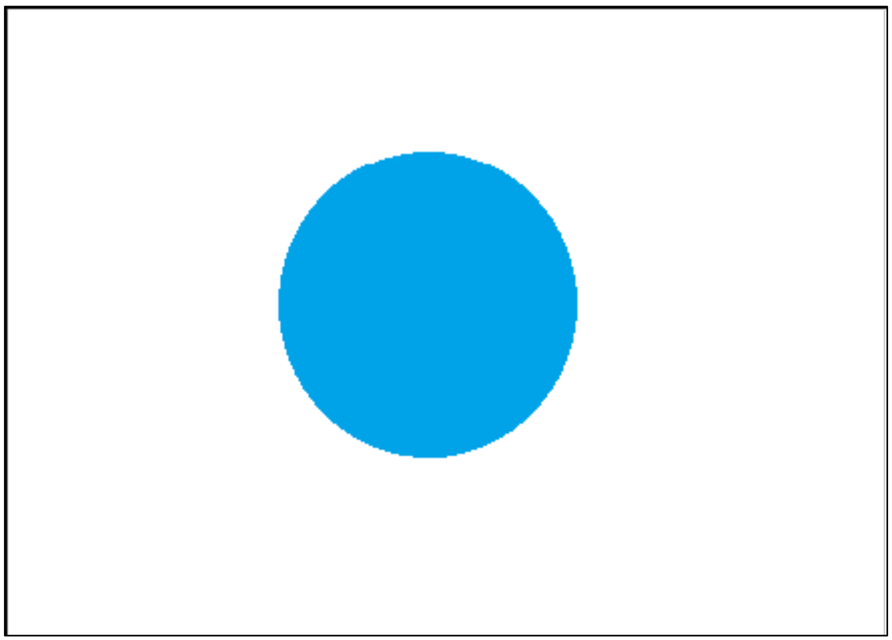


Image

$\frac{\partial f(x,y)}{\partial y}$ with $[-1,\ 1]^T$ and correlation

Examples: Partial Derivatives of an Image

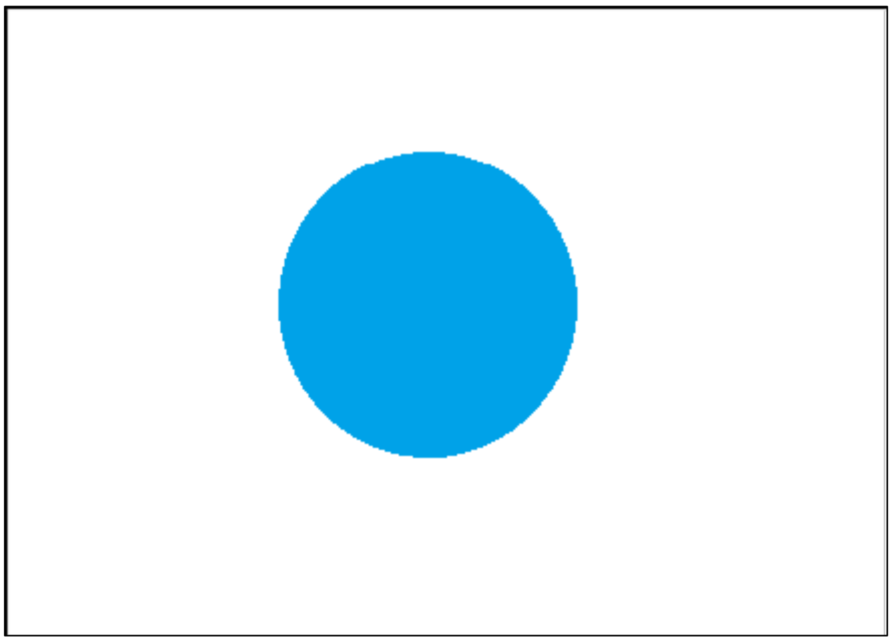
- How does the horizontal derivative using the filter $[-1, 1]$ look like?



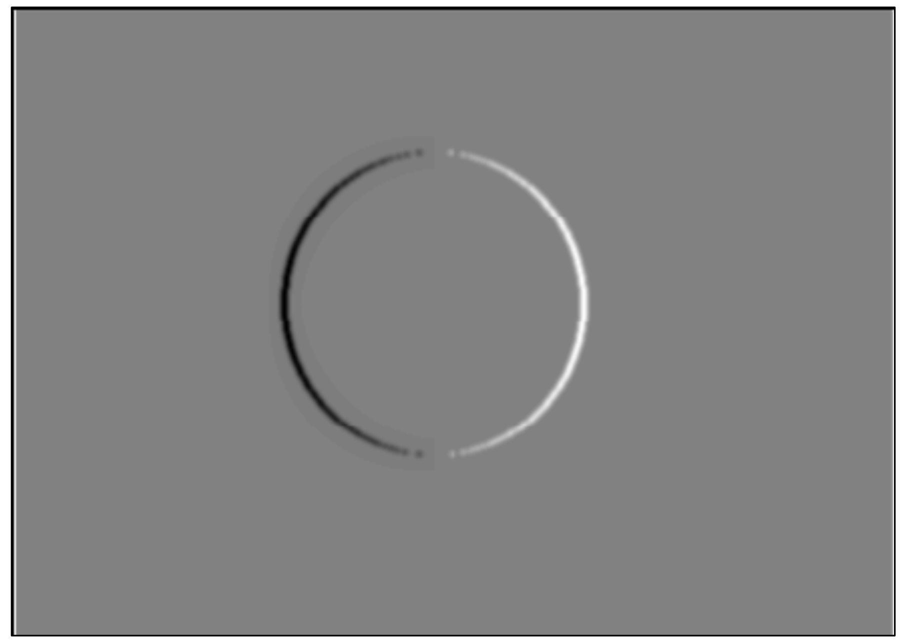
Image

Examples: Partial Derivatives of an Image

- How does the horizontal derivative using the filter [−1, 1] look like?



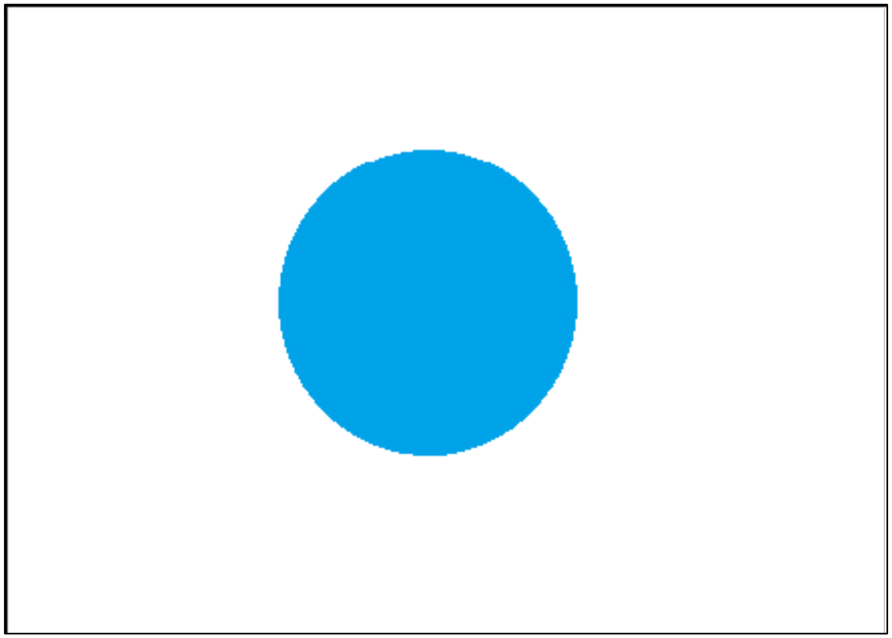
Image



$\frac{\partial f(x,y)}{\partial x}$ with [−1, 1] and correlation

Examples: Partial Derivatives of an Image

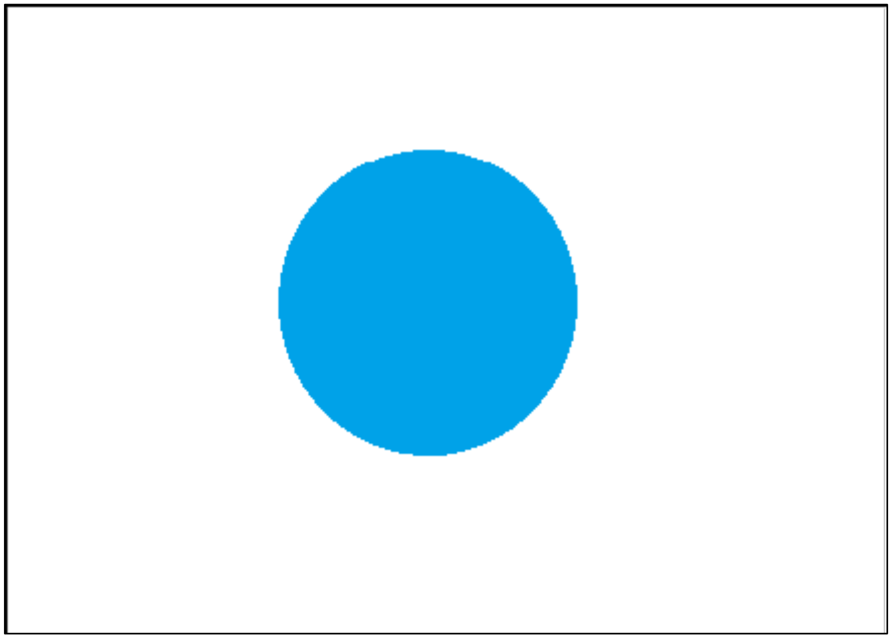
- How about the vertical derivative using filter $[-1, 1]^T$?



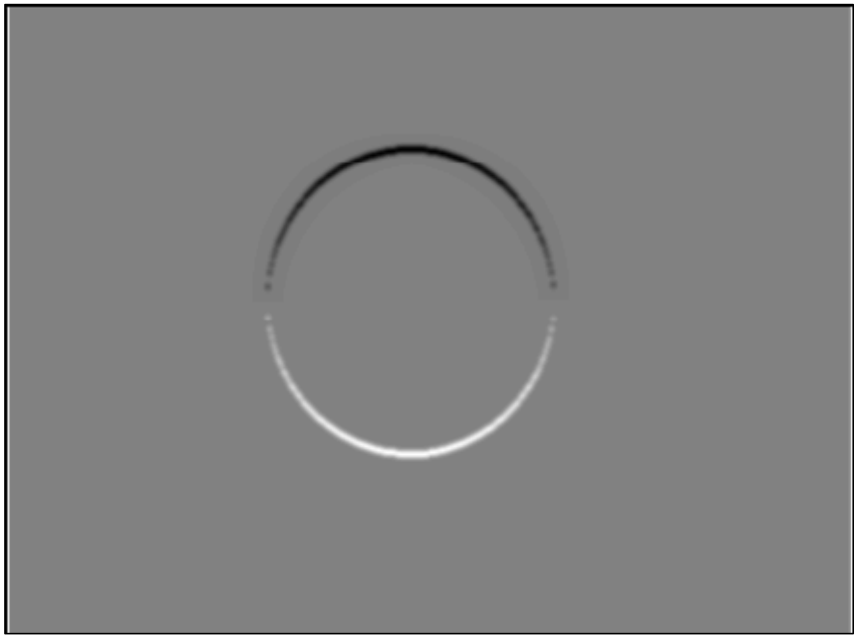
Image

Examples: Partial Derivatives of an Image

- How about the vertical derivative using filter $[-1,\ 1]^T$?



Image

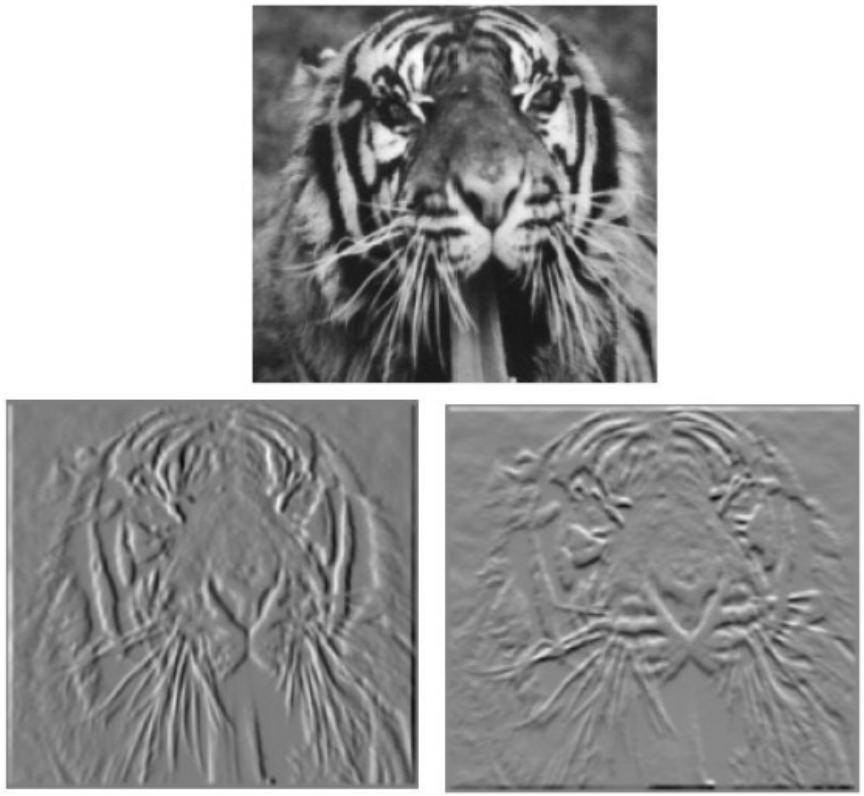


$\frac{\partial f(x,y)}{\partial y}$ with $[-1,\ 1]^T$ and correlation

Examples: Partial Derivatives of an Image

$$\frac{\partial f(x,y)}{\partial x}$$

$$\begin{bmatrix} -1 & 1 \end{bmatrix}$$



$$\frac{\partial f(x,y)}{\partial y}$$

$$\begin{bmatrix} -1 \\ 1 \end{bmatrix} \text{ ? or } \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

[Source: K. Grauman]

Figure: Using correlation filters

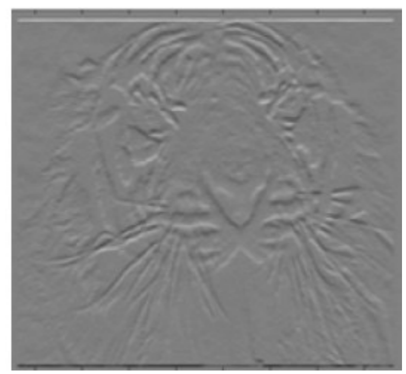
Finite Difference Filters

Prewitt: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

Sobel: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

Roberts: $M_x = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

```
>> My = fspecial('sobel');
>> outim = imfilter(double(im), My);
>> imagesc(outim);
>> colormap gray;
```



[Source: K. Grauman]

Image Gradient

- The gradient of an image $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

Image Gradient

- The gradient of an image $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

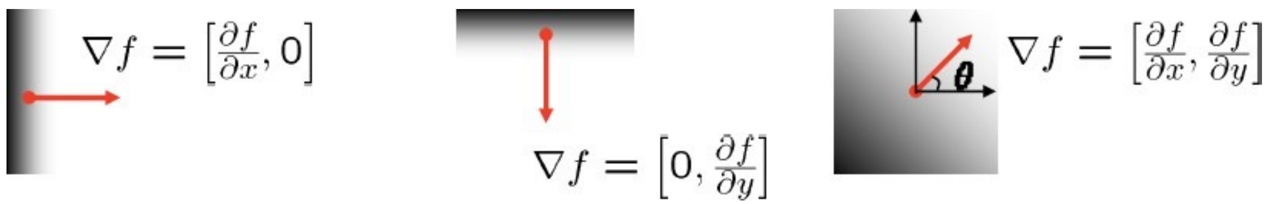
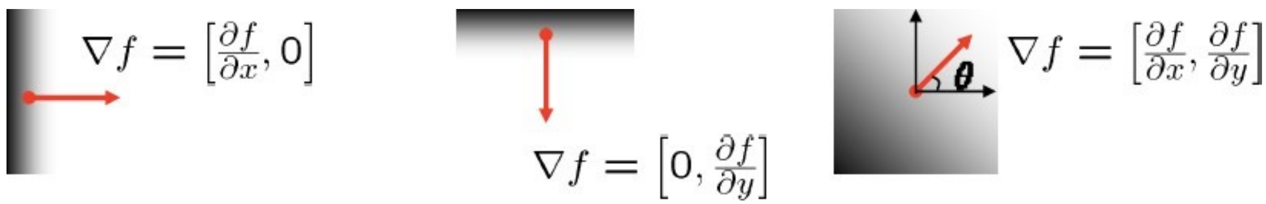


Image Gradient

- The gradient of an image $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

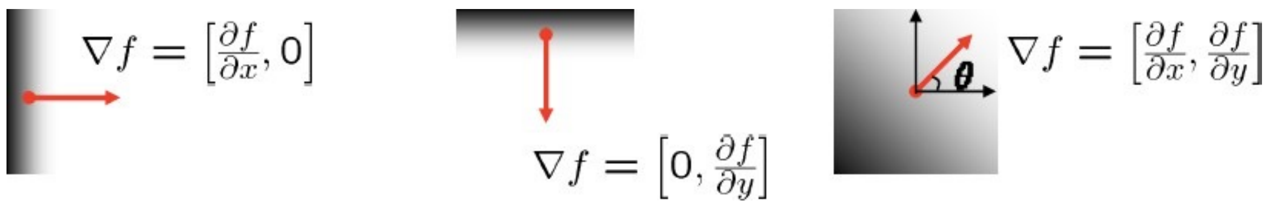


- The gradient direction (orientation of edge normal) is given by:

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

Image Gradient

- The gradient of an image $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$



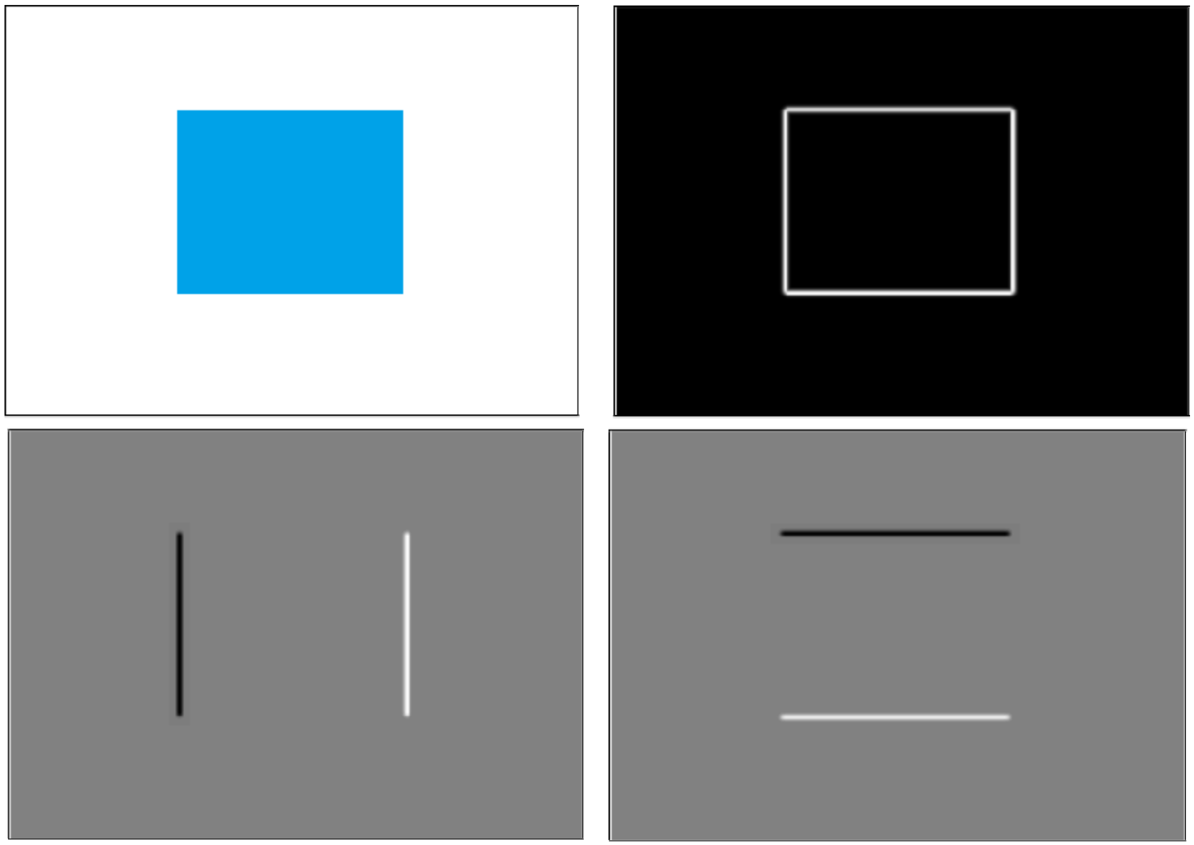
- The gradient direction (orientation of edge normal) is given by:

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

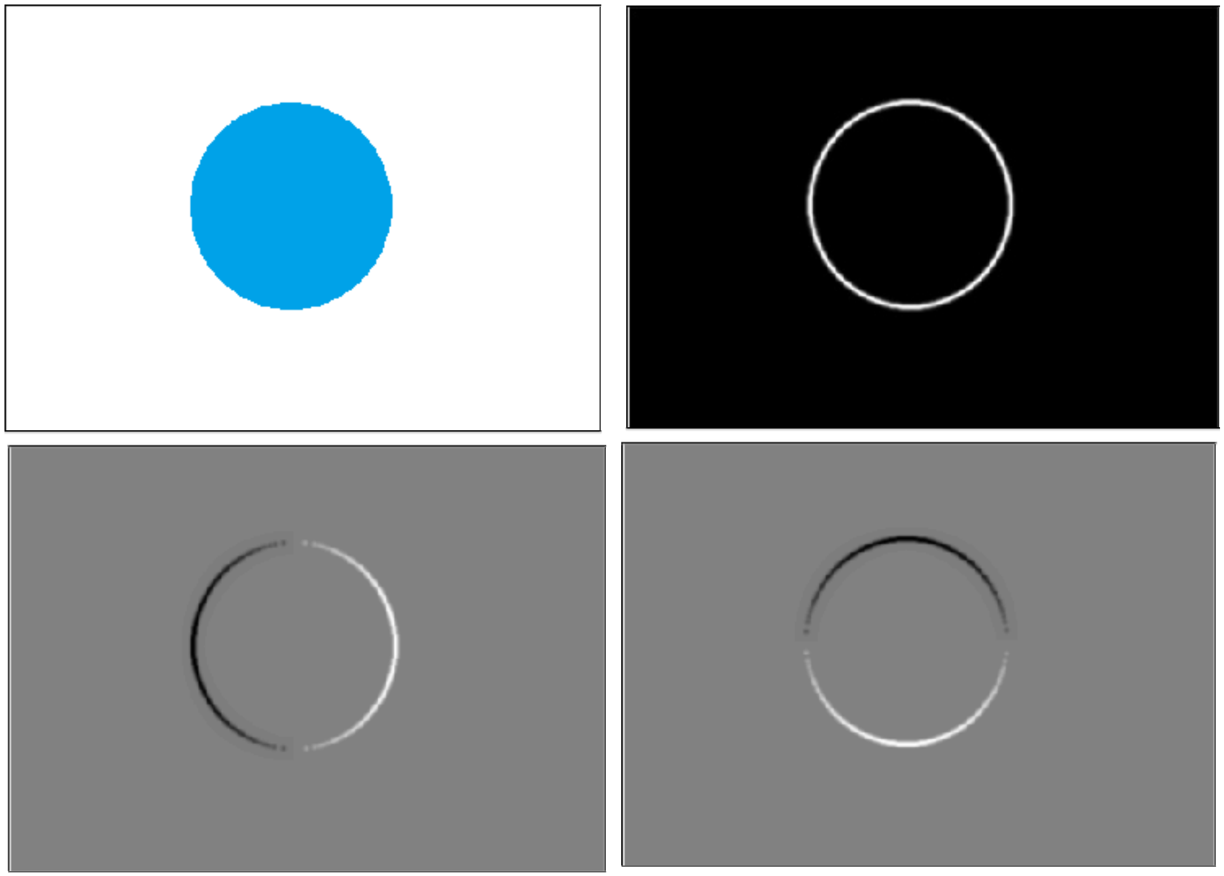
The edge strength is given by the magnitude $\|\nabla_f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$

[Source: S. Seitz]

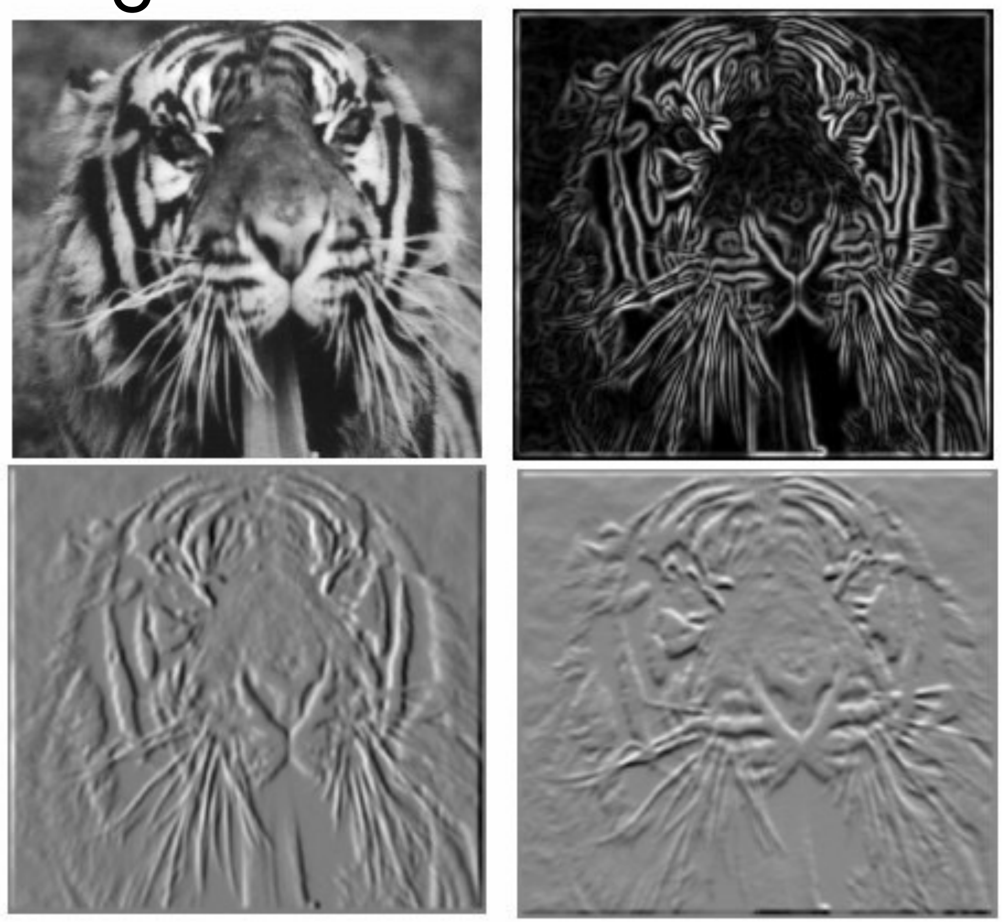
Example: Image Gradient



Example: Image Gradient



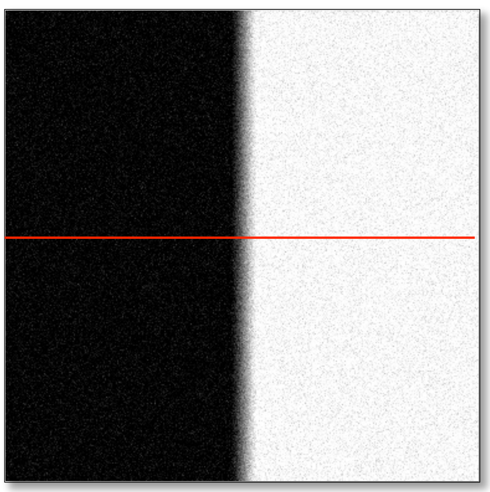
Example: Image Gradient

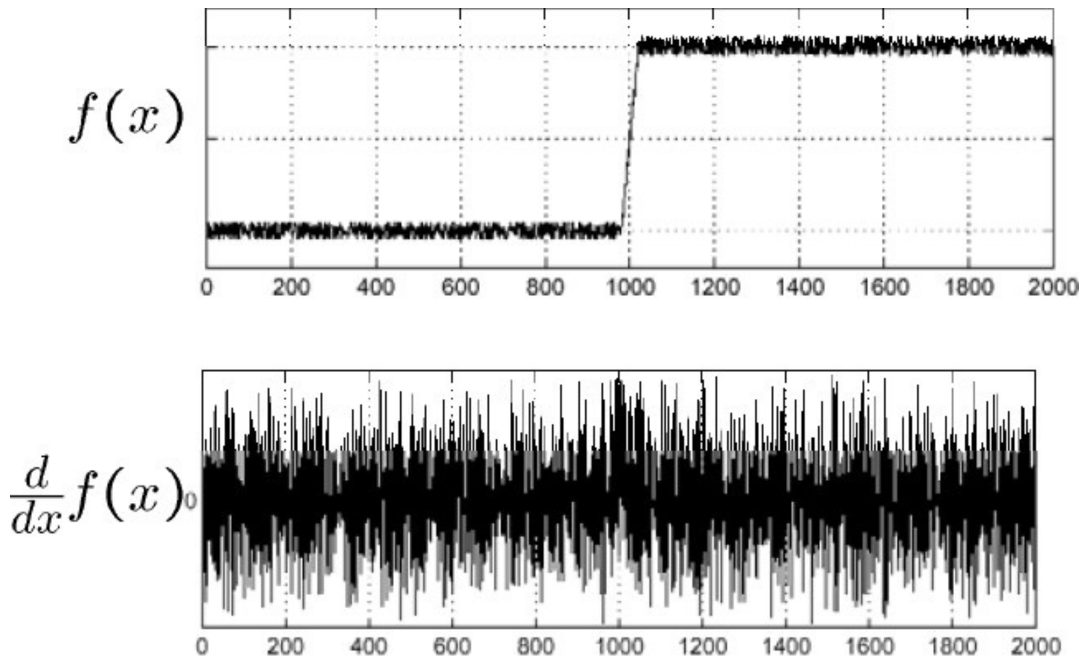


[Source: S. Lazebnik]

Effects of noise

- What if our image is noisy? What can we do?
- Consider a single row or column of the image.
- Plotting intensity as a function of position gives a signal.

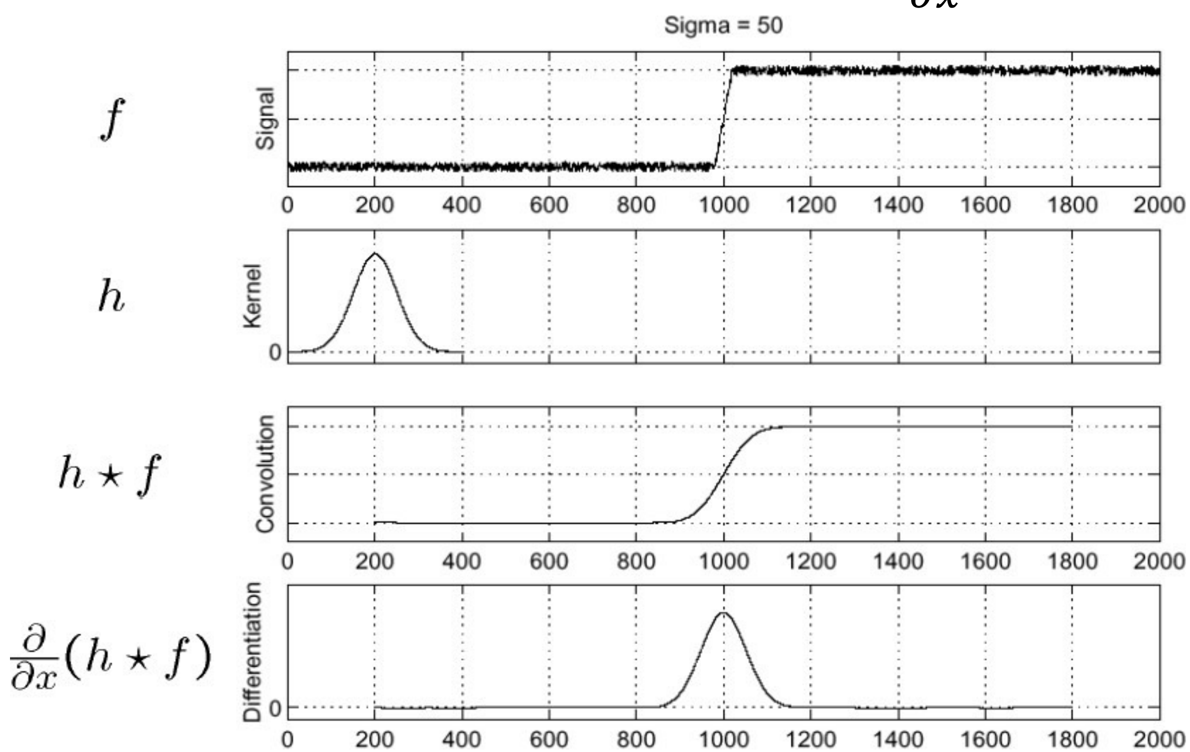

Noisy input Image



[Source: S. Seitz]

Effects of noise

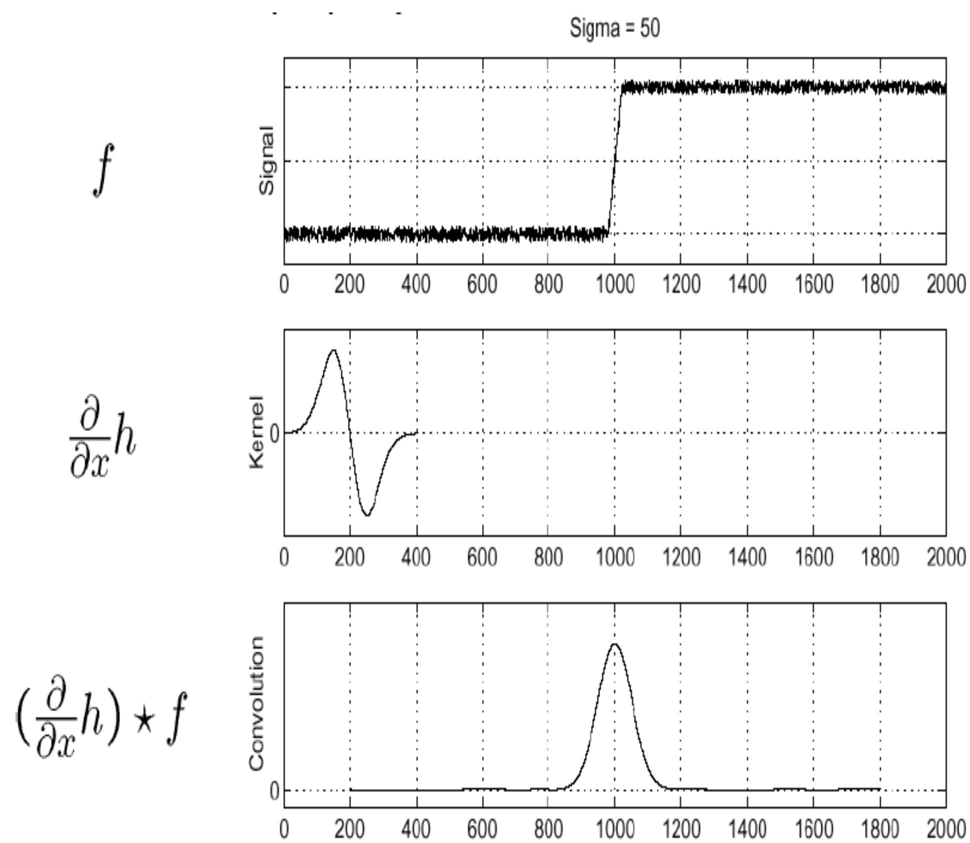
- Smooth first with h (e.g. Gaussian), and look for peaks in $\frac{\partial}{\partial x}(h^* f)$



[Source: S. Seitz]

Derivative theorem of convolution

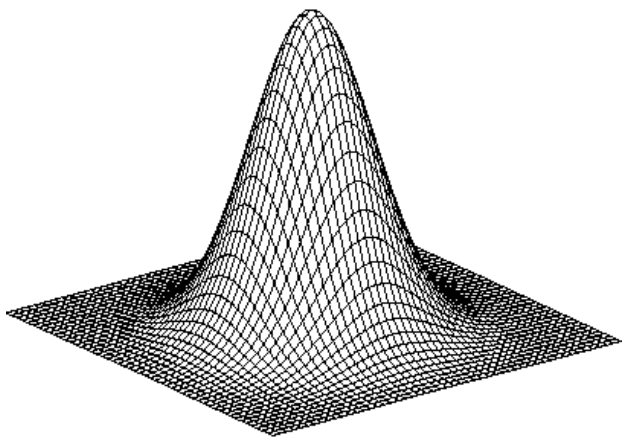
- Differentiation property of convolution
- $\frac{\partial}{\partial x}(h * f) = \left(\frac{\partial h}{\partial x}\right) * f = h * \left(\frac{\partial f}{\partial x}\right)$



- From last time, why does this work?
- It saves one operation

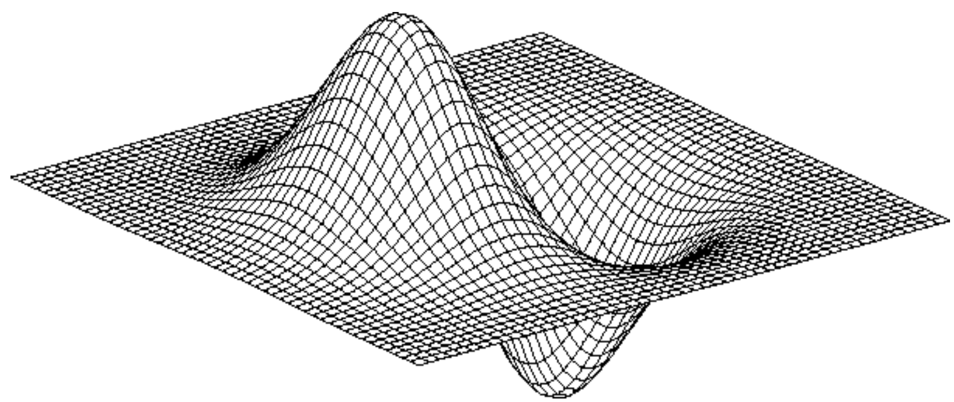
[Source: S. Seitz]

2D Edge Detection Filters



Gaussian

$$f_\sigma(x, y) = \frac{1}{2\pi\sigma^2} exp^{-\frac{x^2+y^2}{2\sigma^2}}$$

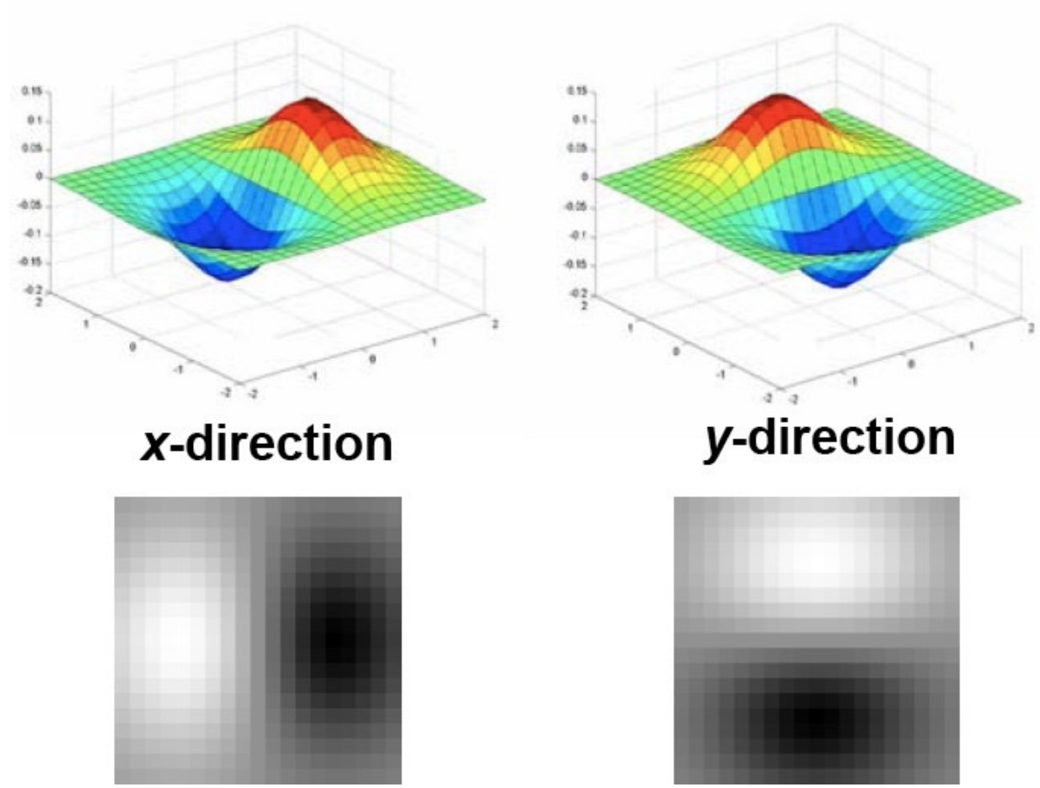


Derivative of Gaussian (x)

$$\frac{\partial}{\partial x} h_\sigma(x, y)$$

[Source: S. Seitz]

Derivative of Gaussians



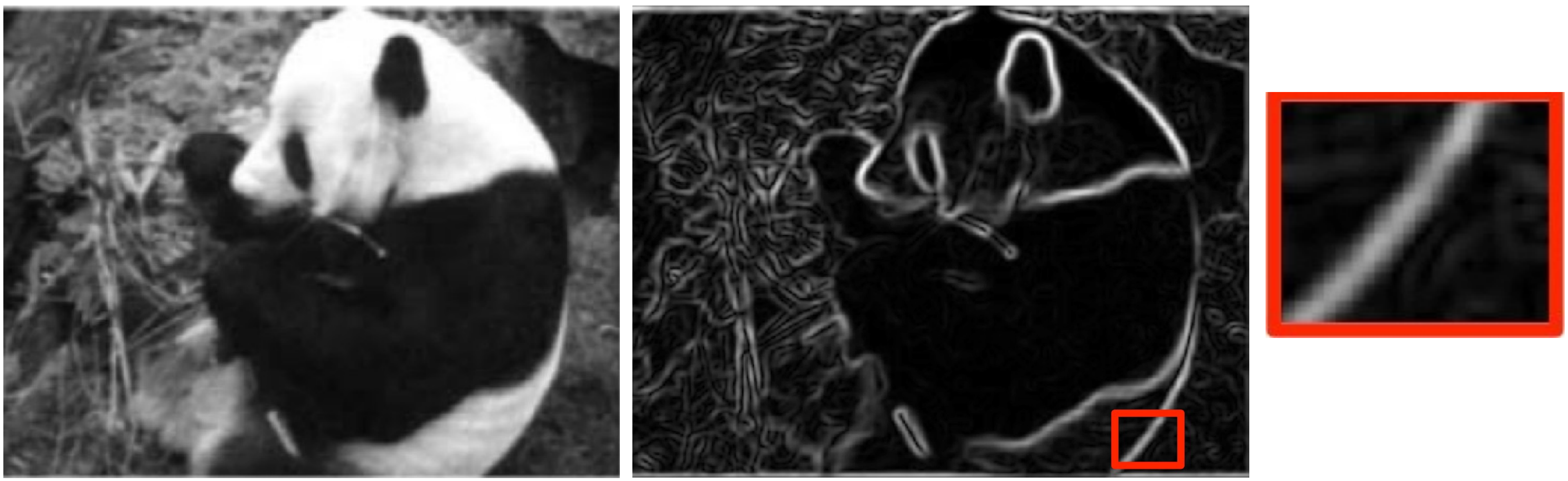
[Source: K. Grauman]

Example

• Applying the Gaussian derivatives to image

[Source: K. Grauman]

Example

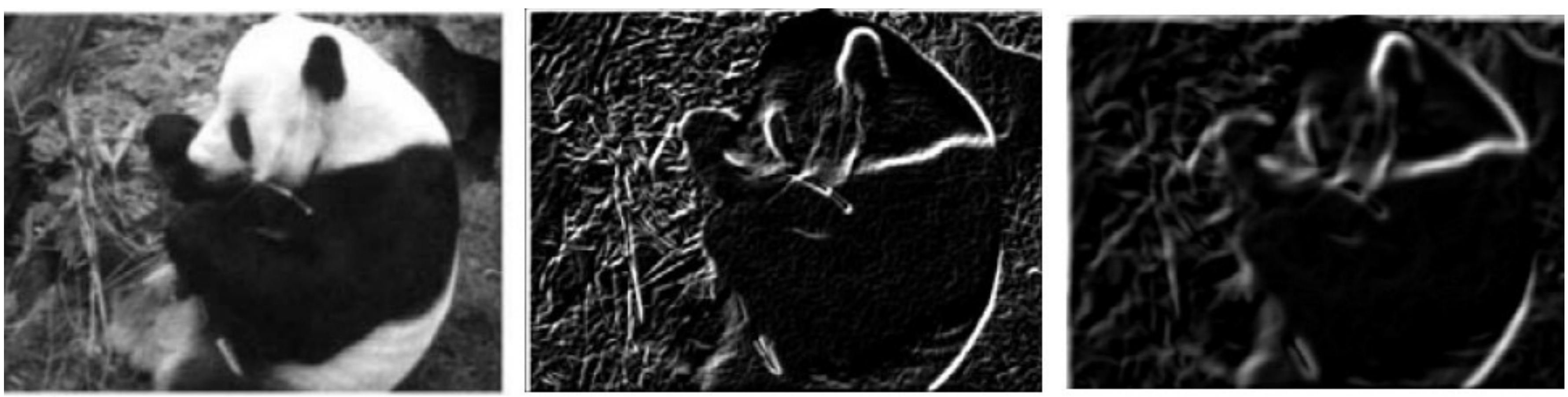


- Applying the Gaussian derivatives to image

[Source: K. Grauman]

Effect of σ on derivatives

- The detected structures differ depending on the Gaussian's scale parameter:
- Larger values: detects edges of larger scale
- Smaller values: detects finer structures



σ = 1 pixel   σ = 3 pixels

[Source: K. Grauman]

Canny Edge Detector

- OpenCV: cv2.Canny()
  - Filter image with derivative of Gaussian (horizontal and vertical directions) Find magnitude and orientation of gradient
  - Non-maximum suppression
  - Linking and thresholding (hysteresis):
    - Define two thresholds: low and high
    - Use the high threshold to start edge curves and the low threshold to continue them

[Source: D. Lowe and L. Fei-Fei]

Locating Edges – Canny's Edge Detector

- Example "peppers" image

Locating Edges – Canny's Edge Detector

Figure: Canny's approach takes gradient magnitude

Locating Edges – Canny's Edge Detector

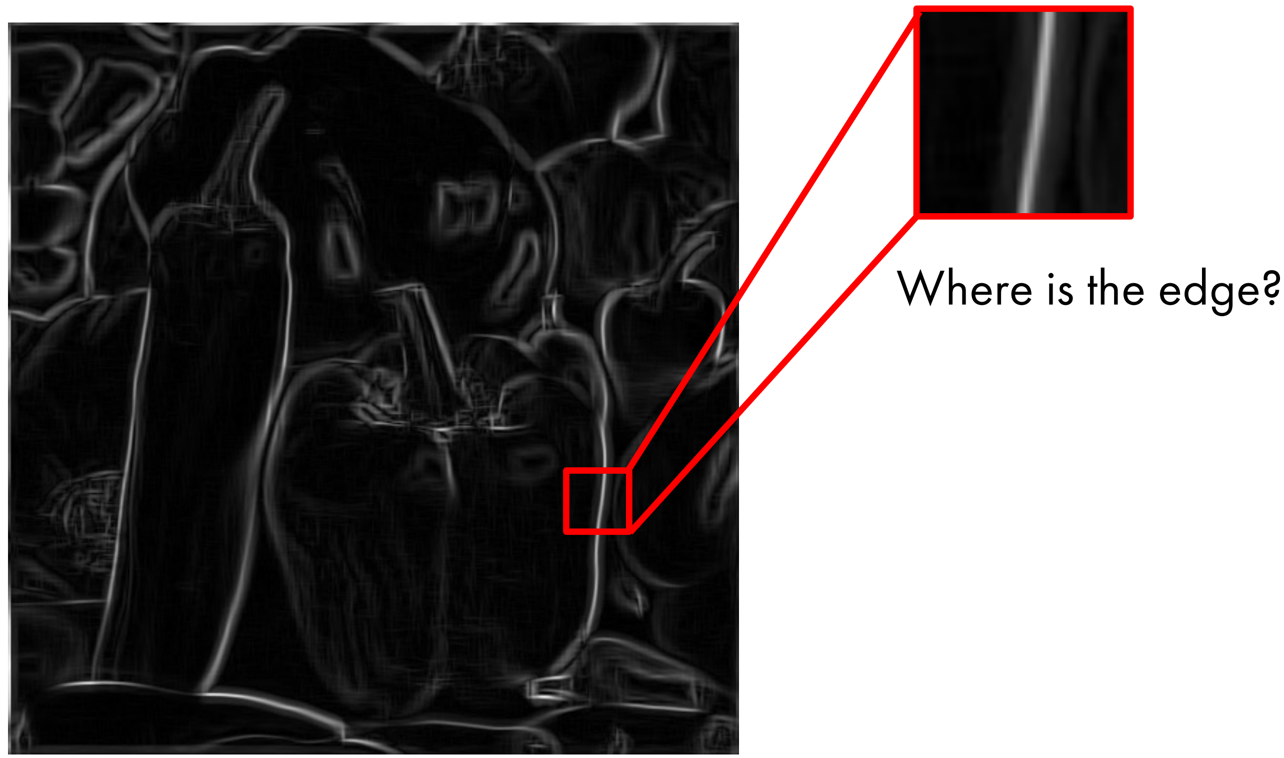


Figure: Canny's approach takes gradient magnitude

Non-Maxima Suppression

- Check if pixel is local maximum along gradient direction
- If yes, take it

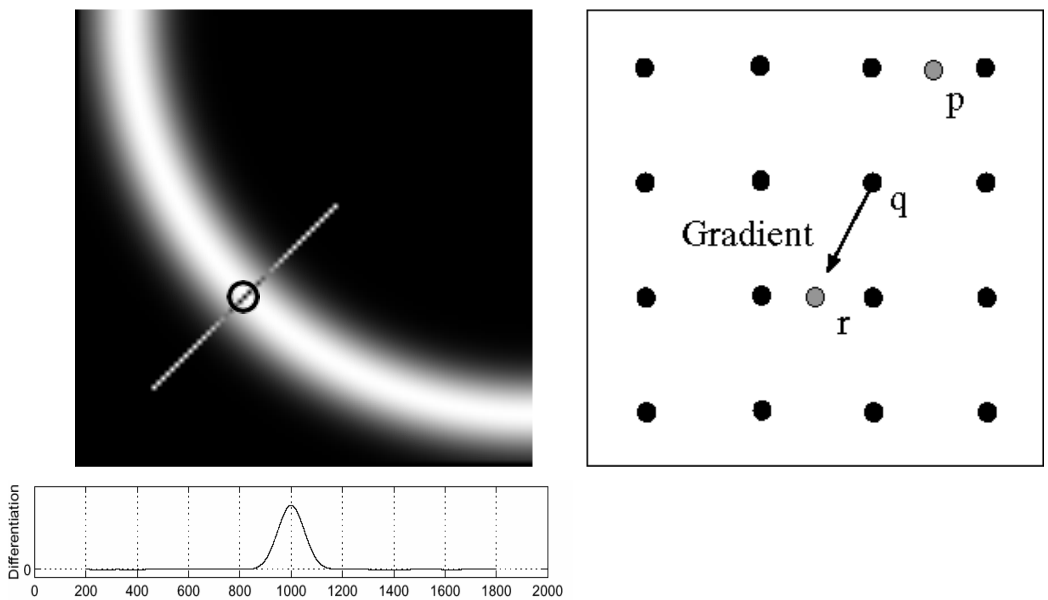
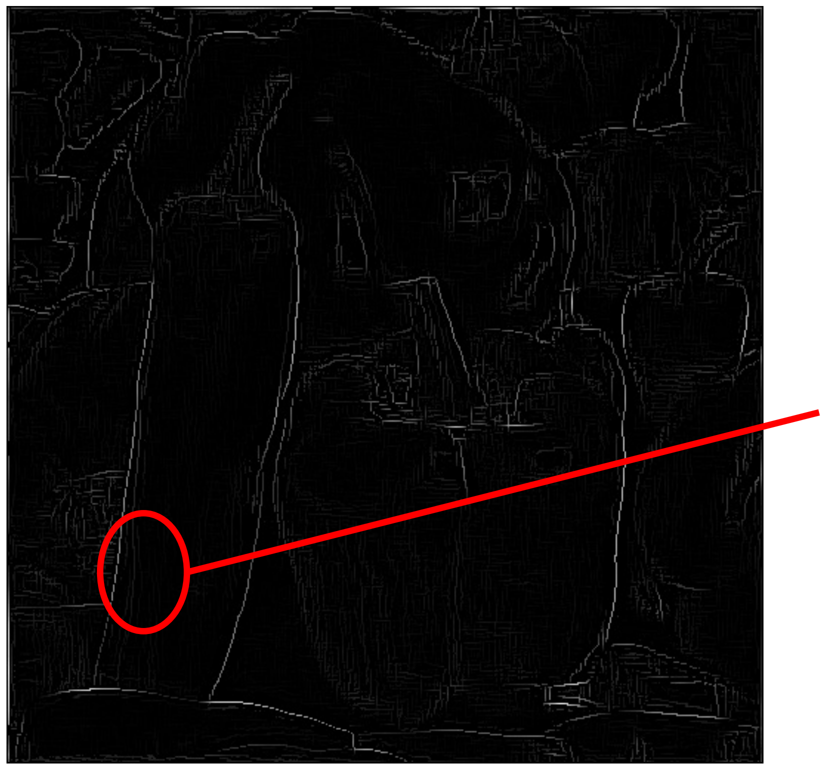


Figure: Gradient magnitude

[Source: N. Snavely]

Finding Edges

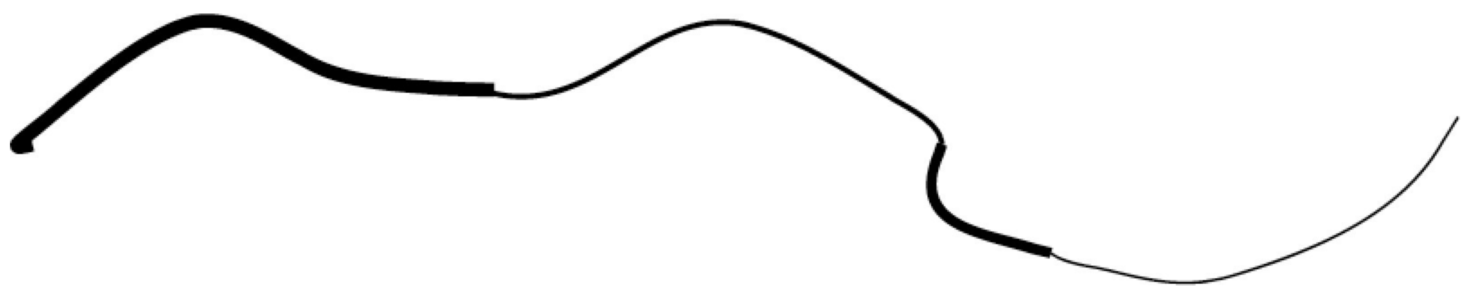


Problem, some pixels did not survive the thresholding

Figure: Problem with thresholding

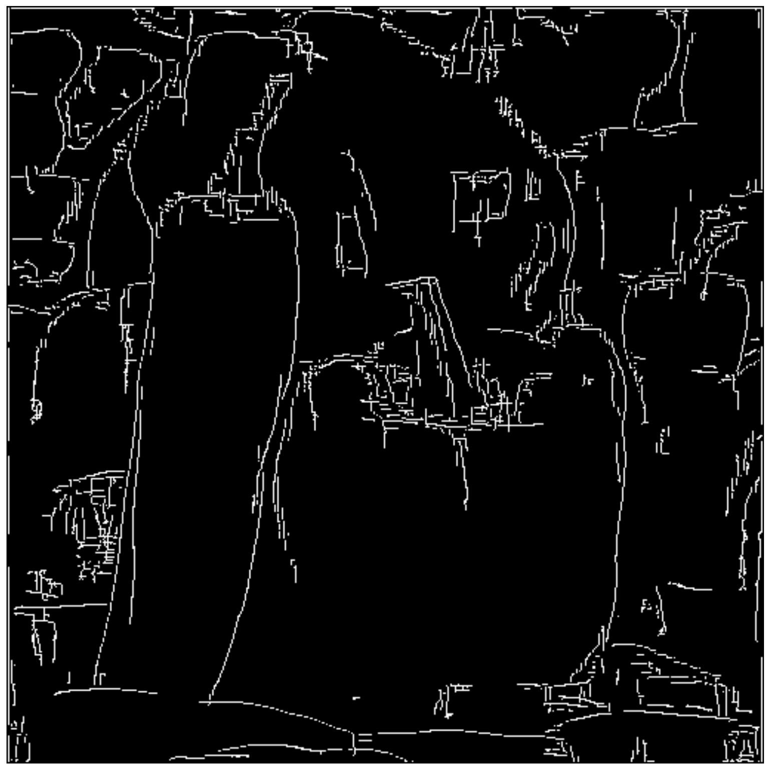
Hysteresis thresholding

- Use a high threshold to start edge curves, and a low threshold to continue them



[Source: K. Grauman]

Hysteresis



Hysteresis thresholding

original image

high threshold (strong edges)

low threshold (weak edges)

hysteresis threshold

[Source: L. Fei Fei]

Canny Edge Detector

- OpenCV: cv2.Canny()
  - Filter image with derivative of Gaussian (horizontal and vertical directions) Find magnitude and orientation of gradient
  - Non-maximum suppression
  - Linking and thresholding (hysteresis):
    - Define two thresholds: low and high
    - Use the high threshold to start edge curves and the low threshold to continue them

[Source: D. Lowe and L. Fei-Fei]

Canny Edge Detector (again)

- large σ (in step 1) detects "large-scale" edges
- small σ detects fine edges

original

Canny with $\sigma = 1$

Canny with $\sigma = 2$

[Source: S. Seitz]

Canny edge detector

- Still one of the most widely used edge detectors in computer vision
- J. Canny, A Computational Approach To Edge Detection, IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.
- Depends on several parameters: σ of the blur and the thresholds

[Slide: R. Urtasun]

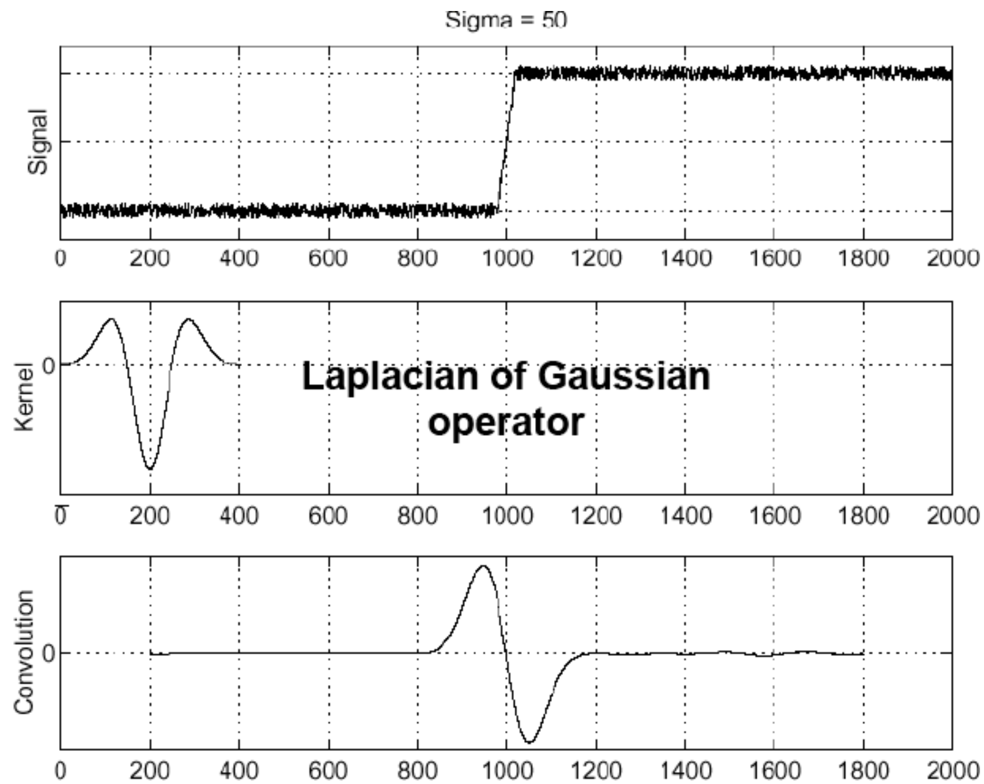
Another Way of Finding Edges: Laplacian of Gaussians

- Edge by detecting zero-crossings of bottom graph

$$f$$

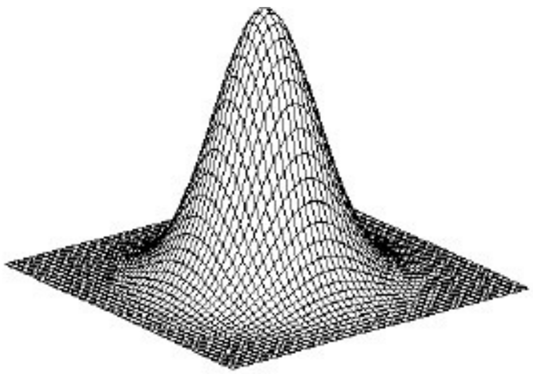
$$\frac{\partial^2}{\partial x^2}h$$

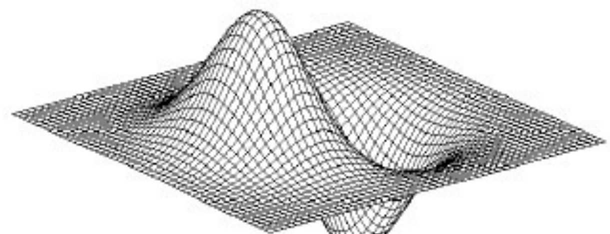
$$(\frac{\partial^2}{\partial x^2}h) \star f$$



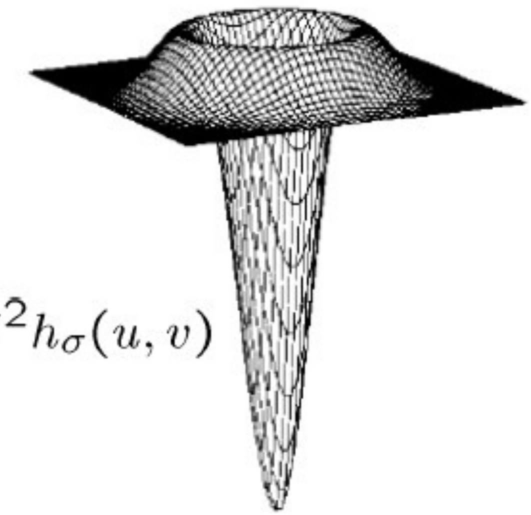
[Source: S. Seitz]

2D Edge Filtering





Laplacian of Gaussian



Gaussian

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

derivative of Gaussian

$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

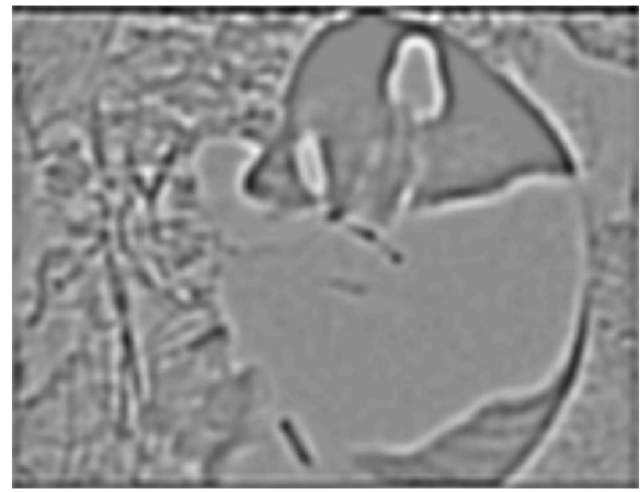
$$\nabla^2 h_\sigma(u,v)$$

With ∇^2 the Laplacian operator $\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$

[Source: S. Seitz]

Example

σ = 1 pixels

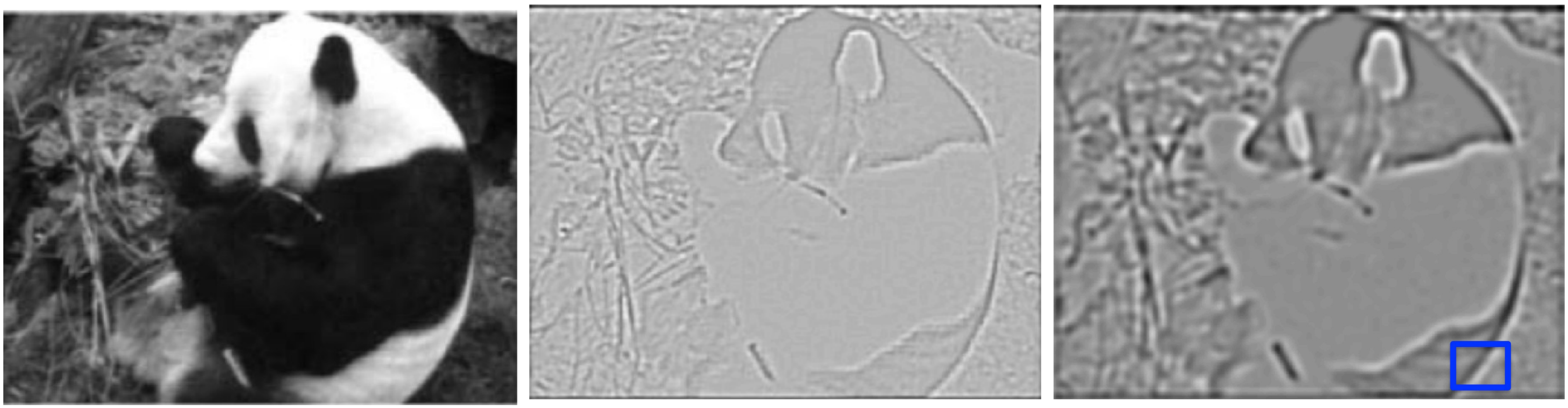


σ = 3 pixels

- Applying the Laplacian operator to image

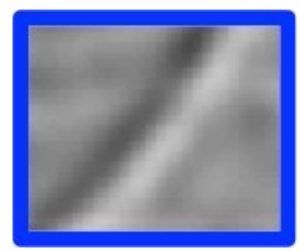
[Source: S. Seitz]

Example



σ = 1 pixels

σ = 3 pixels

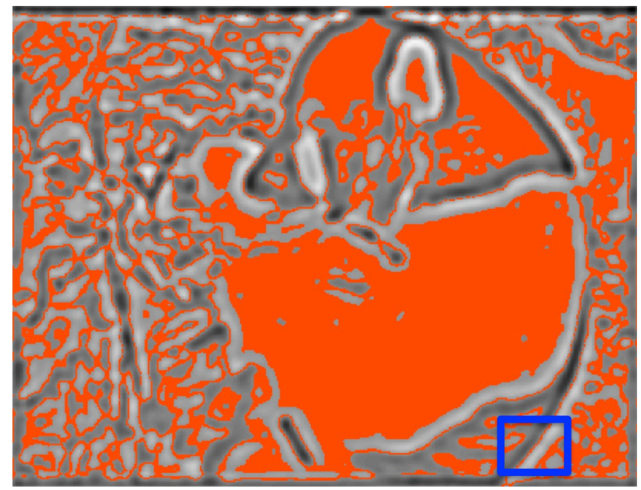


- Applying the Laplacian operator to image

[Source: S. Seitz]

Example

σ = 1 pixels



σ = 3 pixels

- Applying the Laplacian operator to image

[Source: S. Seitz]

A More ‘Modern’ Approach

- This is “old-style” Computer Vision. We are now in the era of successful Machine Learning techniques.

- Question: Can we use ML to do a better job at finding edges?

A More 'Modern' Approach

- This is "old-style" Computer Vision. We are now in the era of successful Machine Learning techniques.

- Question: Can we use ML to do a better job at finding edges?

We will see later.

A More ‘Modern’ Approach

- This is “old-style” Computer Vision. We are now in the era of successful Machine Learning techniques.

- Question: Can we use ML to do a better job at finding edges?

OR Should we see right now?

Holistically-Nested Edge Detection

Saining Xie
Dept. of CSE and Dept. of CogSci
University of California, San Diego
9500 Gilman Drive, La Jolla, CA 92093
s9xie@eng.ucsd.edu

Zhuowen Tu
Dept. of CogSci and Dept. of CSE
University of California, San Diego
9500 Gilman Drive, La Jolla, CA 92093
ztu@ucsd.edu

:1504.06375v2 [cs.CV] 4 Oct 2015

Abstract

*We develop a new edge detection algorithm that addresses two important issues in this long-standing vision problem: (1) holistic image training and prediction; and (2) multi-scale and multi-level feature learning. Our proposed method, holistically-nested edge detection (HED), performs image-to-image prediction by means of a deep learning model that leverages fully convolutional neural networks and deeply-supervised nets. HED automatically learns rich hierarchical representations (guided by deep supervision on side responses) that are important in order to resolve the challenging ambiguity in edge and object boundary detection. We significantly advance the state-of-the-art on the BSD500 dataset (ODS F-score of .*782*) and the NYU Depth dataset (ODS F-score of .*746*), and do so with an improved speed (*0.4*s per image) that is orders of magnitude faster than some recent CNN-based edge detection algorithms.*

1. Introduction

In this paper, we address the problem of detecting edges and object boundaries in natural images. This problem is

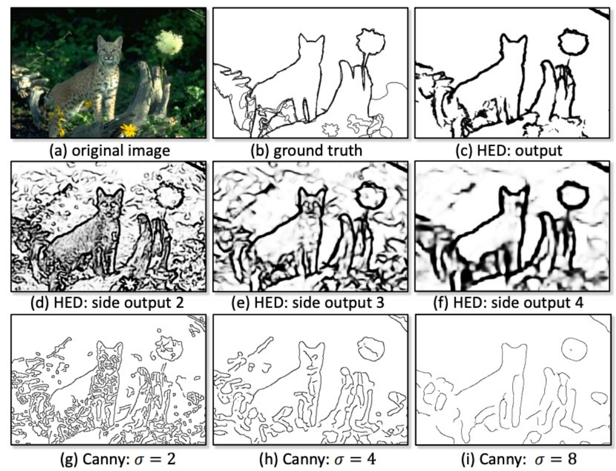


Figure 1. Illustration of the proposed HED algorithm. In the first row: (a) shows an example test image in the BSD500 dataset [28]; (b) shows its corresponding edges as annotated by human subjects; (c) displays the HED results. In the second row: (d), (e), and (f), respectively, show side edge responses from layers 2, 3, and 4 of our convolutional neural networks. In the third row: (g), (h), and (i), respectively, show edge responses from the Canny detector [4] at the scales $\sigma = 2.0$, $\sigma = 4.0$, and $\sigma = 8.0$. HED shows a clear advantage in consistency over Canny.

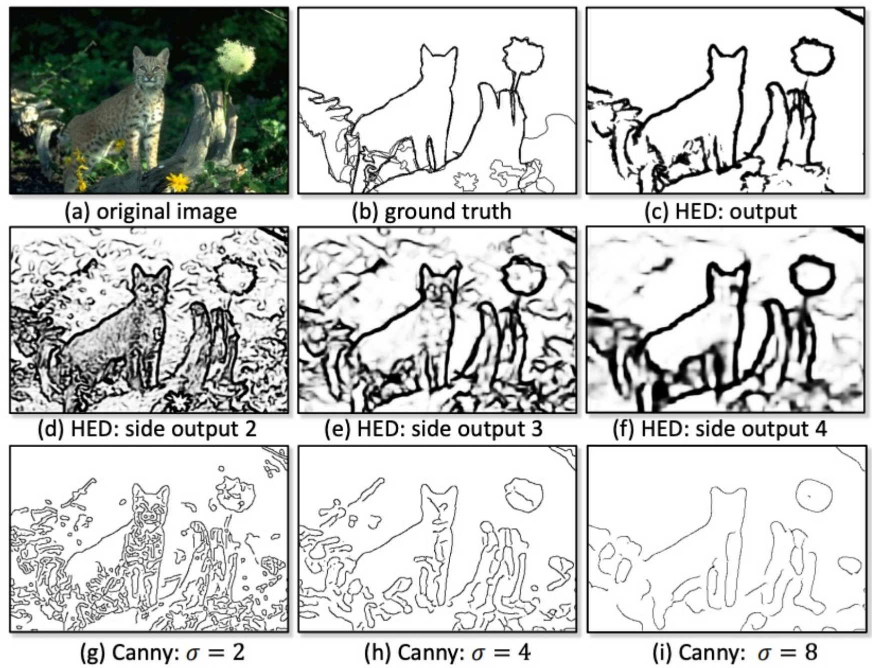


Figure 1. Illustration of the proposed HED algorithm. In the first row: (a) shows an example test image in the BSD500 dataset [28]; (b) shows its corresponding edges as annotated by human subjects; (c) displays the HED results. In the second row: (d), (e), and (f), respectively, show side edge responses from layers 2, 3, and 4 of our convolutional neural networks. In the third row: (g), (h), and (i), respectively, show edge responses from the Canny detector [4] at the scales $\sigma = 2.0$, $\sigma = 4.0$, and $\sigma = 8.0$. HED shows a clear advantage in consistency over Canny.

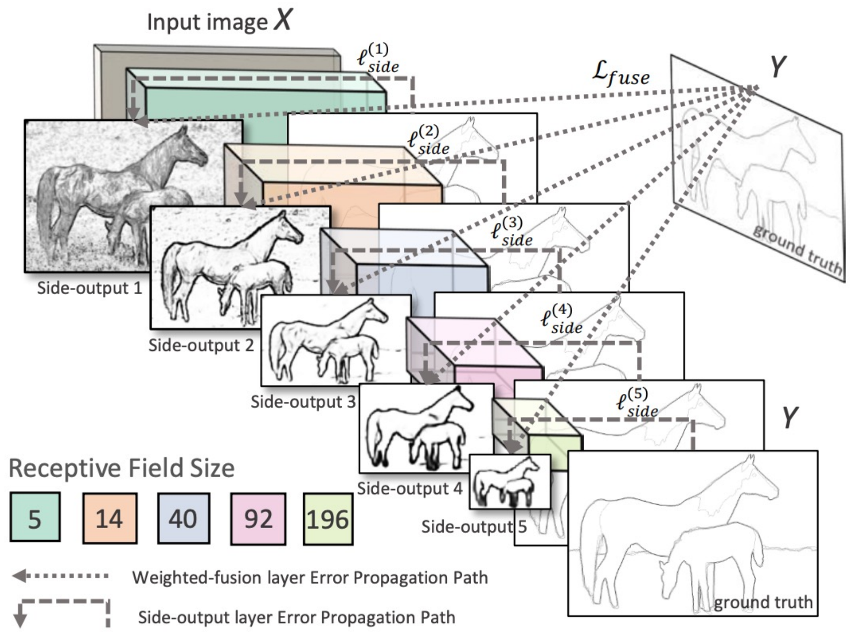


Figure 3. Illustration of our network architecture for edge detection, highlighting the error backpropagation paths. Side-output layers are inserted after convolutional layers. Deep supervision is imposed at each side-output layer, guiding the side-outputs towards edge predictions with the characteristics we desire. The outputs of HED are multi-scale and multi-level, with the side-output-plane size becoming smaller and the receptive field size becoming larger. One weighted-fusion layer is added to automatically learn how to combine outputs from multiple scales. The entire network is trained with multiple error propagation paths (dashed lines).

Summary – Stuff You Should Know

Not so good:

- Horizontal image gradient: Subtract intensity of left neighbor from pixel's intensity (filtering with $[-1, 1]$)
- Vertical image gradient: Subtract intensity of bottom neighbor from pixel's intensity (filtering with $[-1, 1]^T$)

Much better (more robust to noise):

- Horizontal image gradient: Apply derivative of Gaussian with respect to x to image filtering
- Vertical image gradient: Apply derivative of Gaussian with respect to y to image
- Magnitude of gradient: compute the horizontal and vertical image gradients, square them, sum them, and √ the sum
- Edges: Locations in image where magnitude of gradient is high
- Phenomena that causes edges: rapid change in surface's normals, depth discontinuity, rapid changes in color, change in illumination

Summary – Stuff You Should Know

- Properties of gradient's magnitude:
    - Zero far away from edge
    - Positive on both sides of the edge
    - Highest value directly on the edge
    - Higher σ emphasizes larger structures

- Canny edge detector:
    - Compute gradient's direction and magnitude
    - Non-maxima suppression
    - Thresholding at two levels and linking

- OpenCV functions:
    - cv2.GaussianBlur()
    - cv2.Sobel():)
    - cv2.Laplacian())
    - cv2.Canny()

Next time...

- Image pyramids