COMP0026: Image Processing

Image Filtering

Q: 从151页开始才是b站内容。

UCL

[Matlab]

COMP0026: Image Processing

Image Filtering

```
I = imread('moon.tif');
h = fspecial('unsharp');
I2 = imfilter(I,h);
imshow(I), title('Original Image')
figure, imshow(I2), title('Filtered Image')
```

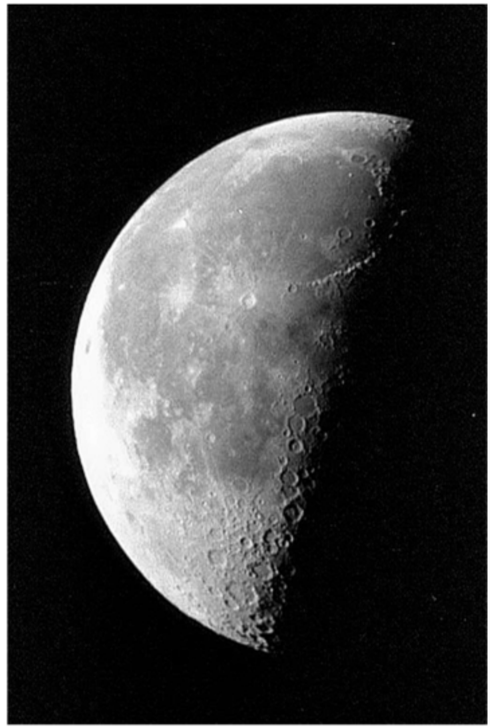
UCL

[Matlab]

COMP0026: Image Processing

Image Filtering

```
I = imread('moon.tif');
h = fspecial('unsharp');
I2 = imfilter(I,h);
imshow(I), title('Original Image')
figure, imshow(I2), title('Filtered Image')
```



UCL

[Matlab]

Lectures will be Recorded

Image Filtering

input image ⟶ filter function ⟶ output image

Image Filtering

input image ⟶ filter function ⟶ output image

Functions

linear/non-linear

Image Filtering

input image ⟶ filter function ⟶ output image

Functions
- **linear/non-linear**

Neighborhoods
- **local/non-local**

Goals

Goals

- **Low-level data (e.g., pixel) processing operations**

Goals

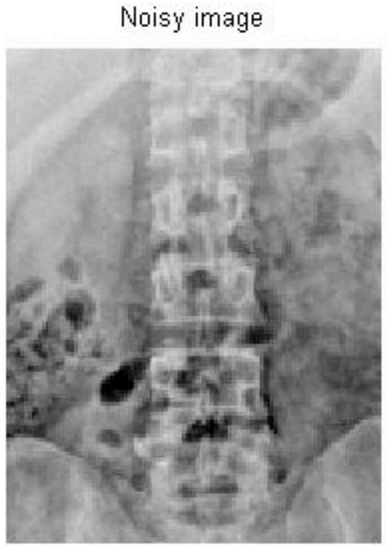
- **Low-level data (e.g., pixel) processing operations**

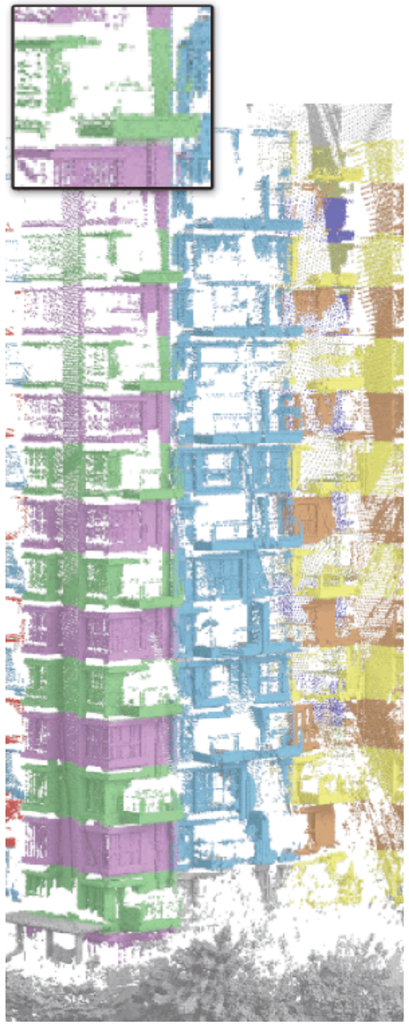
- **Smoothing and noise reduction**

Goals

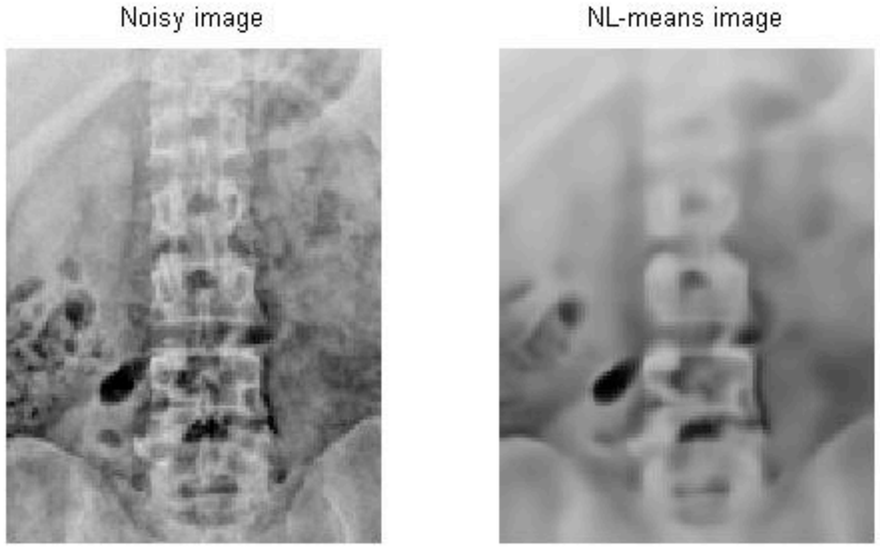
- Low-level data (e.g., pixel) processing operations
- Smoothing and noise reduction
- Feature extraction and enhancement

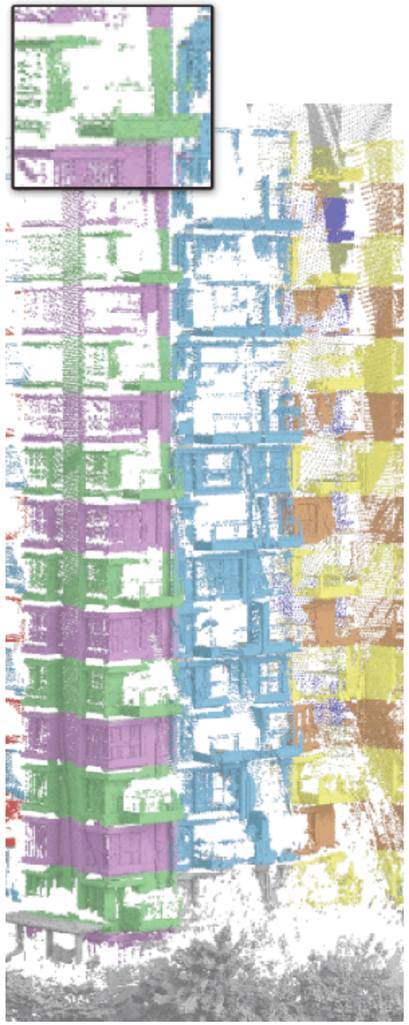
Non-local Filtering



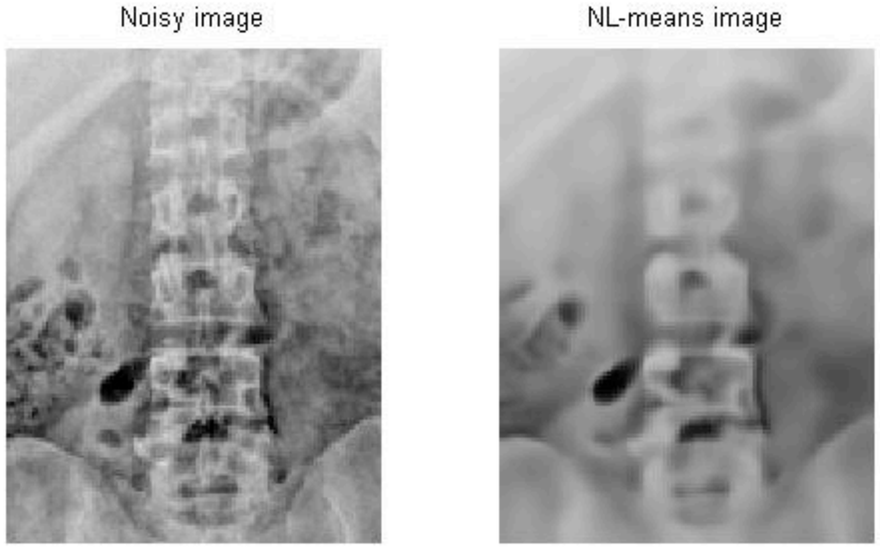


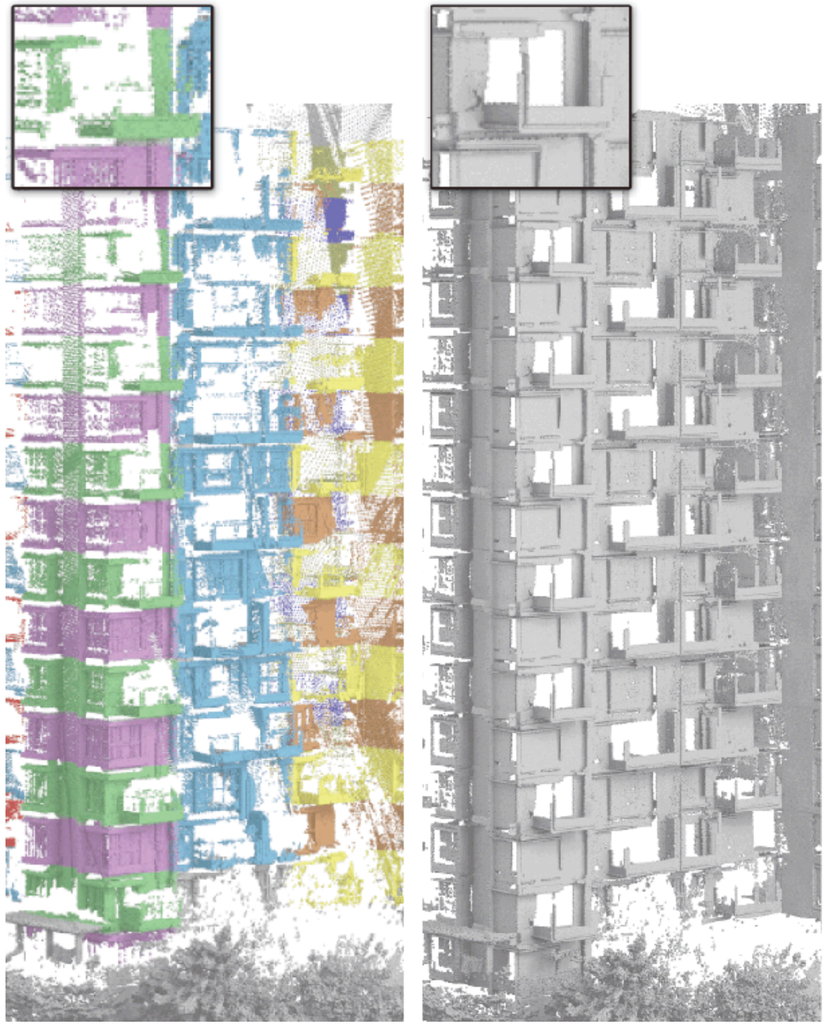
Non-local Filtering





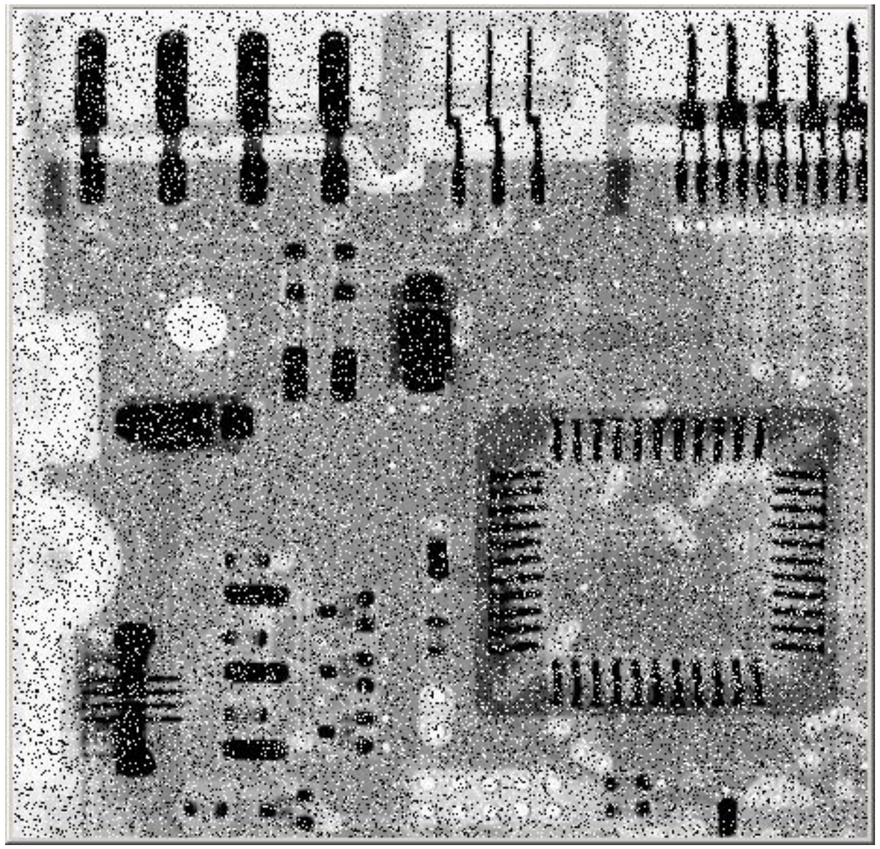
Non-local Filtering





Denoising: Remove “Noise”

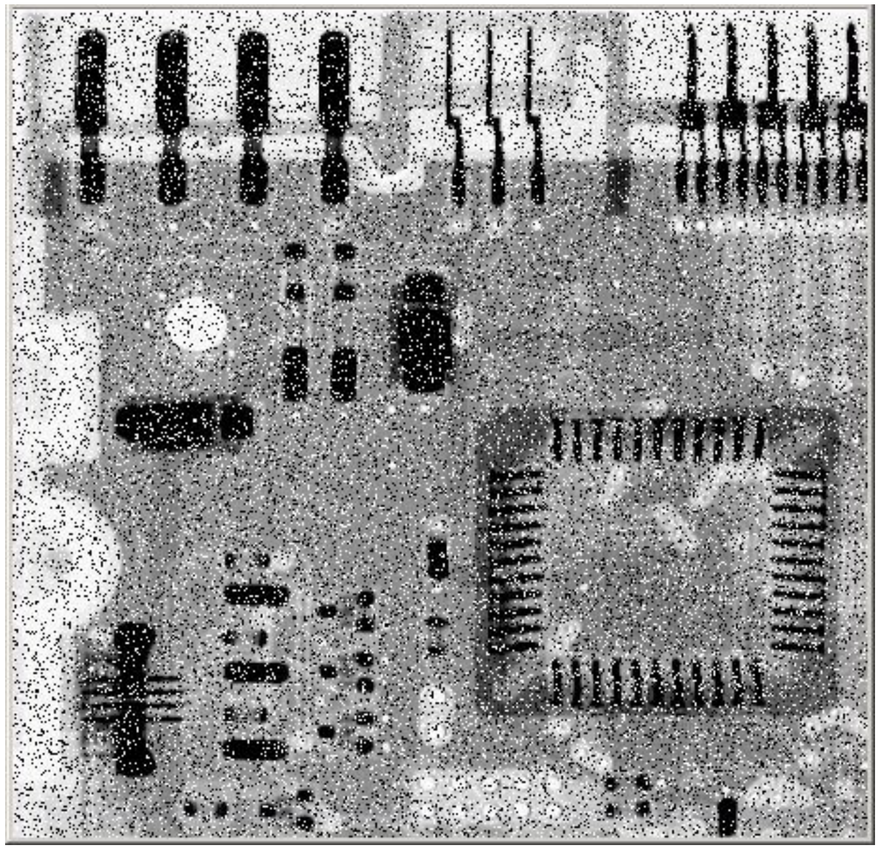
(if you know how to model that noise specifically!)

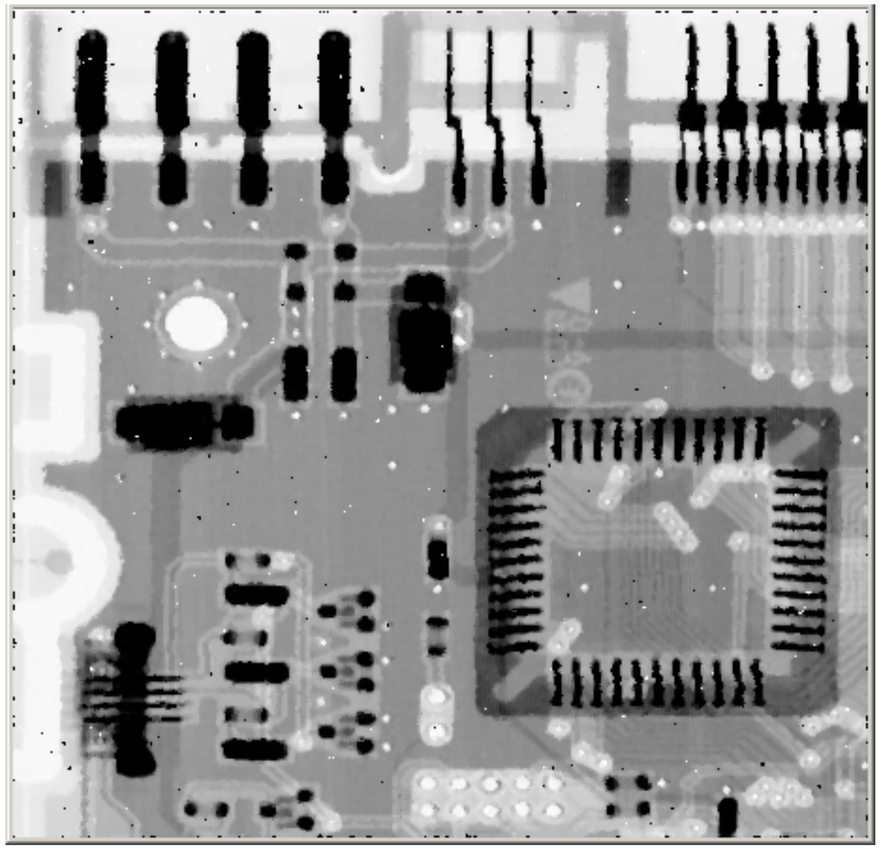


Input image from DIP by Gonzalez+Woods, 2008

Denoising: Remove “Noise”

(if you know how to model that noise specifically!)





median filter

Input image from DIP by Gonzalez+Woods, 2008

Outline

- Linear filtering
- Convolution operations
  - Smoothing
  - Low-pass and high-pass filters
  - Sharpen
- Filtering in the Fourier domain
- Rank filters

Linear Mappings

Linear Mappings

- **Equivalently, *L* is *linear* if**

Linear Mappings

- **Equivalently, *L* is *linear* if**

$$L(I_1 + I_2) = L(I_1) + L(I_2)$$

Linear Mappings

- **Equivalently, *L* is *linear* if**

$$L(I_1 + I_2) = L(I_1) + L(I_2)$$

$$L(\alpha I) = \alpha L(I) \quad \forall \alpha$$

(i.e., *L* preserves linear combinations)

Linear Mappings

- **Equivalently, *L* is *linear* if**

$$L(I_1 + I_2) = L(I_1) + L(I_2)$$

$$L(\alpha I) = \alpha L(I) \quad \forall \alpha$$

(i.e., *L* preserves linear combinations)

$$L(\alpha I_1 + \beta I_2) = \alpha L(I_1) + \beta L(I_2)$$

Convolutional Filters

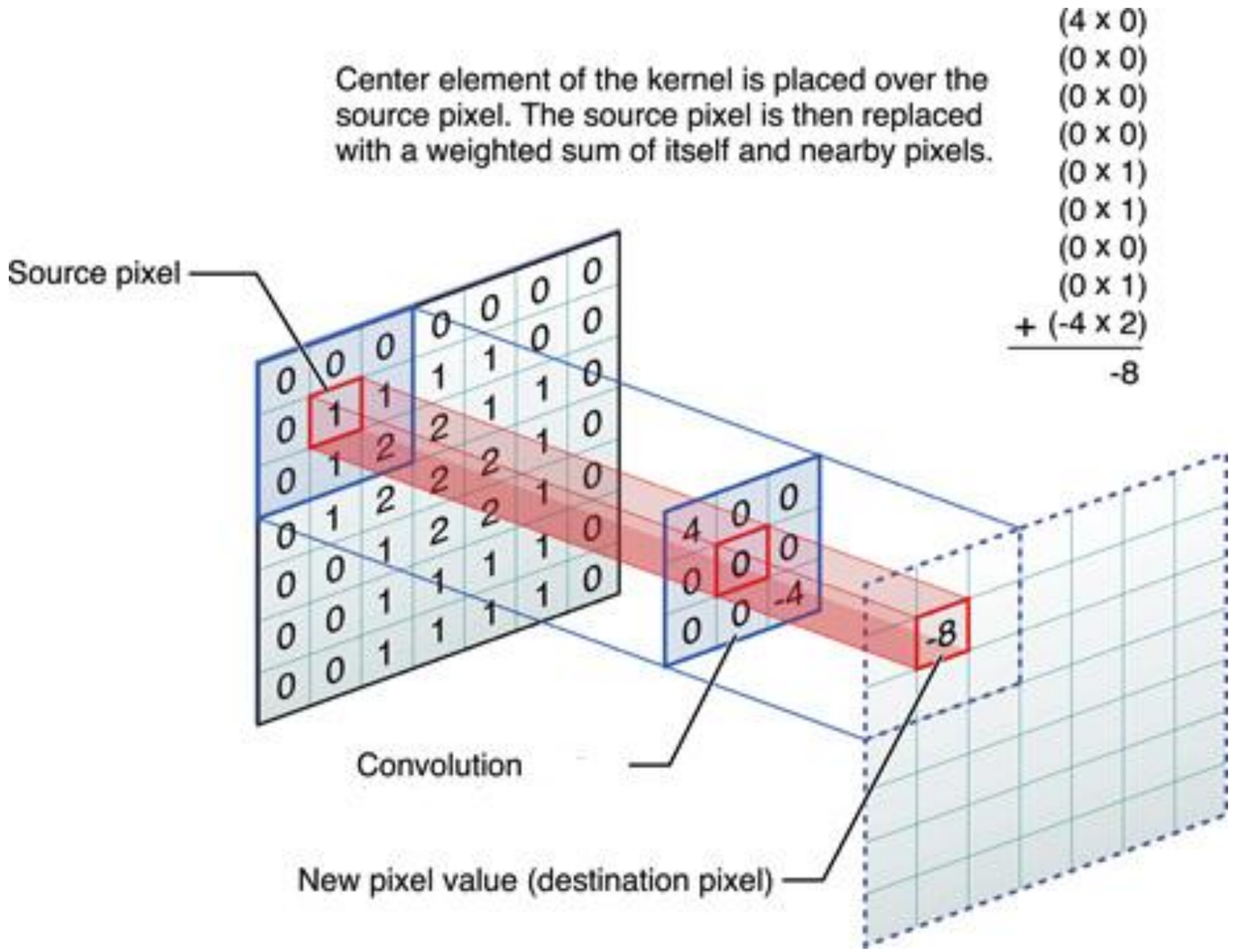
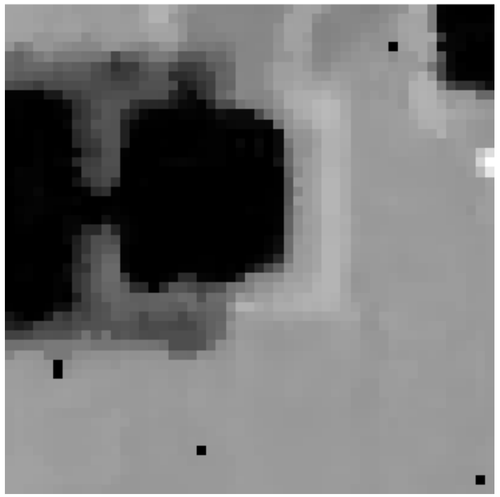
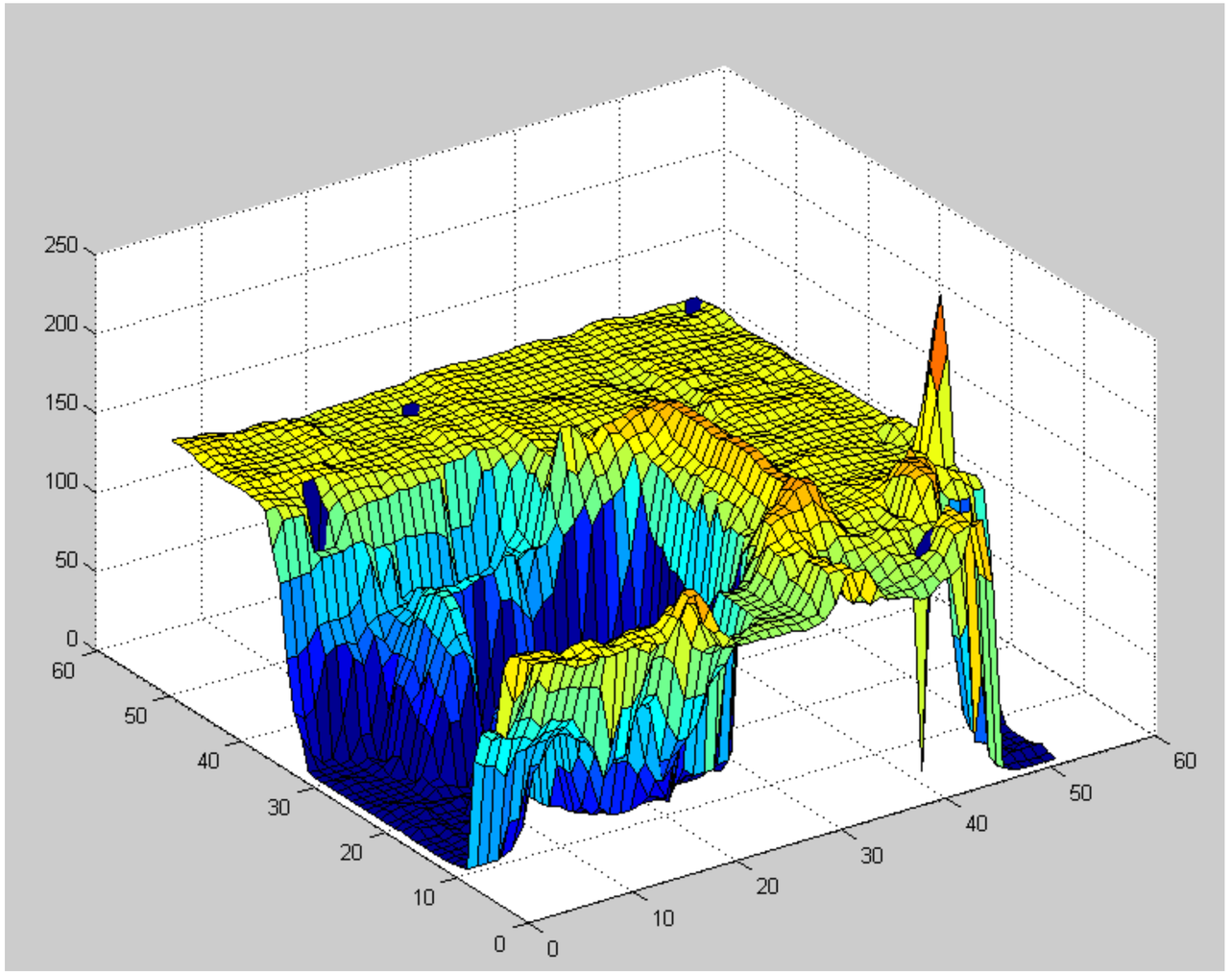


Image ~ Heightfield

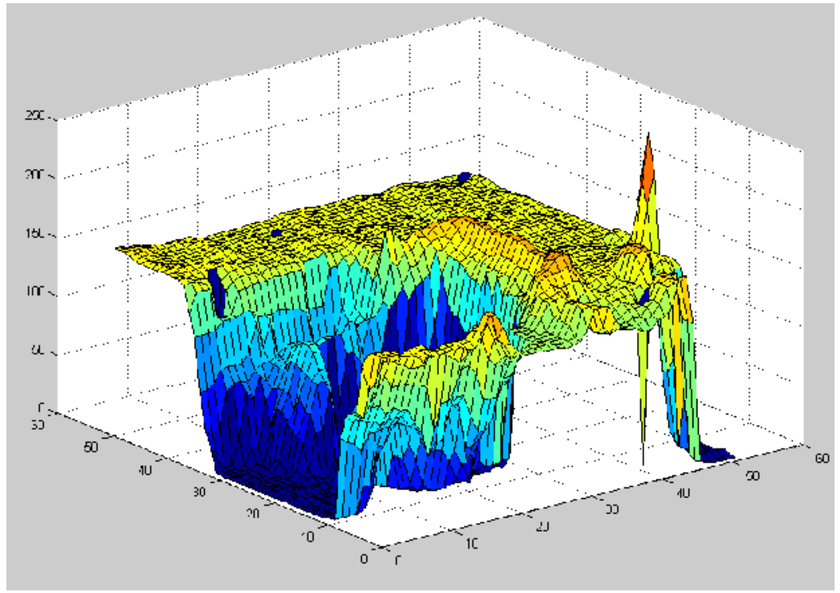
I

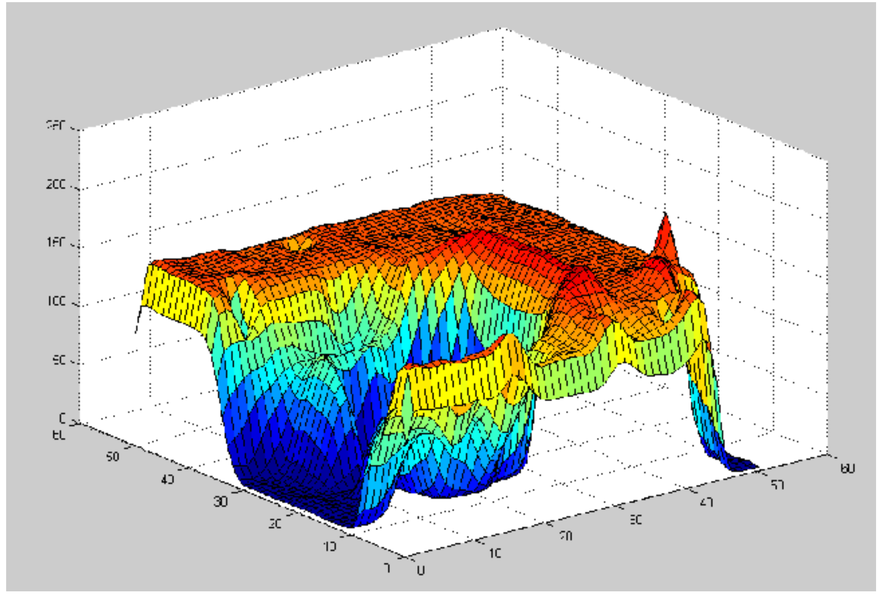


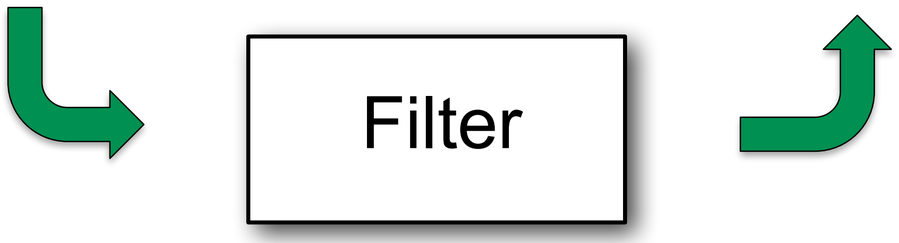
z = *I*'s intensity



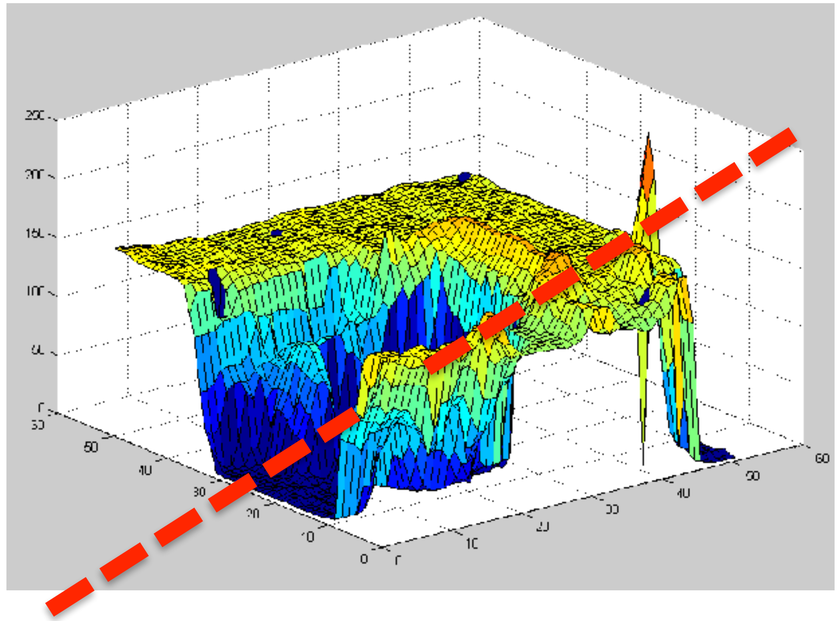
Filter: Let Intensity Through Selectively

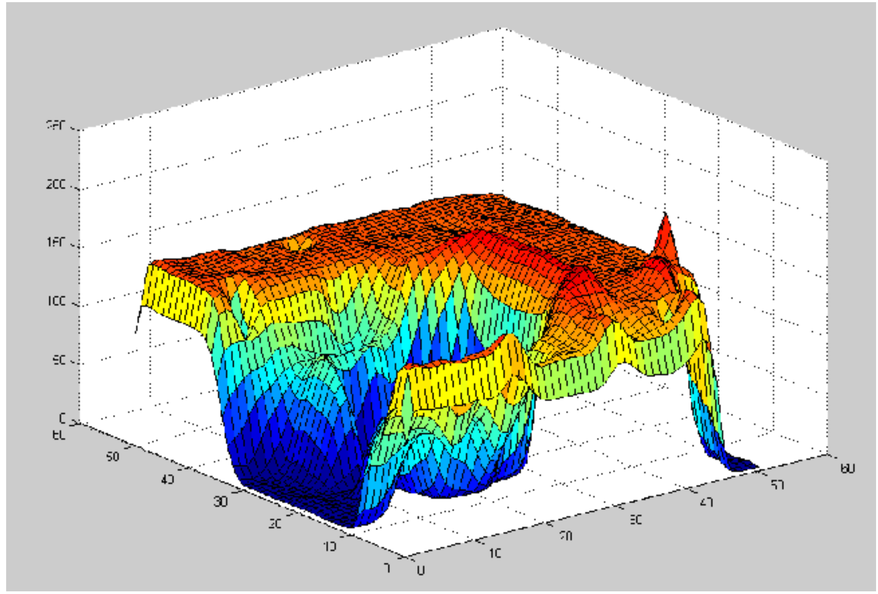


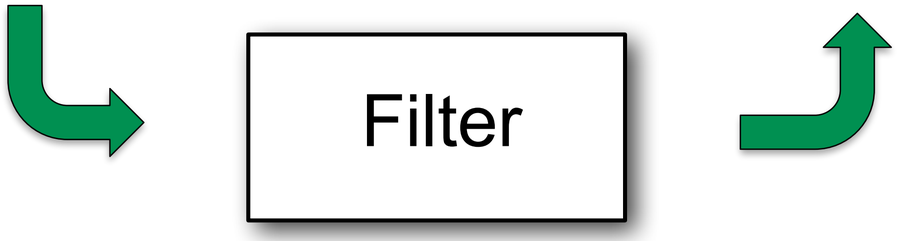




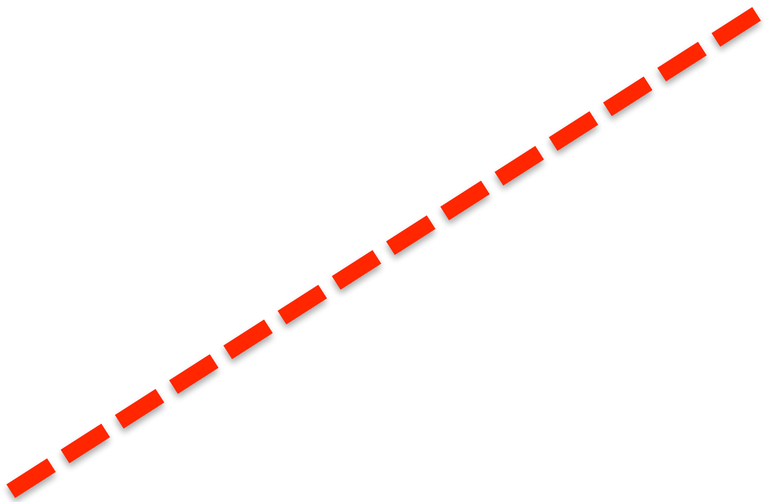
Filter: Let Intensity Through Selectively





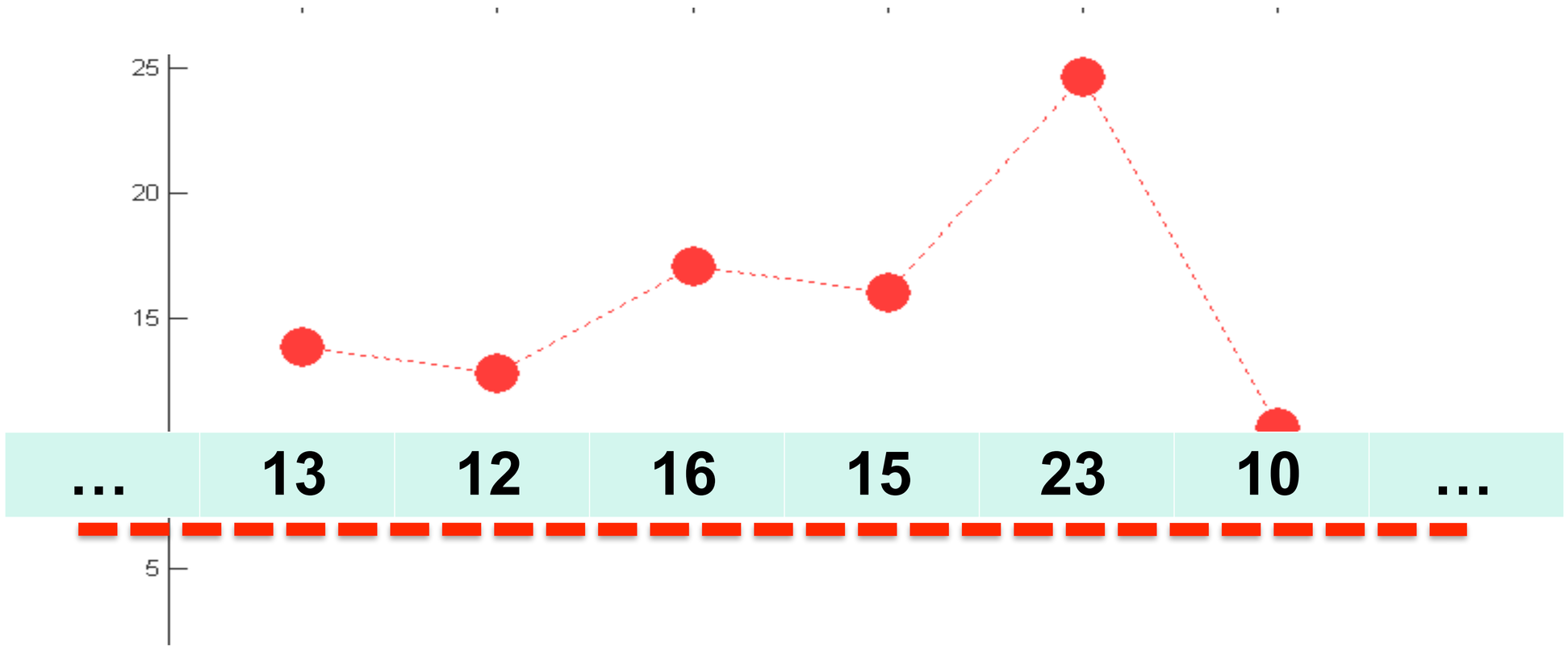


Filter: Let Intensity Through Selectively

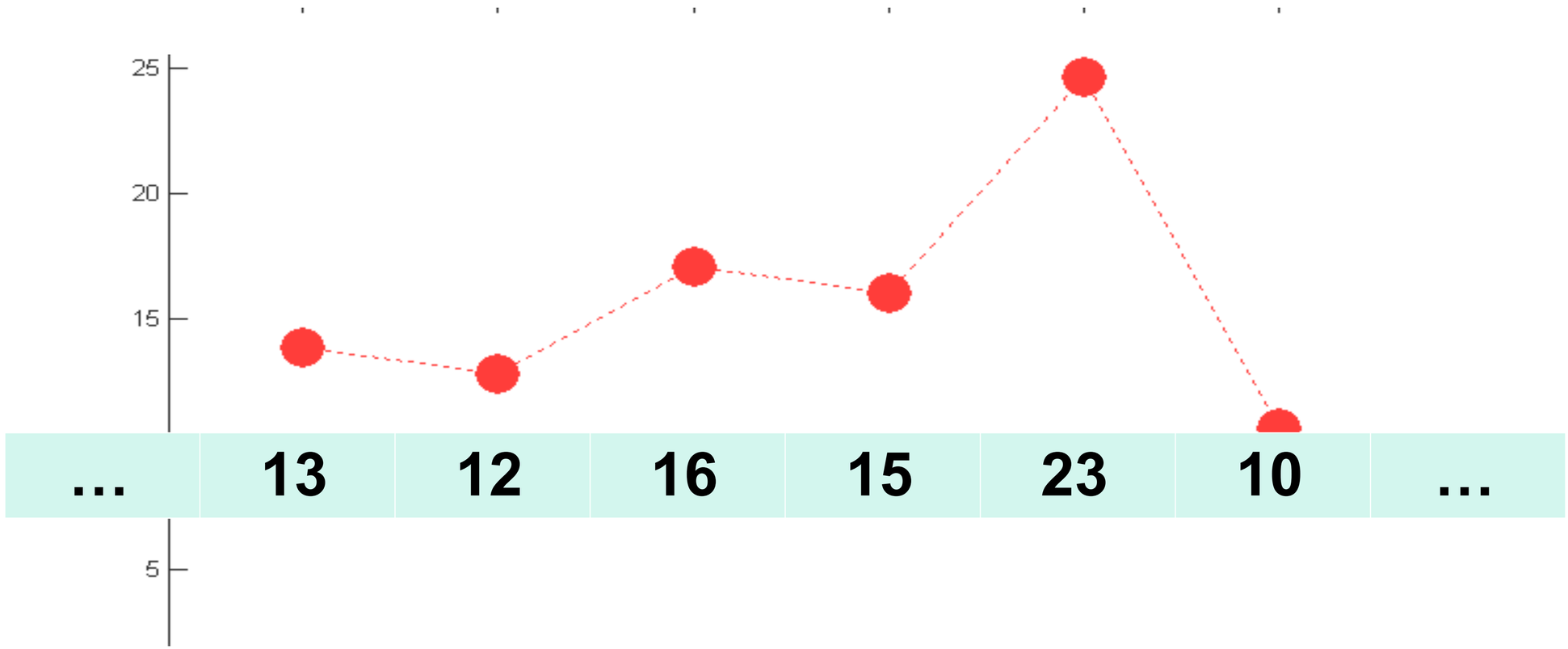


Filter: Let Intensity Through Selectively

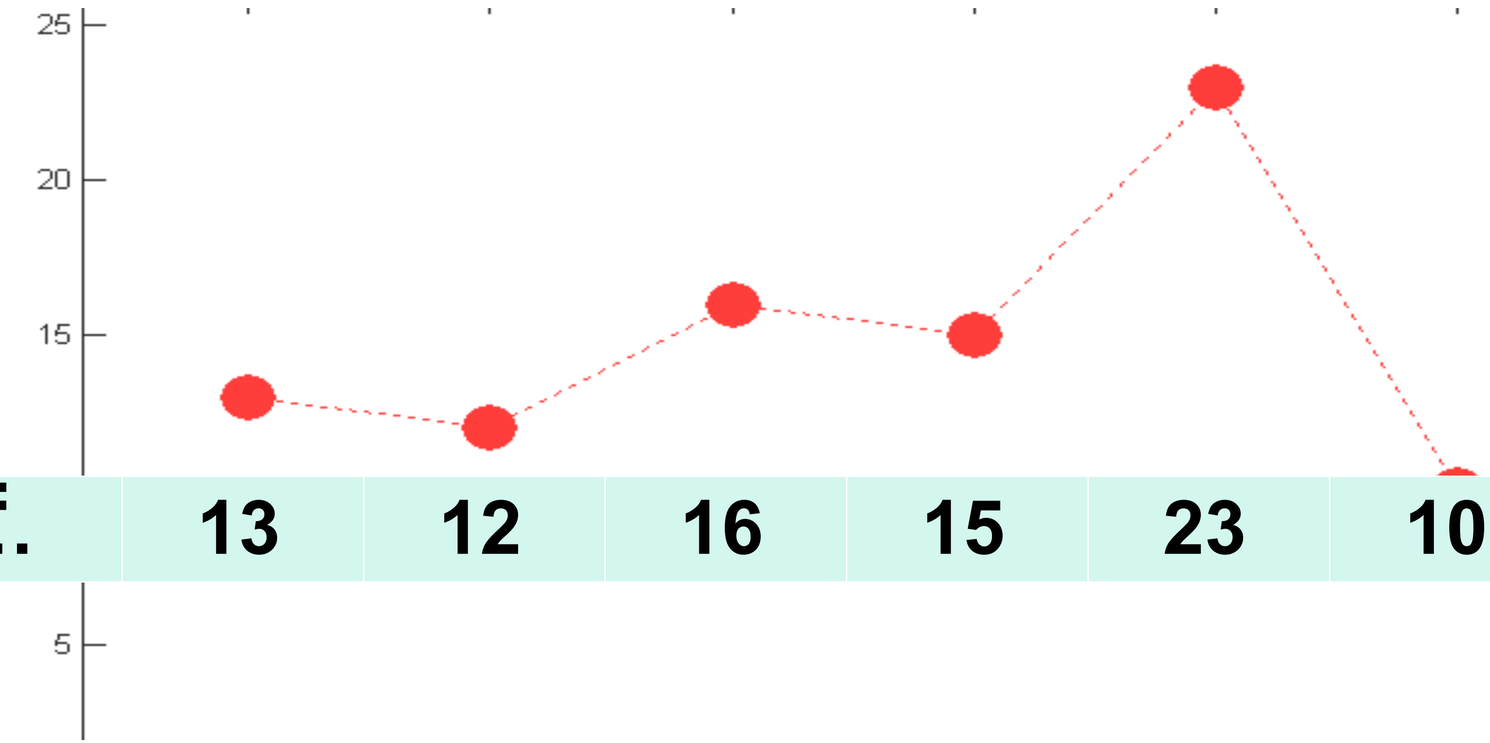
A Row of Pixel Intensities



A Row of Pixel Intensities



Linear Operations: Weighted Sum

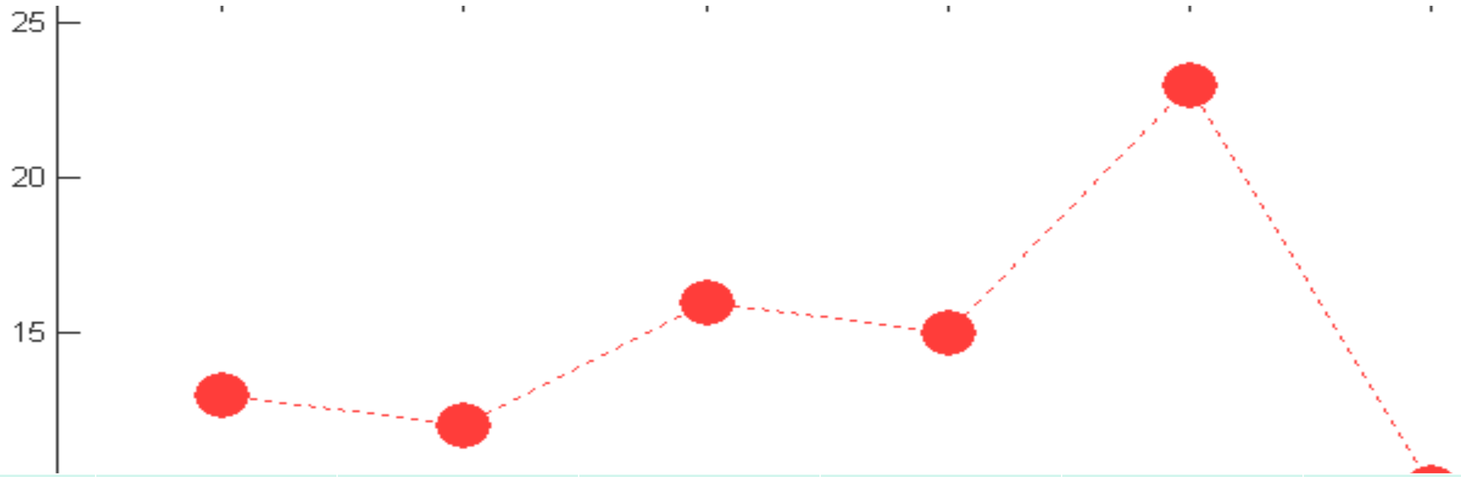


$I =$

...	13	12	16	15	23	10	...

$I' =$

...	??	??	??	??	...

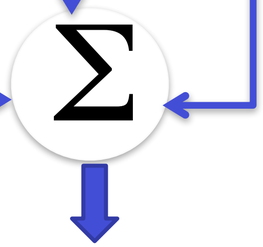


$I =$

...	13	12	16	15	23	10	...

$K =$

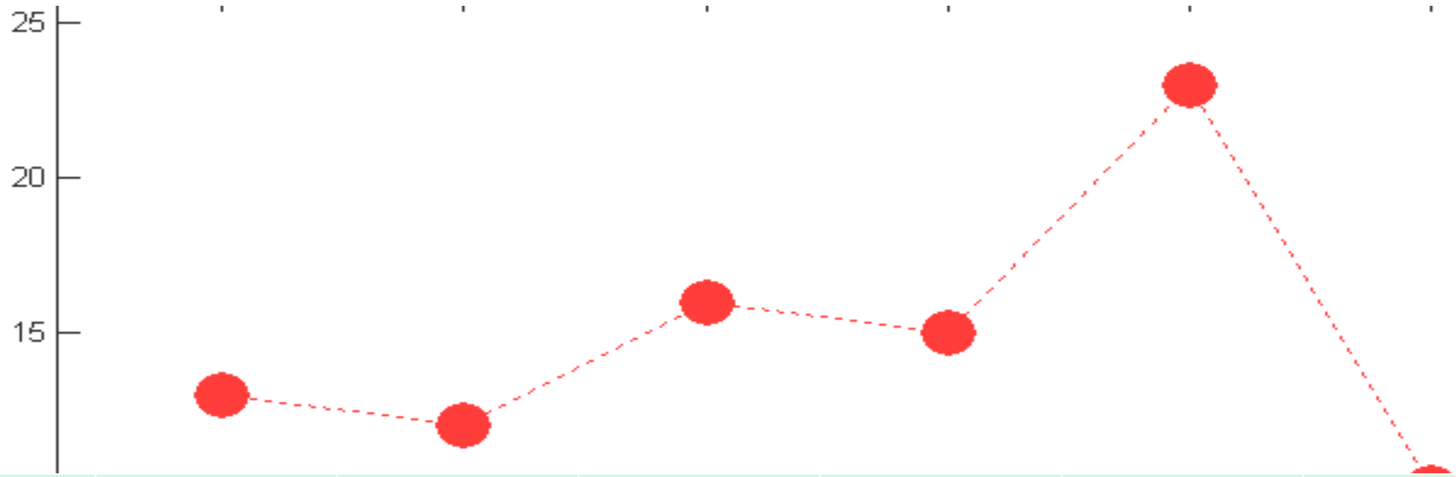
*	*	*
1/3	1/3	1/3



Neighborhood: $N(x) = x \pm 1$
Operation: Average

$I' =$

...	??	??	??	??	...

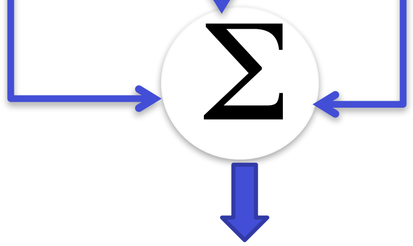


$I =$

...	13	12	16	15	23	10	...

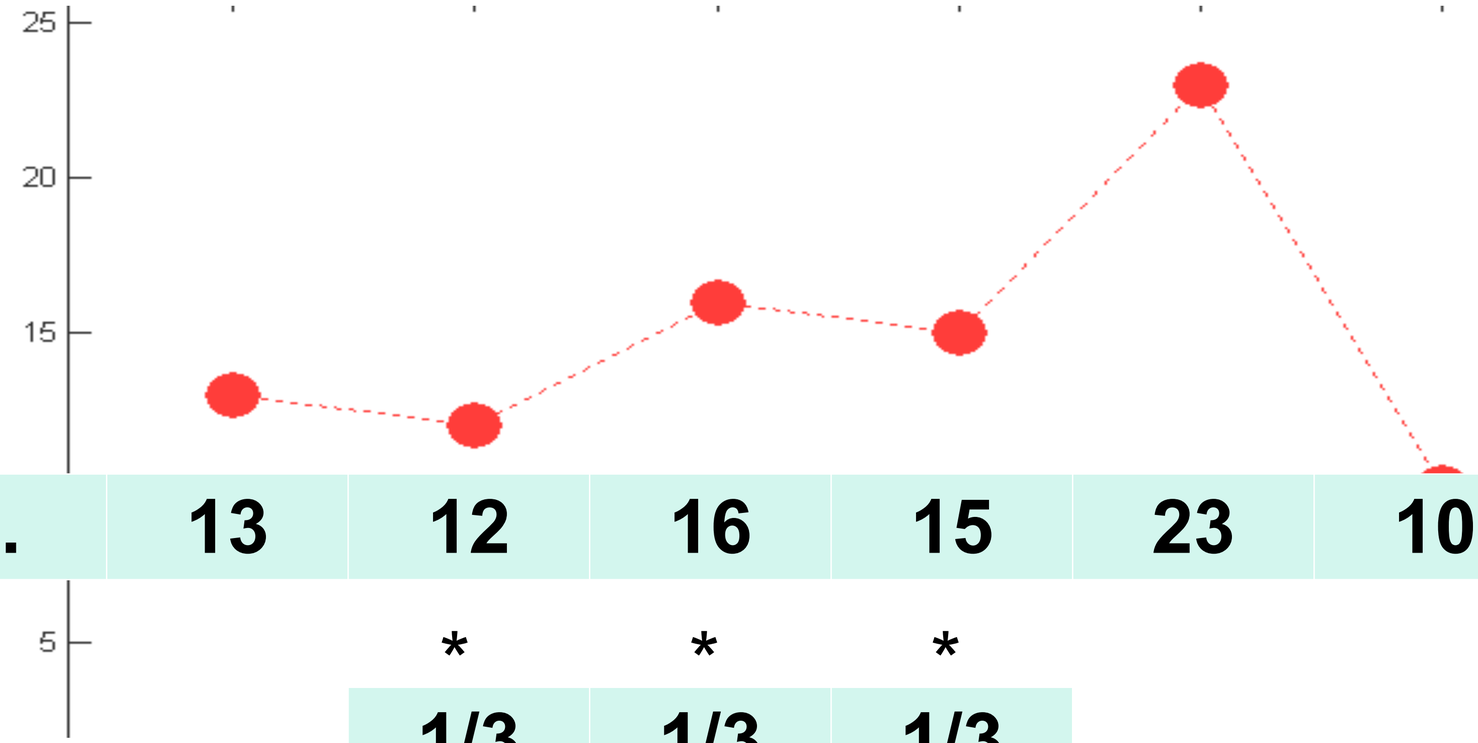
$K =$

	*		*		*	
	1/3		1/3		1/3	



$I' =$

...	13.66	??	??	??	...



$I =$

...	13	12	16	15	23	10	...

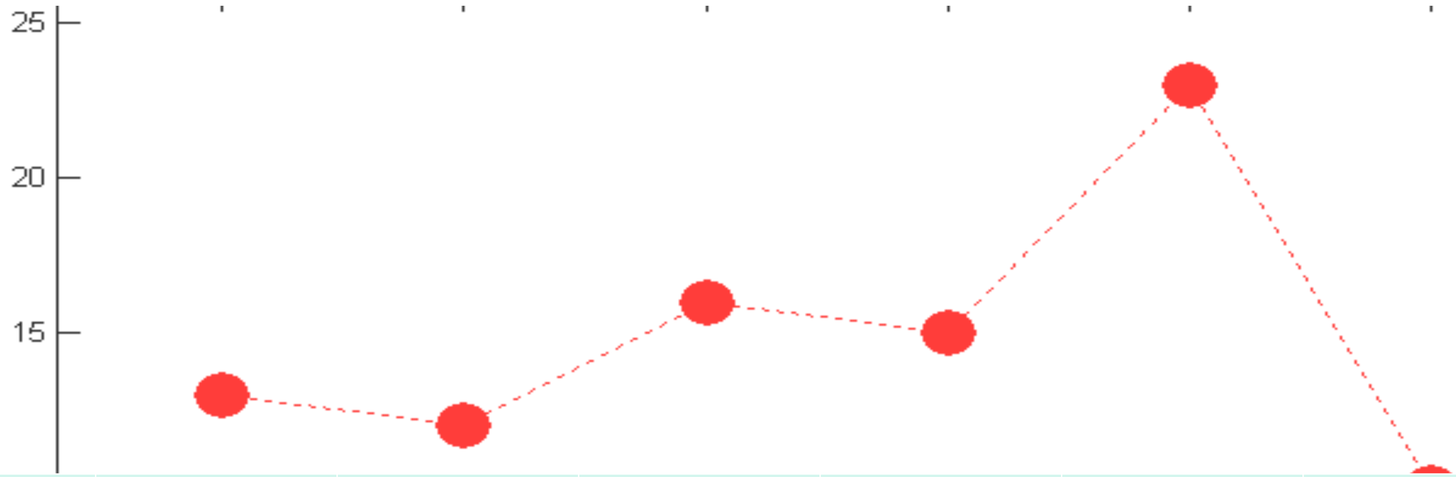
$K =$

	*		*		*	
	1/3		1/3		1/3	

Σ

$I' =$

...	13.66	??	??	??	...

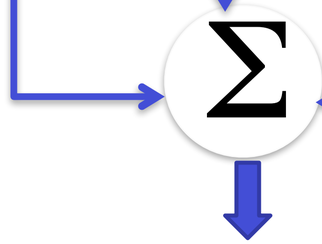


$I =$

...	13	12	16	15	23	10	...

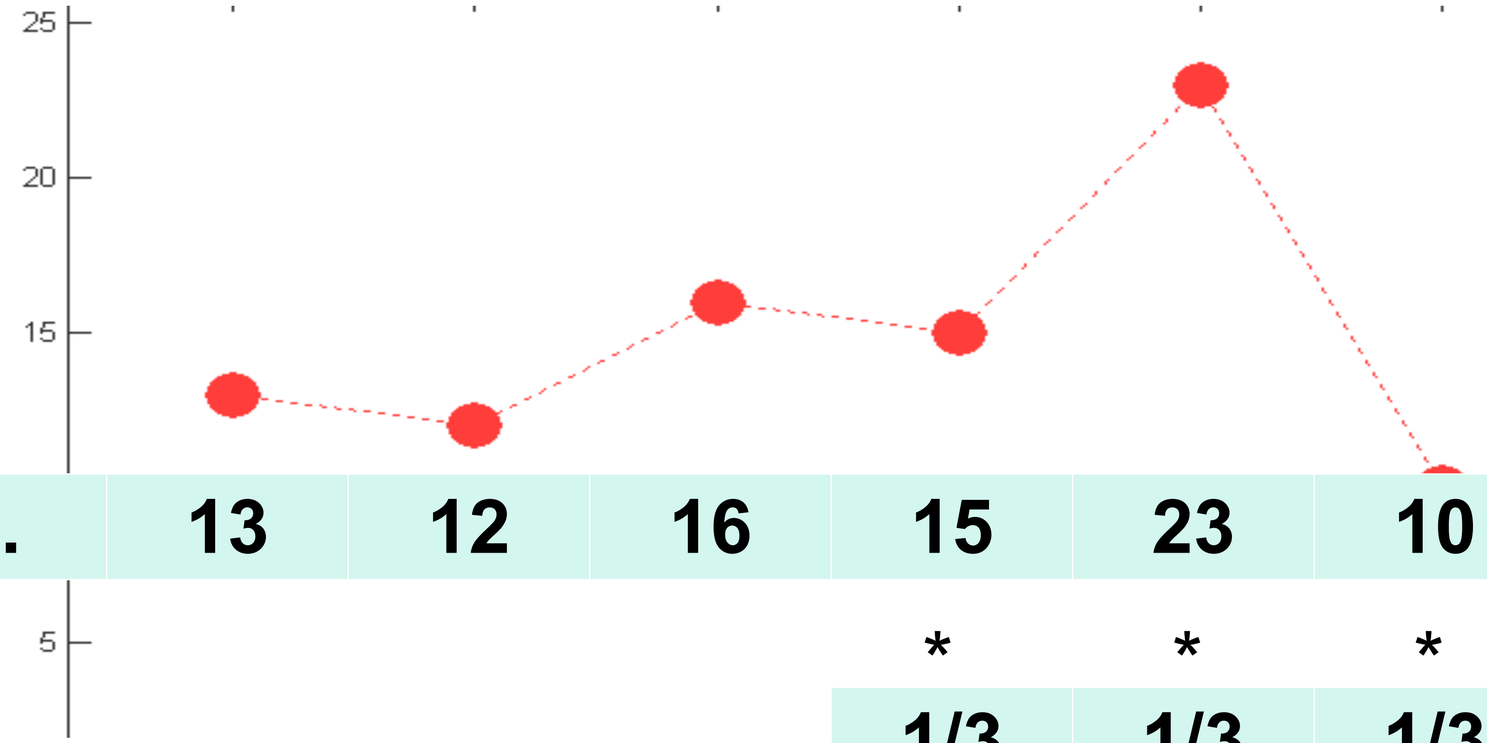
$K =$

	*		*		*	
	1/3		1/3		1/3	



$I' =$

...	13.66	14.33	??	??	...



$I =$

…	13	12	16	15	23	10	…

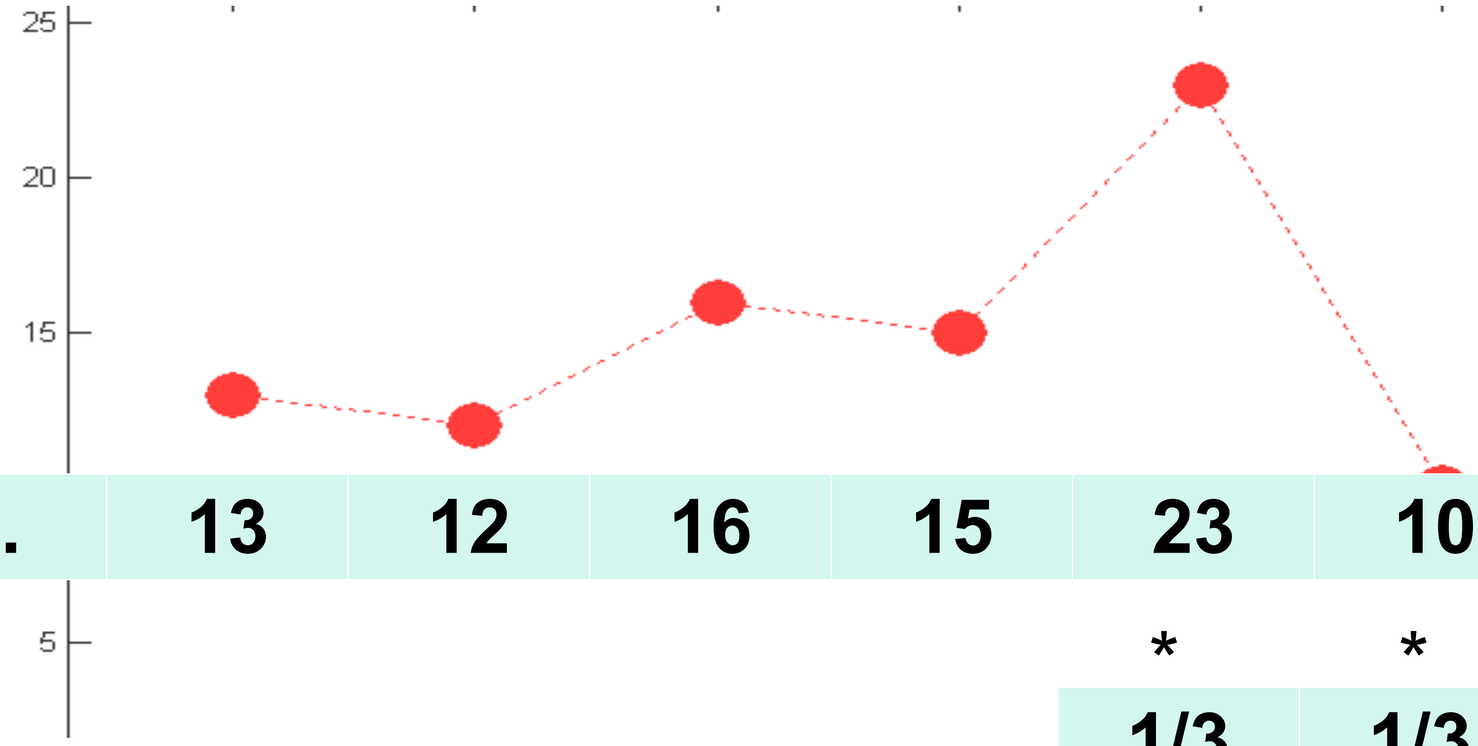
$K =$

			*	*	*	
			1/3	1/3	1/3	

Σ

$I' =$

…	13.66	14.33	18	??	…

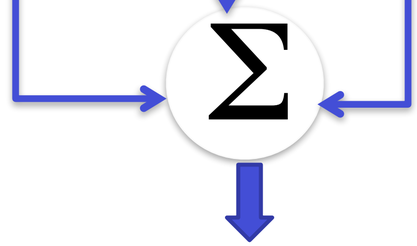


$I =$

…	13	12	16	15	23	10	…

$K =$

	*		*		*	
1/3		1/3		1/3		



$I' =$

…	13.66	14.33	18	16	…

Correlation

Correlation

- **Linear operation in 1D:**

$$I'(x) = \sum_{i=-a}^{a} K(i)I(x+i)$$

Correlation

- **Linear operation in 1D:**

$$I'(x) = \sum_{i=-a}^{a} K(i)I(x+i)$$

- ***I* is the input image; *I'* is the output of the operation**

Correlation

- **Linear operation in 1D:**

$$I'(x) = \sum_{i=-a}^{a} K(i)I(x+i)$$

- ***I* is the input image; *I'* is the output of the operation**
- ***K* is the *kernel* of the operation; many choices!**

Correlation

- **Linear operation in 1D:**

$$I'(x) = \sum_{i=-a}^{a} K(i)I(x+i)$$

- ***I* is the input image; *I'* is the output of the operation**
- ***K* is the *kernel* of the operation; many choices!**
- ***N*(*x*) is a neighbourhood of size (*2a+1*)**

Correlation

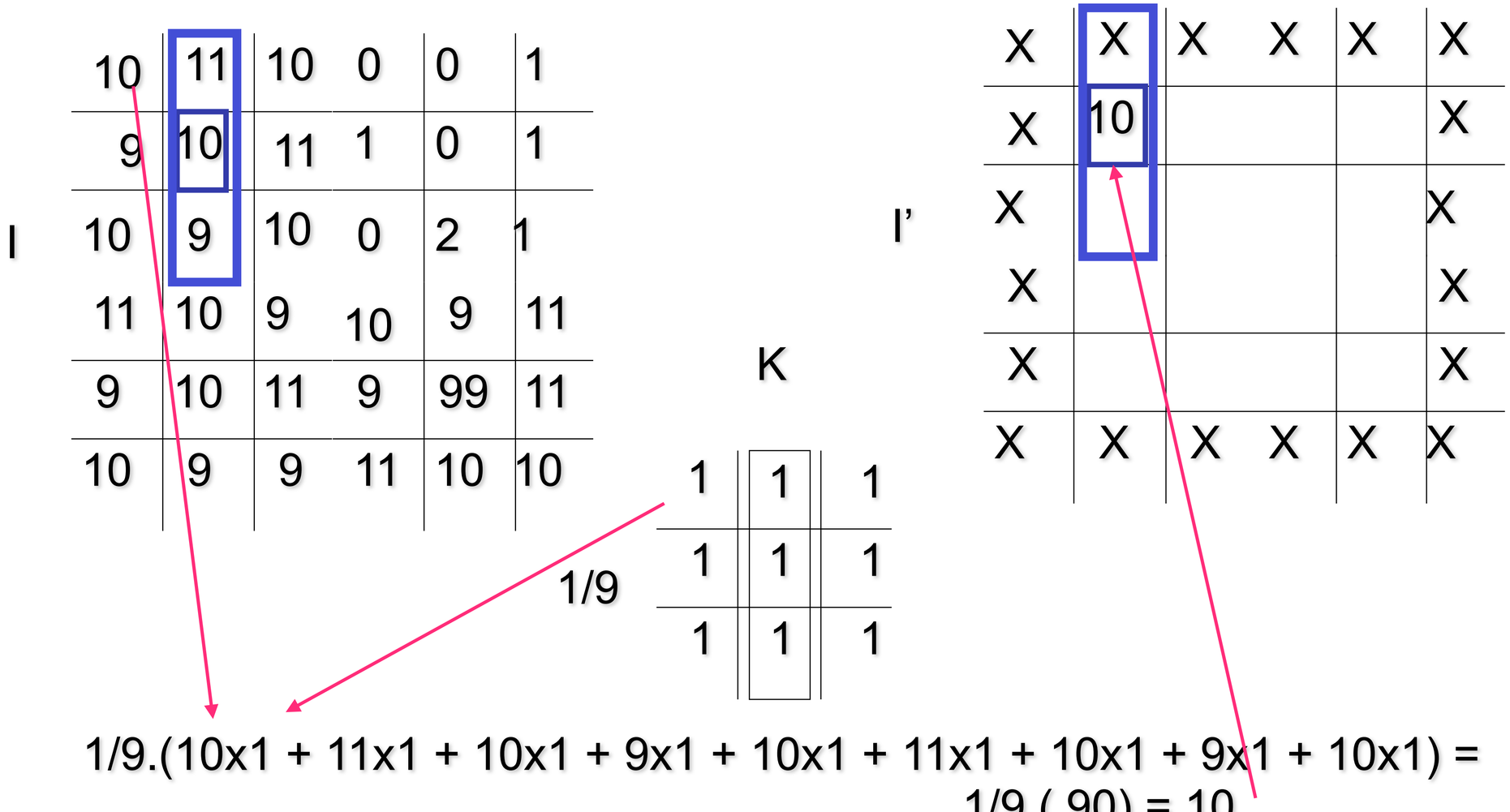
- **Linear operation in 1D:**

$$I'(x) = \sum_{i=-a}^{a} K(i) I(x+i)$$

- **Linear operation in 2D:**

$$I'(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} K(i,j) I(x+i, y+j)$$

- ***I* is the input image; *I'* is the output of the operation**
- ***K* is the *kernel* of the operation; many choices!**
- ***N*(*x*,*y*) is a neighbourhood of size (*2a+1)* x *(2b+1)***



Slide credit: Christopher Rasmussen

$$I'(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} K(i,j) I(x+i, y+j)$$

I

10	11	10	0	0	1
9	10	11	1	0	1
10	9	10	0	2	1
11	10	9	10	9	11
9	10	11	9	99	11
10	9	9	11	10	10

K

1	1	1
1	1	1
1	1	1

1/9

I’

X	X	X	X	X	X
X	10				X
X					X
X					X
X					X
X	X	X	X	X	X

1/9.(10x1 + 11x1 + 10x1 + 9x1 + 10x1 + 11x1 + 10x1 + 9x1 + 10x1) =
1/9.(90) = 10

Slide credit: Christopher Rasmussen

Correlation with a Unit-Impulse Image

$I =$

0	0	0	1	0	0	0	0

$K =$

5	6	7

$I' =$

_	_	_	_	_	_	_	_

Correlation with a Unit-Impulse Image

$I =$

0	0	0	1	0	0	0	0

$K =$

5	6	7

$I' =$

0	0	7	6	5	0	0	0

Correlation reveals copy of *K*, rotated by 180°

Correlation with a Unit-Impulse Image

$I =$

0	0	0	1	0	0	0	0

$K =$

5	6	7

$I' =$

0	0	7	6	5	0	0	0

Correlation reveals copy of *K*, rotated by 180°

Would be nice to get result that is **NOT** rotated!

Correlation/Convolution

$$I'(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} K(i,j) I(x+i, y+j)$$

Correlation/Convolution

$$I'(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} K(i,j) I(x+i, y+j)$$

Convolution

Correlation/Convolution

$$I'(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} K(i,j) I(x+i, y+j)$$

Convolution

$$\begin{aligned} I'(x,y) &= \sum_{i=-a}^{a} \sum_{j=-b}^{b} K(i,j) I(x-i, y-j) \\ &= \sum_{i=-a}^{a} \sum_{j=-b}^{b} K(-i,-j) I(x+i, y+j) \end{aligned}$$

Correlation/Convolution

$$I'(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} K(i,j) I(x+i, y+j)$$

Convolution

$$\begin{aligned} I'(x,y) &= \sum_{i=-a}^{a} \sum_{j=-b}^{b} K(i,j) I(x-i, y-j) \\ &= \sum_{i=-a}^{a} \sum_{j=-b}^{b} K(-i,-j) I(x+i, y+j) \end{aligned}$$

$$K(i,j) = K(-i,-j) \quad \Leftrightarrow \quad \text{convolution} \equiv \text{correlation}$$

Correlation/Convolution

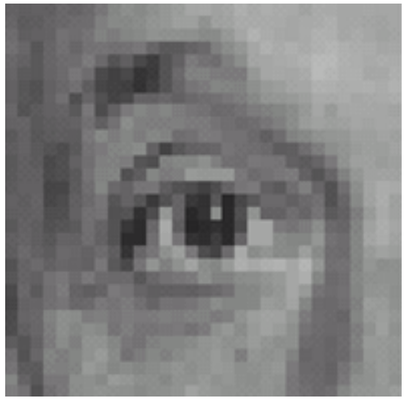
$$I'(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} K(i,j) I(x+i, y+j)$$

Convolution

$$I'(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} K(i,j) I(x-i, y-j)$$

$$I' = K * I$$

Linear Filtering (warm-up)



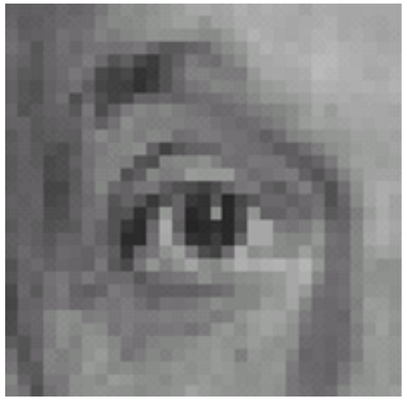
Original

0	0	0
0	1	0
0	0	0

?

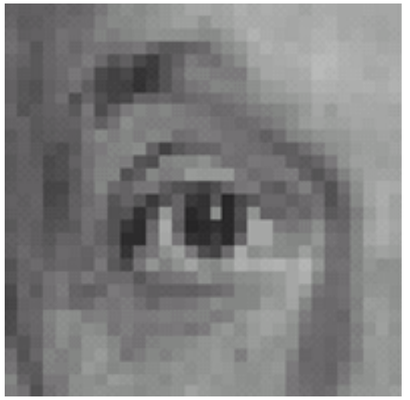
Slide credit: D.A. Forsyth

Linear Filtering (warm-up)



Original

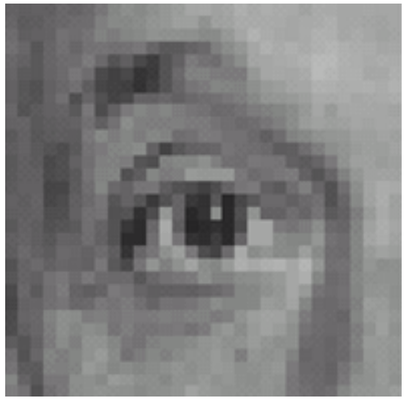
0	0	0
0	1	0
0	0	0



Filtered
(no change)

Linear Filtering (Convolution)

$$I'(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} K(i,j) I(x-i, y-j)$$



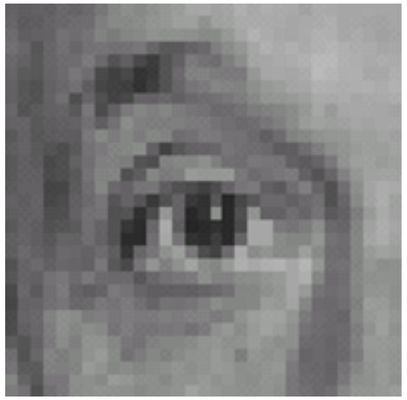
Original

0	0	0
1	0	0
0	0	0

?

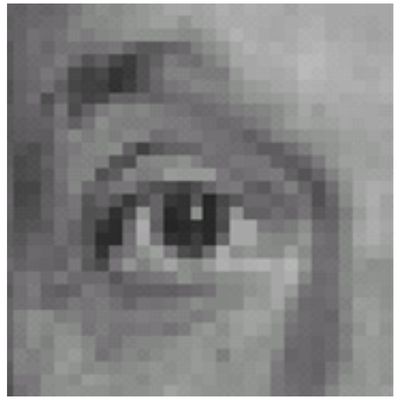
Linear Filtering (Convolution)

$$I'(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} K(i,j) I(x-i, y-j)$$



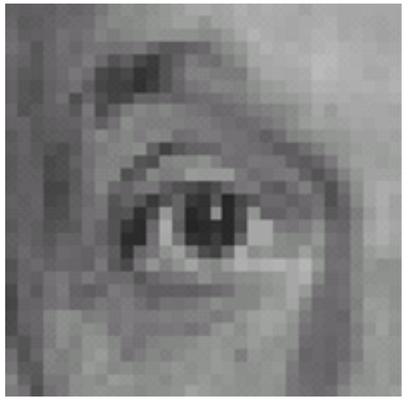
Original

0	0	0
1	0	0
0	0	0



Shifted left
By 1 pixel

Linear Filtering

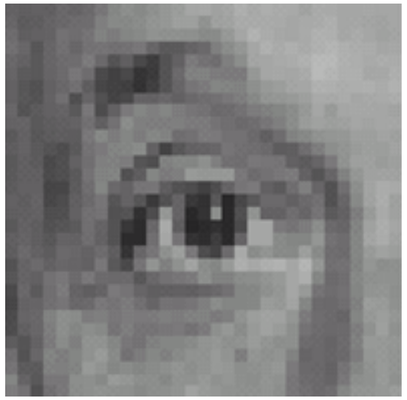


Original

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

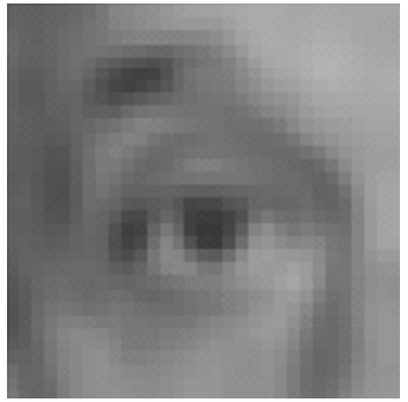
?

Linear Filtering



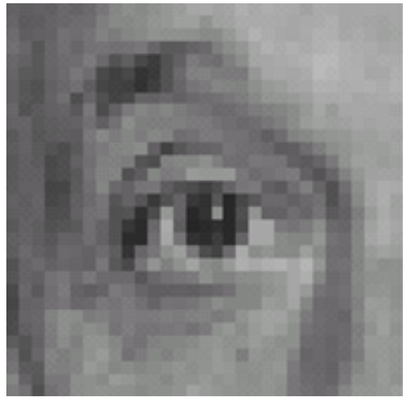
Original

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



Blur (with a box filter)

Linear Filtering

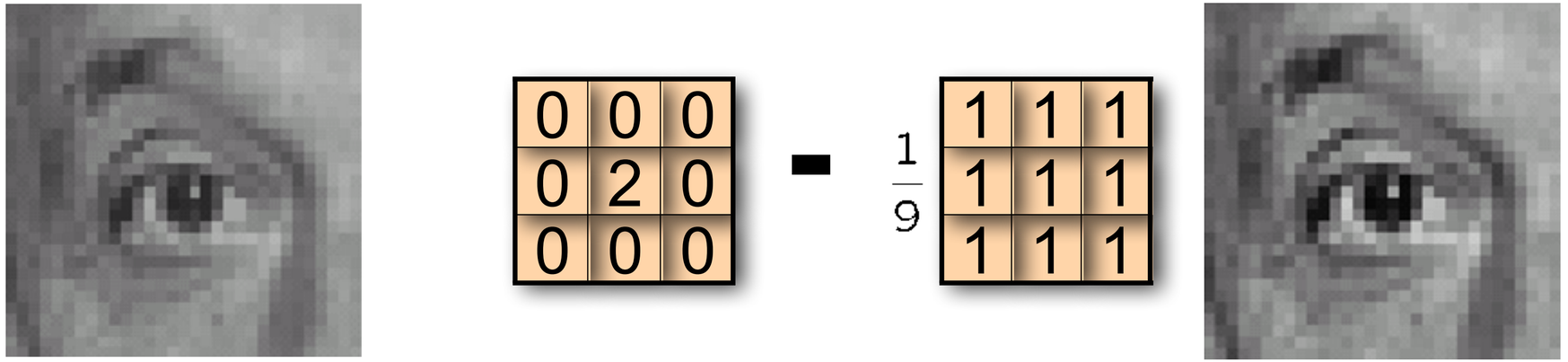


Original

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{matrix} - \frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} \quad ?$$

(Note that filter sums to 1)

Linear Filtering

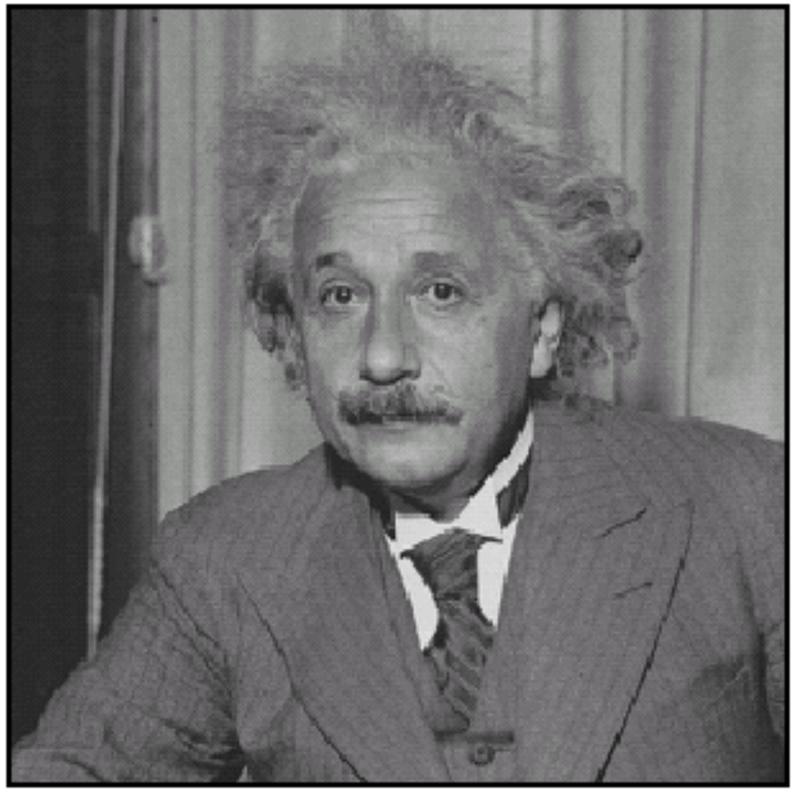


Original

Sharpening filter

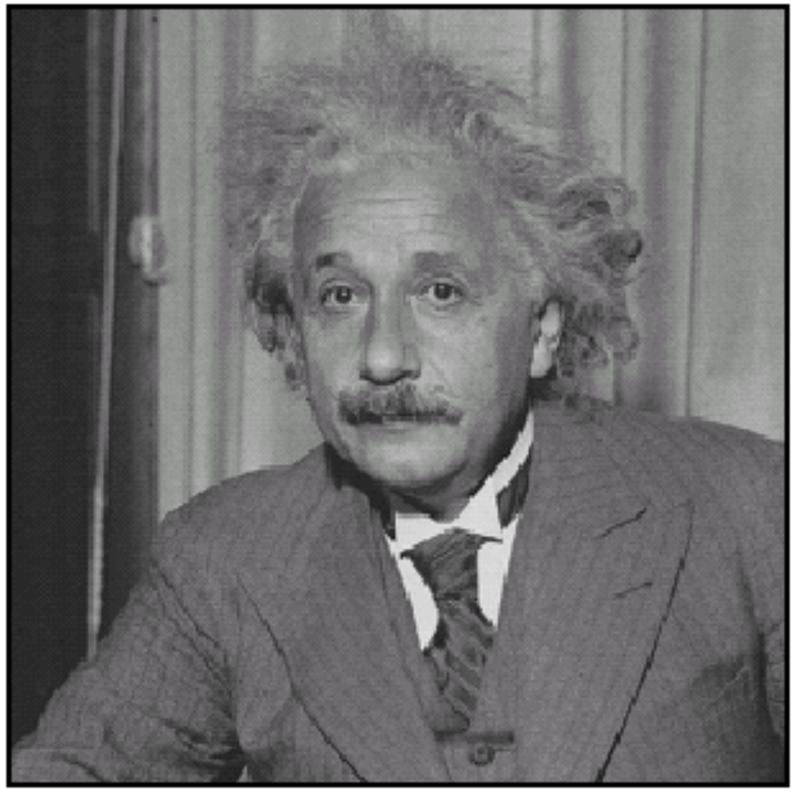
- Amplifies differences with local average
- Also known as Laplacian

Sharpening

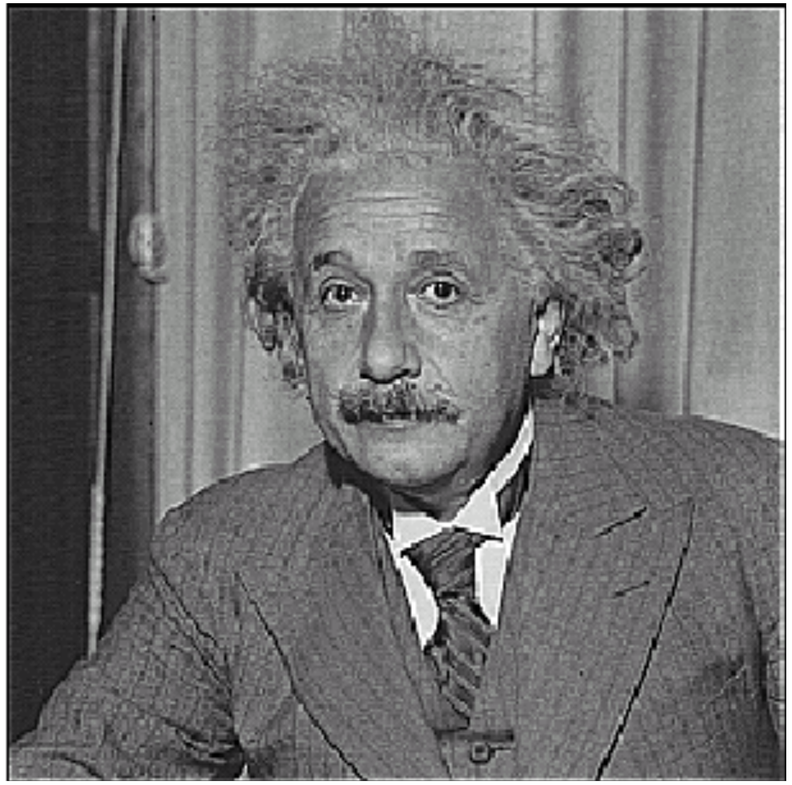


before

Sharpening



before



after

Example

K=ones(9,9);

I2=conv2(I,K);

Example

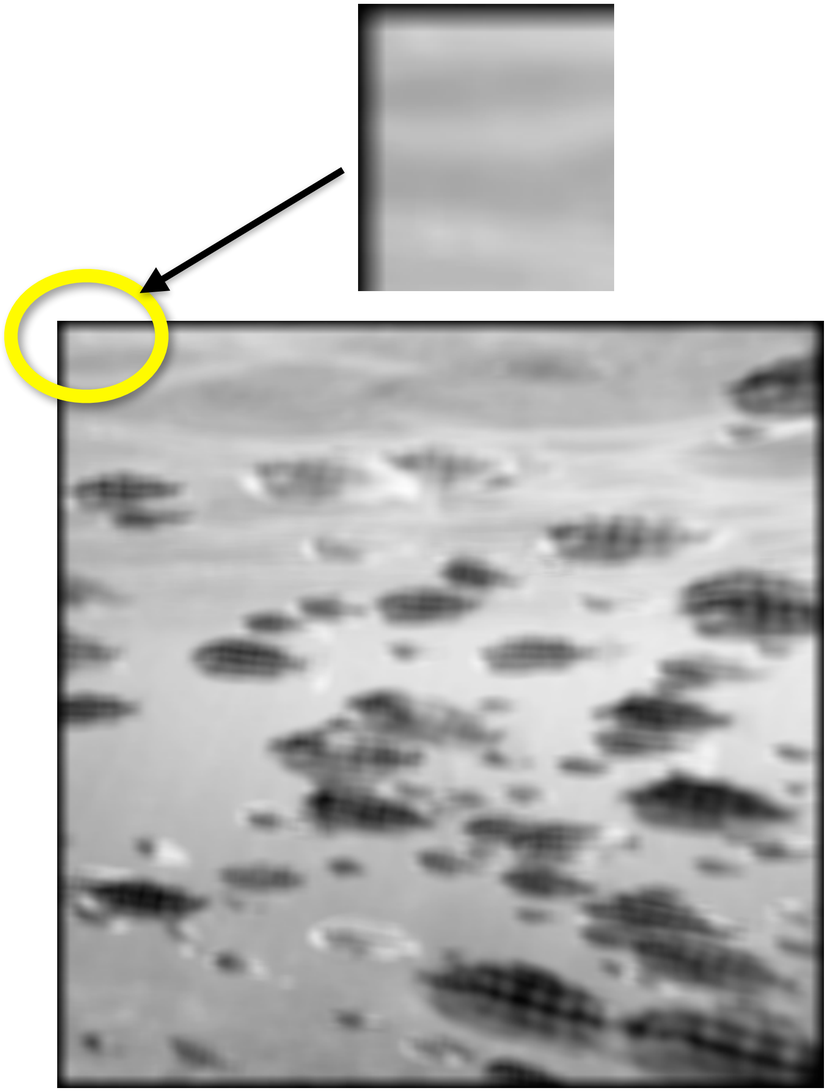
K=ones(9,9);

I2=conv2(I,K);

Example

K=ones(9,9);

I2=conv2(I,K);



‘Boring’ Details

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

‘Boring’ Details

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

‘Boring’ Details

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

‘Boring’ Details

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

‘Boring’ Details

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

‘Boring’ Details

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

‘Boring’ Details

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

‘Boring’ Details

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

‘Boring’ Details

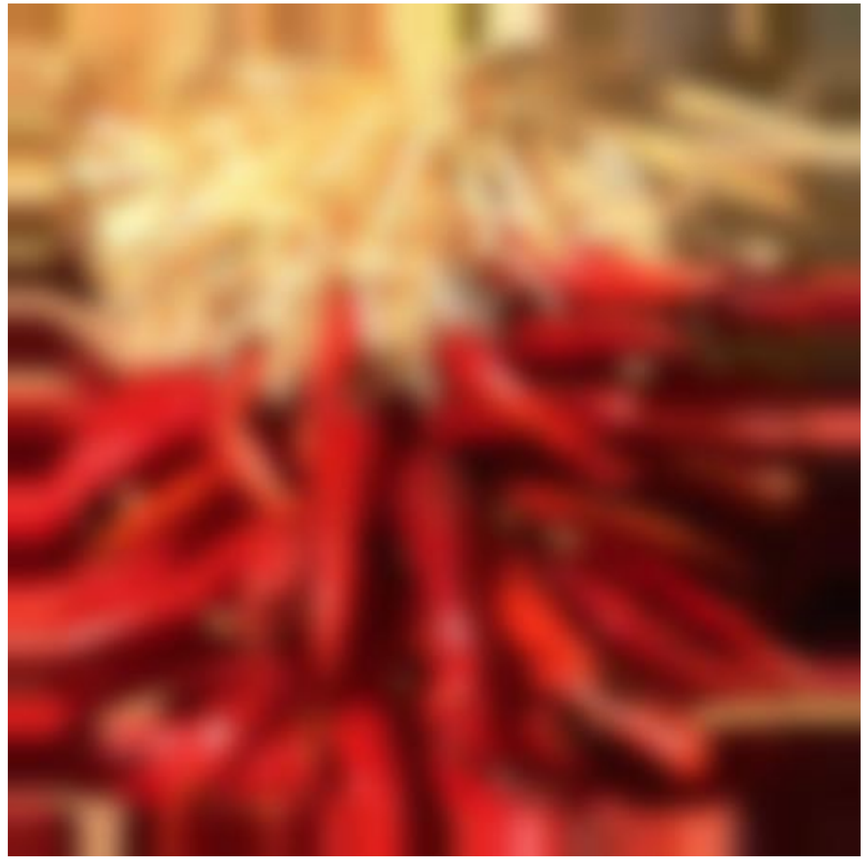
- When filter window falls off the edge of the image?
  - need to extrapolate
  - methods:
    - clip filter (black)
    - wrap around
    - copy edge
    - reflect across edge
    - vary filter near edge

‘Boring’ Details

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

‘Boring’ Details

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



‘Boring’ Details

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

‘Boring’ Details

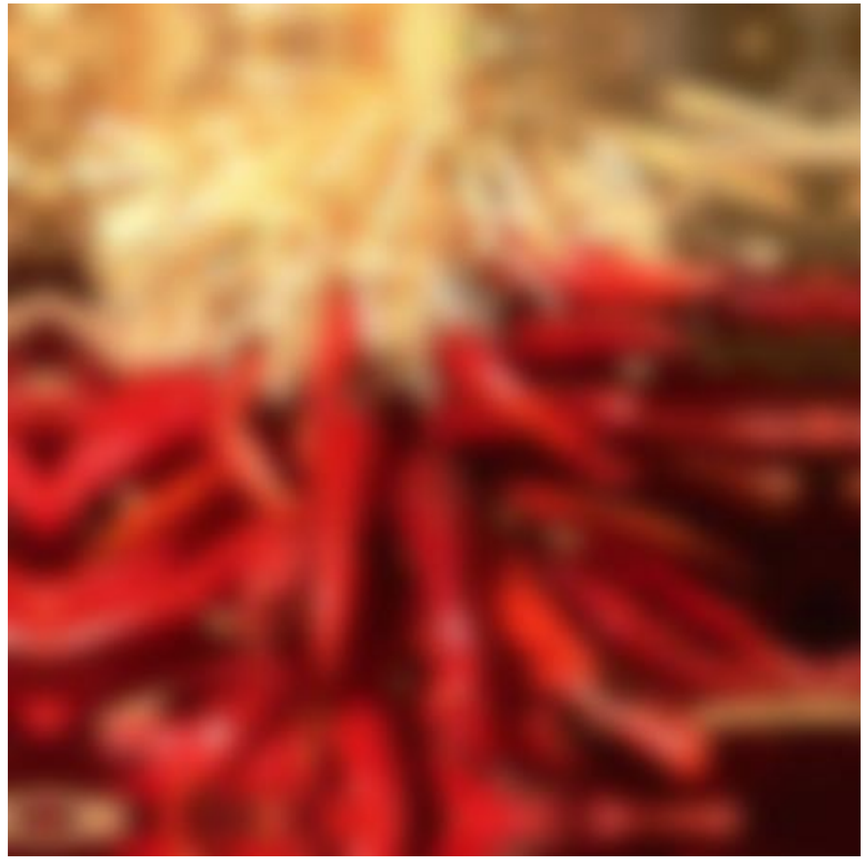
- When filter window falls off the edge of the image?
  - need to extrapolate
  - methods:
    - clip filter (black)
    - wrap around
    - copy edge
    - reflect across edge
    - vary filter near edge

‘Boring’ Details

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

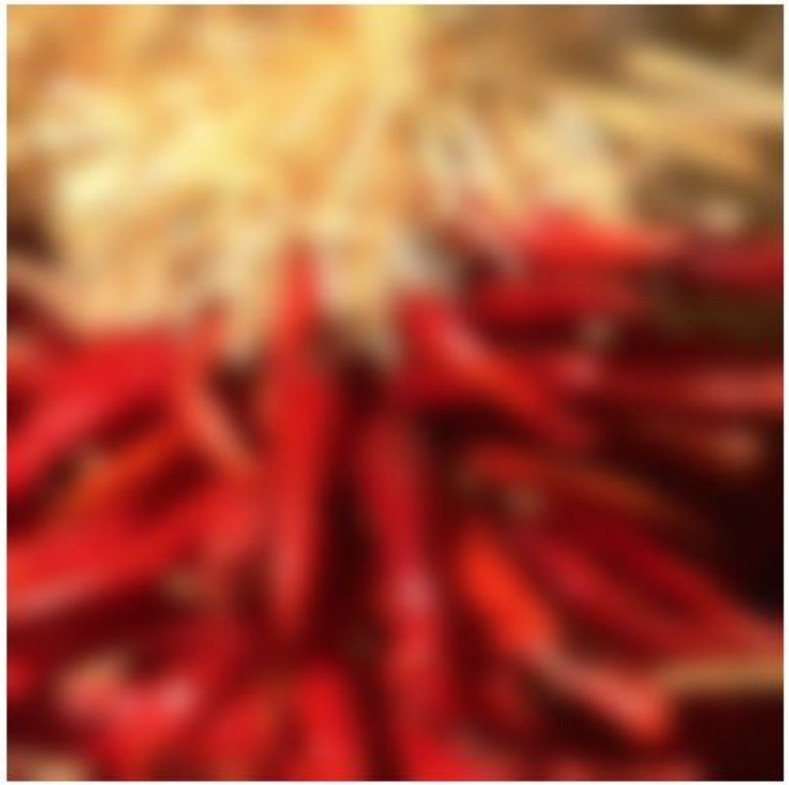
‘Boring’ Details

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



‘Boring’ Details

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



Smoothing Kernels (Low-pass filters)

Mean filter:

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

Smoothing Kernels (Low-pass filters)

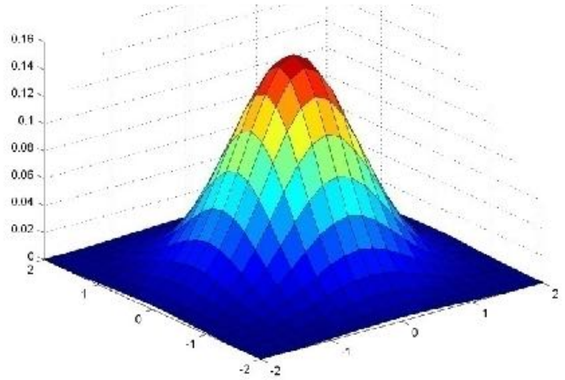
Mean filter:

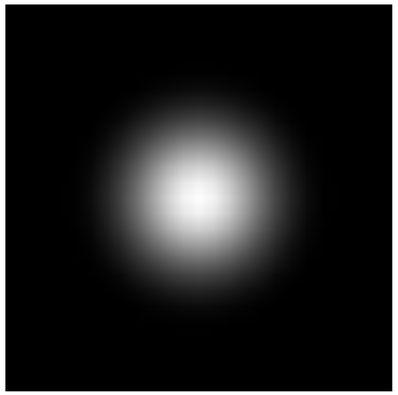
$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Weighted smoothing filters:

$$\frac{1}{10}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Gaussian Kernel





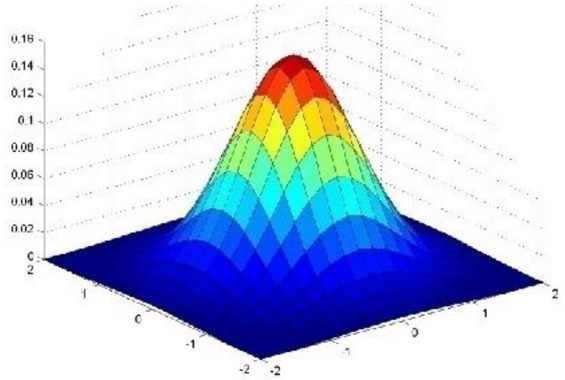
0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

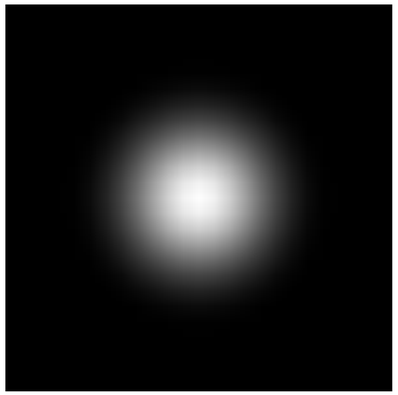
$$G_\sigma := \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

5 x 5, σ = 1

Gaussian Kernel

- **Weight contributions of neighbouring pixels related to nearness**





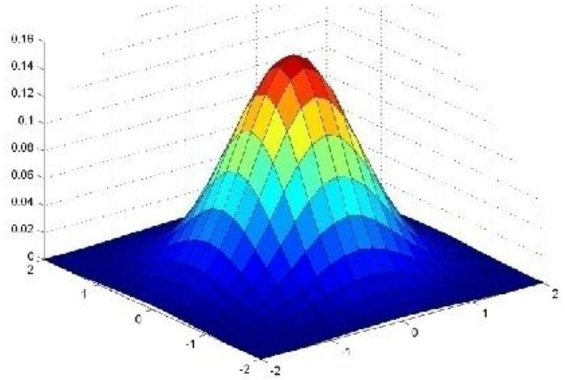
0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

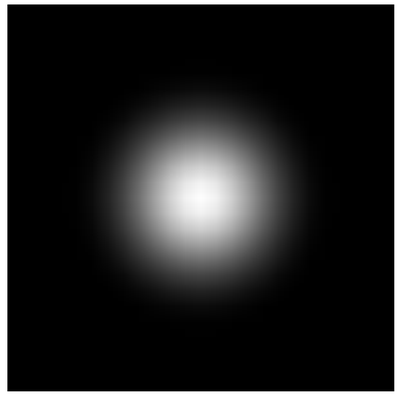
$$G_\sigma := \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

5 x 5, σ = 1

Gaussian Kernel

- **Weight contributions of neighbouring pixels related to nearness**





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

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

5 x 5, σ = 1

- **Constant factor in the front makes volume sum to 1**
  - **(subject to rounding errors, so better to normalize by sum of matrix)**

Review of Sigma (σ)

σ = sigma = standard deviation = √(variance)

$$Data, \mathbf{d} = [d_1 \quad d_2 \quad d_3 \quad \ldots \quad d_n]$$

$$Mean := \quad \mu = \frac{\sum_i \mathbf{d}_i}{n}$$

$$Variance := \sum_i (\mathbf{d}_i - \mu)^2 / n$$

$$StdDeviation := \sigma = \sqrt{Variance}$$

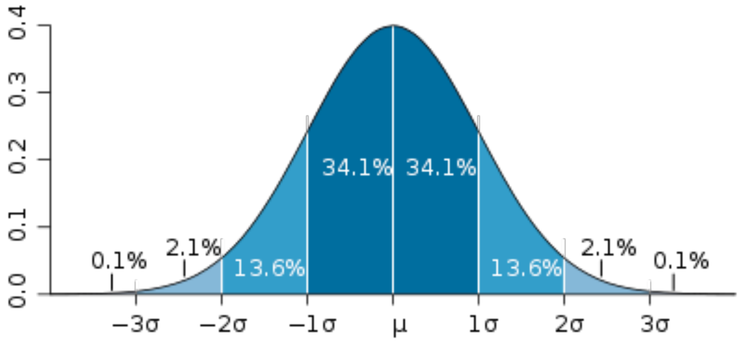


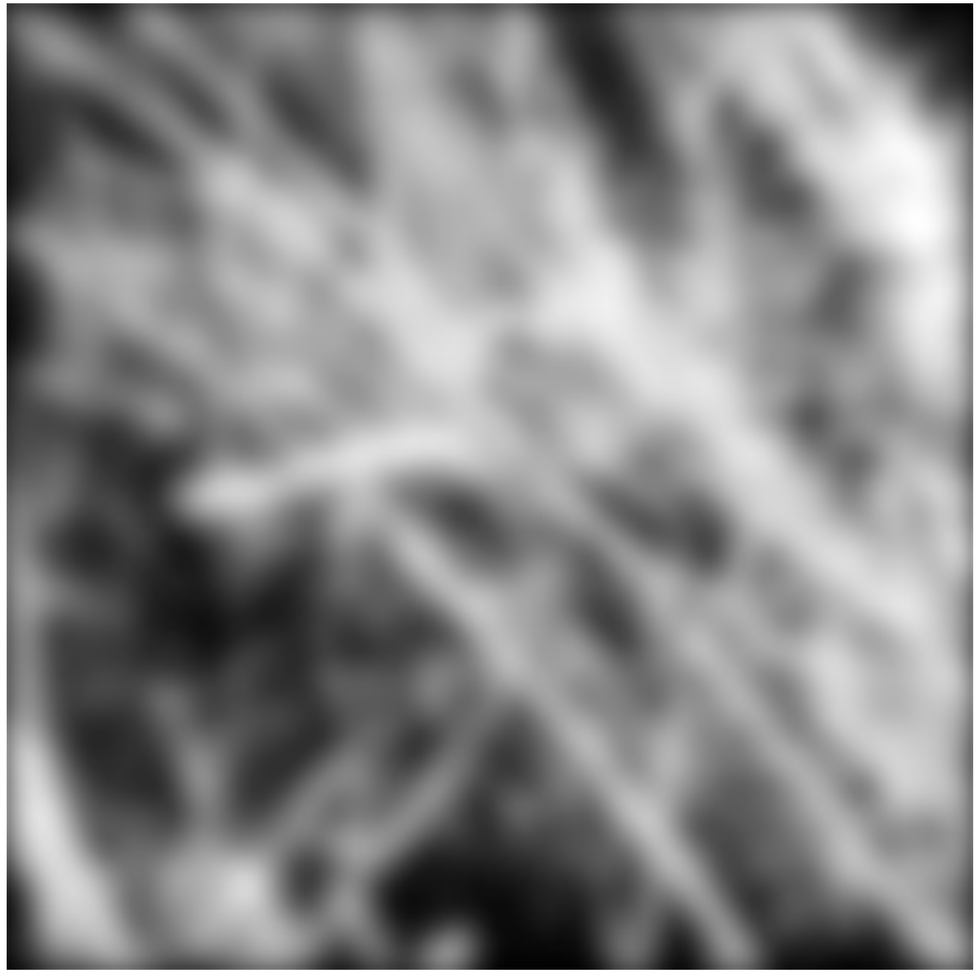
Figure by Petter Strandmark

```
mean(d), var(d), std(d)
```

Smoothing with Gaussian (Blur)

Slide credit: D.A. Forsyth

Smoothing with Gaussian (Blur)

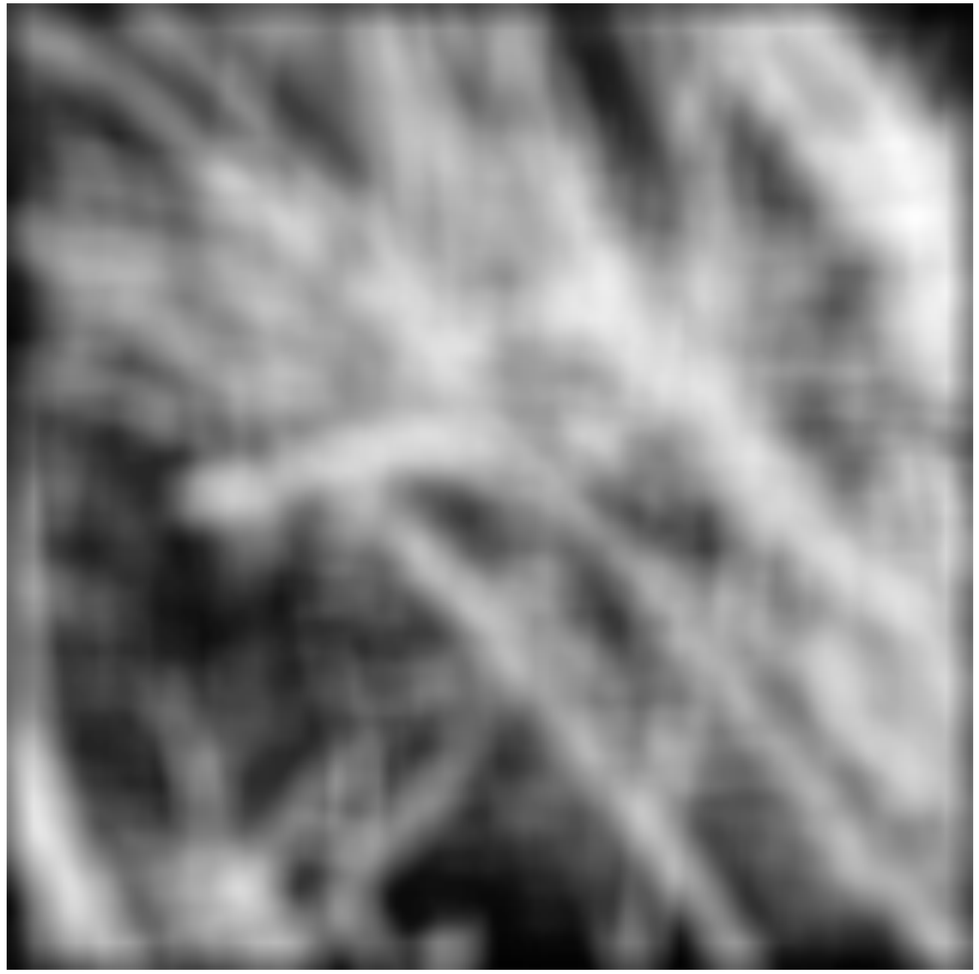


Slide credit: D.A. Forsyth

Smoothing with Box Filter

Slide credit: D.A. Forsyth

Smoothing with Box Filter



Slide credit: D.A. Forsyth

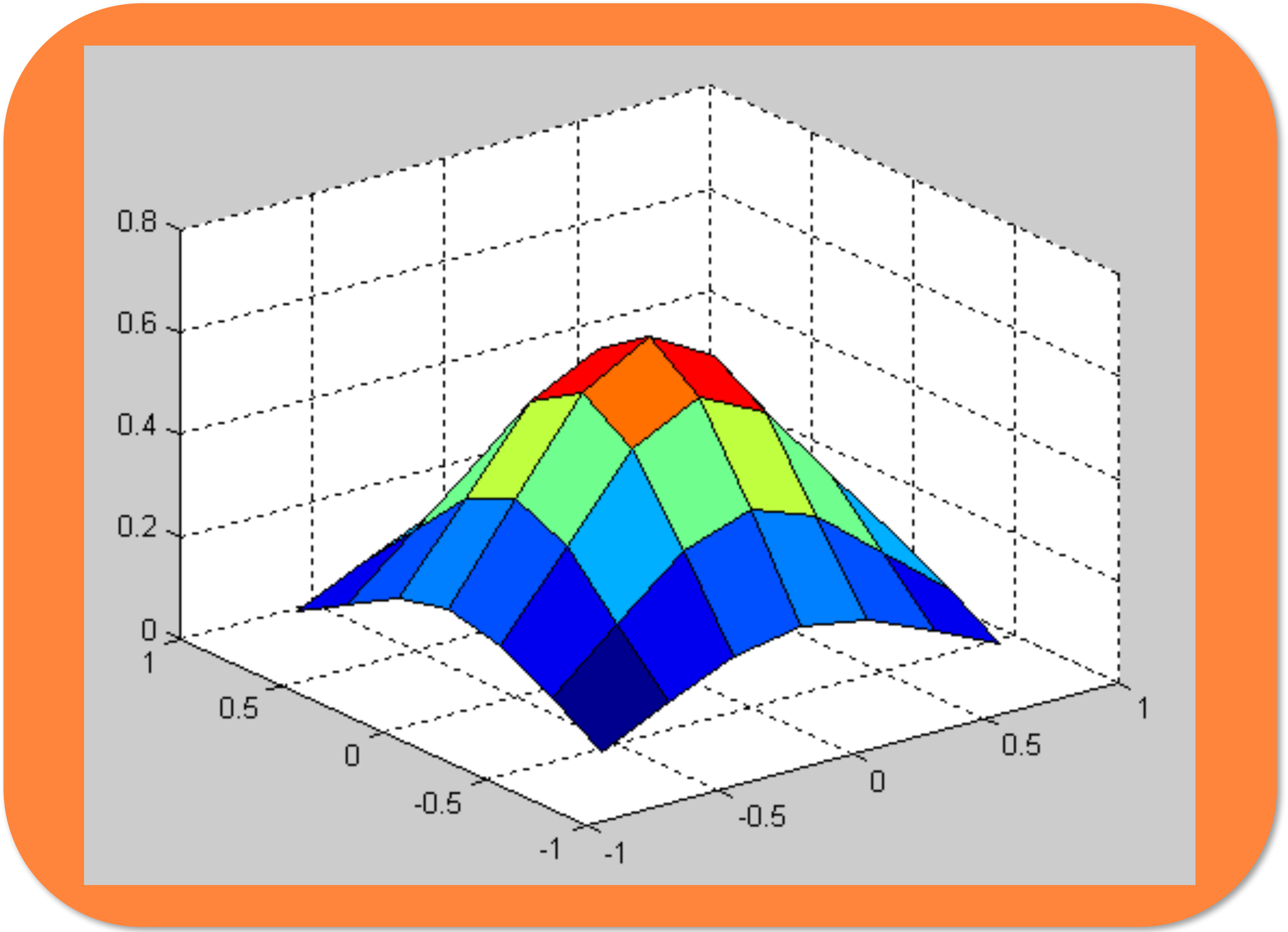
```
X =

   -0.7500   -0.5000   -0.2500         0    0.2500    0.5000    0.7500
   -0.7500   -0.5000   -0.2500         0    0.2500    0.5000    0.7500
   -0.7500   -0.5000   -0.2500         0    0.2500    0.5000    0.7500
   -0.7500   -0.5000   -0.2500         0    0.2500    0.5000    0.7500
   -0.7500   -0.5000   -0.2500         0    0.2500    0.5000    0.7500
   -0.7500   -0.5000   -0.2500         0    0.2500    0.5000    0.7500
   -0.7500   -0.5000   -0.2500         0    0.2500    0.5000    0.7500

Y =

   -0.7500   -0.7500   -0.7500   -0.7500   -0.7500   -0.7500   -0.7500
   -0.5000   -0.5000   -0.5000   -0.5000   -0.5000   -0.5000   -0.5000
   -0.2500   -0.2500   -0.2500   -0.2500   -0.2500   -0.2500   -0.2500
         0         0         0         0         0         0         0
    0.2500    0.2500    0.2500    0.2500    0.2500    0.2500    0.2500
    0.5000    0.5000    0.5000    0.5000    0.5000    0.5000    0.5000
    0.7500    0.7500    0.7500    0.7500    0.7500    0.7500    0.7500

>> Z = (1/(2*pi*sigma.^2)) * exp(-(X.^2 + Y.^2)/(2*sigma.^2));
>> surf(X,Y,Z)    %  shown with sigma = 0.5
>>
>> % Probably want Z = Z / sum(Z(:));
```


0.8
0.6
0.4
0.2
0
1
0.5
0
-0.5
-1
-1
-0.5
0
0.5
1

Separable Kernels

- **Separable filters can be written as *K*(*m*,*n*) = *f*(*m*)*g*(*n*)**
- **For a rectangular neighbourhood with size *(2M+1)*x*(2N+1)*,**

$$I'(m,n) = f \star (g \star I(N(m,n)))$$

Separable Kernels

- **Separable filters can be written as *K*(*m*,*n*) = *f*(*m*)*g*(*n*)**
- **For a rectangular neighbourhood with size *(2M+1)*x*(2N+1)*,**

$$I'(m,n) = f \star (g \star I(N(m,n)))$$

$$I'(m,n) = \sum_{i=-M}^{M} f(i) I''(m-i,n)$$

Separable Kernels

- **Separable filters can be written as *K*(*m*,*n*) = *f*(*m*)*g*(*n*)**
- **For a rectangular neighbourhood with size *(2M+1)*x*(2N+1)*,**

$$I'(m,n) = f \star (g \star I(N(m,n)))$$

$$I'(m,n) = \sum_{i=-M}^{M} f(i) I''(m-i,n)$$

$$I''(m,n) = \sum_{j=-N}^{N} g(j) I(m,n-j)$$

Gaussian Smoothing Kernels

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

Gaussian Smoothing Kernels

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

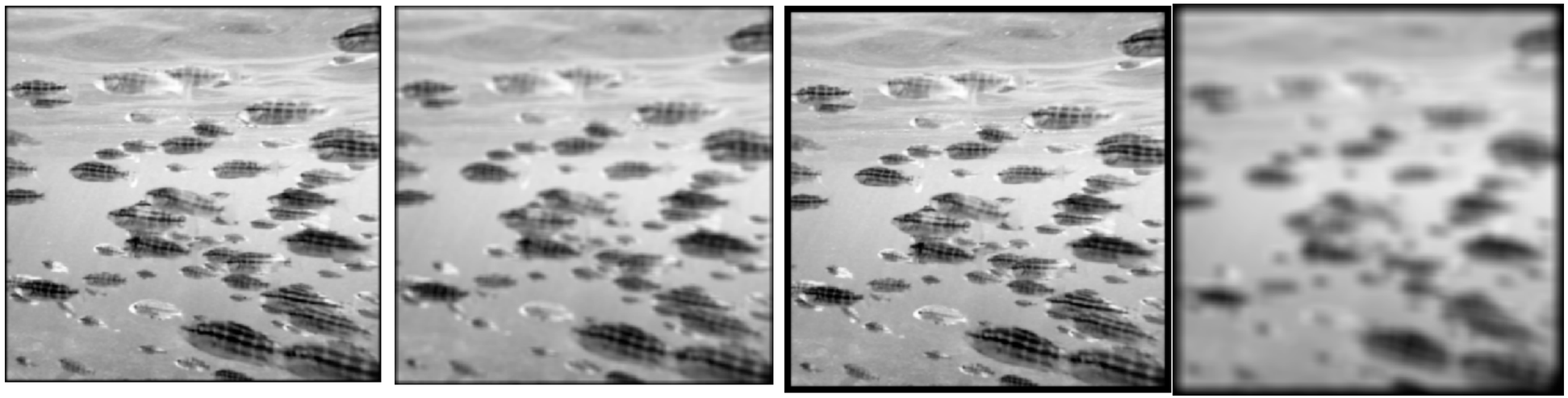
Separable!

To see how svd() can be used to obtain the separated kernels of a rank 1 kernel.

Gaussian Smoothing Kernels

- **Amount of smoothing depends on σ and window size.**
- **Width > 3σ**

7x7; σ =1. 7x7; σ =9. 19x19; σ =1. 19x19; σ =9.



Scale Space

Scale Space

- **Convolution of a Gaussian with standard deviation σ with itself, is a Gaussian with standard deviation $\sigma\sqrt{2}$.**

Scale Space

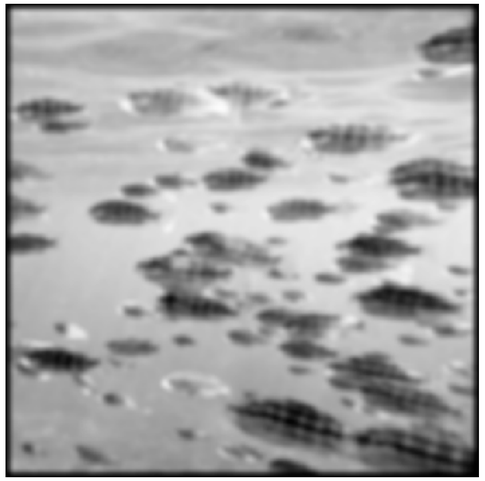
- Convolution of a Gaussian with standard deviation σ with itself, is a Gaussian with standard deviation $\sigma\sqrt{2}$.

- Repeated convolution by a Gaussian filter produces the *scale space* of an image.

Scale Space Example

11x11; σ =3.

1



2

3

4

5

6

Gaussian Smoothing Kernel Top-5

Gaussian Smoothing Kernel Top-5

1. **Rotationally symmetric**

Gaussian Smoothing Kernel Top-5

1. **Rotationally symmetric**
2. **Has a single lobe/mode**
   - **Neighbour's influence decreases monotonically**

Gaussian Smoothing Kernel Top-5

1. **Rotationally symmetric**
2. **Has a single lobe/mode**
   - **Neighbour's influence decreases monotonically**
3. **Still one lobe in frequency domain**
   - **No corruption from high frequencies**

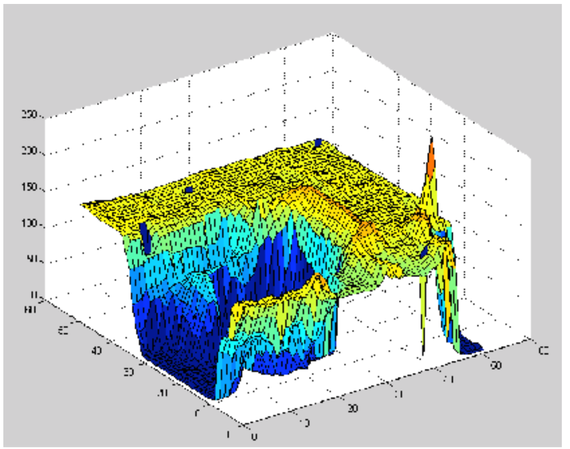
Gaussian Smoothing Kernel Top-5

1. **Rotationally symmetric**
2. **Has a single lobe/mode**
   - **Neighbour's influence decreases monotonically**
3. **Still one lobe in frequency domain**
   - **No corruption from high frequencies**
4. **Simple relationship to σ**

Gaussian Smoothing Kernel Top-5

1. **Rotationally symmetric**
2. **Has a single lobe/mode**
   - **Neighbour's influence decreases monotonically**
3. **Still one lobe in frequency domain**
   - **No corruption from high frequencies**
4. **Simple relationship to σ**
5. **Easy to implement efficiently**

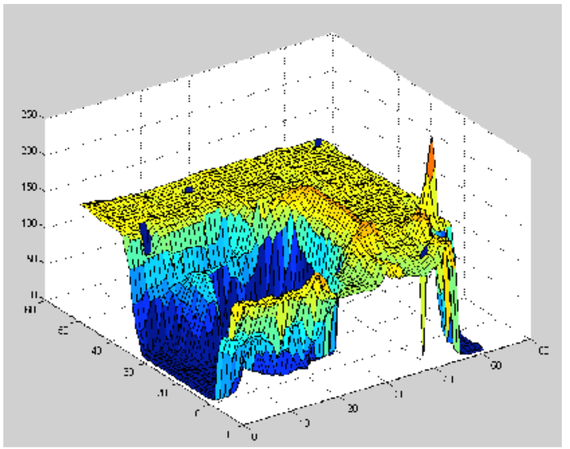
Smoothing vs. Sharpening



z = *I*'s
intensity

Smoothing vs. Sharpening

- Smoothing ↔ Integration, i.e.,
  - Summing
  - Sand-papering the height field's slope

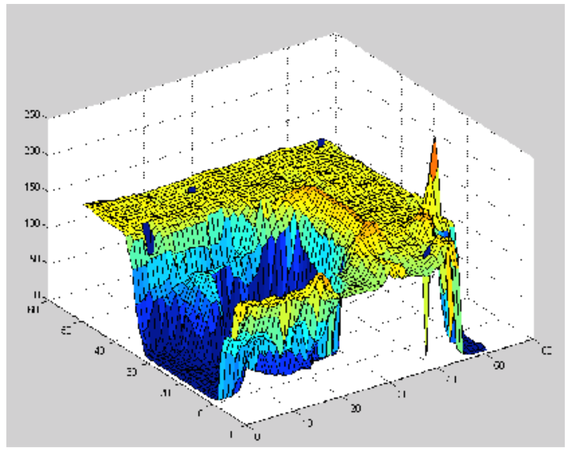


z = *I*'s intensity

Smoothing vs. Sharpening

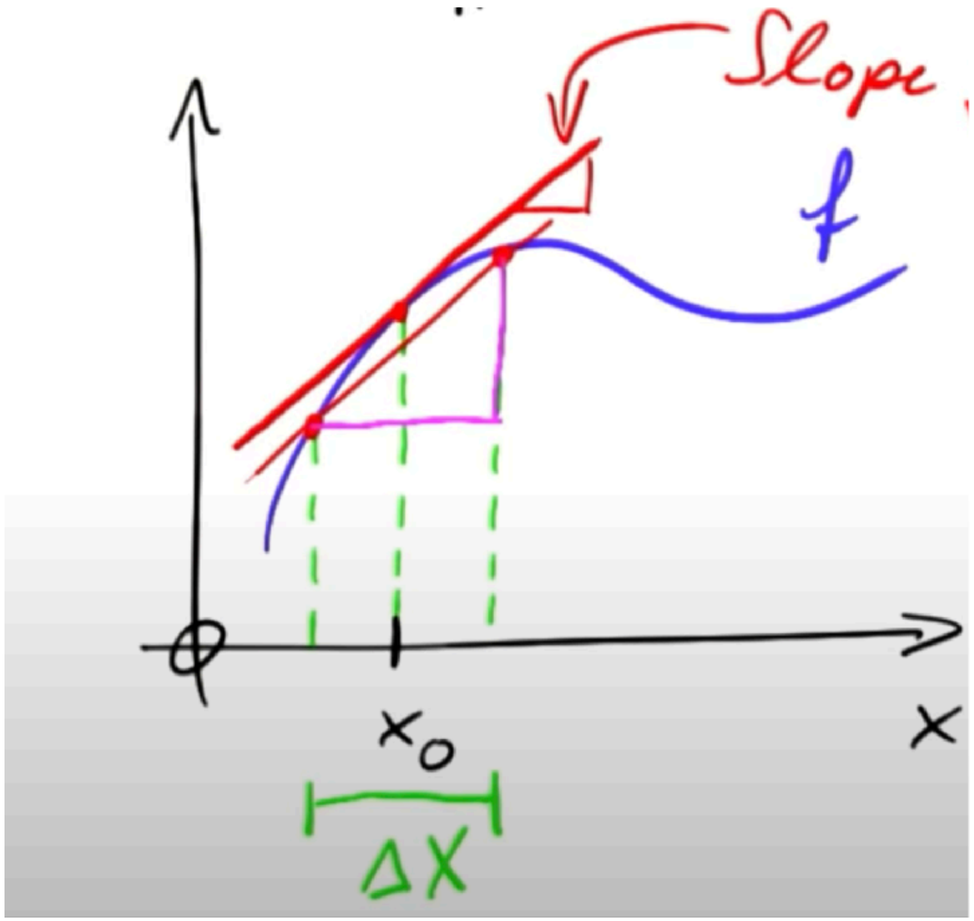
- Smoothing ↔ Integration, i.e.,
  - Summing
  - Sand-papering the height field's slope

- Sharpening ↔ Differentiation, i.e.,
  - Subtracting
  - Accentuating the height field's slope



z = *I*'s intensity

1st and 2nd Derivative



Differentiation and Convolution

Differentiation and Convolution

- **Recall, for 2D function, f(x,y):**

$$\frac{\partial f}{\partial x} = \lim_{\epsilon \to 0} \left(\frac{f(x+\epsilon, y) - f(x, y)}{\epsilon} \right)$$

Differentiation and Convolution

- **Recall, for 2D function, f(x,y):**

$$\frac{\partial f}{\partial x} = \lim_{\epsilon \to 0} \left(\frac{f(x+\epsilon, y) - f(x, y)}{\epsilon} \right)$$

- **This is linear and shift invariant, so the result of a convolution**

-1	**1**

Differentiation and Convolution

- **Recall, for 2D function, f(x,y):**

$$\frac{\partial f}{\partial x} = \lim_{\epsilon \to 0} \left(\frac{f(x+\epsilon, y) - f(x, y)}{\epsilon} \right)$$

- **This is linear and shift invariant, so the result of a convolution**

-1	1

- We could approximate this as (which is a convolution)

$$\frac{\partial f}{\partial x} \approx \frac{f(x_{n+1}, y) - f(x_n, y)}{\Delta x}$$

1st and 2nd Derivative

The derivative of a function f at a point x is defined by the limit

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Approximation of the derivative when h is small

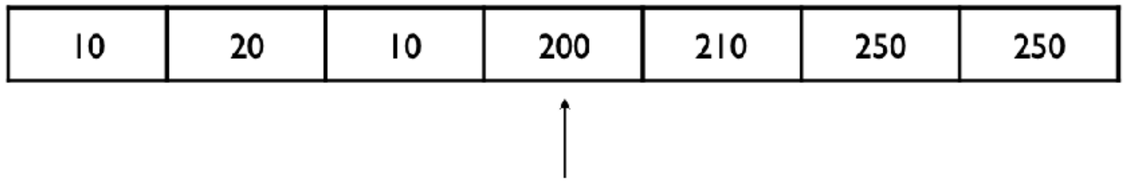
This definition is based on the 'forward difference' but ...

1st and 2nd Derivative

it turns out that using the 'central difference' is more accurate

$$f'(x) = \lim_{h \to 0} \frac{f(x + 0.5h) - f(x - 0.5h)}{h}$$

How do we compute the derivative of a discrete signal?

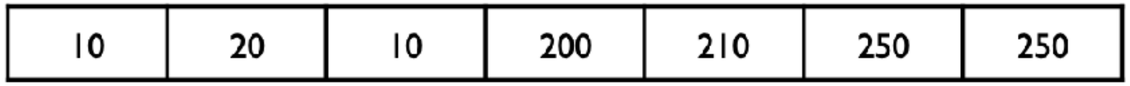


1st and 2nd Derivative

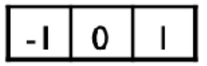
it turns out that using the 'central difference' is more accurate

$$f'(x) = \lim_{h \to 0} \frac{f(x + 0.5h) - f(x - 0.5h)}{h}$$

How do we compute the derivative of a discrete signal?

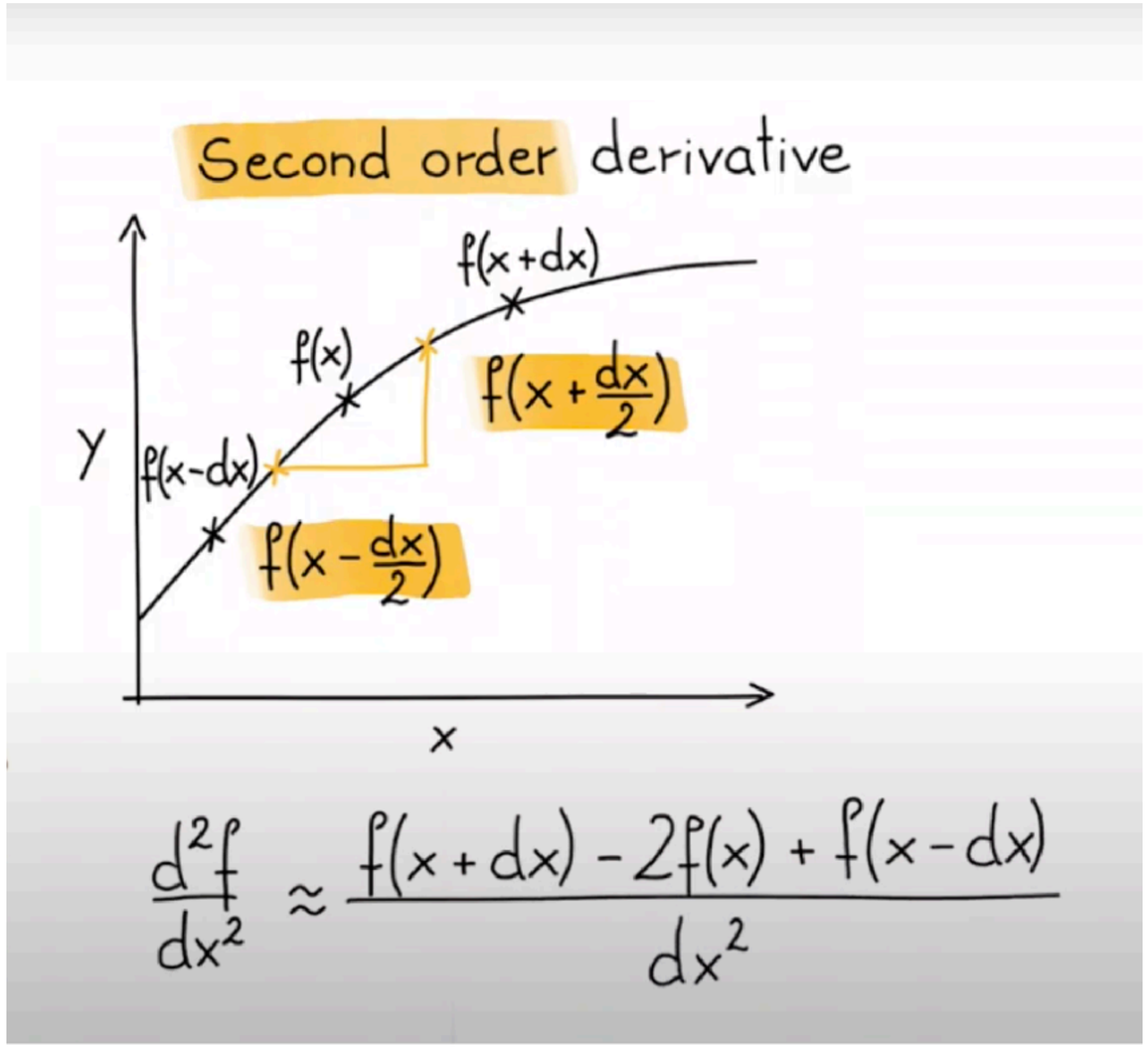


$$f'(x) = \frac{f(x+1) - f(x-1)}{2} = \frac{210 - 10}{2} = 100$$



1D derivative filter

2nd Derivative



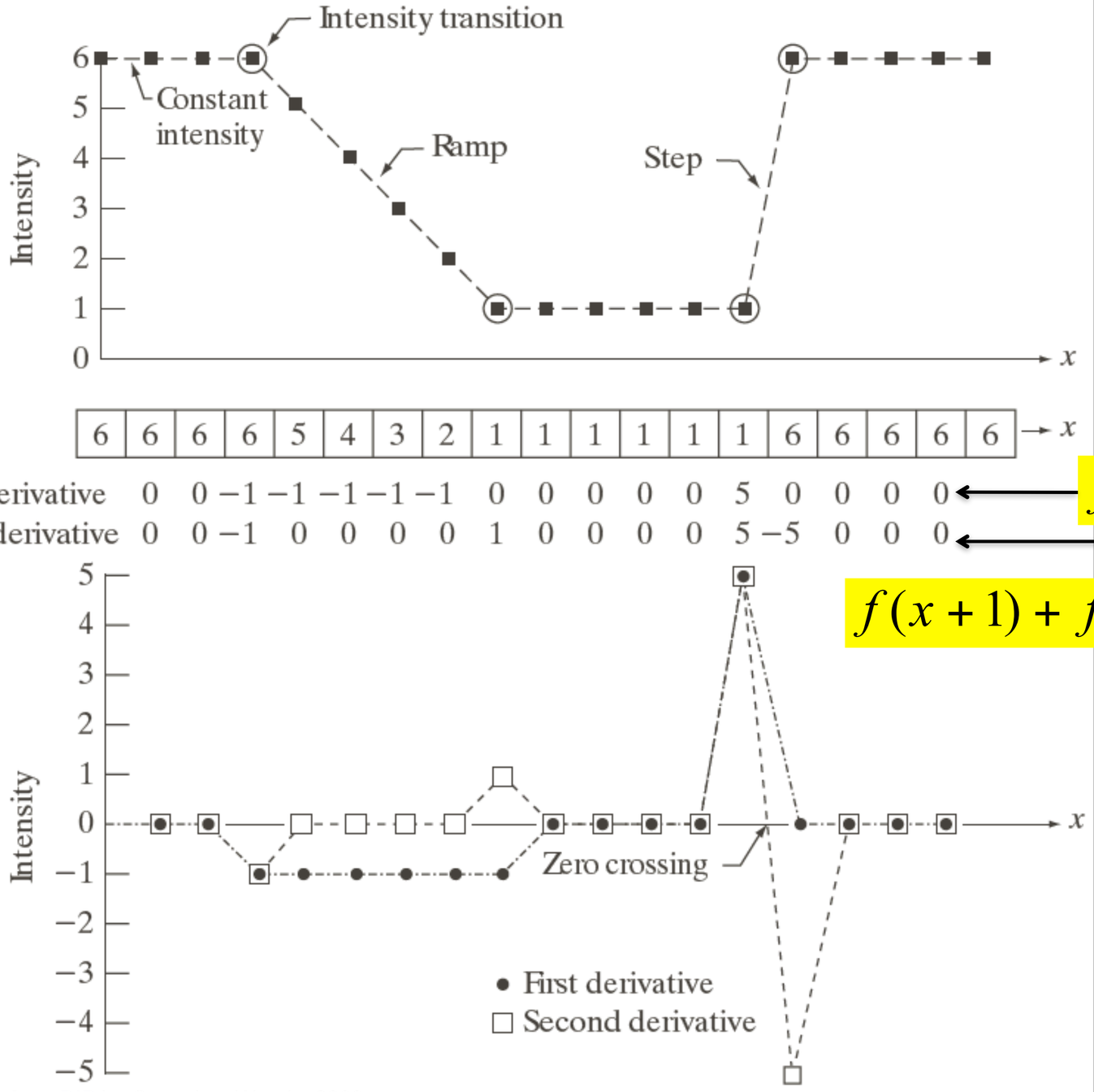
1st and 2nd Derivative

(Look ahead)

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

(Look ahead and look back)

$$\frac{\partial^2 f}{\partial x} = f(x+1) + f(x-1) - 2f(x)$$



$$f(x+1) - f(x)$$

$$f(x+1) + f(x-1) - 2f(x)$$

Figure from DIP by Gonzalez+Woods, 2008

Vertical Gradients from Finite Differences

Slide credit: D.A. Forsyth

Differential Filters

Prewitt operator:

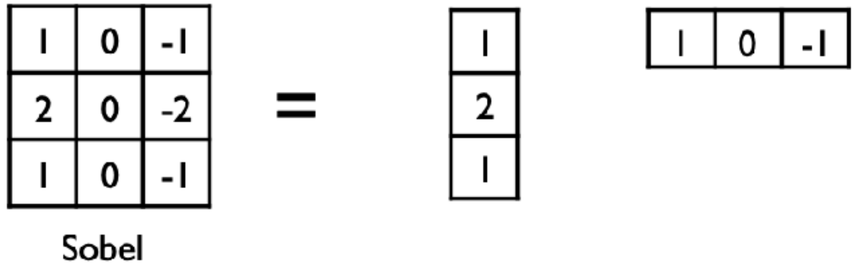
$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Sobel operator:

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

Differential Filters

Decomposing the Sobel filter



Sobel

1st Derivative: From 1D to 2D

$$\frac{\partial f}{\partial x} = f(x+1) - f(x) \qquad \text{in 1D} \quad \text{2D:} \qquad \nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} g_x \\ g_y \end{bmatrix}$$

- Gradient is anisotropic
- $mag(\nabla f) = \sqrt{g_x^2 + g_y^2}$ is isotropic
- Orientation of gradient: $\alpha(x, y) = \tan^{-1}\left[\frac{g_y}{g_x}\right]$

Imagine g_x and g_y as vectors that form two sides of a right triangle.
The result of vector-addition points in the direction of the gradient!

2nd Derivative: From 1D to 2D

$$\frac{\partial^2 f}{\partial x} = f(x+1) + f(x-1) - 2f(x) \quad \text{in 1D} \qquad \text{2D:} \quad \nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y)$$

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y)$$

0	1	0
1	-4	1
0	1	0

1	1	1
1	-8	1
1	1	1

Incorporating diagonals, isotropic in increments of 45°

-1	-1	-1
-1	8	-1
-1	-1	-1

Laplacian: use this (negative) version of kernel for convenience

Image Sharpening

Image Sharpening

- **Also known as *enhancement***

Image Sharpening

- Also known as *enhancement*

- Increases the high frequency components to enhance edges

Image Sharpening

- Also known as *enhancement*
- Increases the high frequency components to enhance edges
- $I' = I + \alpha(k * I)$, where k is a high-pass filter kernel and α is a scalar in [0,1]

Image Sharpening: Example 1

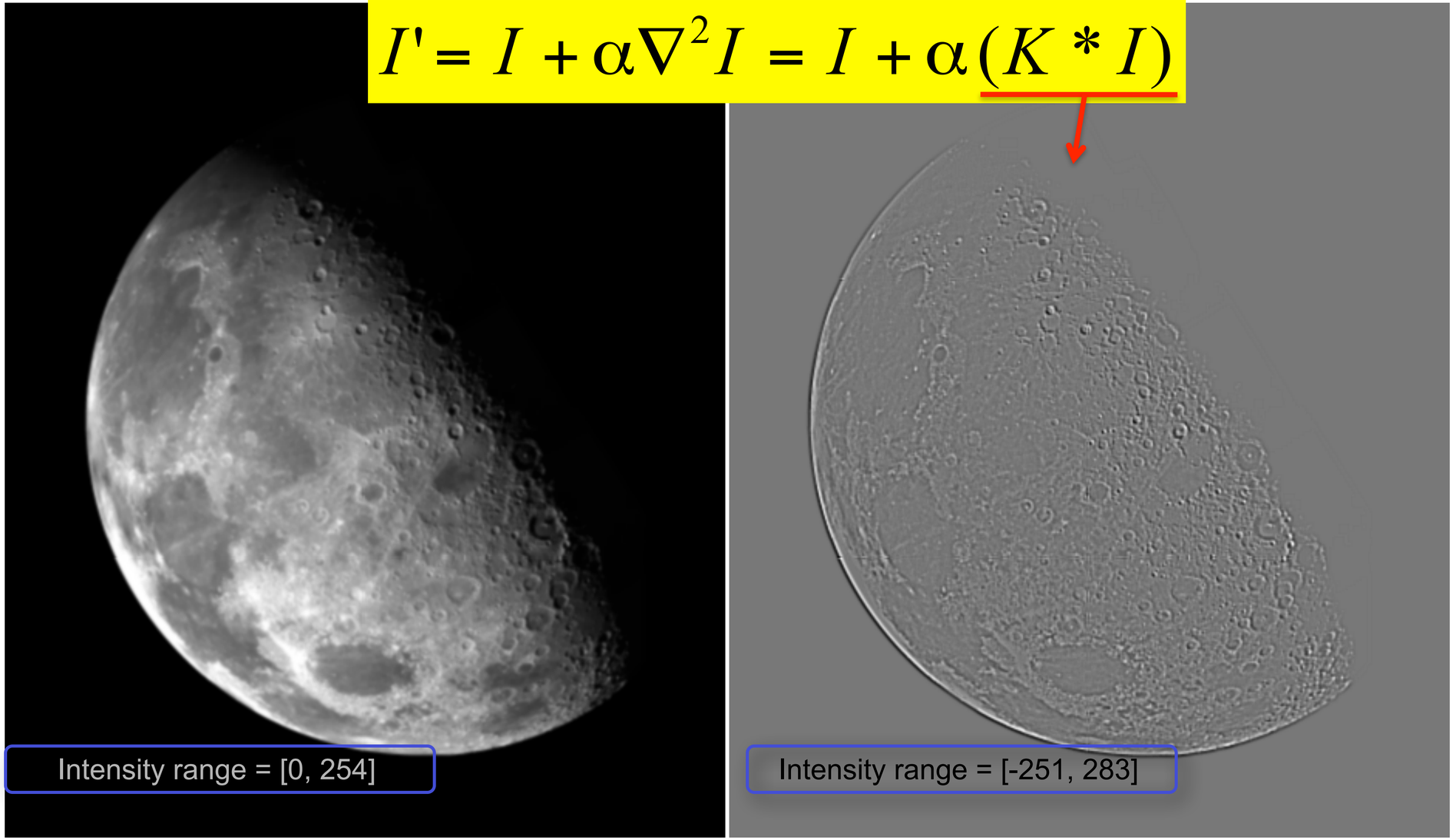
$$I' = I + \alpha \nabla^2 I = I + \alpha (K * I)$$

-1	-1	-1
-1	8	-1
-1	-1	-1

Figure from NASA, obtained on DIP

Image Sharpening: Example 1

$$I' = I + \alpha \nabla^2 I = I + \alpha (K * I)$$



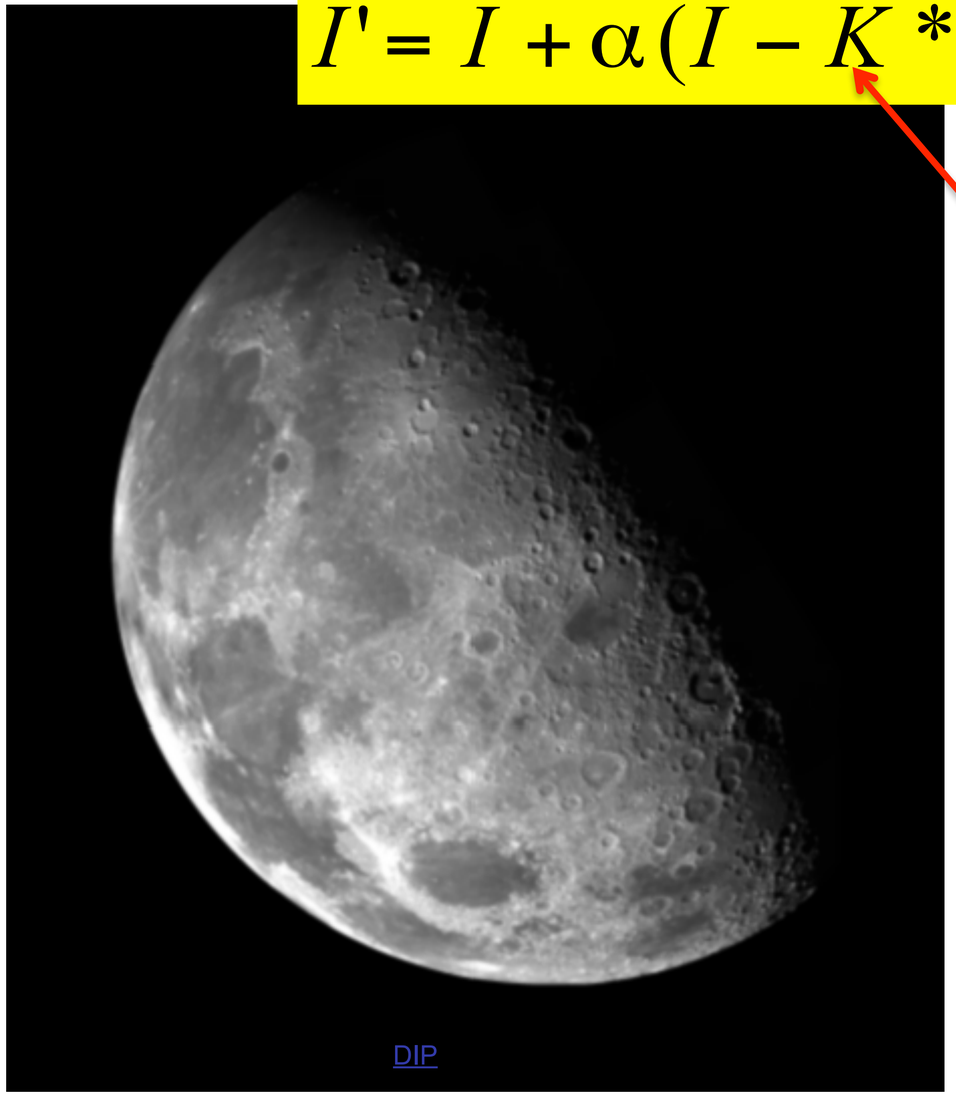
I $K*I$

Image Sharpening: Example 1

Image Sharpening: Example 1

Image Sharpening: Example 2

$$I' = I + \alpha(I - K * I) \Leftarrow \text{"Unsharp Mask"}$$



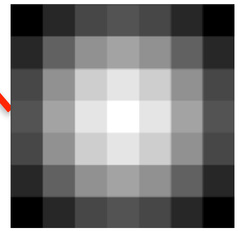
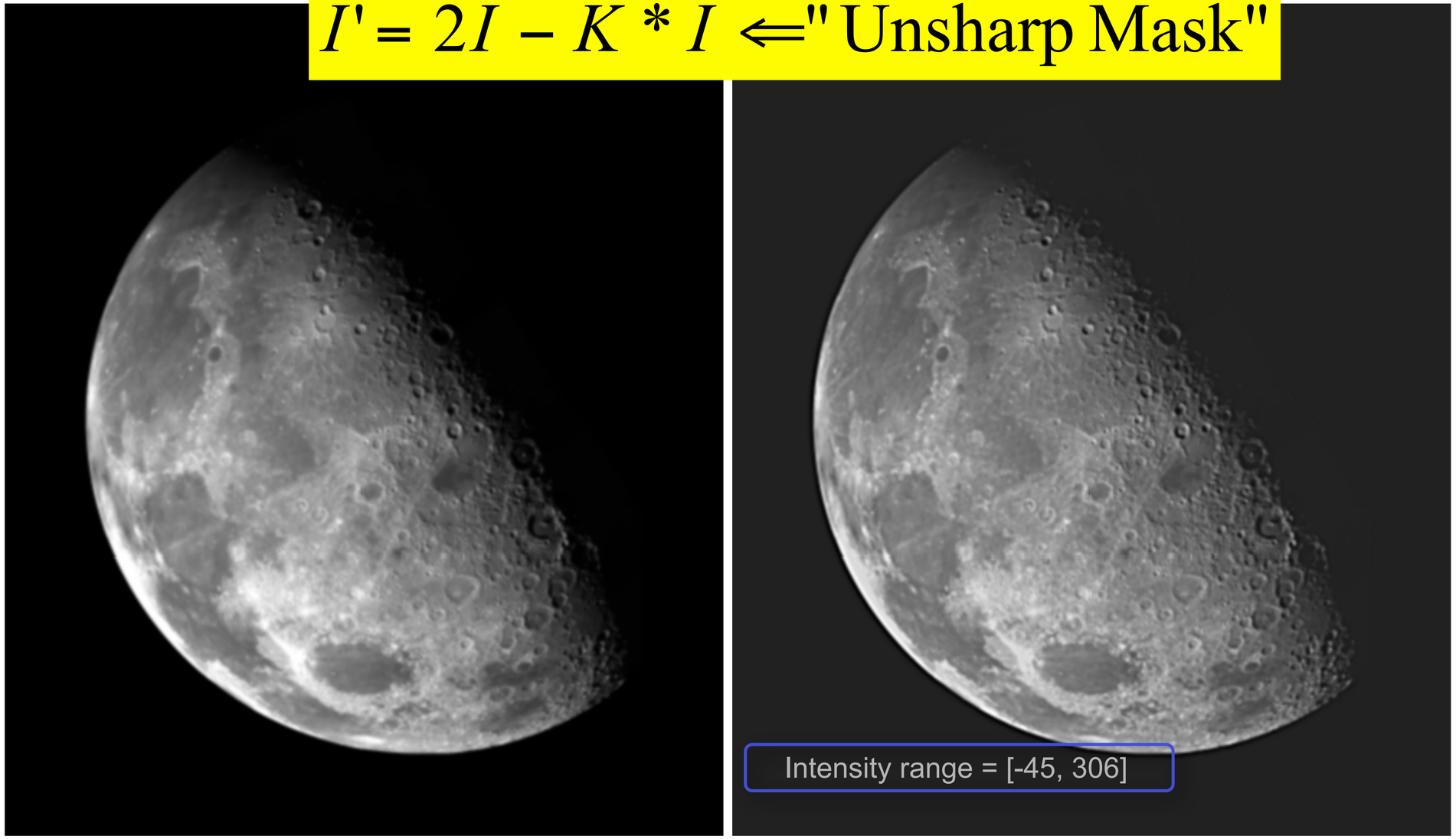


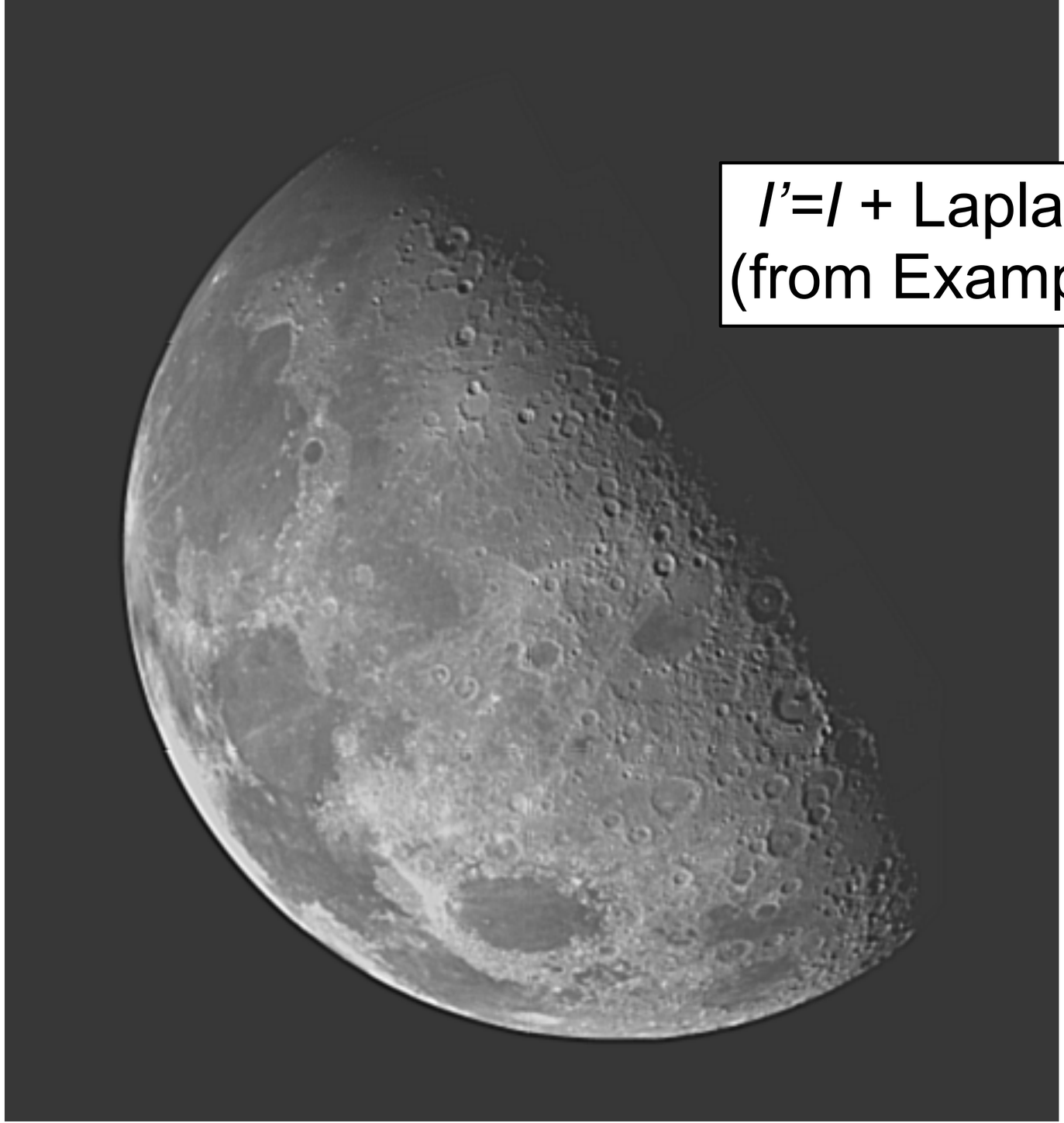
Image Sharpening: Example 2

$$I' = 2I - K * I \Leftarrow \text{"Unsharp Mask"}$$



I

I’= Unsharp Mask
(from Example 2)



I’=I + Laplacian
(from Example 1)

Filters are Templates

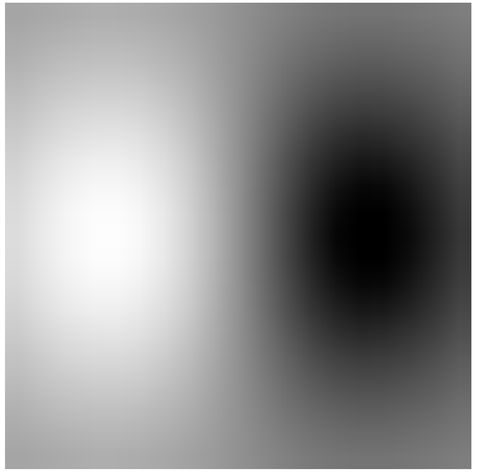
Slide credit: D.A. Forsyth

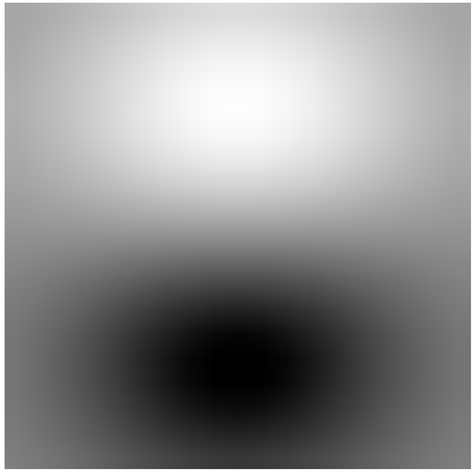
Filters are Templates

- **Filter at some point can be seen as taking a dot-product between the image and some vector**

Filters are Templates

- Filter at some point can be seen as taking a dot-product between the image and some vector
- Filtering the image is a set of dot products
  - filters look like the effects they are intended to find
  - filters find effects they look like





Slide credit: D.A. Forsyth

Scaled Representations

Scaled Representations

- Find *correlations/convolutions* at all scales
  - e.g., when finding hands or faces, we don't know what size they will be in a particular image
  - Template size is constant, but image size changes

Slide credit: D.A. Forsyth

Scaled Representations

- Find *correlations/convolutions* at all scales
  - e.g., when finding hands or faces, we don't know what size they will be in a particular image
  - Template size is constant, but image size changes

- Efficient search for *correspondence*
  - look at coarse scales, then refine with finer scales
  - much less cost, but may miss best match

Slide credit: D.A. Forsyth

Scaled Representations

- Find *correlations/convolutions* at all scales
  - e.g., when finding hands or faces, we don't know what size they will be in a particular image
  - Template size is constant, but image size changes

- Efficient search for *correspondence*
  - look at coarse scales, then refine with finer scales
  - much less cost, but may miss best match

- Examining all *levels of detail*
  - Find edges with different amounts of blur
  - Find textures with different spatial frequencies *(levels of detail)*

Slide credit: D.A. Forsyth

Fourier Domain

Fourier Domain

Fourier Domain

- The Fourier Domain represents the image by the sum of weighted frequency components

Fourier Domain

- The Fourier Domain represents the image by the sum of weighted frequency components

- Many image processing operations are simpler in the Fourier Domain

Fourier Domain

- The Fourier Domain represents the image by the sum of weighted frequency components

- Many image processing operations are simpler in the Fourier Domain

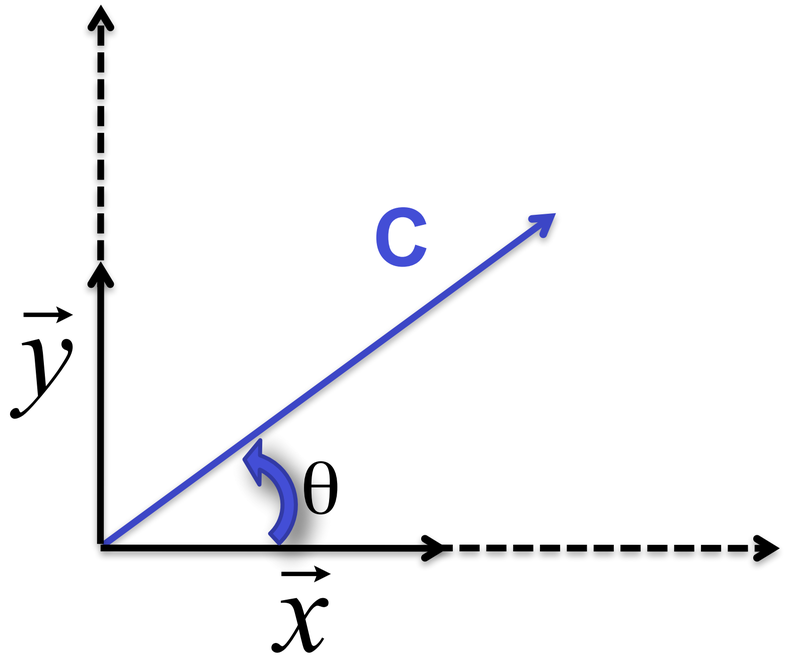
- Convolutions are simpler in the Fourier Domain

Jean Baptiste Joseph Fourier (1768-1830)

Fourier Series:

Any periodic function can be rewritten as a weighted sum of sines and cosines of different frequencies.

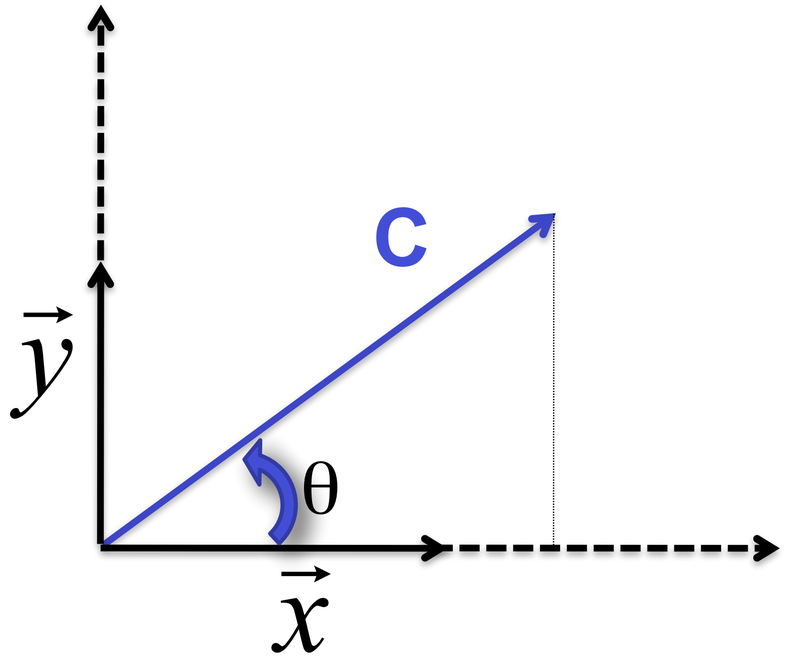
Geometry Review



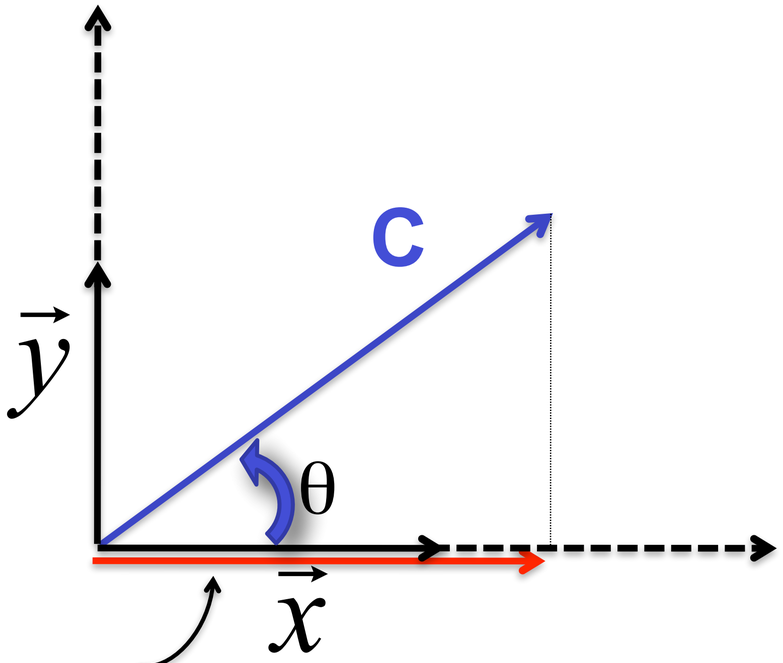
(unit-vector on x-axis)

Matlab:
θ = atan2(y, x)

Geometry Review

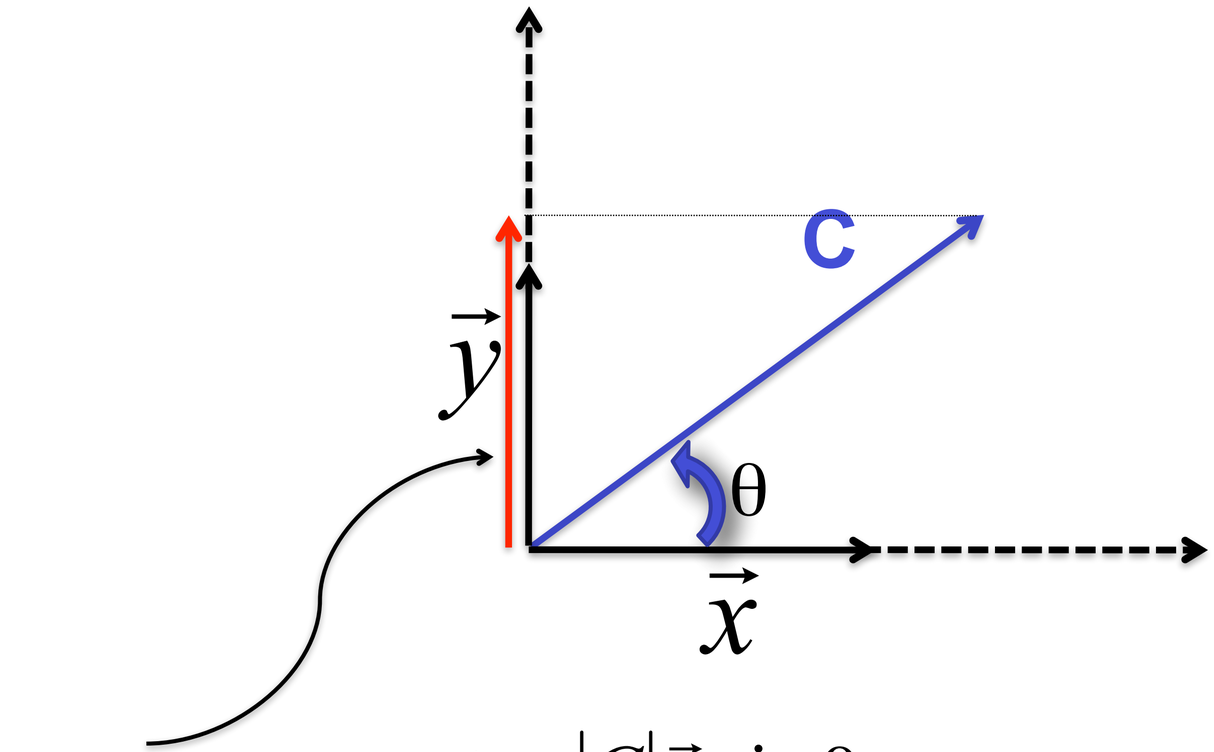


Geometry Review



$$\text{x} - \text{component} = |C|\vec{x}\cos\theta$$

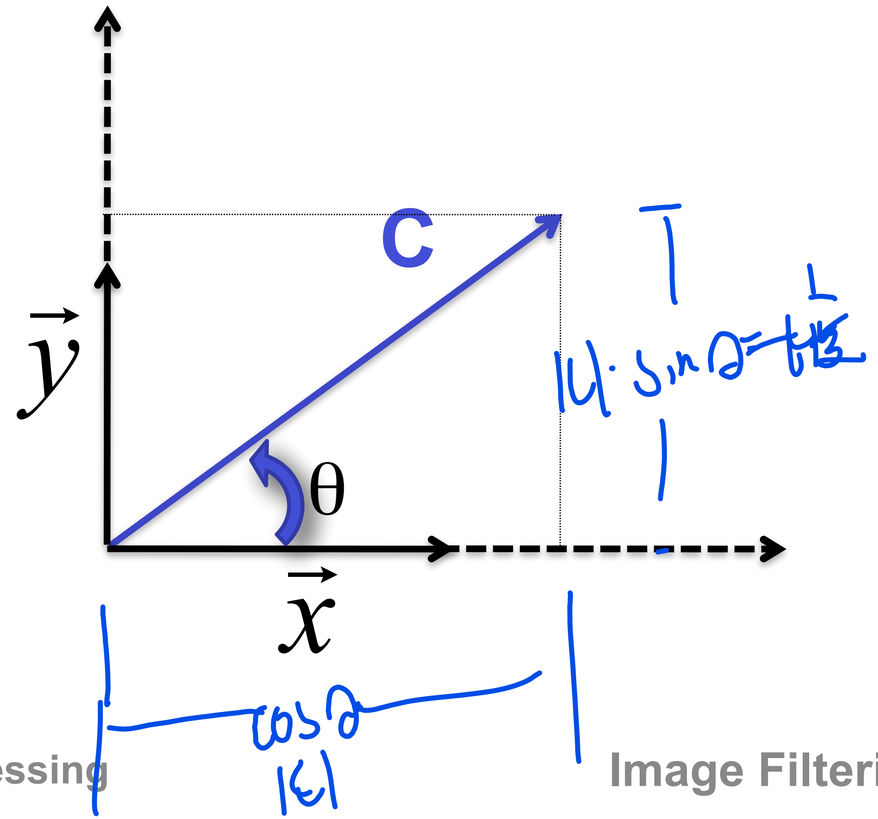
Geometry Review



$$\text{y} - \text{component} = |C|\vec{y}\sin\theta$$

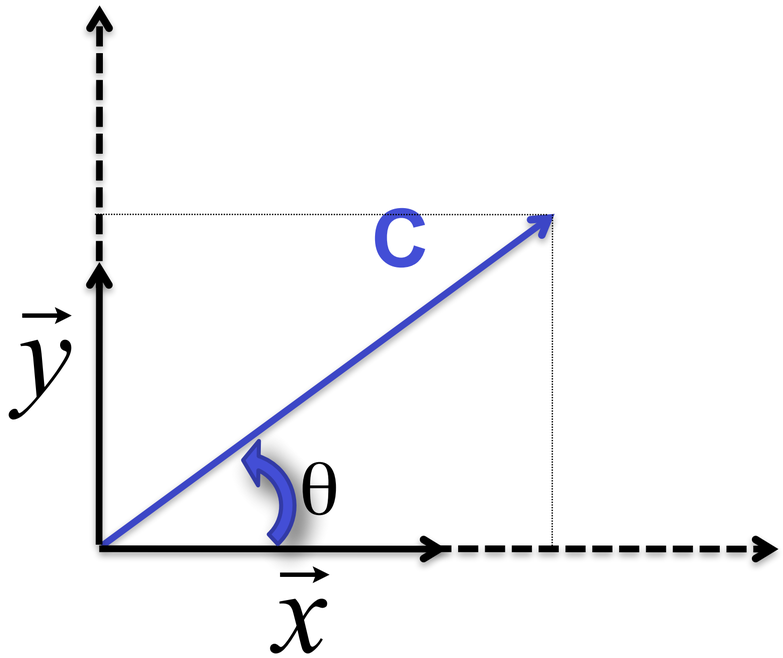
Geometry Review

$$C = |C|\left(\vec{x}\cos\theta + \vec{y}\sin\theta\right)$$



Geometry Review

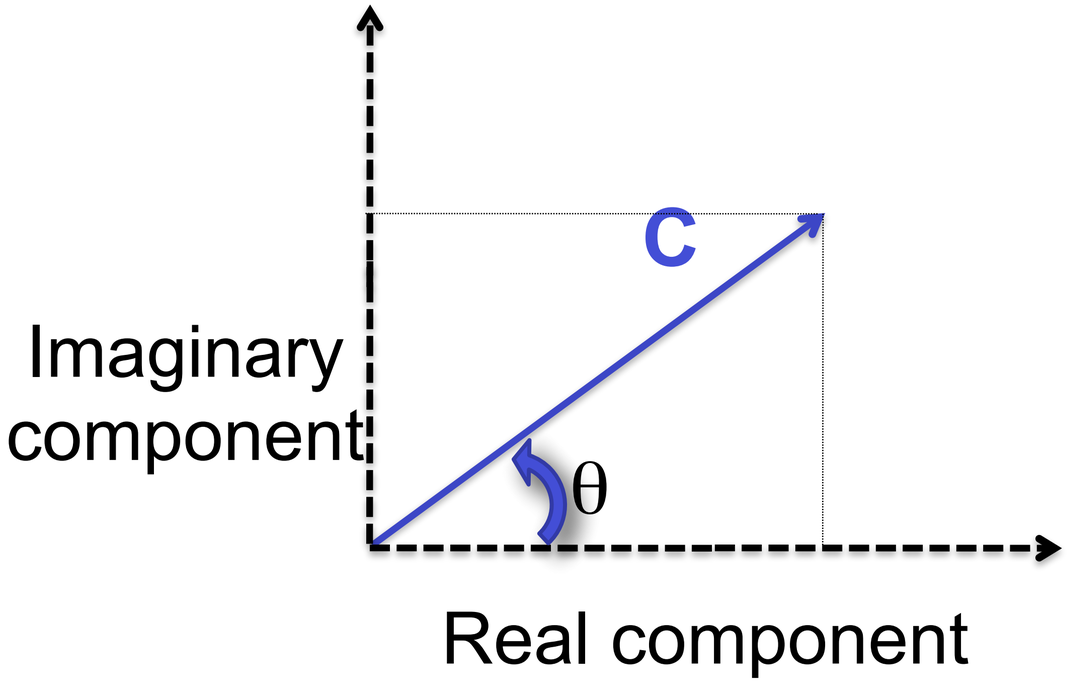
$$C = |C|\left(\vec{x}\cos\theta + \vec{y}\sin\theta\right)$$



Now, use these polar coordinates to represent imaginary numbers!

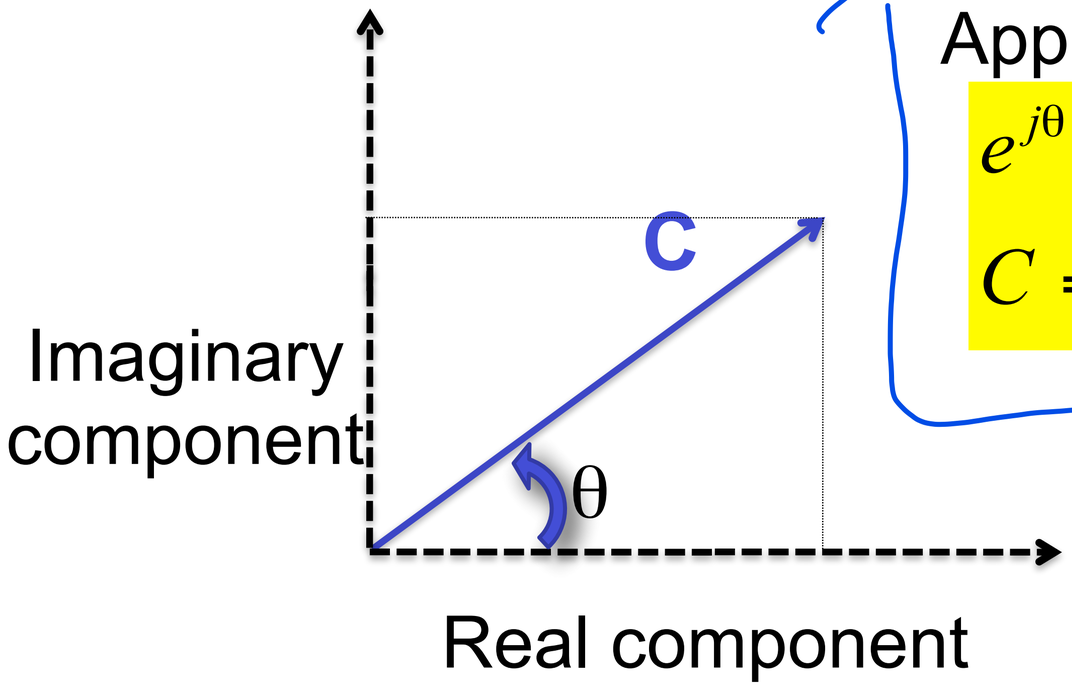
Geometry Review

$$C = |C|(\cos\theta + j\sin\theta)$$



Geometry Review

$$C = |C|(\cos\theta + j\sin\theta)$$



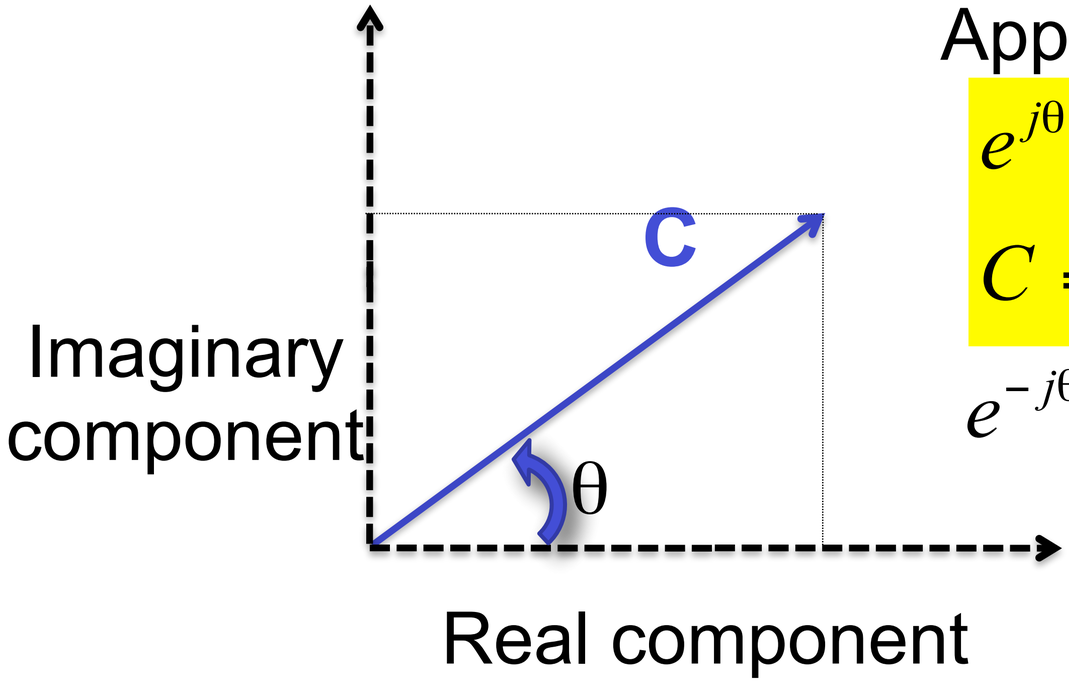
Apply Euler's formula*

$$e^{j\theta} = \cos\theta + j\sin\theta$$

$$C = |C|e^{j\theta}$$

Geometry Review

$$C = |C|(\cos\theta + j\sin\theta)$$



Apply Euler's formula*

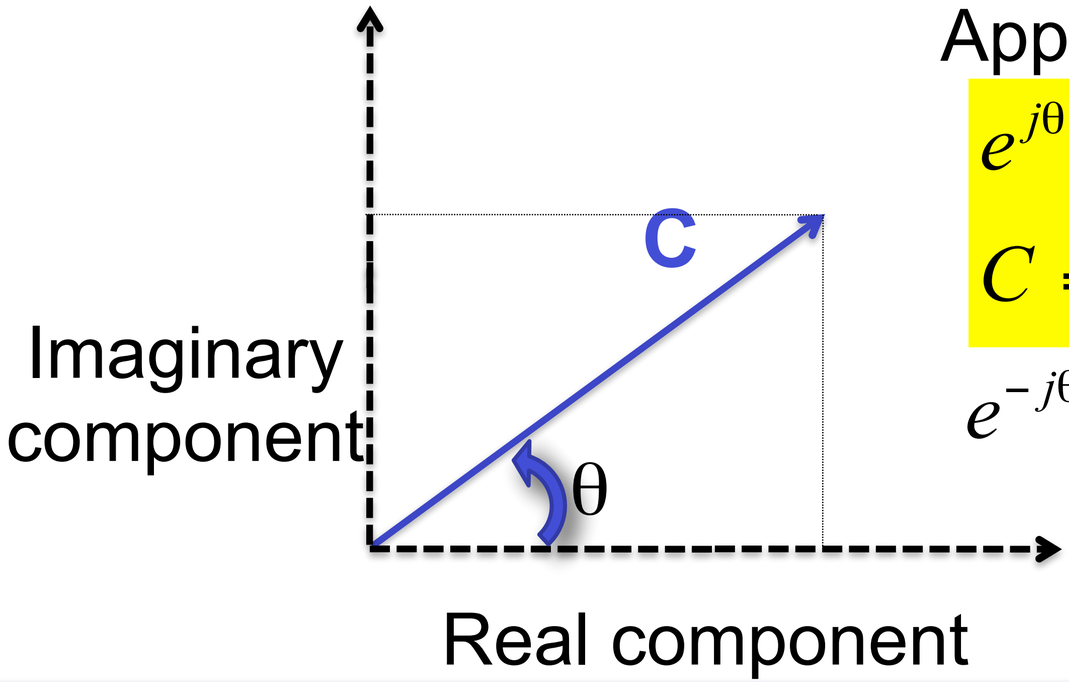
$$e^{j\theta} = \cos\theta + j\sin\theta$$

$$C = |C|e^{j\theta}$$

$$e^{-j\theta} = \cos\theta - j\sin\theta$$

Geometry Review

$$C = |C|(\cos\theta + j\sin\theta)$$



Apply Euler's formula*

$$e^{j\theta} = \cos\theta + j\sin\theta$$

$$C = |C|e^{j\theta}$$

$$e^{-j\theta} = \cos\theta - j\sin\theta$$

Now let's represent all points using this polar notation. If the points came from a periodic function (of space or time, using variable *t* here), we could model them using a Fourier Series!

Fourier Transform (Continuous, 1D)

$$\mathcal{F}\{f(t)\} = F(\mu) = \int_{-\infty}^{\infty} f(t) e^{-j2\pi\mu t} dt$$

The way to convert back from Fourier domain is the Inverse Fourier Transform (more later):

$$f(t) = \int_{-\infty}^{\infty} F(\mu) e^{j2\pi\mu t} d\mu$$

Fourier Transform (Continuous, 1D)

$$\mathcal{F}\{f(t)\} = F(\mu) = \int_{-\infty}^{\infty} f(t) e^{-j2\pi\mu t} dt$$

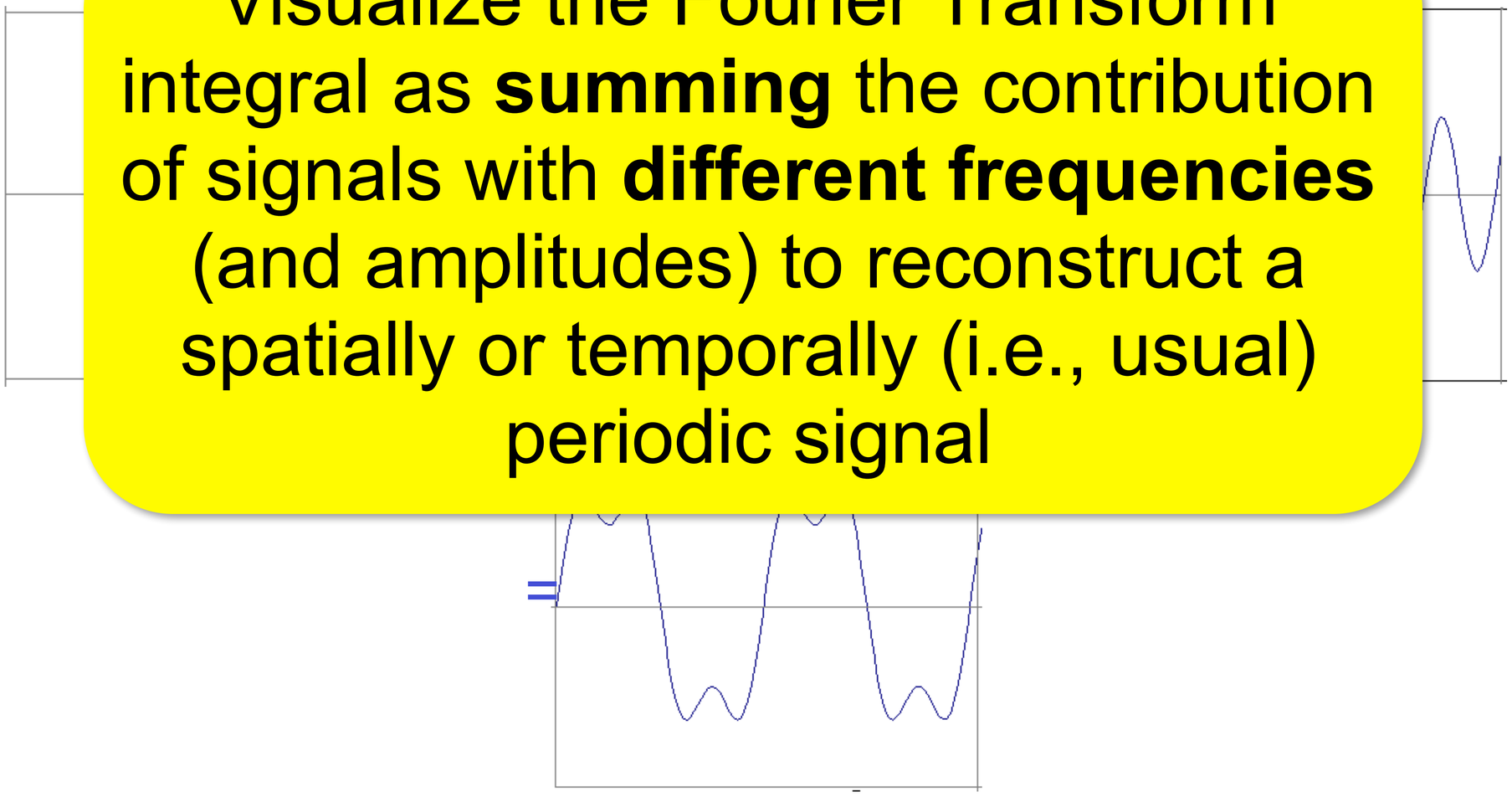
In Frequency domain, think of μ as cycles/sec (Hz)
(according to units of t)

The way to convert back from Fourier domain is the Inverse Fourier Transform (more later):

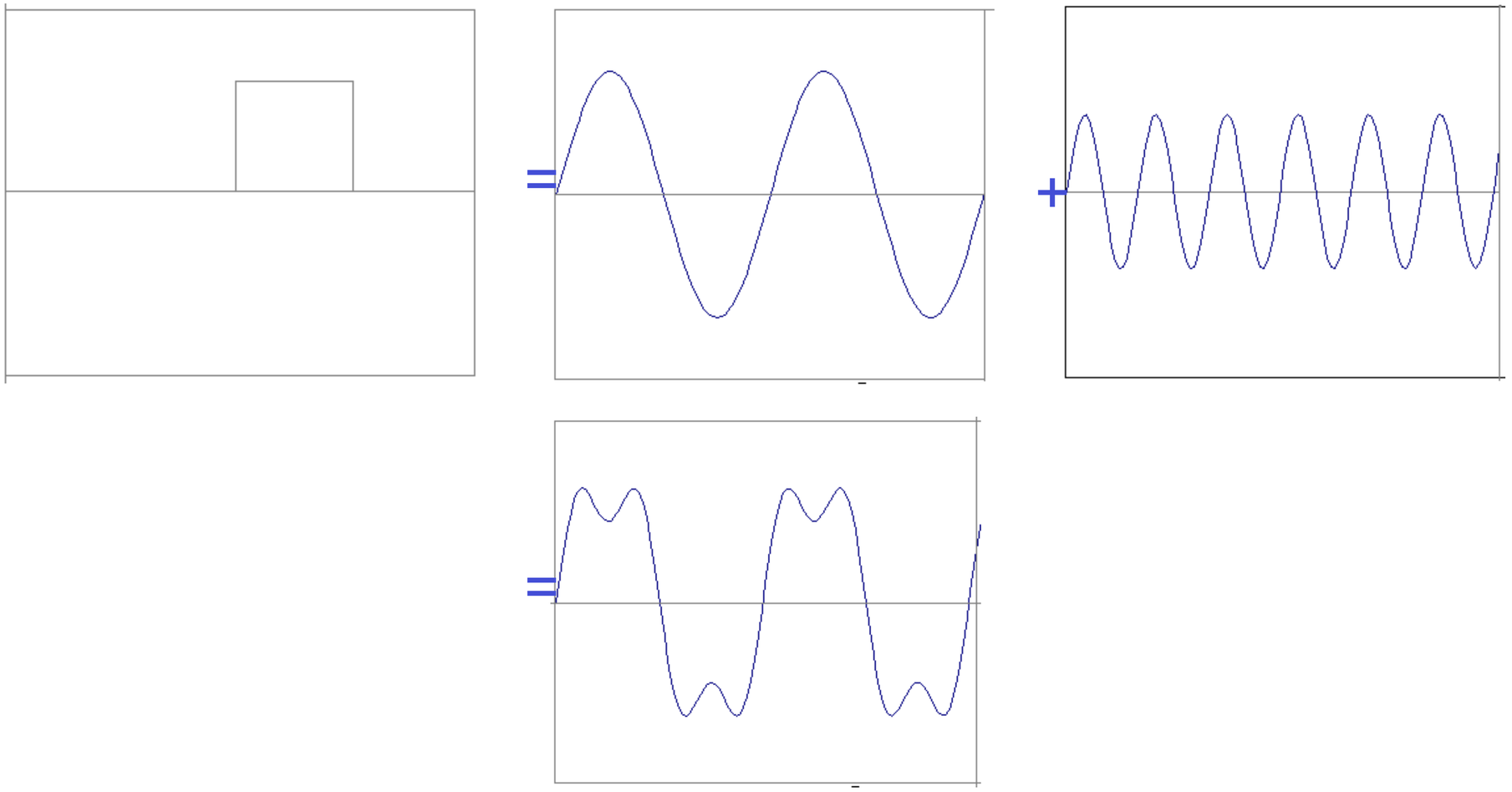
$$f(t) = \int_{-\infty}^{\infty} F(\mu) e^{j2\pi\mu t} d\mu$$

Frequency Spectra

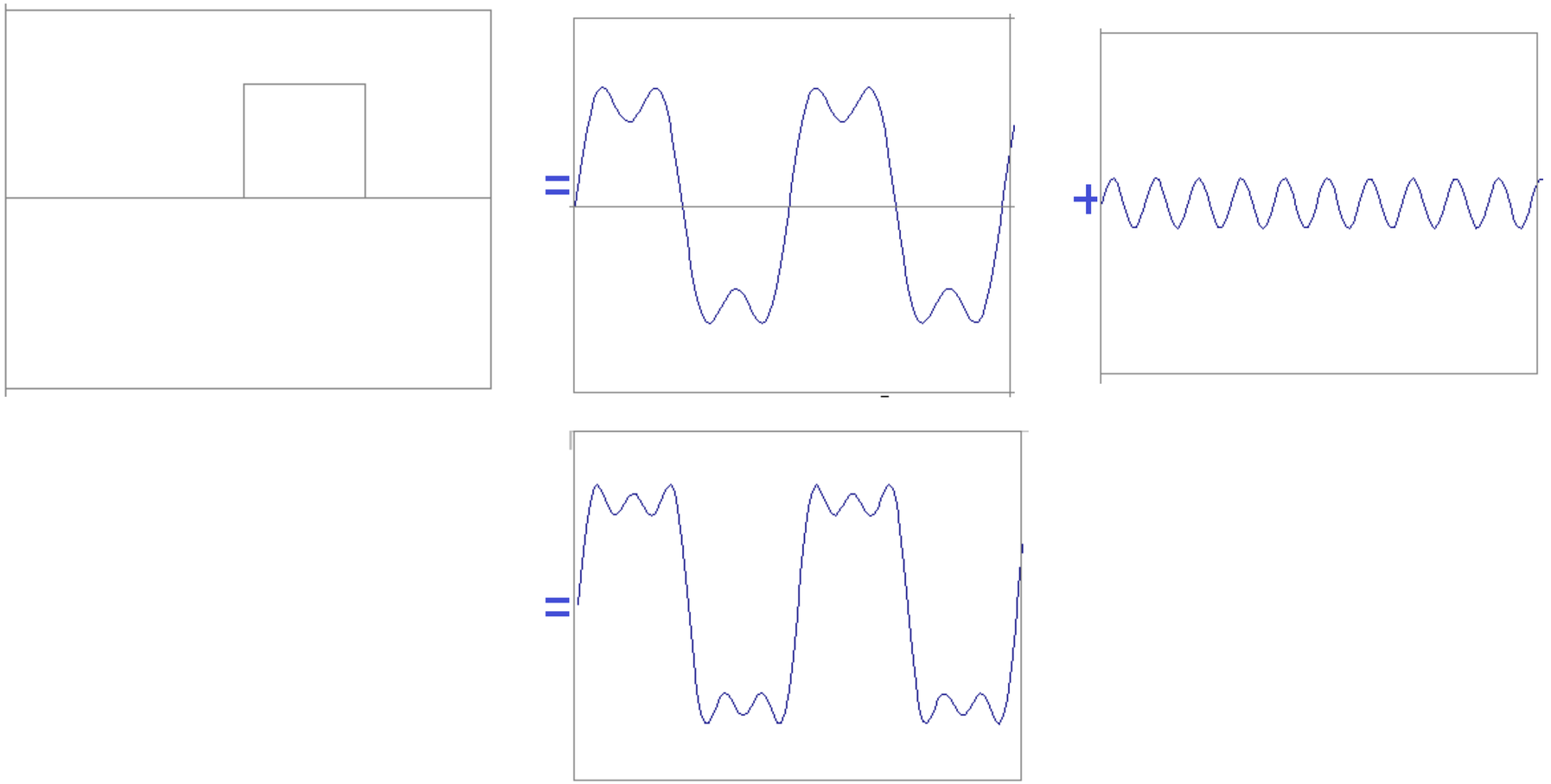
Visualize the Fourier Transform integral as **summing** the contribution of signals with **different frequencies** (and amplitudes) to reconstruct a spatially or temporally (i.e., usual) periodic signal



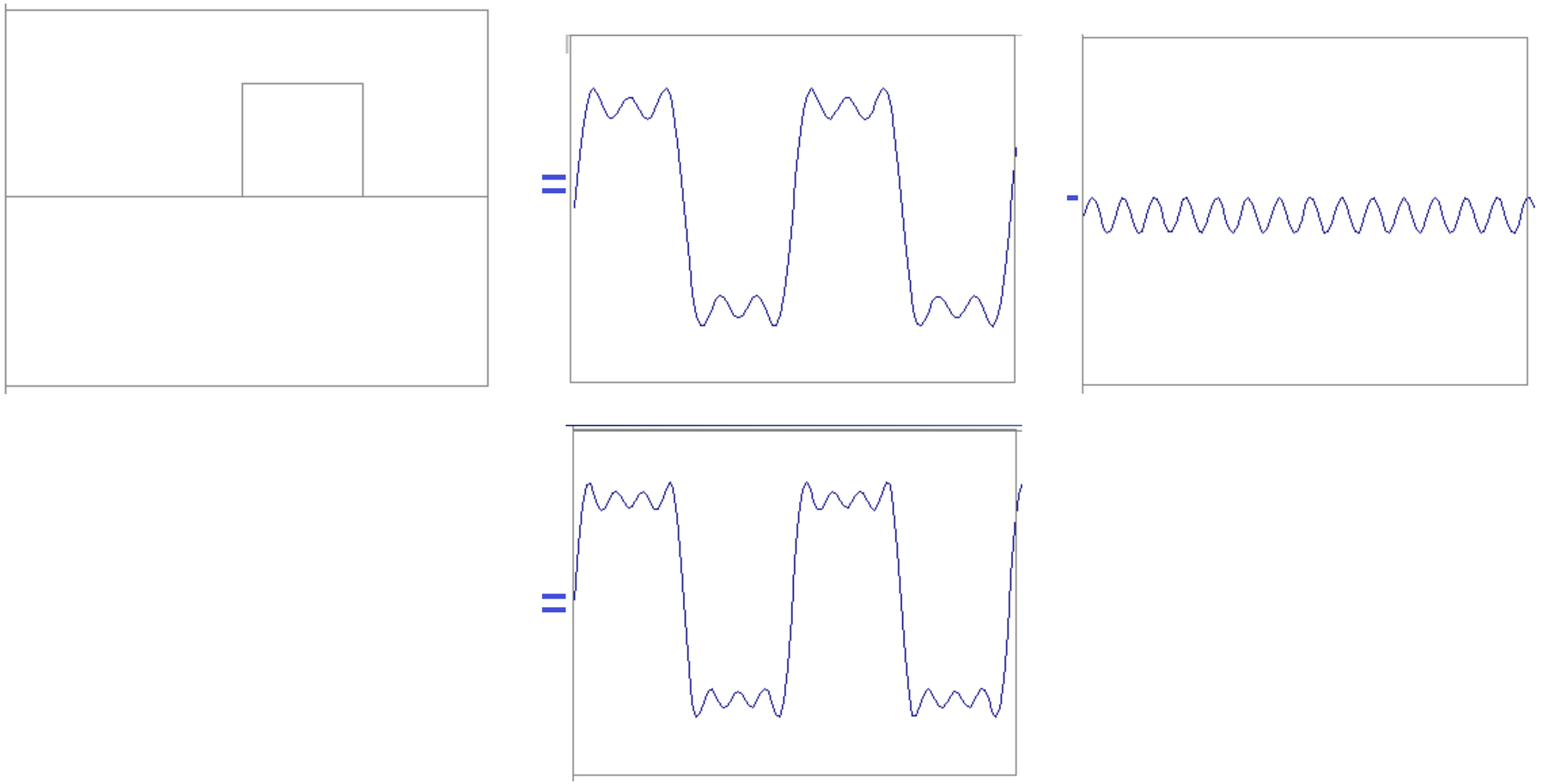
Frequency Spectra



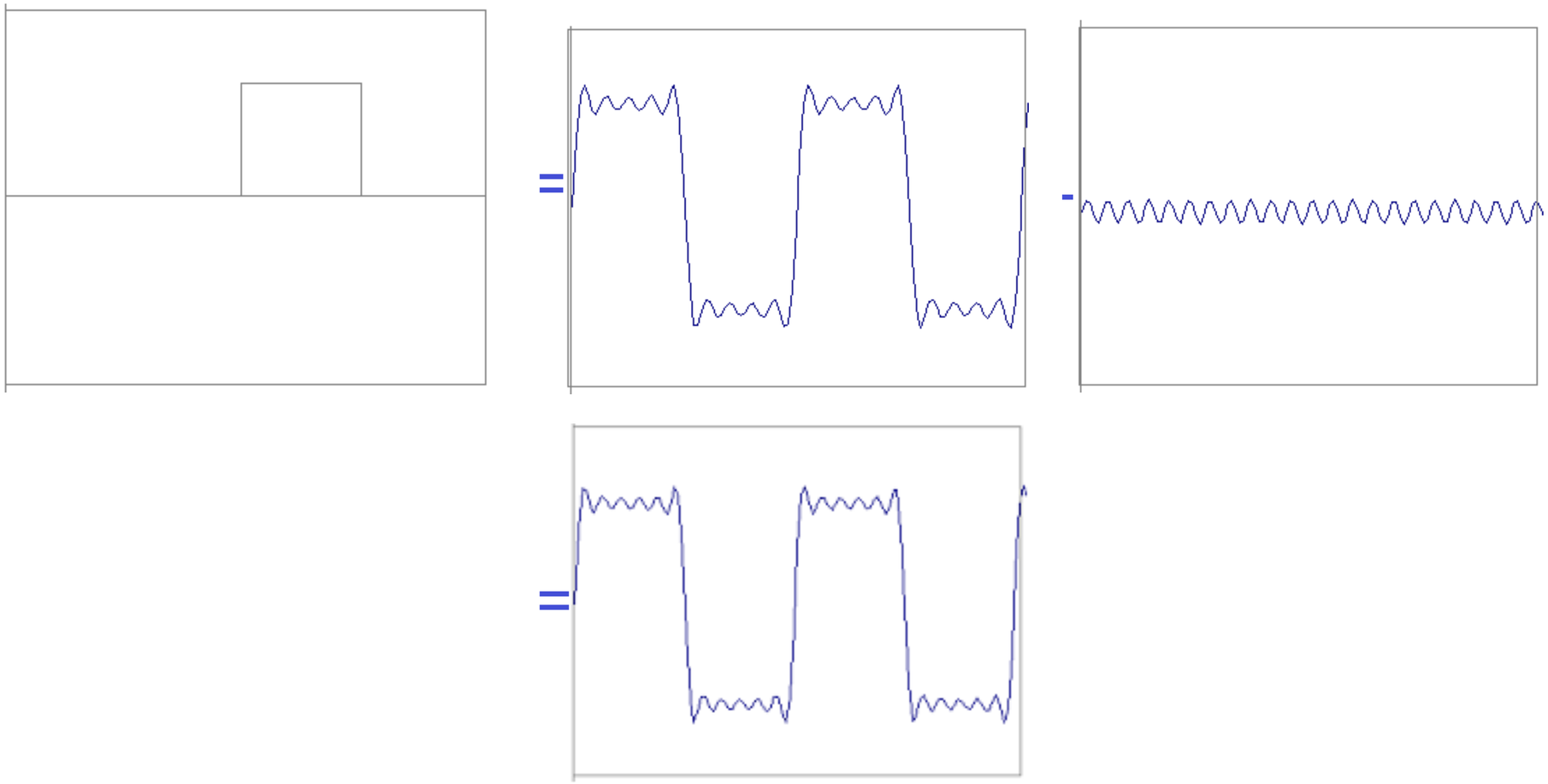
Frequency Spectra



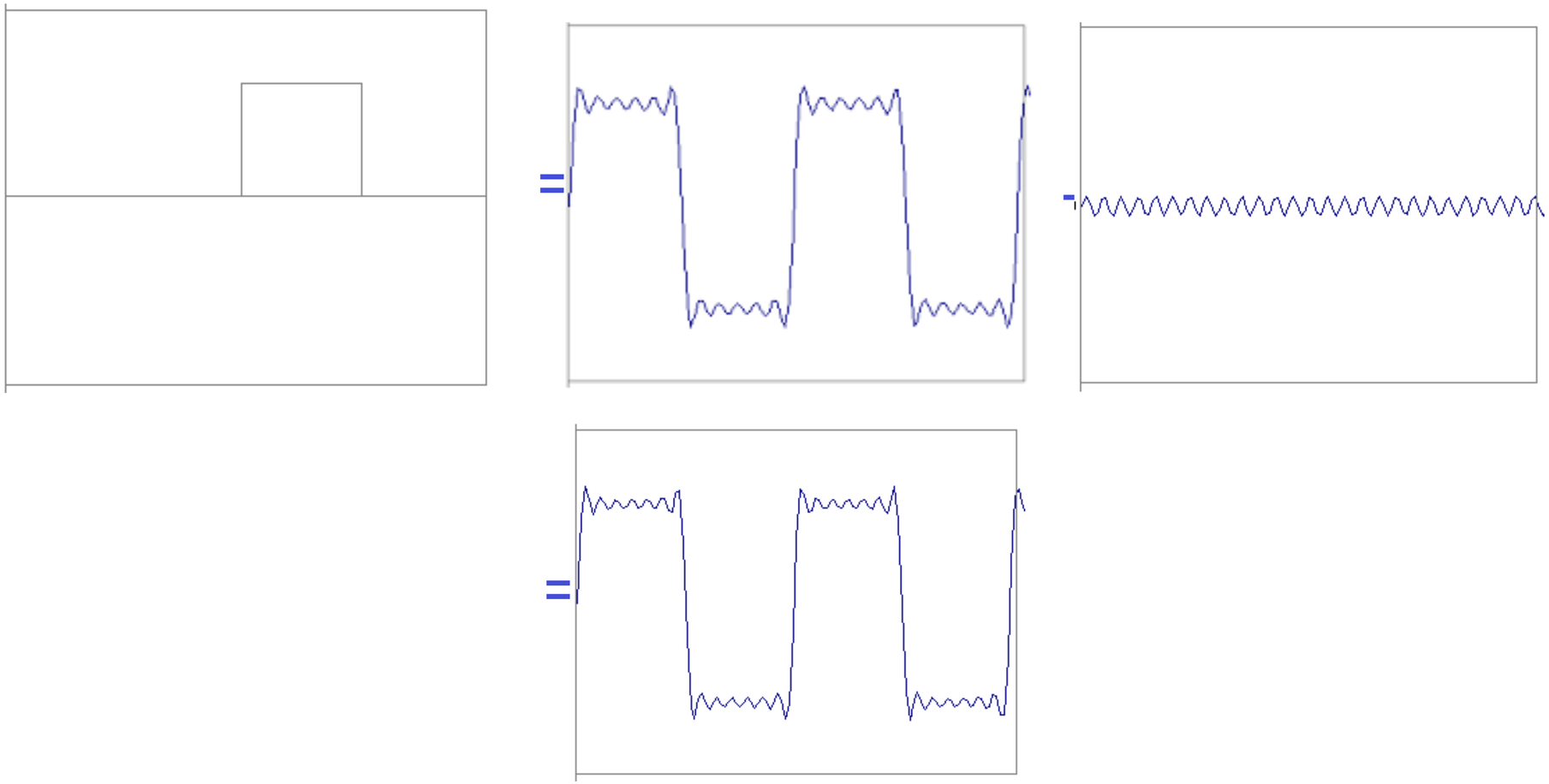
Frequency Spectra



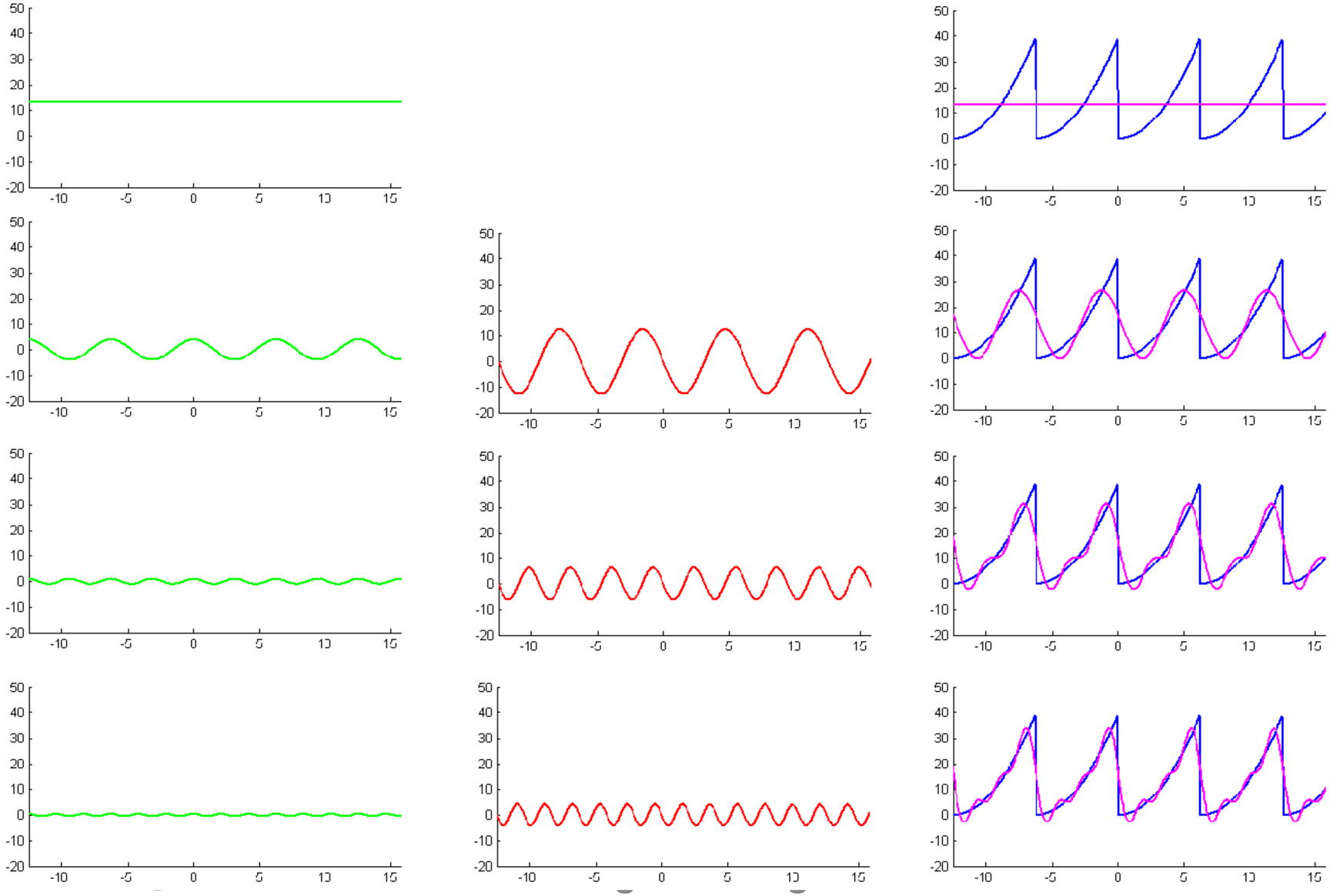
Frequency Spectra

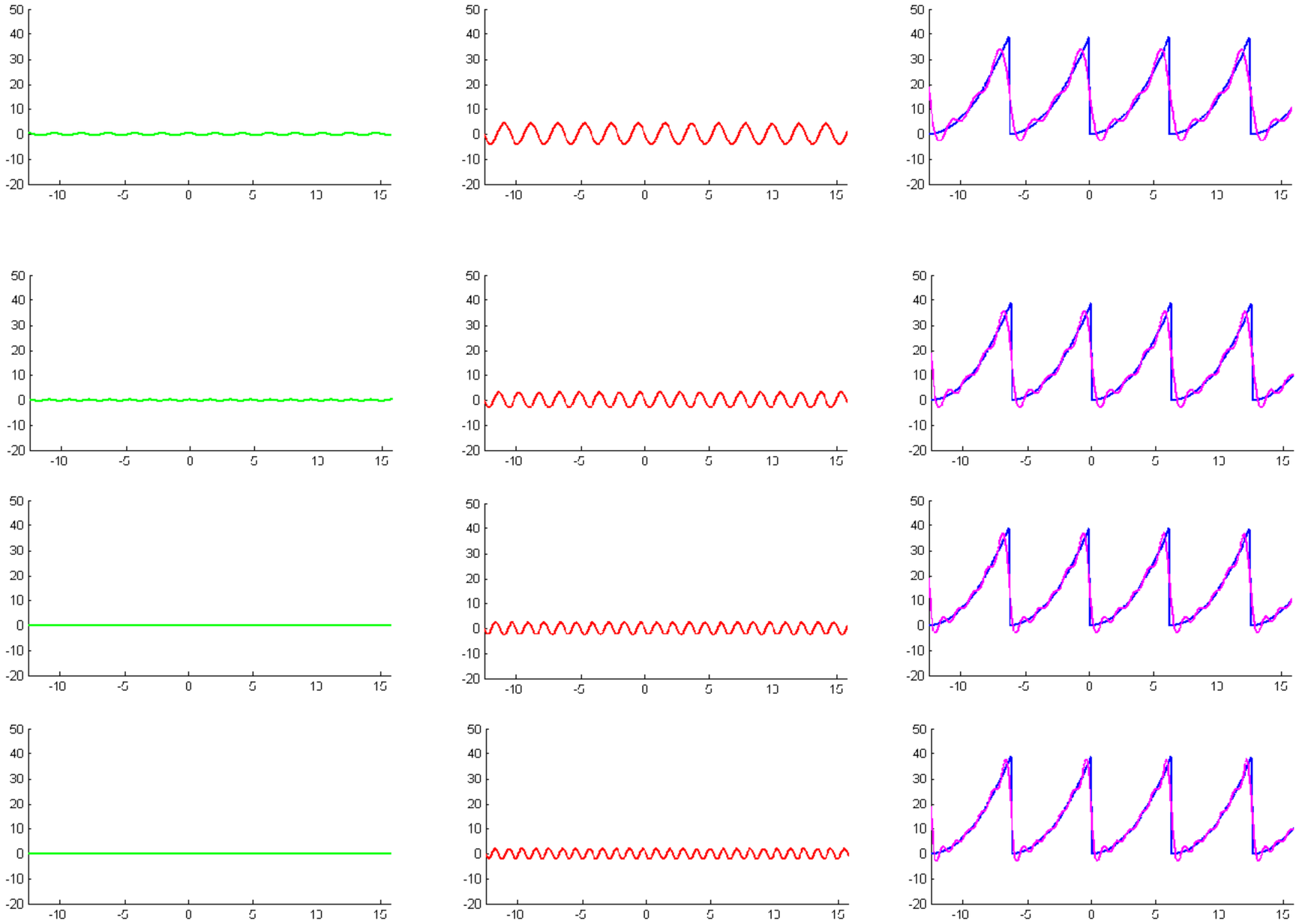


Frequency Spectra



x^2 Example




50
40
30
20
10
0
-10
-20
-10
-5
0
5
10
15

Discrete Fourier Transform (1D)

$$F(u) = \sum_{x=0}^{M-1} f(x) e^{-j 2\pi u x / M} \qquad u = 0, 1, 2, \ldots, M-1$$

Takes a discretized signal of length M (e.g., a row of an image, indexed by x) and returns a new (complex) function of length M that is indexed by u.

What is u? It represents frequency of sinusoidal component

Discrete Fourier Transform (1D)

Discrete Fourier Transform (DFT)

$$F(u) = \sum_{x=0}^{M-1} f(x) e^{-j 2\pi u x / M} \qquad u = 0, 1, 2, \ldots, M-1$$

Discrete Fourier Transform (DFT)

$$F(u) = \sum_{x=0}^{M-1} f(x) \left[\cos(2\pi u x / M) - j \sin(2\pi u x / M) \right]$$

Discrete Fourier Transform (1D)

Discrete Fourier Transform (DFT)

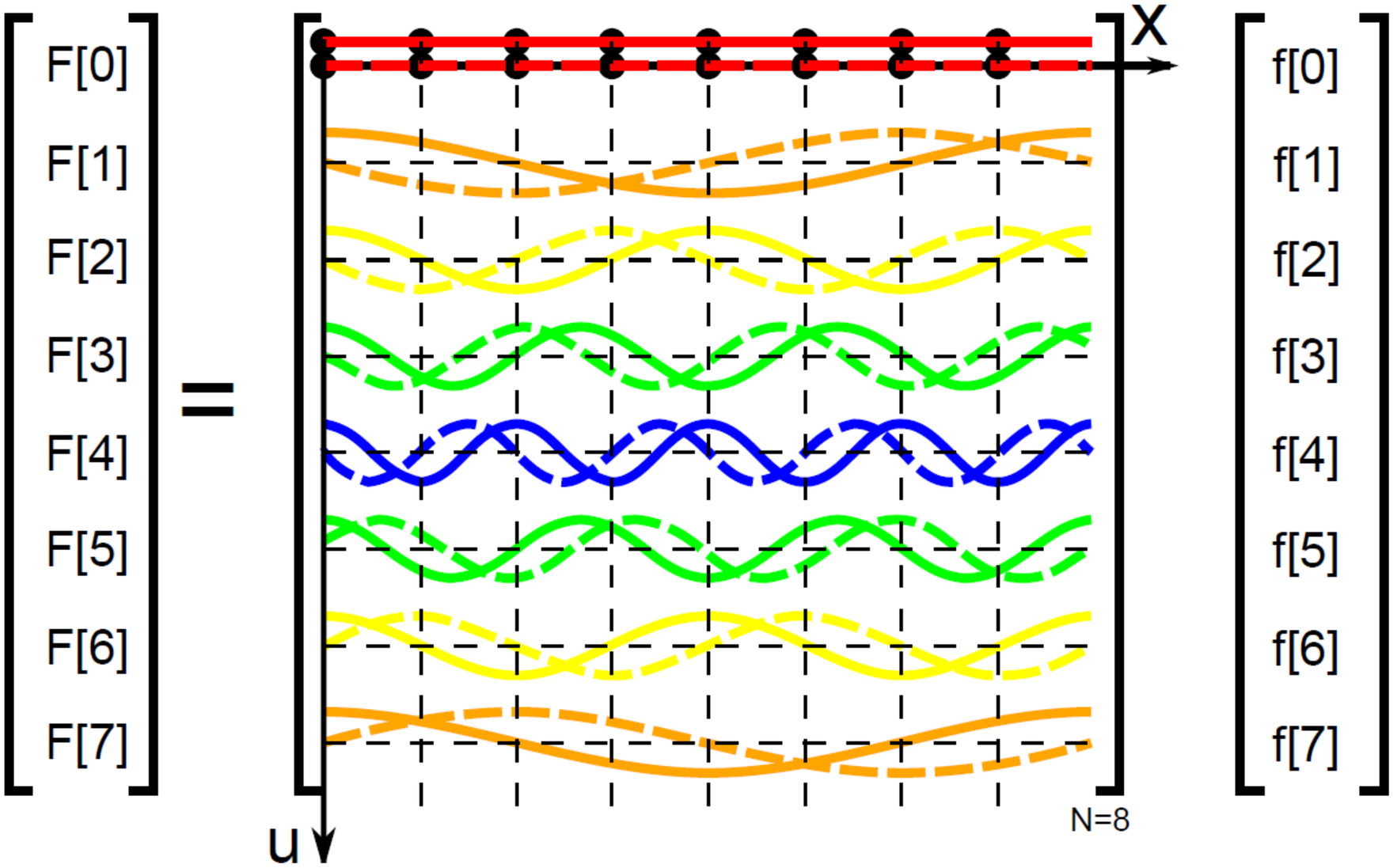
$$F(u) = \sum_{x=0}^{M-1} f(x) e^{-j 2\pi u x / M} \qquad u = 0, 1, 2, \ldots, M-1$$

To understand this apply Euler's formula*

$$e^{j\theta} = \cos\theta + j \sin\theta$$

Discrete Fourier Transform (DFT)

$$F(u) = \sum_{x=0}^{M-1} f(x) \left[\cos(2\pi u x / M) - j \sin(2\pi u x / M) \right]$$



Can think of this as a change of basis: we move from space in which signal is represented as coefficients of delta functions to frequency where signal is represented as coefficients of sinusoidal basis functions

Fourier Transform (Derivation: Continuous 1D Discrete)

(again) $$F(\mu) = \int_{-\infty}^{\infty} f(t) e^{-j 2\pi \mu t} dt$$

Model the sampling of *f(t)* as multiplication with a sampling function:

$$\tilde{f}(t) = \sum_{x=-\infty}^{\infty} f(t) \delta(t - x\Delta t)$$

Fourier Transform (Derivation: Continuous 1D Discrete)

(again) $$F(\mu) = \int_{-\infty}^{\infty} f(t) e^{-j 2\pi \mu t} dt$$

Model the sampling of *f(t)* as multiplication with a sampling function:

$$\tilde{f}(t) = \sum_{x=-\infty}^{\infty} f(t) \delta(t - x\Delta t)$$

What is δ(.) and why does this give us a comb-filter?

Fourier Transform (Derivation: Continuous 1D Discrete)

(again) $$F(\mu) = \int_{-\infty}^{\infty} f(t) e^{-j 2\pi \mu t} dt$$

Model the sampling of *f(t)* as multiplication with a sampling function:

$$\tilde{f}(t) = \sum_{x=-\infty}^{\infty} f(t) \delta(t - x\Delta t)$$

$$\Rightarrow \tilde{F}(\mu) = \int_{-\infty}^{\infty} \tilde{f}(t) e^{-j 2\pi \mu t} dt$$

Fourier Transform (Derivation: Continuous 1D Discrete)

(again) $$F(\mu) = \int_{-\infty}^{\infty} f(t) e^{-j 2\pi \mu t} dt$$

Model the sampling of *f(t)* as multiplication with a sampling function:

$$\tilde{f}(t) = \sum_{x=-\infty}^{\infty} f(t) \delta(t - x\Delta t)$$

$$\Rightarrow \tilde{F}(\mu) = \int_{-\infty}^{\infty} \tilde{f}(t) e^{-j 2\pi \mu t} dt$$ (continued…)

$$\widetilde{F}(\mu) = \int_{-\infty}^{\infty} \widetilde{f}(t) e^{-j2\pi\mu t}\, dt$$

$$= \int_{-\infty}^{\infty} \sum_{x=-\infty}^{\infty} f(t)\delta(t - x\Delta t) e^{-j2\pi\mu t}\, dt$$

$$= \sum_{x=-\infty}^{\infty} \int_{-\infty}^{\infty} f(t)\delta(t - x\Delta t) e^{-j2\pi\mu t}\, dt$$

(Integral is only non - zero when $t = x\Delta t$)

$$= \sum_{x=-\infty}^{\infty} f(x) e^{-j2\pi\mu x\Delta t}$$

$$\widetilde{F}(\mu) = \int_{-\infty}^{\infty} \widetilde{f}(t) e^{-j2\pi\mu t}\, dt$$

$$= \int_{-\infty}^{\infty} \sum_{x=-\infty}^{\infty} f(t)\delta(t - x\Delta t) e^{-j2\pi\mu t}\, dt$$

$$= \sum_{x=-\infty}^{\infty} \int_{-\infty}^{\infty} f(t)\delta(t - x\Delta t) e^{-j2\pi\mu t}\, dt$$

(Integral is only non - zero when $t = x\Delta t$)

$$= \sum_{x=-\infty}^{\infty} f(x) e^{-j2\pi\mu x\Delta t}$$

(continued…)

$$\widetilde{F}(\mu) = \sum_{x=-\infty}^{\infty} f(x) e^{-j 2\pi \mu x \Delta t} \quad \Big\}$$ (from last page)

$\text{freq} = \dfrac{1}{\Delta t}$ and we want *M* equally-spaced samples,
so sweep continuous frequency-space
with samples μ, where $\mu = \dfrac{u}{M}\dfrac{1}{\Delta t}$

Discrete Fourier Transform (DFT)

$$F(u) = \sum_{x=0}^{M-1} f(x) e^{-j 2\pi u x / M} \qquad u = 0, 1, 2, \ldots, M-1$$

Inverse Discrete Fourier Transform (IDFT)

$$f(x) = \frac{1}{M} \sum_{u=0}^{M-1} F(u) e^{j 2\pi u x / M} \qquad x = 0, 1, 2, \ldots, M-1$$

Discrete Fourier Transform (2D)

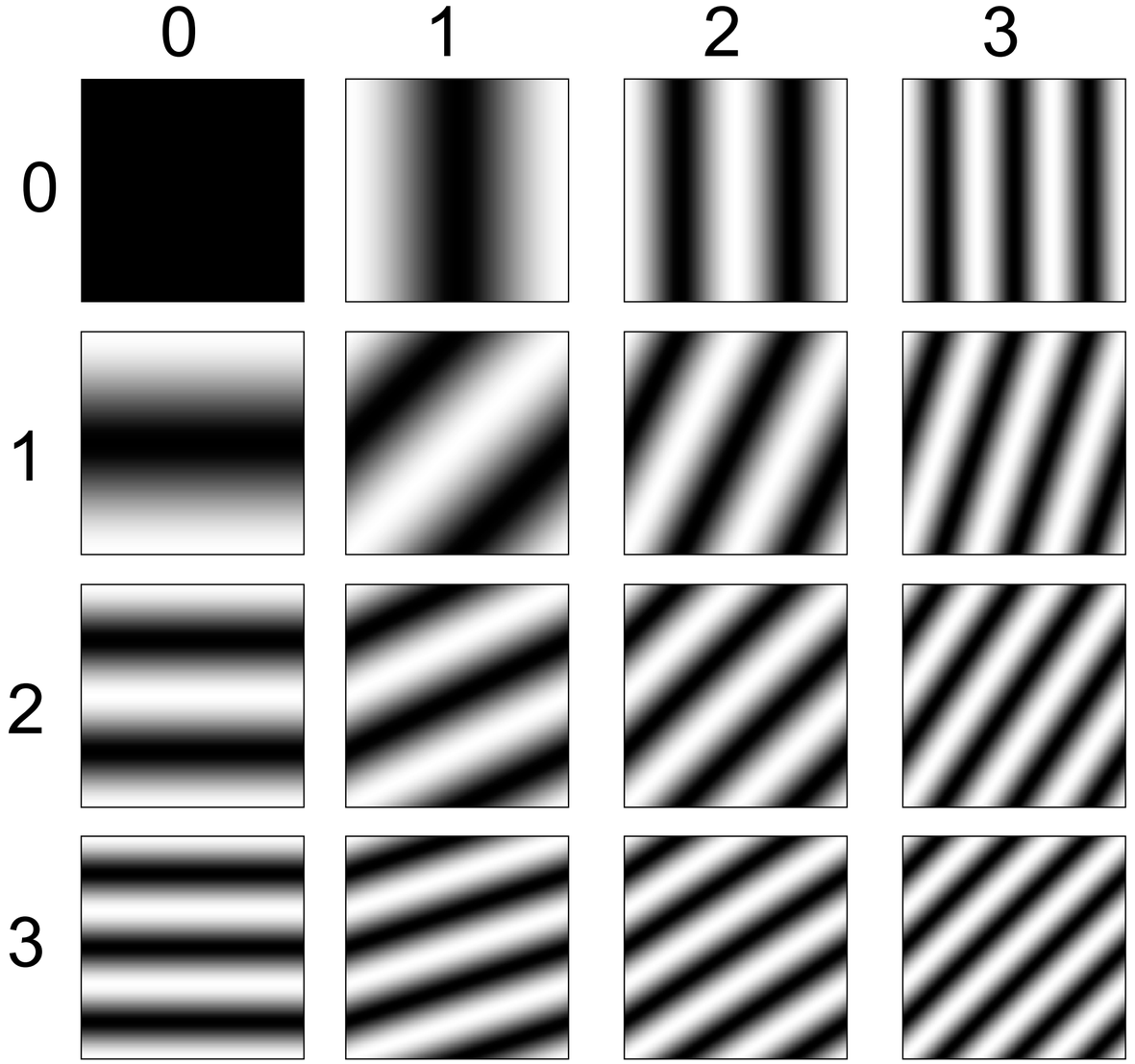
$$F(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

where:
u = 0, 1, 2, …, M-1
v = 0, 1, 2, …, N-1

$$f(x,y) = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

where:
x = 0, 1, 2, …, M-1
y = 0, 1, 2, …, N-1

Fourier Basis Functions



2D Fourier Domain Example

Full image

2D Fourier Domain Example

Full image

First 1 basis fn

2D Fourier Domain Example

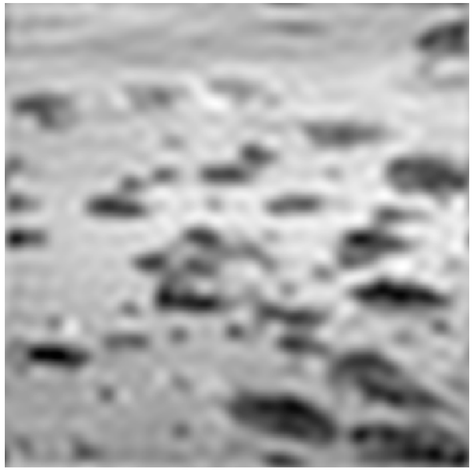
Full image

First 1 basis fn

First 4 basis fns

First 9 basis fns

First 16 basis fns



First 400 basis fns

Fast Fourier Transform

Fast Fourier Transform

- **Naïve DFT is O(N^4)**

Fast Fourier Transform

- **Naïve DFT is O(N^4)**

- **The FFT computes the DFT of an *N*x*N* image in O($N^2 \log_2 N$) time if $N = 2^M$**

Fast Fourier Transform

- **Naïve DFT is O(N^4)**
- **The FFT computes the DFT of an *N*x*N* image in O($N^2 \log_2 N$) time if $N = 2^M$**
- **FFTW = Fastest FT in the West** (http://www.fftw.org/)

Image Spectra

Image Spectra

- ***F*** **is a 2D** ***M*****x*****N*** **array of complex numbers.**

Image Spectra

- F is a 2D MxN array of complex numbers.
- Common to plot $|F|$ and $\arg(F)$ as images

Image Spectra

- F is a 2D MxN array of complex numbers.
- Common to plot $|F|$ and $\arg(F)$ as images
- $|F|$ is the magnitude or amplitude spectrum

Image Spectra

- F is a 2D MxN array of complex numbers.
- Common to plot $|F|$ and $\arg(F)$ as images
- $|F|$ is the magnitude or amplitude spectrum
- $\arg(F)$ is the phase spectrum

Image Spectra

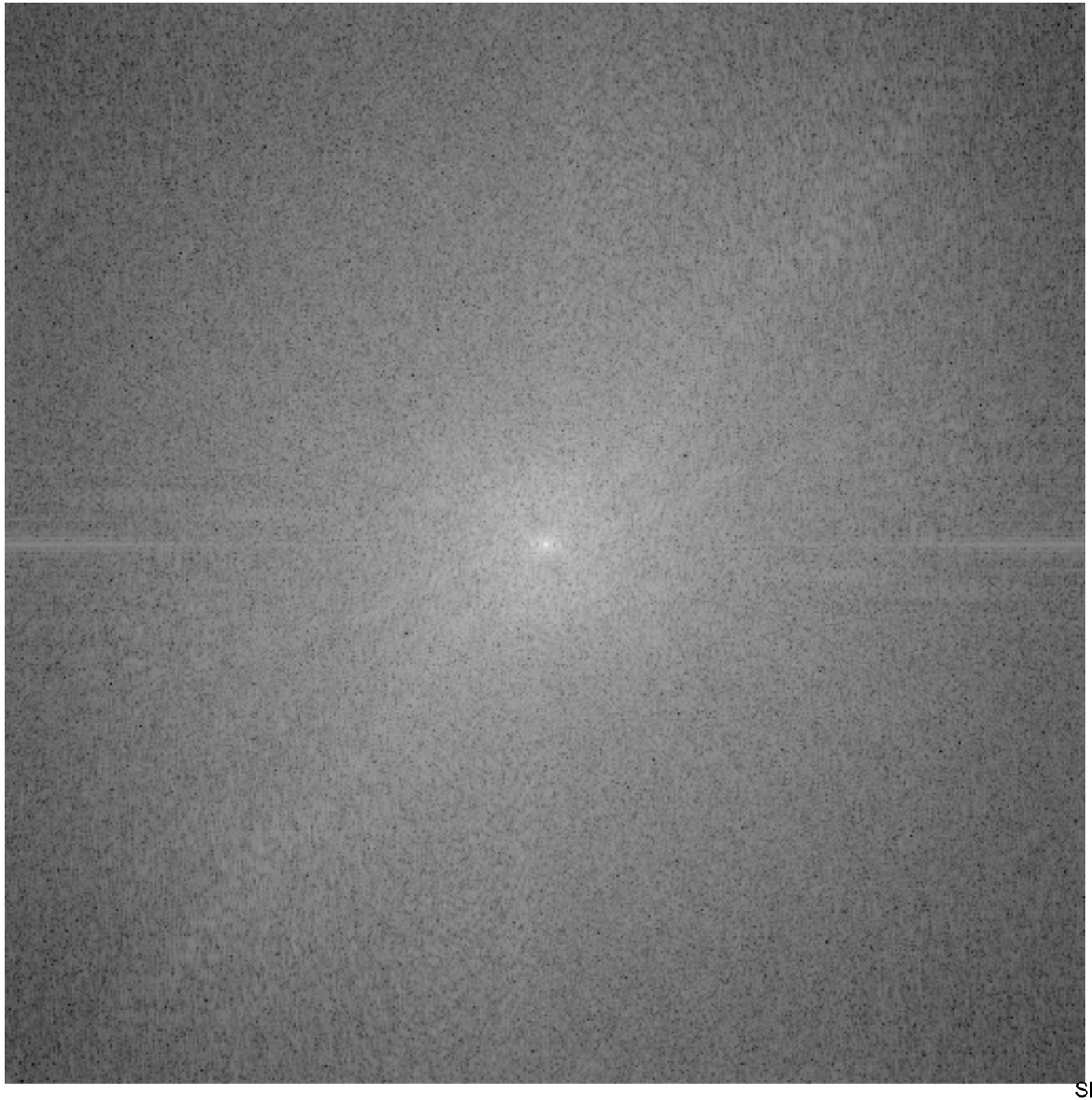
- F is a 2D MxN array of complex numbers.
- Common to plot $|F|$ and $\arg(F)$ as images
- $|F|$ is the magnitude or amplitude spectrum
- $\arg(F)$ is the phase spectrum
- F is periodic:
  $F(u,v) = F(u+M,v) = F(u,v+N) = F(u+M,v+N)$.

Image Spectra

- F is a 2D MxN array of complex numbers.
- Common to plot $|F|$ and $\arg(F)$ as images
- $|F|$ is the magnitude or amplitude spectrum
- $\arg(F)$ is the phase spectrum
- F is periodic:
  $F(u,v) = F(u+M,v) = F(u,v+N) = F(u+M,v+N)$.
- F has conjugate symmetry so
  $|F(u,v)|=|F(-u,-v)|$

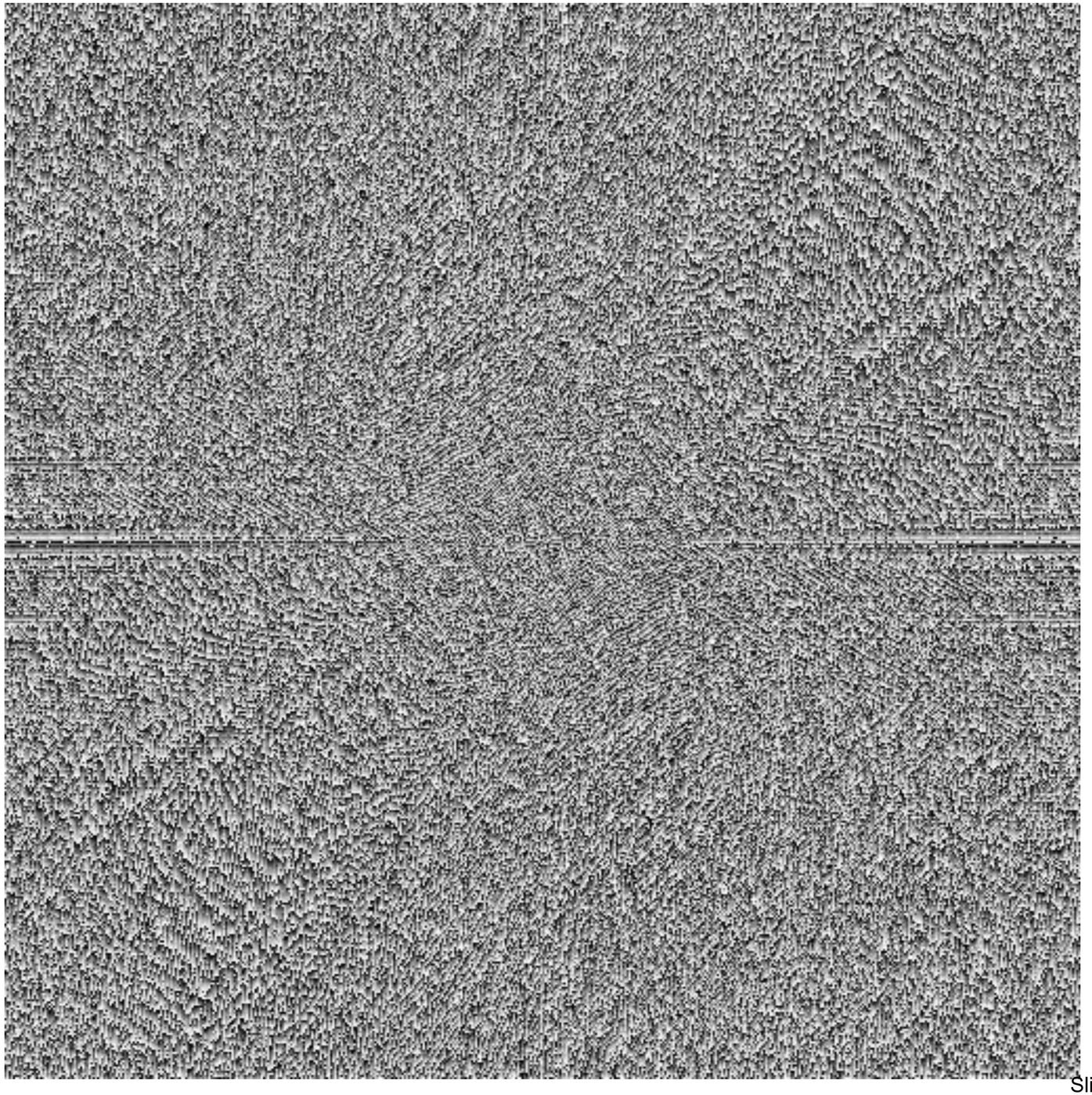
Slide by Alexei Efros

This is the magnitude transform of the cheetah pic



Slide by Alexei Efros

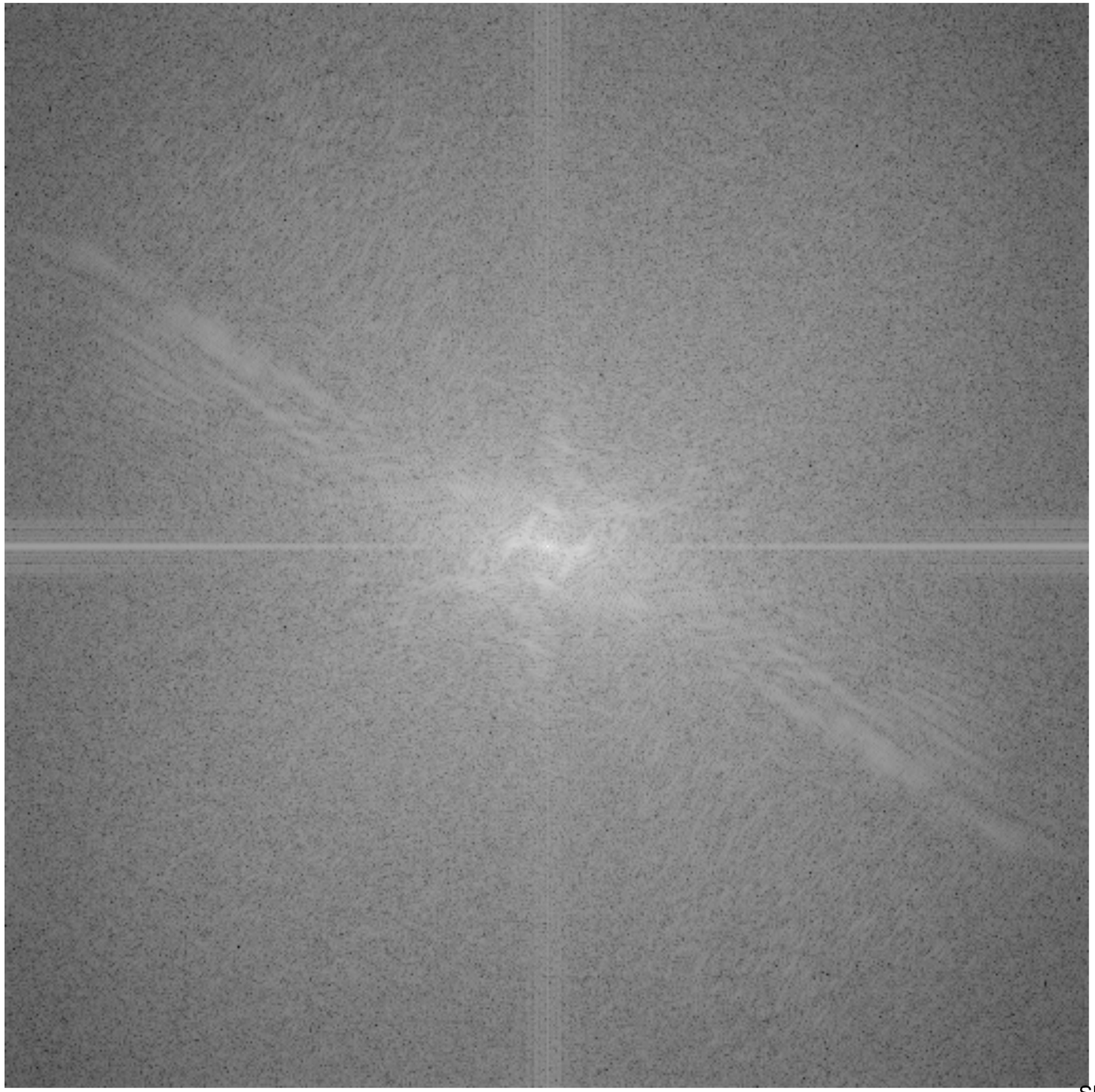
This is the phase transform of the cheetah pic



Slide by Alexei Efros

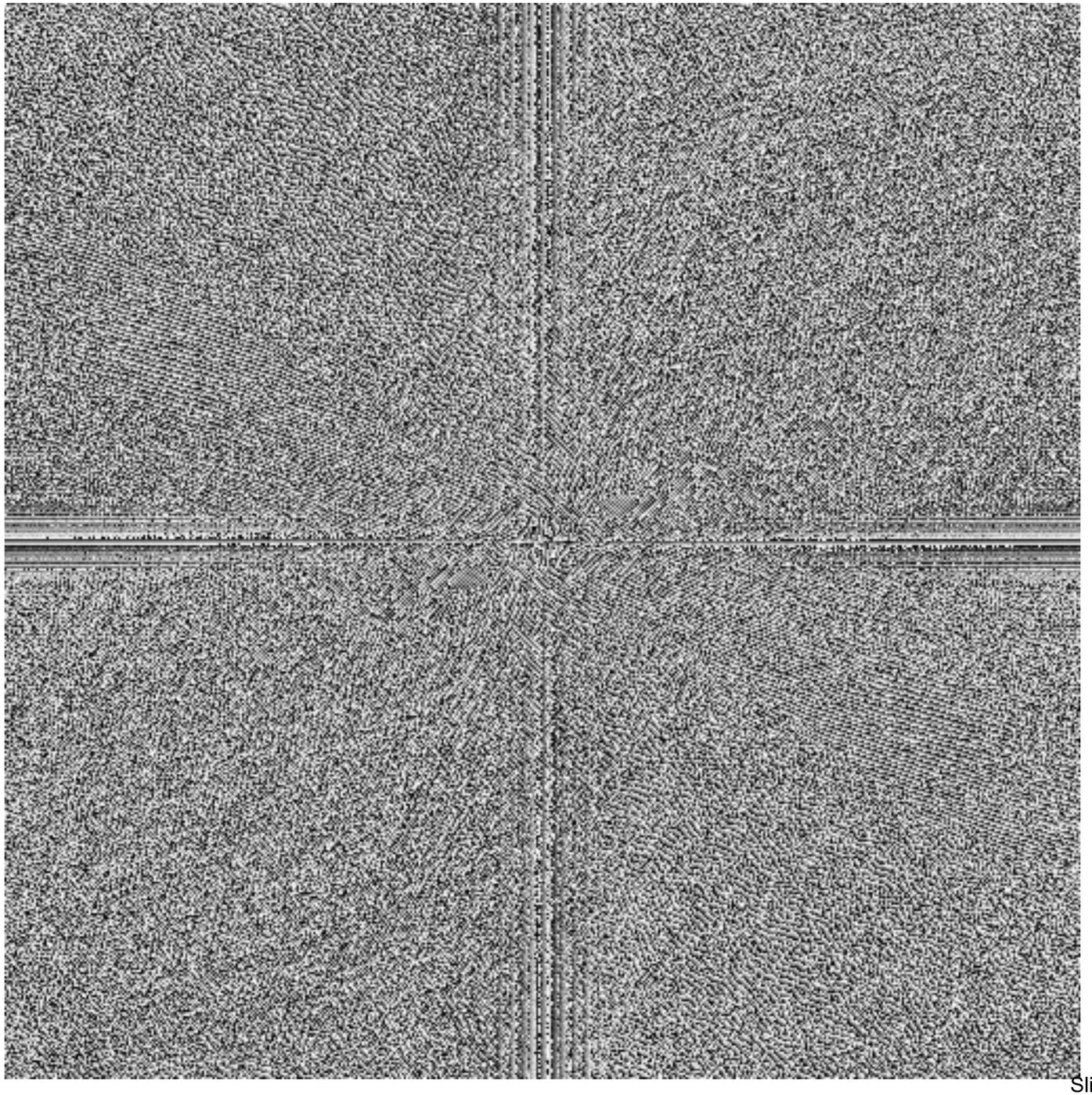
Slide by Alexei Efros

This is the magnitude transform of the zebra pic

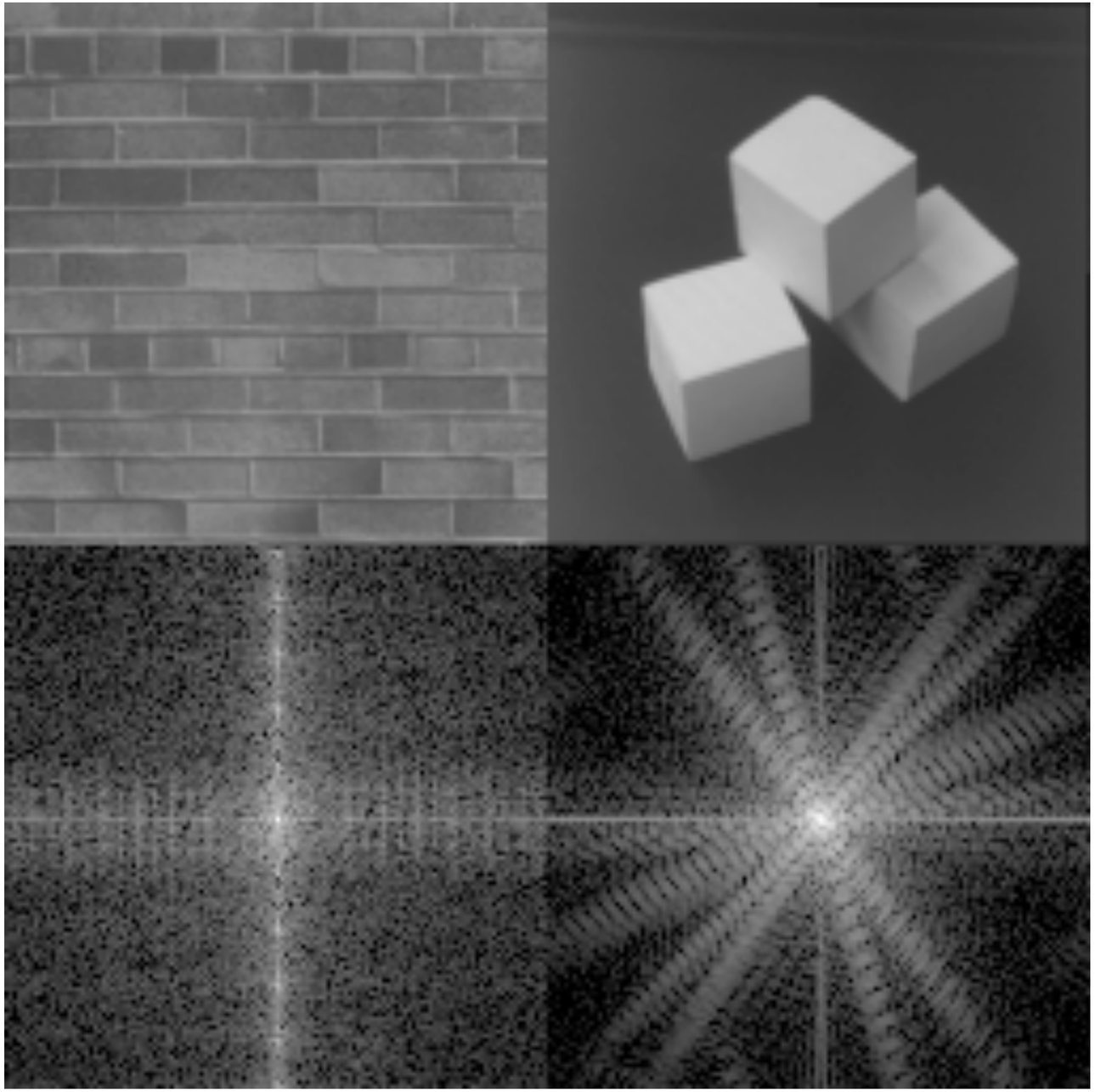


Slide by Alexei Efros

This is the phase transform of the zebra pic



Slide by Alexei Efros



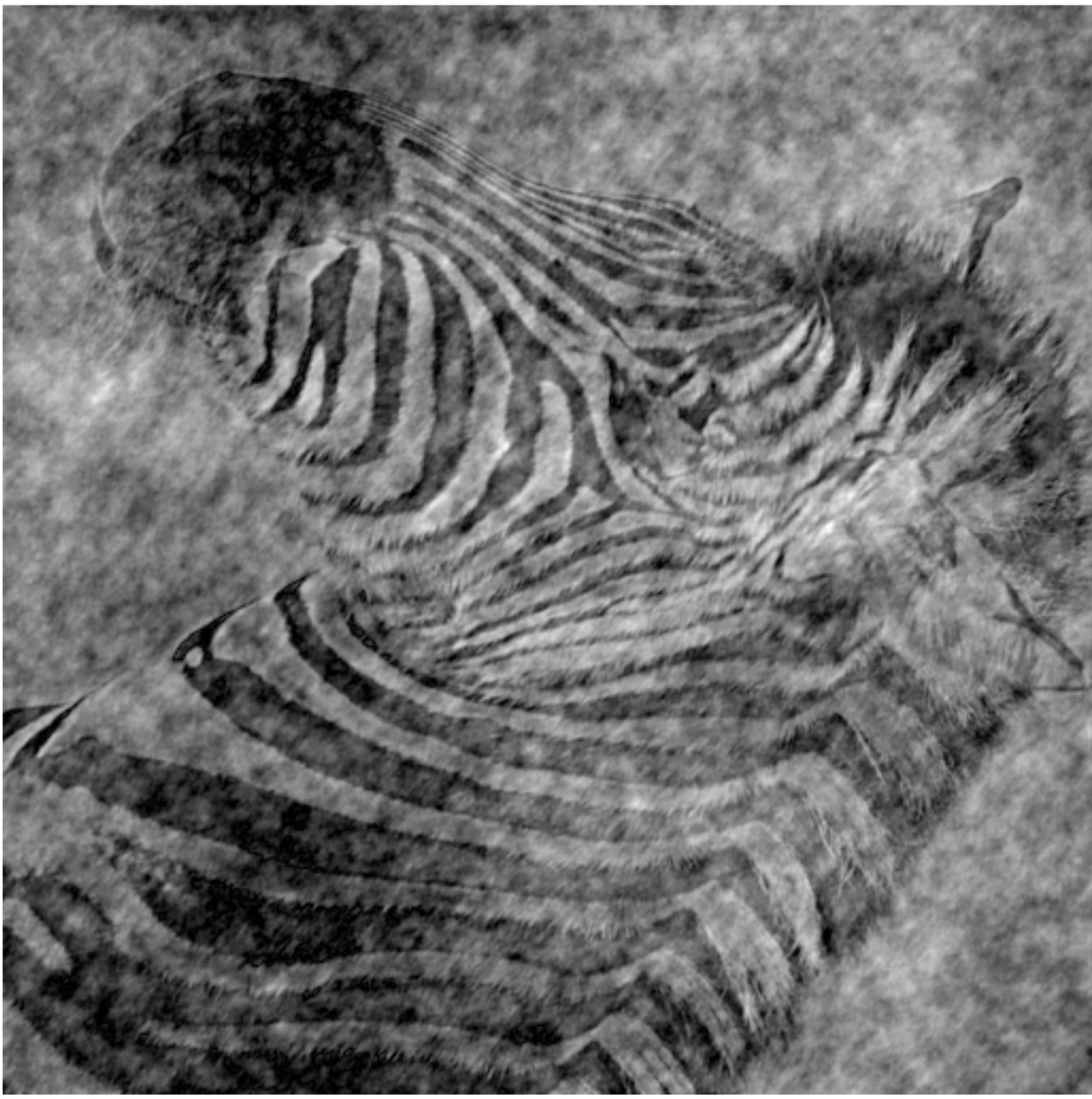
Oriented Components

Curious things about FT on images

- The magnitude spectra of all natural images are quite similar
  - Heavy on low-frequencies,
    falling off in high frequencies

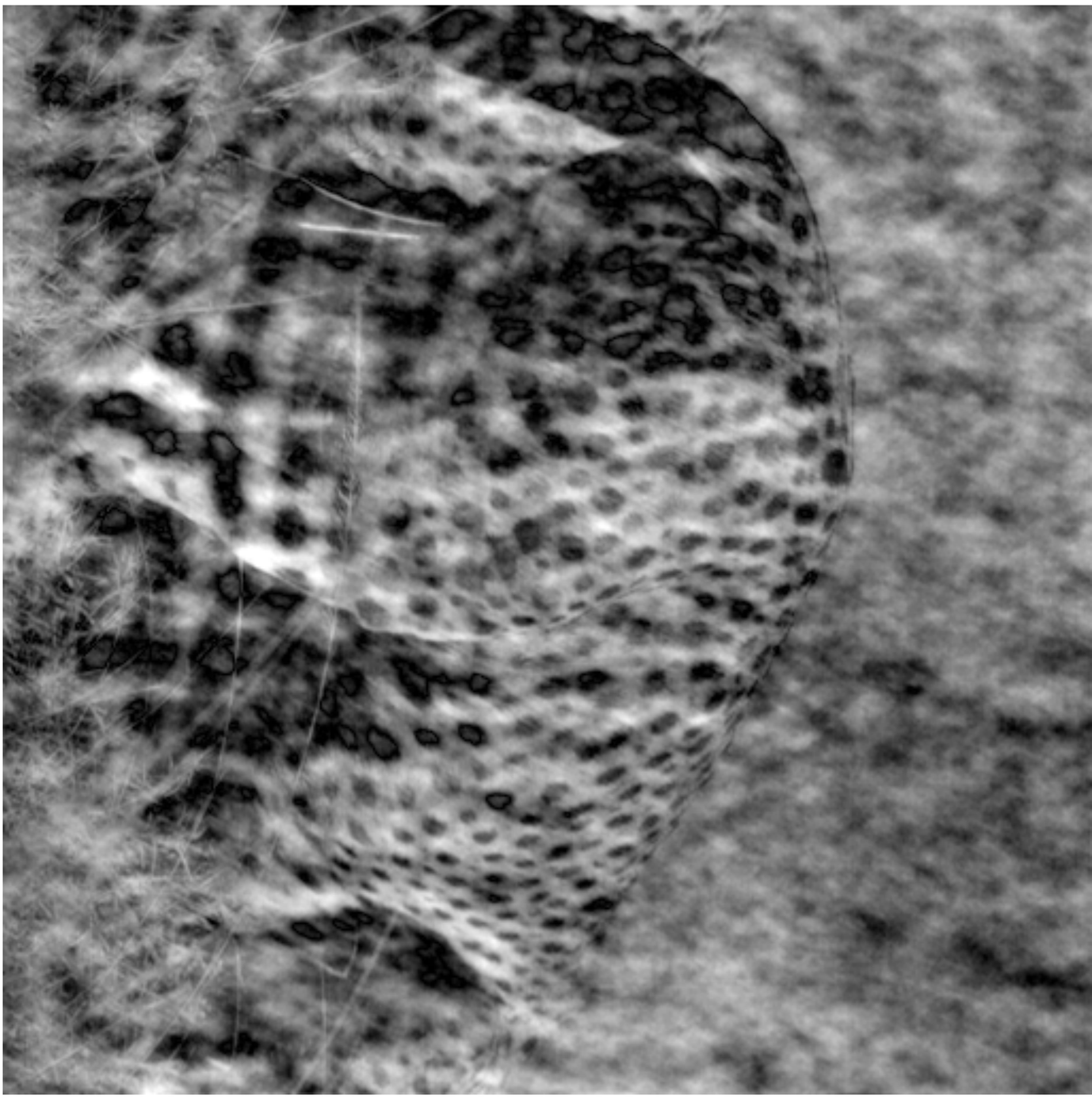
- Most information in the image is carried in the phase, not the amplitude
  - Not quite clear why

Reconstruction with zebra phase,
cheetah magnitude



Slide by Alexei Efros

Reconstruction with cheetah phase,
zebra magnitude



Slide by Alexei Efros

Filtering in the Fourier Domain (why we care)

Filtering in the Fourier Domain (why we care)

- The Convolution Theorem states that

$$\Im[f * g] = \Im[f]\Im[g]$$

where $\Im$ denotes the Fourier Transform

Filtering in the Fourier Domain (why we care)

- The Convolution Theorem states that

$$\Im[f * g] = \Im[f]\Im[g]$$

where $\Im$ denotes the Fourier Transform

- To convolve image *f* by kernel *g*:
  - Compute DFTs *F* and *G* of *f* and *g*
  - Multiply *F* by *G*
  - Compute the inverse DFT of *FG*.

```
function g = dftfilt(f, H)
%DFTFILT Performs frequency domain filtering.
%   From DIPUM by Gonzalez, Woods, and Eddins 2004
%   G = DFTFILT(F, H) filters F in the frequency domain using the
%   filter transfer function H. The output, G, is the filtered
%   image, which has the same size as F.  DFTFILT automatically pads
%   H to be the same size as F.  Function PADDEDSIZE can be used to
%   determine an appropriate size for H.
%
%   DFTFILT assumes that F is real and that H is a real, uncentered
%   circularly-symmetric filter function.

%   Copyright 2002-2004 R. C. Gonzalez, R. E. Woods, & S. L. Eddins
%   Digital Image Processing Using MATLAB, Prentice-Hall, 2004
%   $Revision: 1.5 $  $Date: 2003/08/25 14:28:22 $

% Obtain the FFT of the padded input.
F = fft2(f, size(H, 1), size(H, 2));

% Perform filtering.
g = real(ifft2(H.*F));

% Crop to original size.
g = g(1:size(f, 1), 1:size(f, 2));
```

See also: fftshift() and ifftshift()

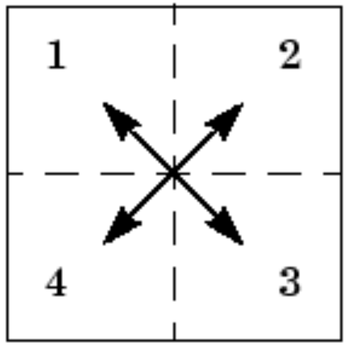
```
function g = dftfilt(f, H)
%DFTFILT Performs frequency domain filtering.
%   From DIPUM by Gonzalez, Woods, and Eddins 2004
%   G = DFTFILT(F, H) filters F in the frequency domain using the
%   filter transfer function H. The output, G, is the filtered
%   image, which has the same size as F.  DFTFILT automatically pads
%   H to be the same size as F.  Function PADDEDSIZE can be used to
%   determine an appropriate size for H.
%
%   DFTFILT assumes that F is real and that H is a real, uncentered
%   circularly-symmetric filter function.

%   Copyright 2002-2004 R. C. Gonzalez, R. E. Woods, & S. L. Eddins
%   Digital Image Processing Using MATLAB, Prentice-Hall, 2004
%   $Revision: 1.5 $  $Date: 2003/08/25 14:28:22 $

% Obtain the FFT of the padded input.
F = fft2(f, size(H, 1), size(H,

% Perform filtering.
g = real(ifft2(H.*F));

% Crop to original size.
g = g(1:size(f, 1), 1:size(f,
```

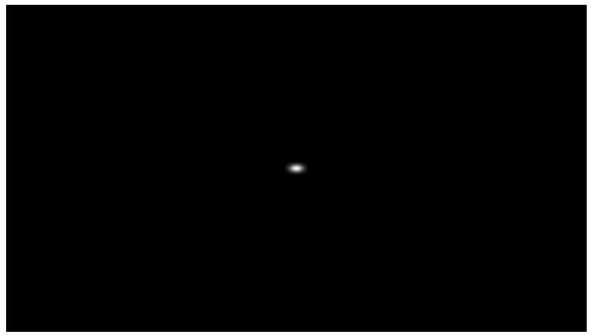


(from Matlab Help)

See also: fftshift() and ifftshift()

Example: 2D Convolution Theorem

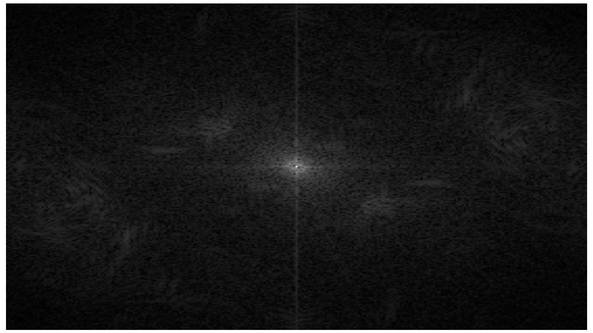
$f(x,y)$ 

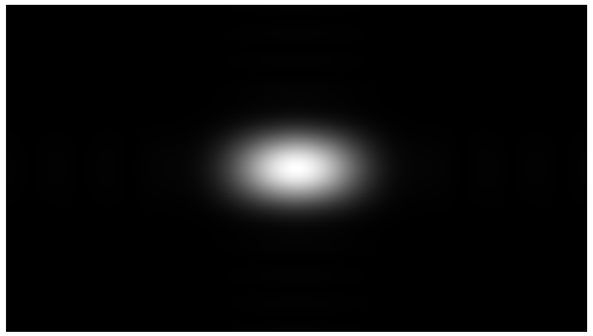
$*$ 

$g(x,y)$

$\Downarrow$

$h(x,y)$

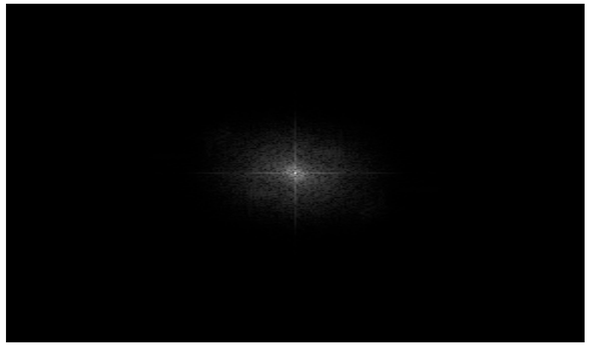


$\times$ 

$|F(s_x,s_y)|$

$|G(s_x,s_y)|$

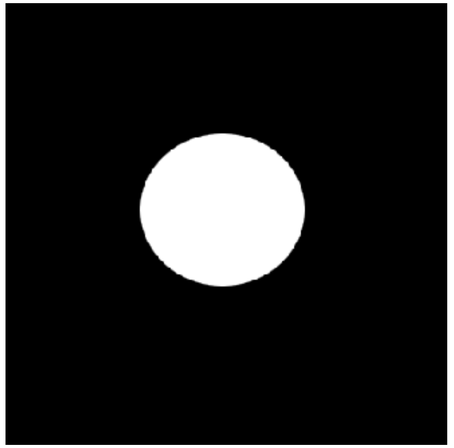
$\Downarrow$

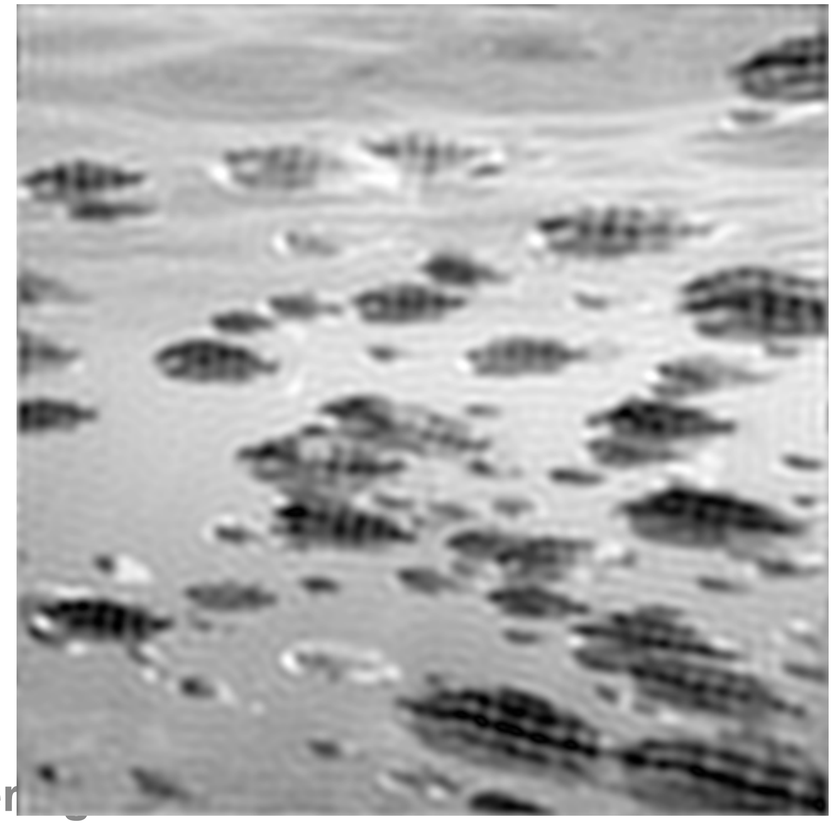


$|H(s_x,s_y)|$

Ideal Low-Pass Filter

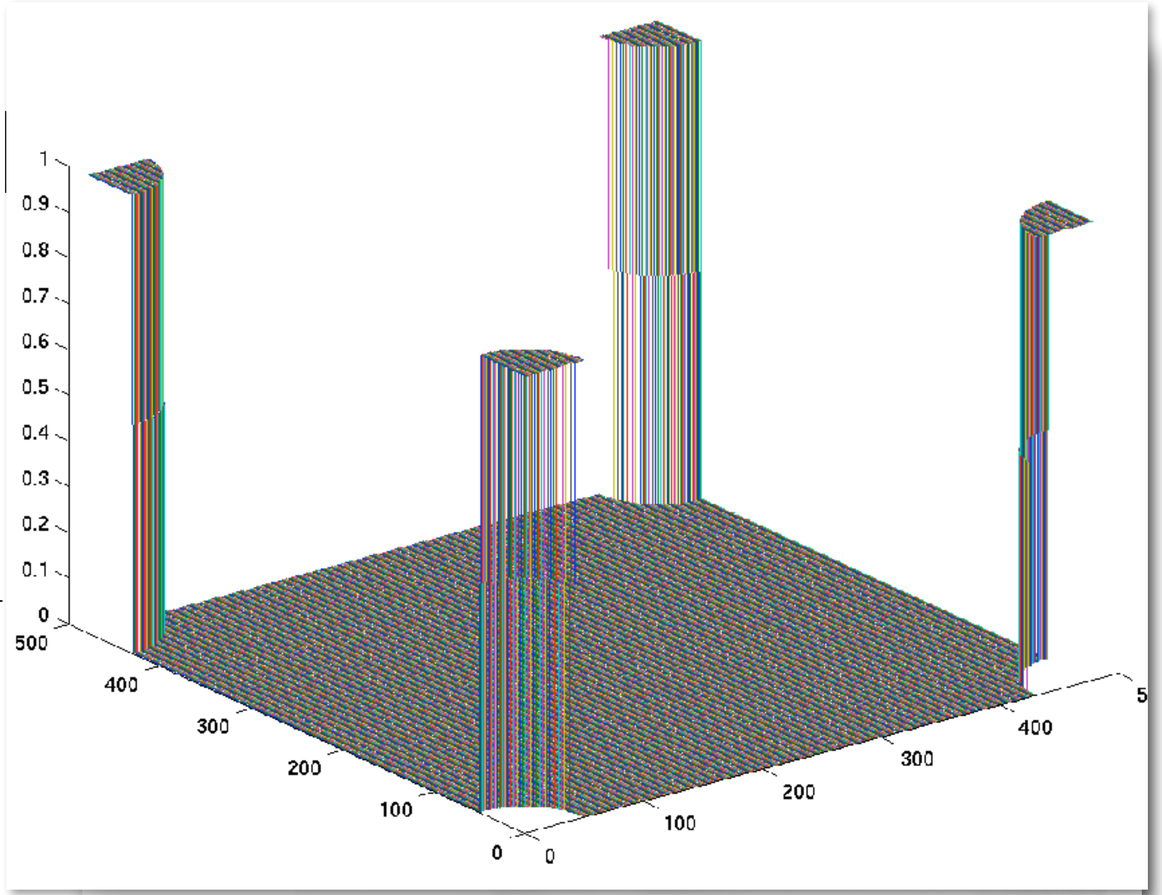
$$K(u,v) = \begin{cases} 0 & u^2 + v^2 > r^2 \\ 1 & \text{otherwise} \end{cases}$$

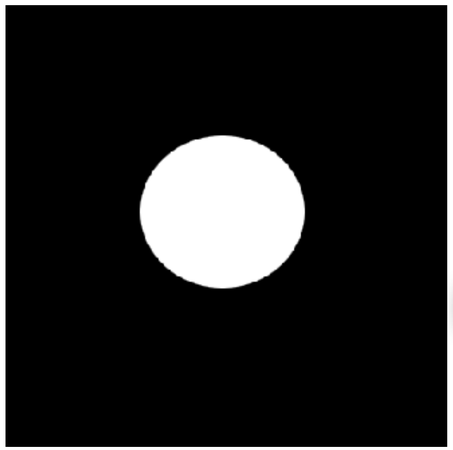


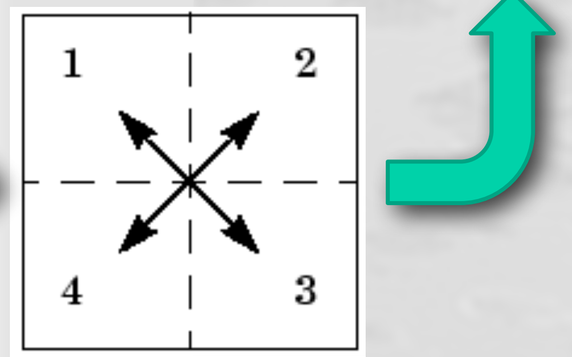


Ideal Low-Pass

$$K(u,v) = \begin{cases} 0 & u^2 + v^2 \\ 1 & \text{otherwi} \end{cases}$$

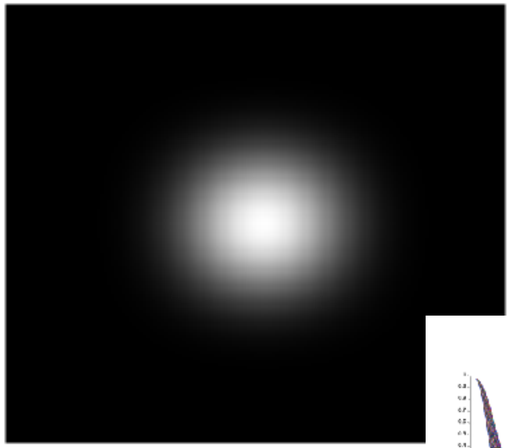


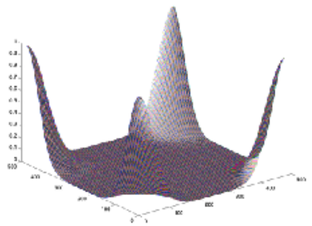


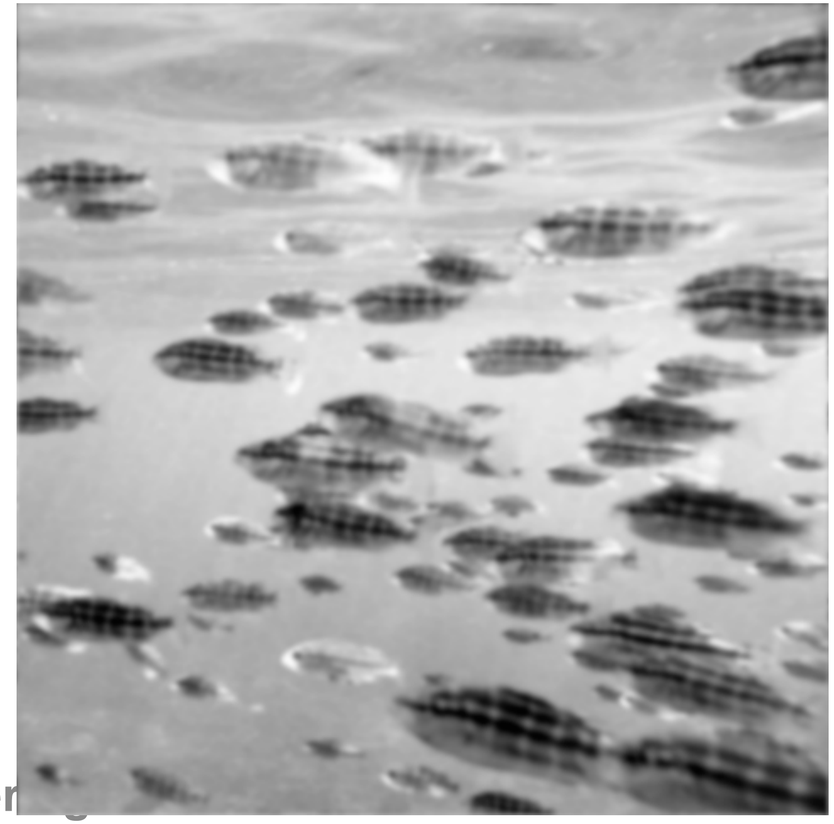


Gaussian Low-Pass Filter

$$K(u,v) = \exp\left(-\frac{u^2+v^2}{2\sigma^2}\right)$$

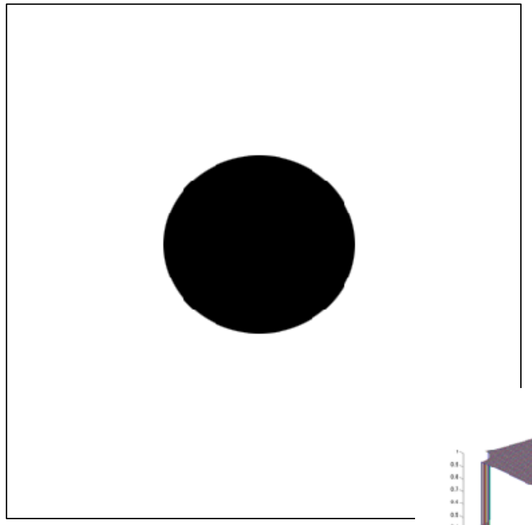


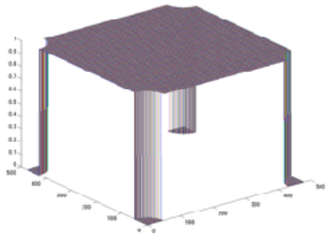


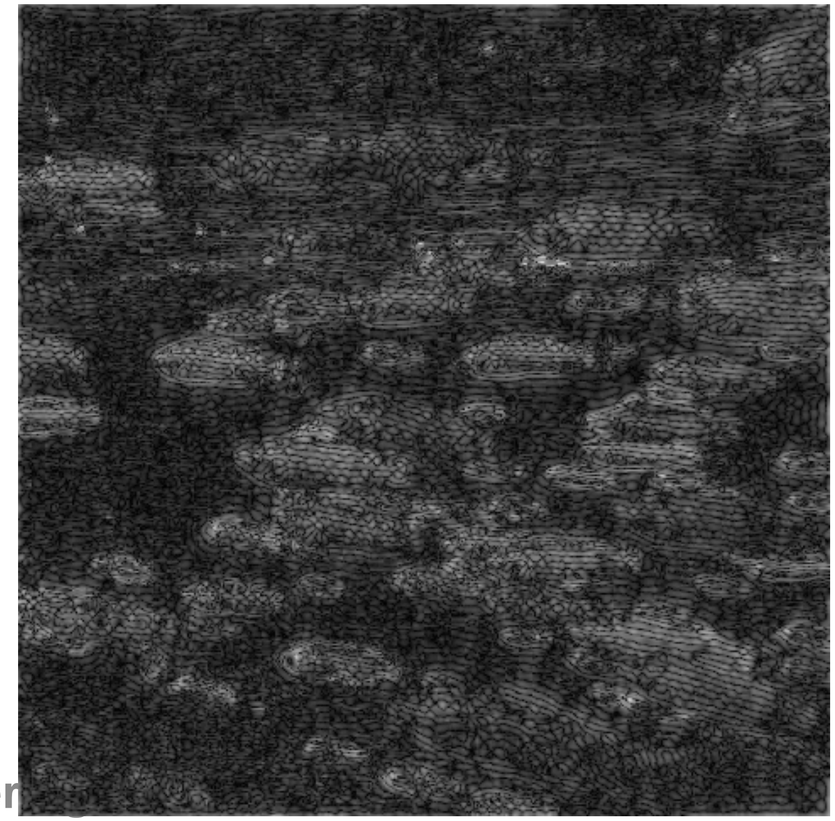


Ideal High-Pass Filter

$$K(u,v) = \begin{cases} 0 & u^2 + v^2 < r^2 \\ 1 & \text{otherwise} \end{cases}$$

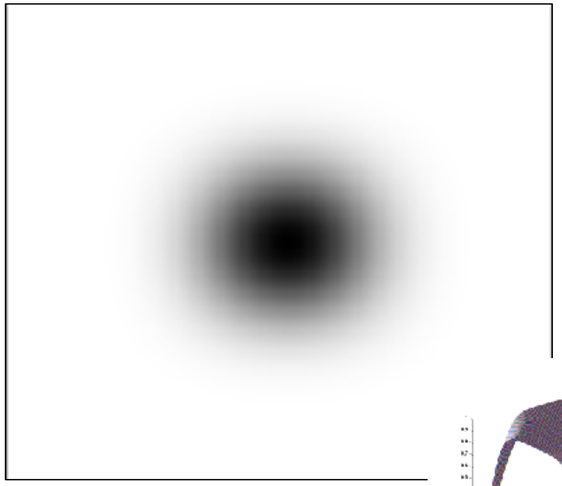


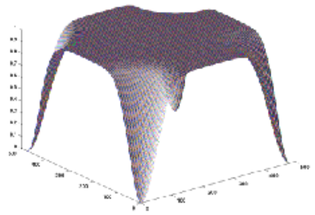


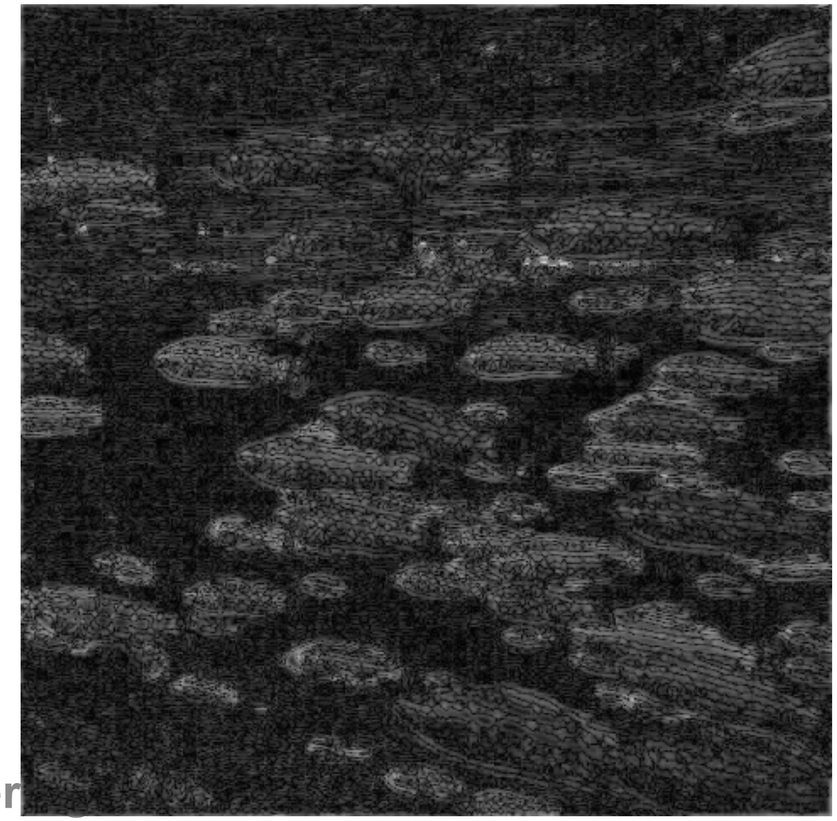


Gaussian High-Pass Filter

$$K(u,v) = 1 - \exp\left(-\frac{u^2+v^2}{2\sigma^2}\right)$$

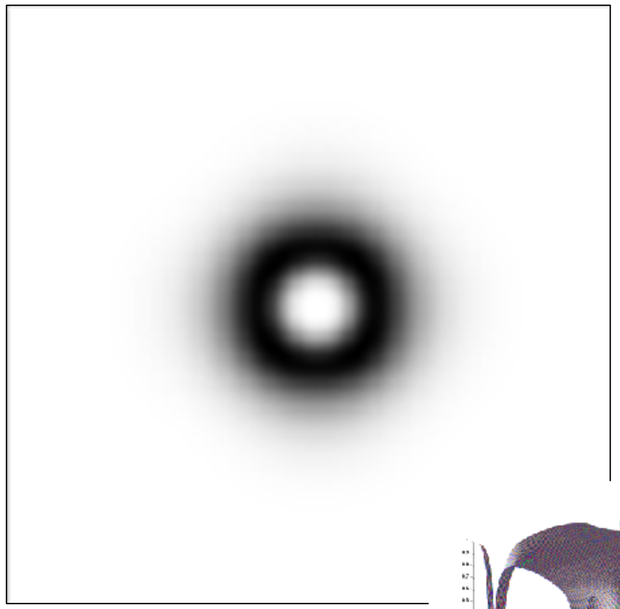


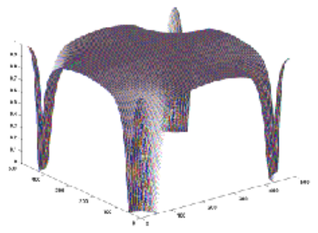


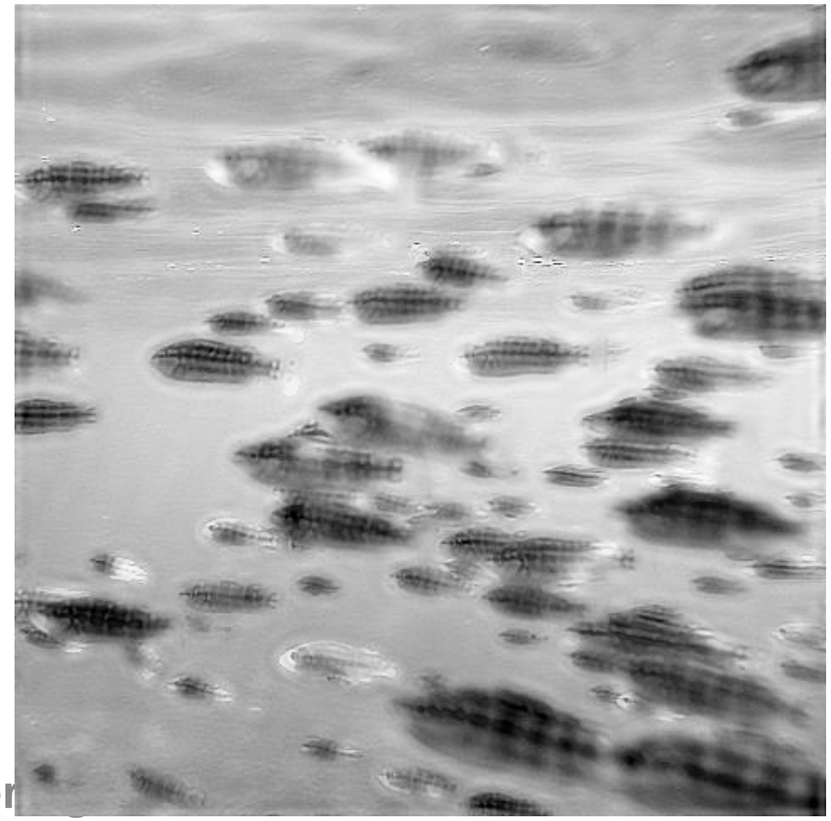


Band-Stop Filter

$$K(u,v) = \frac{1}{1 + (\Omega r / \sqrt{u^2 + v^2 - r_0^2})^{2n}}$$

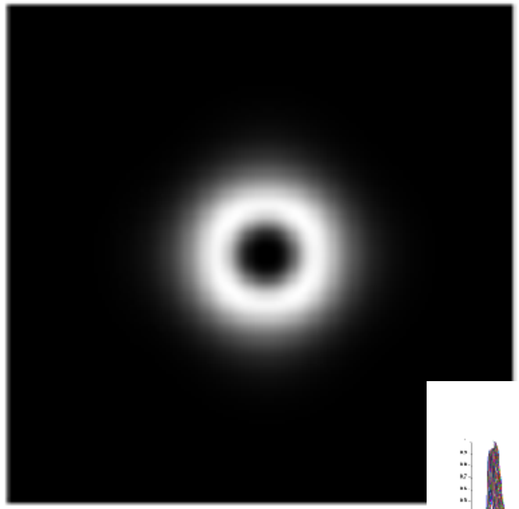


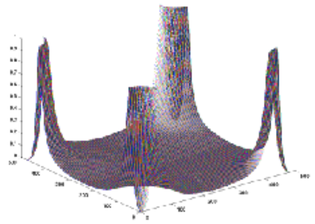


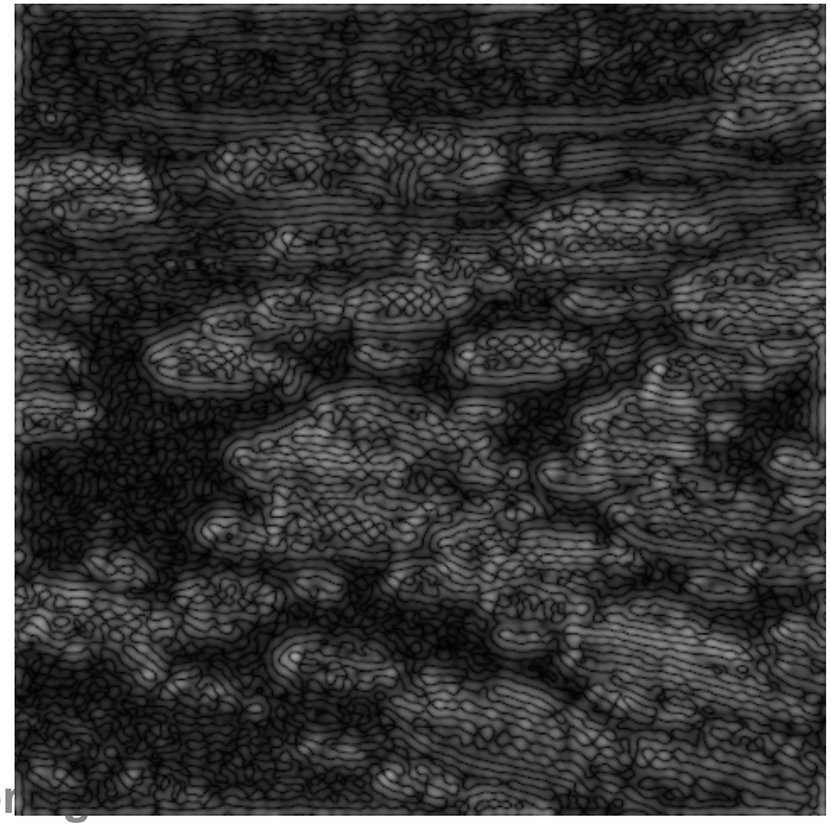


Band-Pass Filter

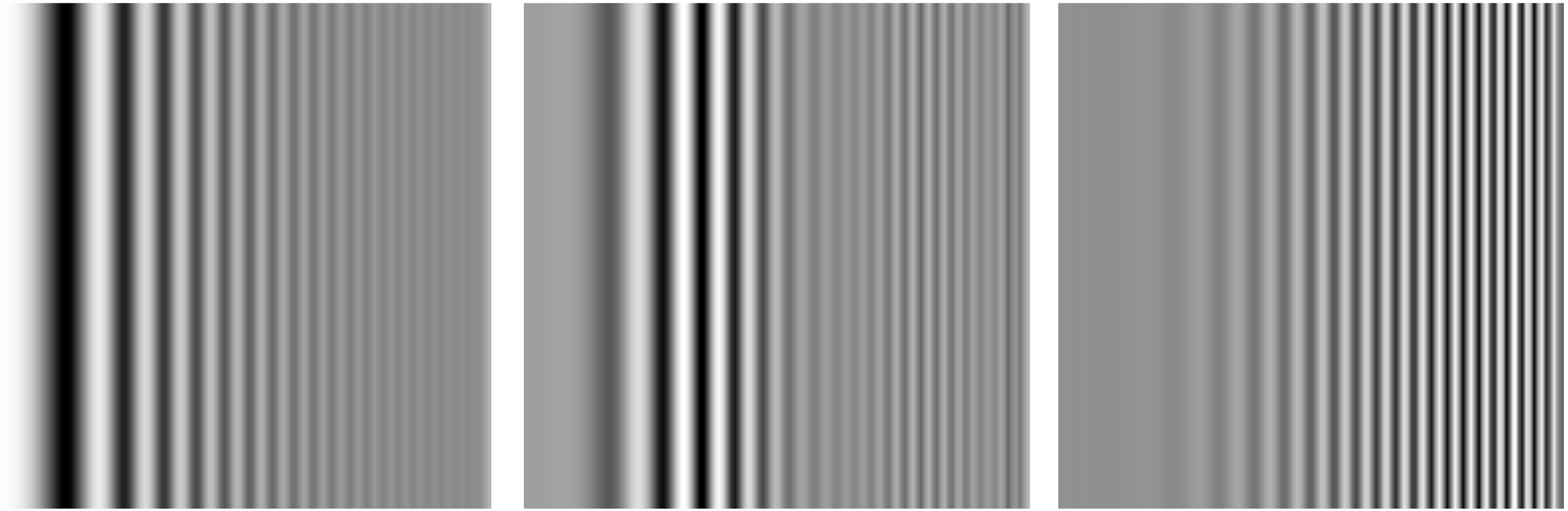
$$K(u,v) = 1 - \frac{1}{1 + (\Omega r / \sqrt{u^2 + v^2 - r_0^2})^{2n}}$$







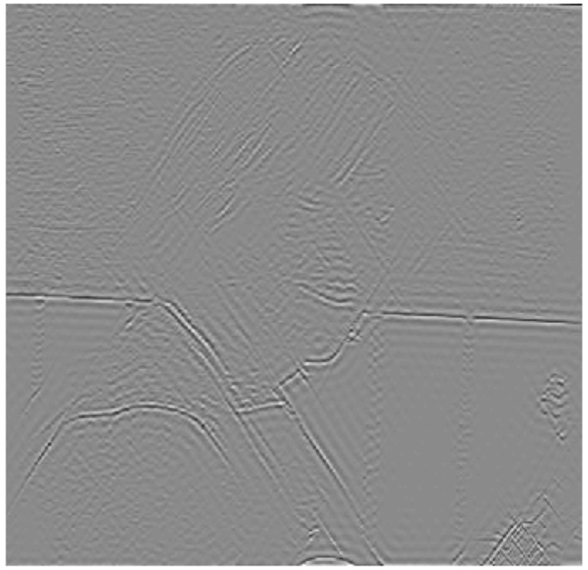
X-pass Filter

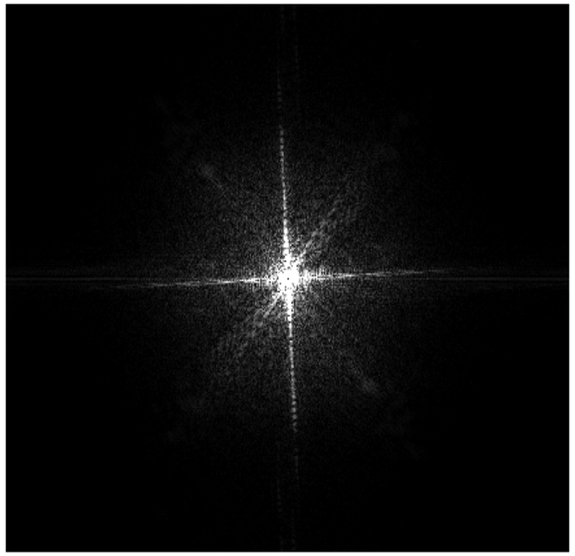


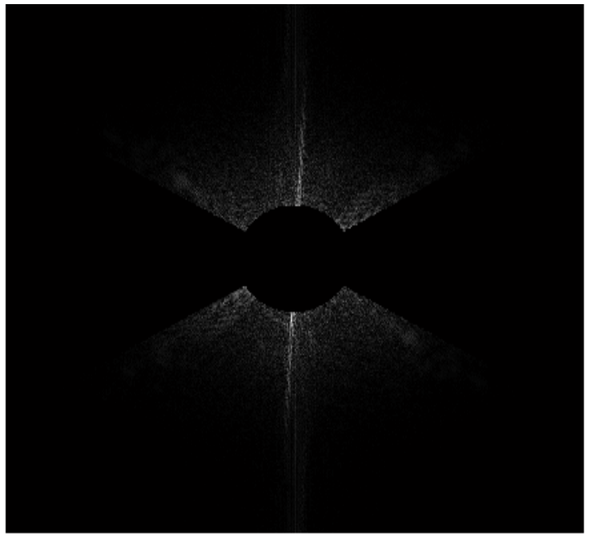
(a) Lowpass filter **(b)** Bandpass filter **(c)** Highpass filter

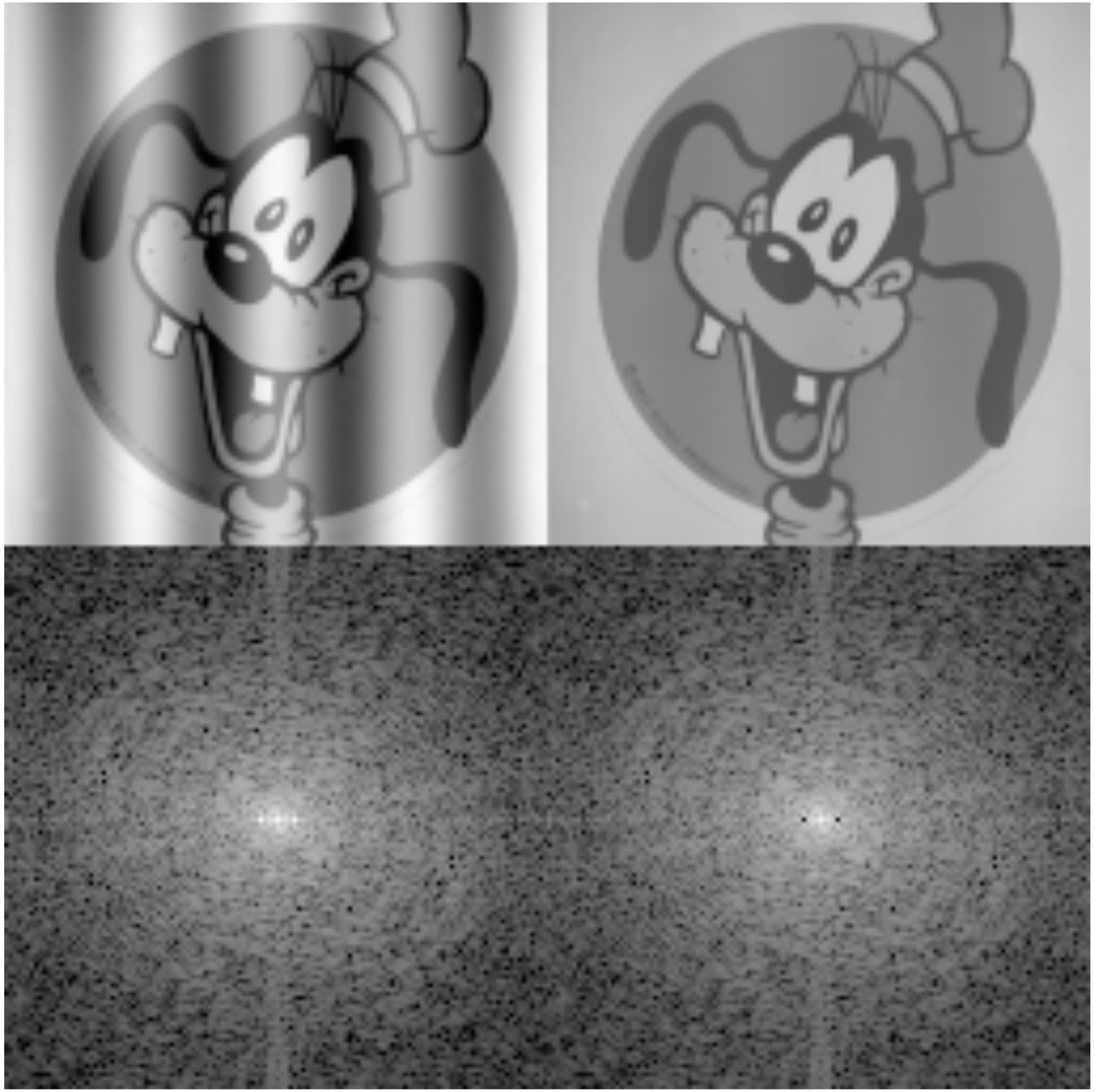
From Young, Gerbrands and van Vliet;
CVOnline.

Orientation Filtering









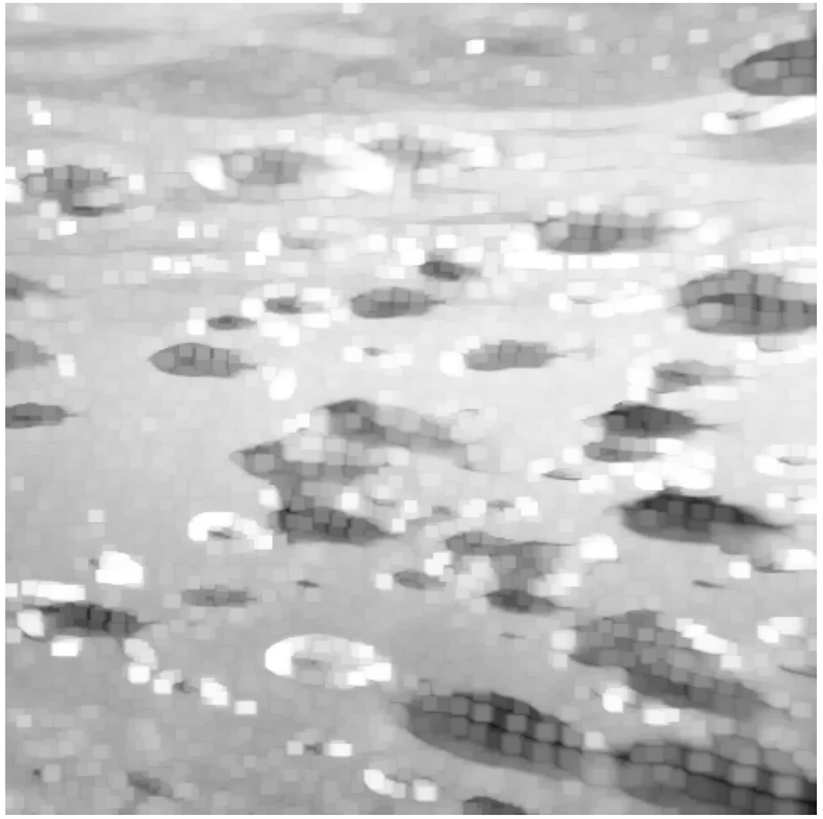
Removing Unwanted Periodic Component

Rank Filters

- **Simple non-linear filters based on the ordering of the grey levels in the neighbourhood.**

Max Filter

$$I'(m,n) = \max\{I(i,j) : (i,j) \in N(m,n)\}$$



Min Filter

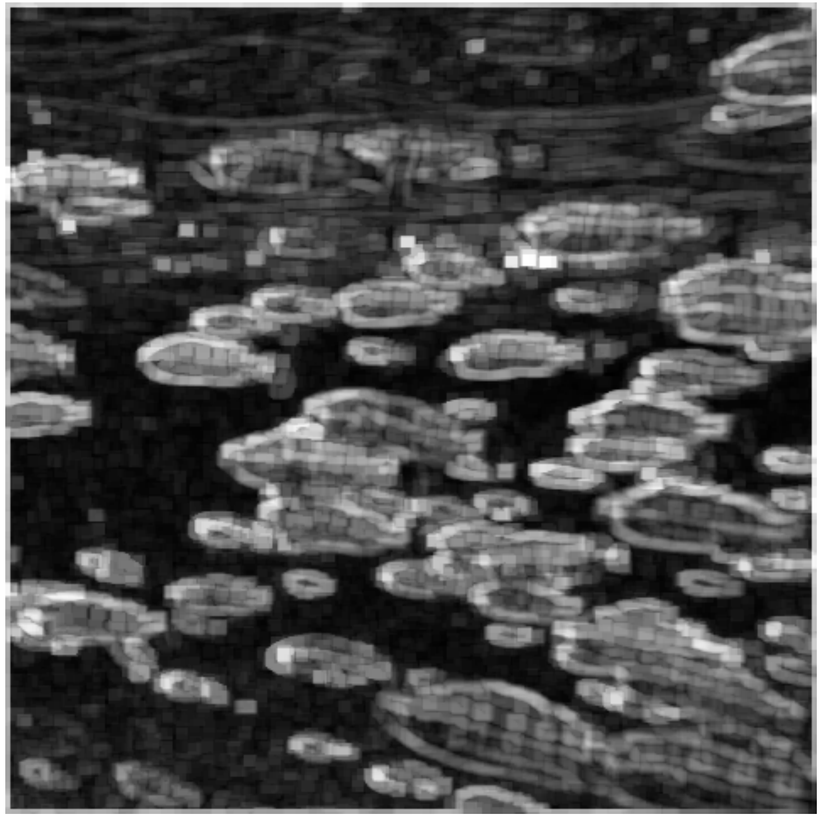
$$I'(m,n) = \min\{I(i,j) : (i,j) \in N(m,n)\}$$

Median Filter

$$I'(m,n) = \text{median}\{I(i,j) : (i,j) \in N(m,n)\}$$

Range Filter

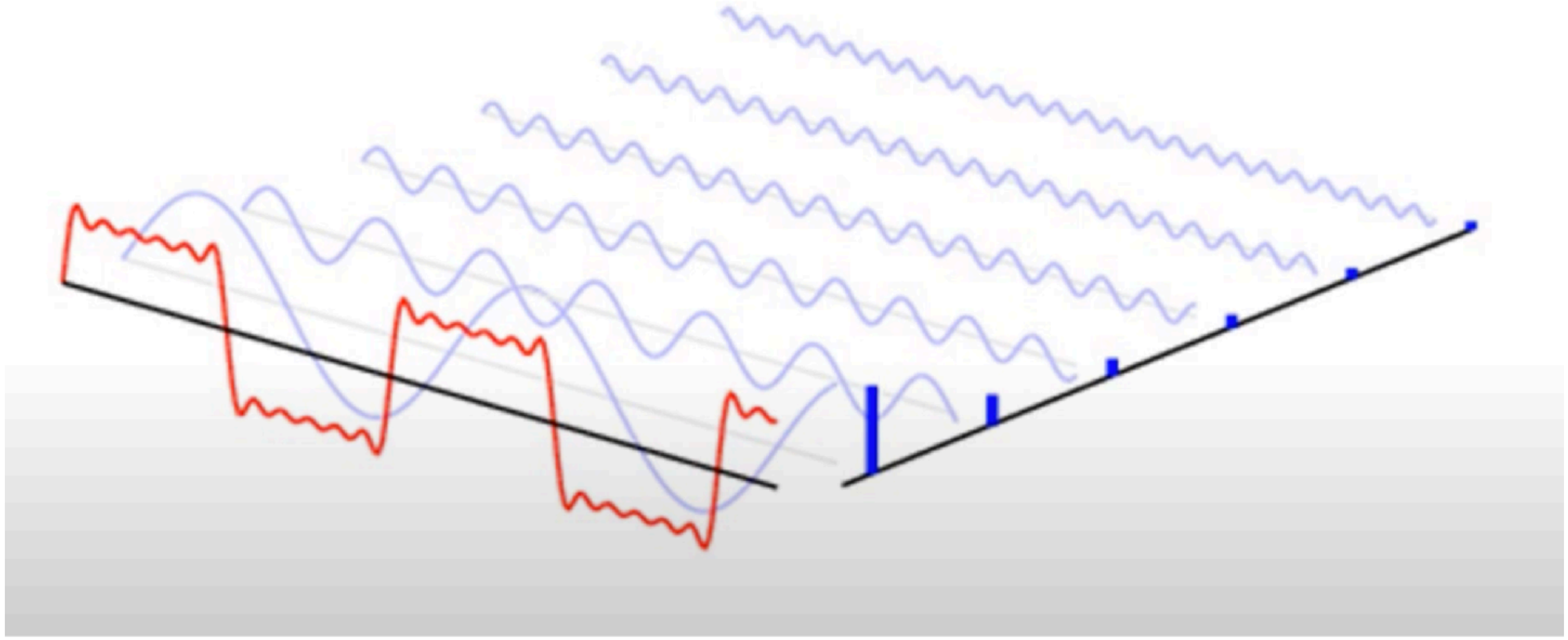
$$I'(m,n) = \max\{I(i,j) : (i,j) \in N(m,n)\} - \min\{I(i,j) : (i,j) \in N(m,n)\}$$

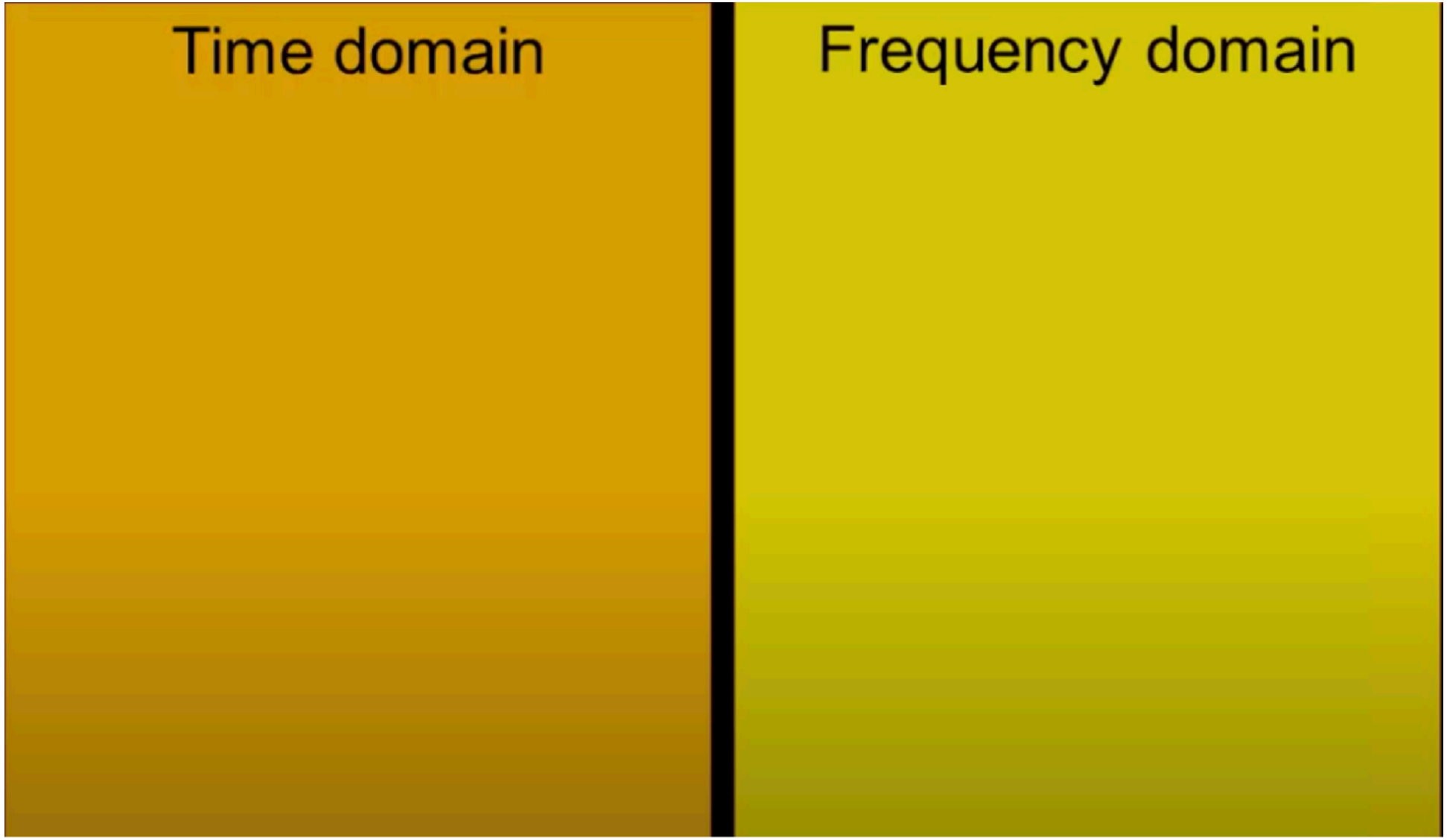


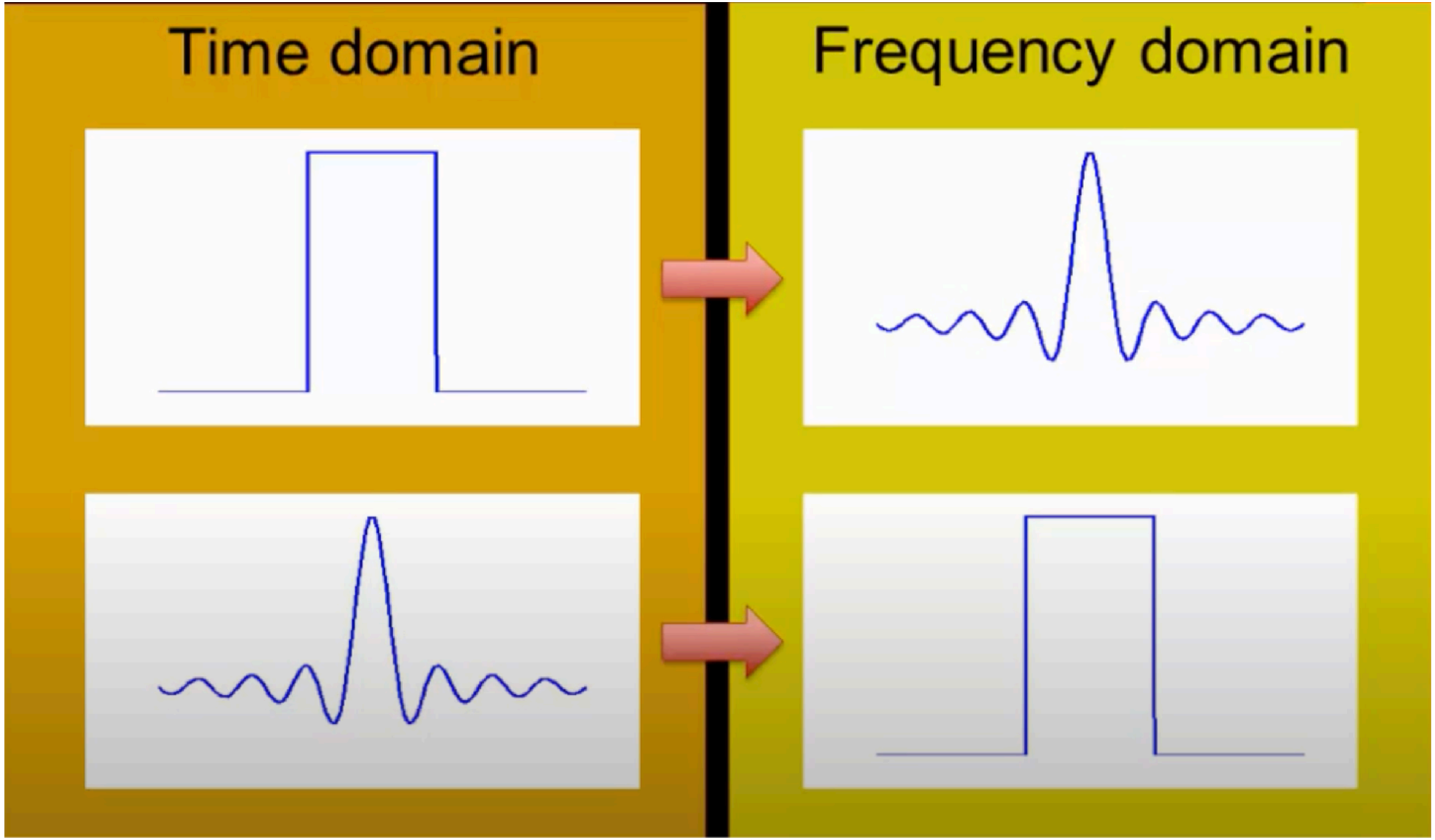
Summary

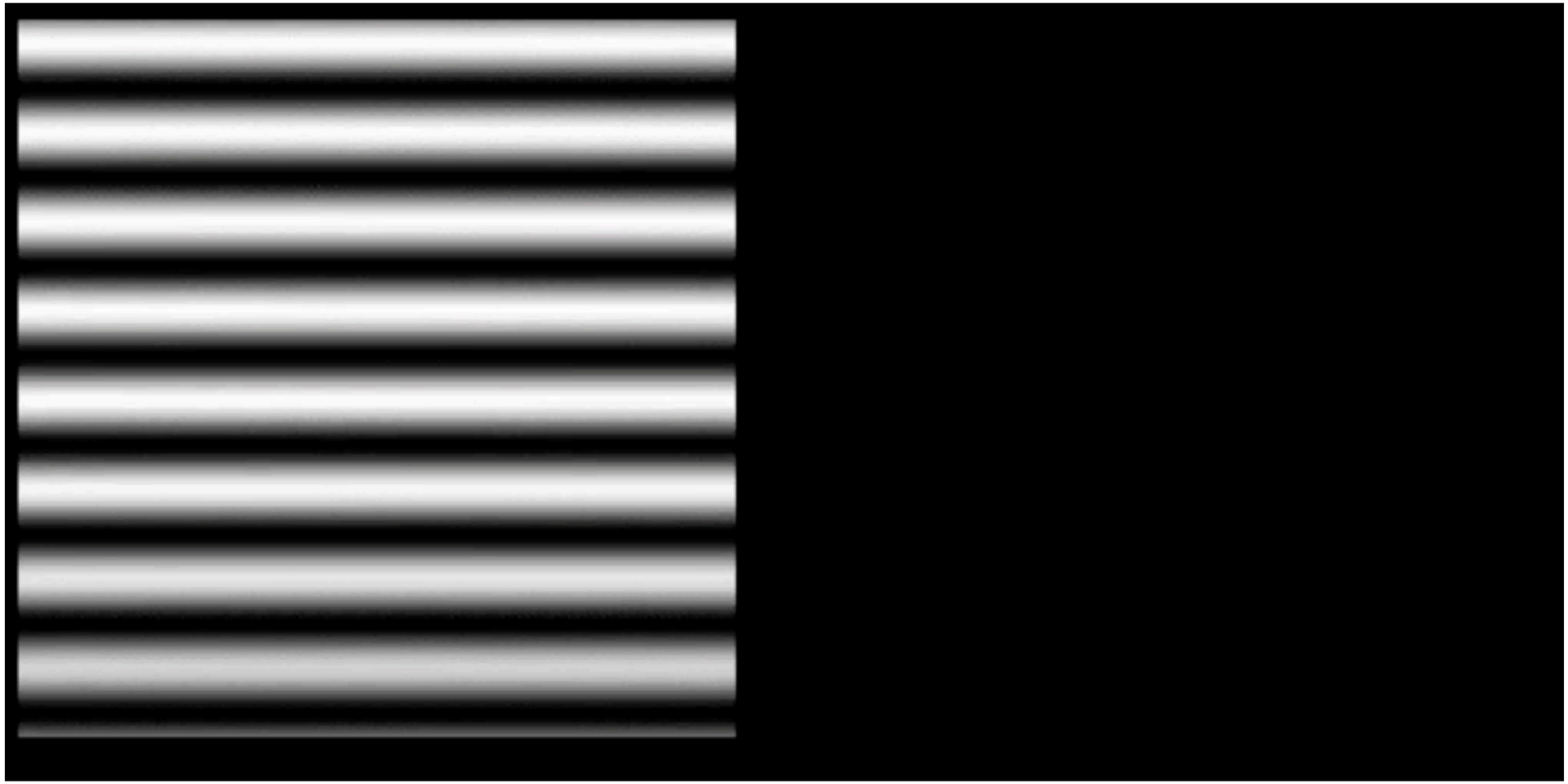
- Linear filtering
- Convolution in the image domain
  - Separable filters
  - Smoothing filters and Gaussian filters
  - Differential filters
- Convolution in the Fourier domain
- Rank filters

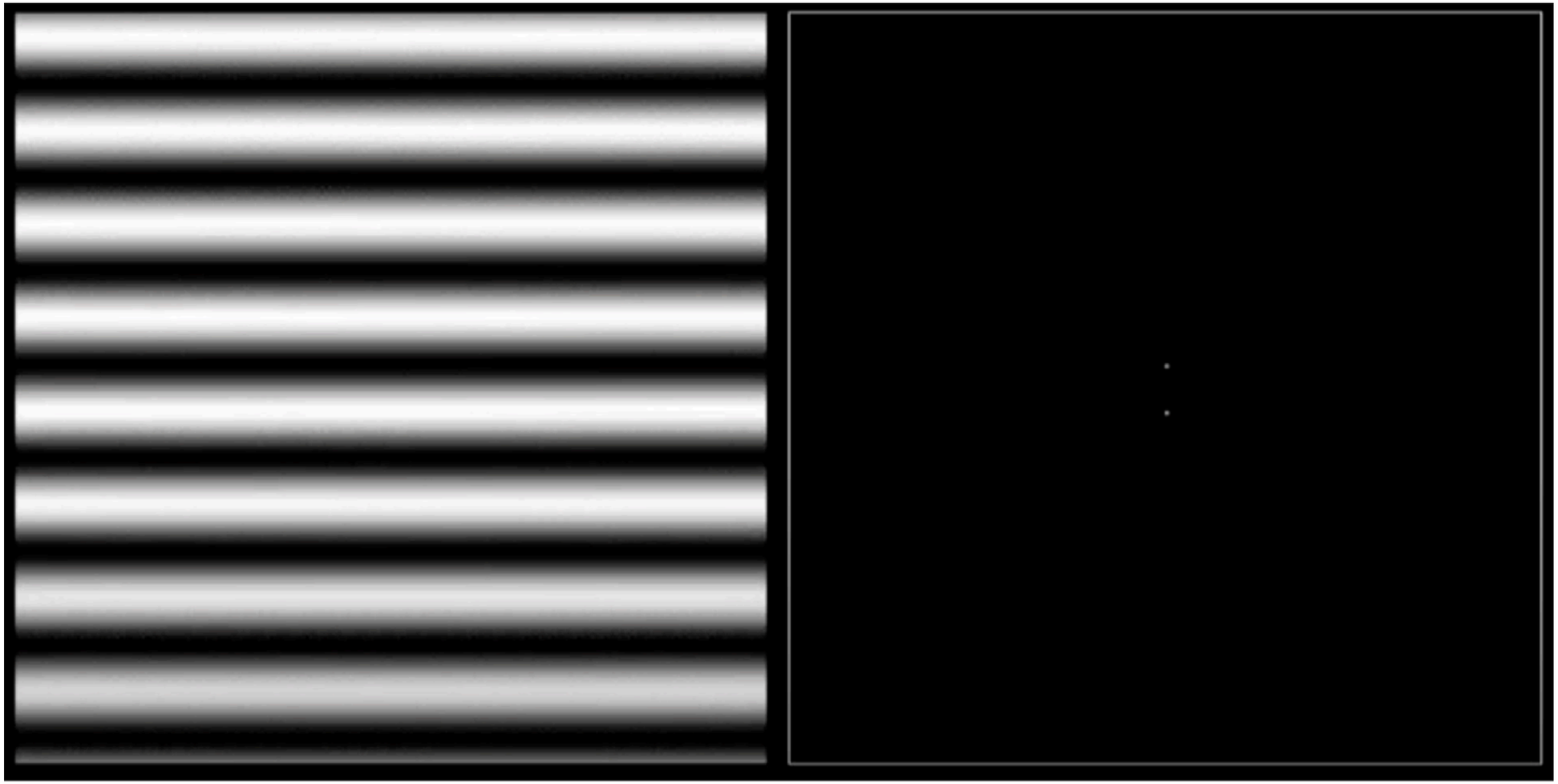
Fourier transform

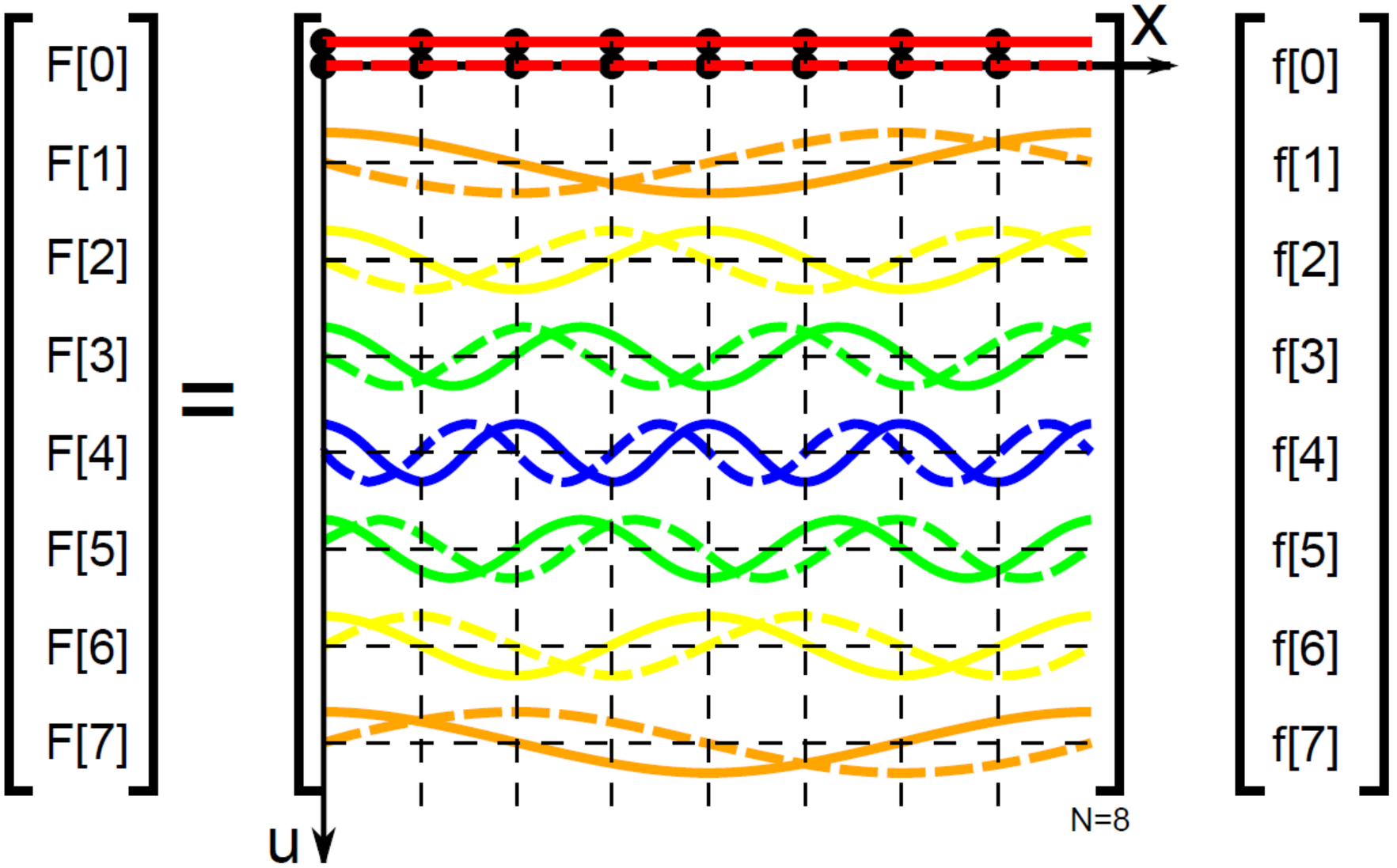



Time domain
Frequency domain

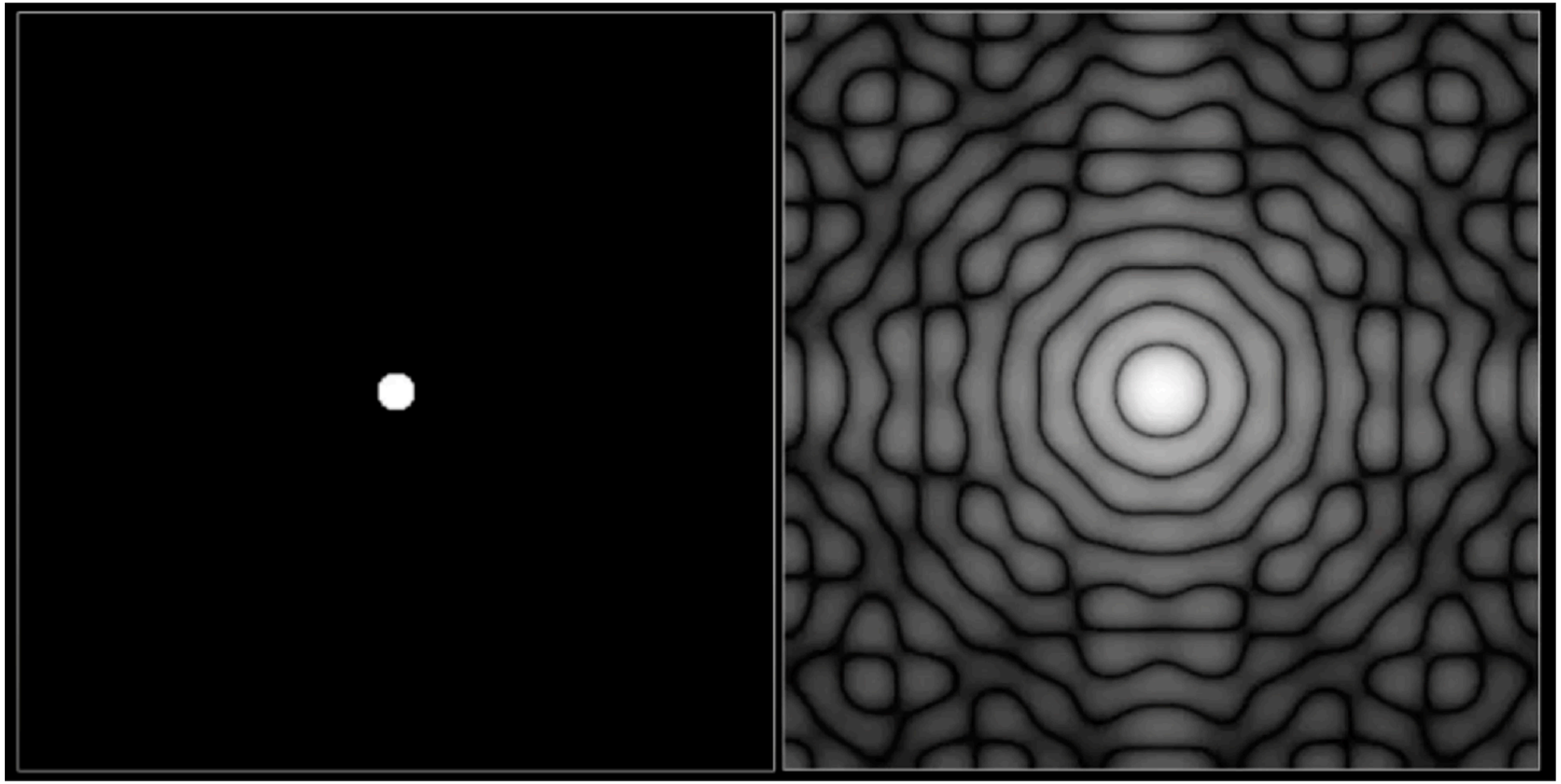

Time domain
Frequency domain

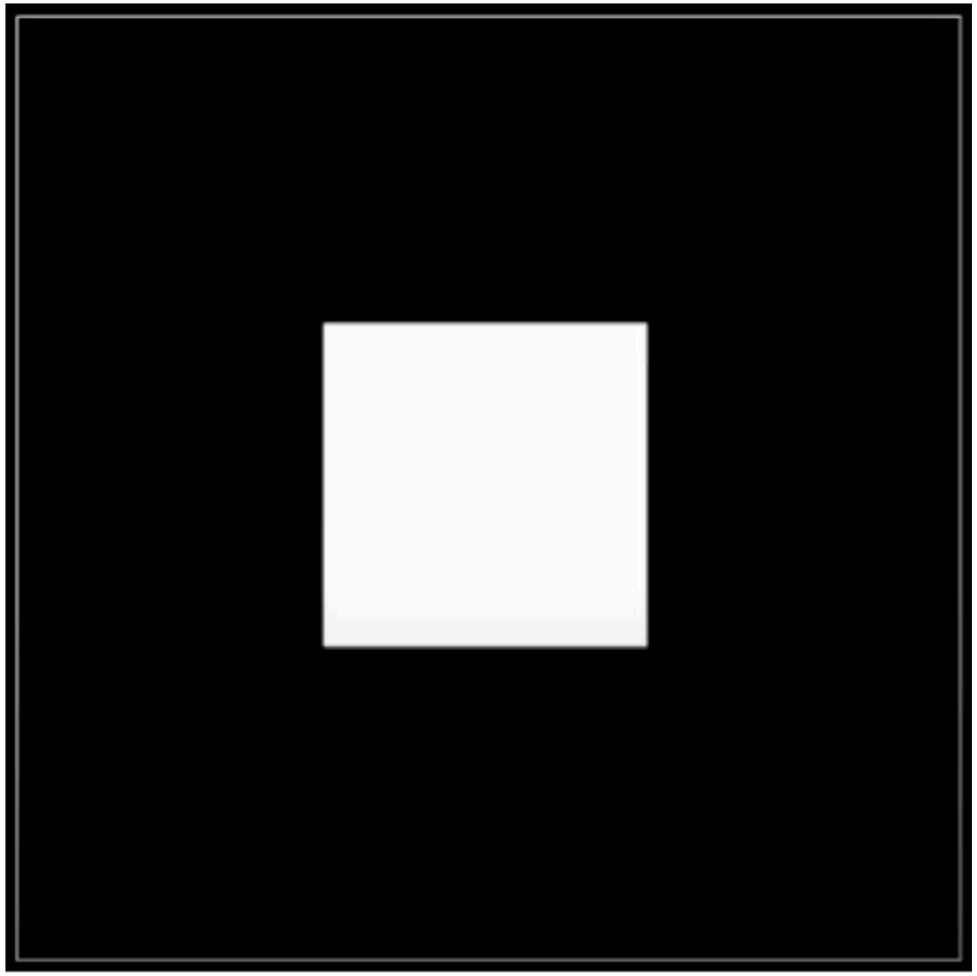


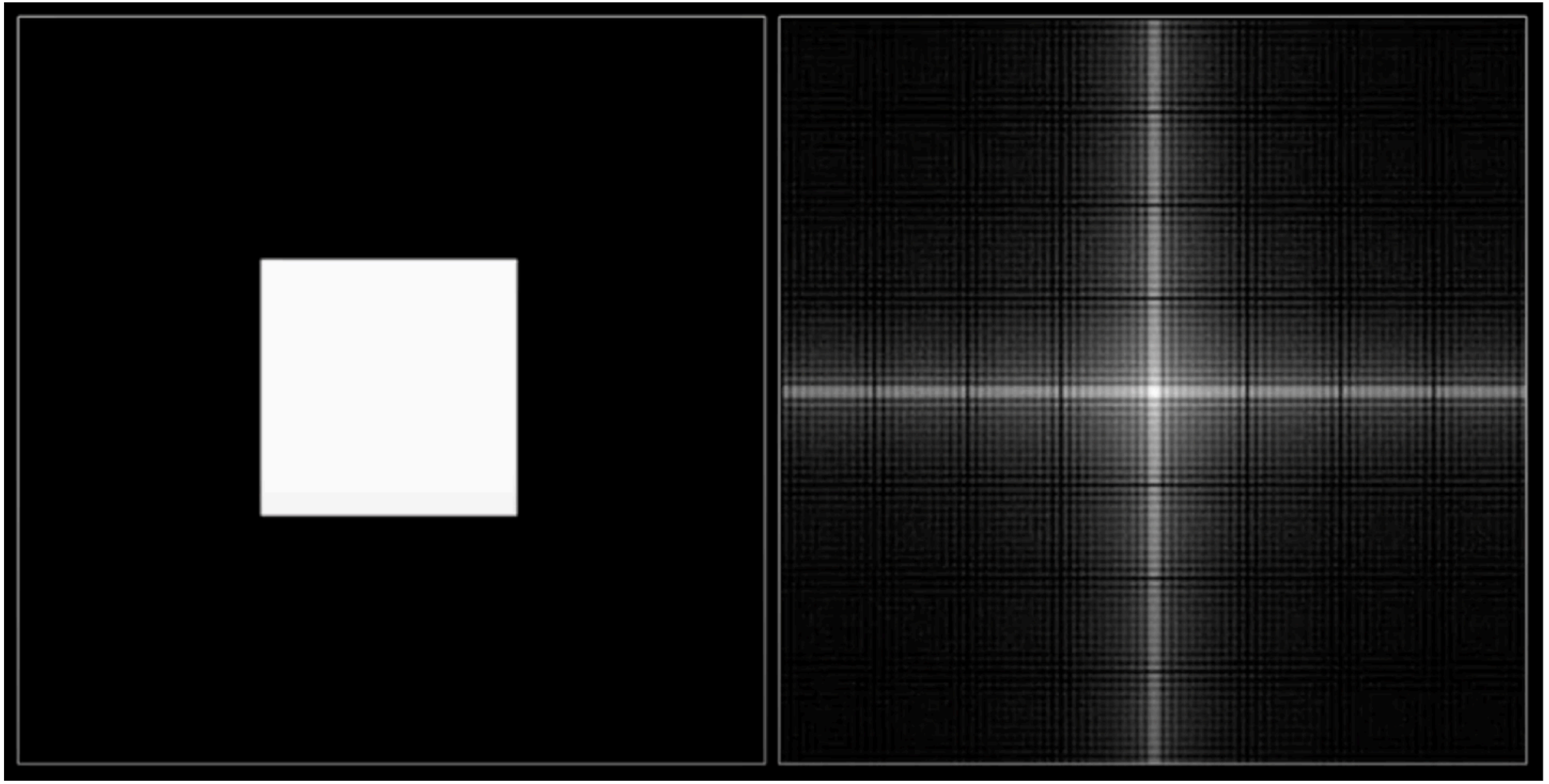




Can think of this as a change of basis: we move from space in which signal is represented as coefficients of delta functions to frequency where signal is represented as coefficients of sinusoidal basis functions







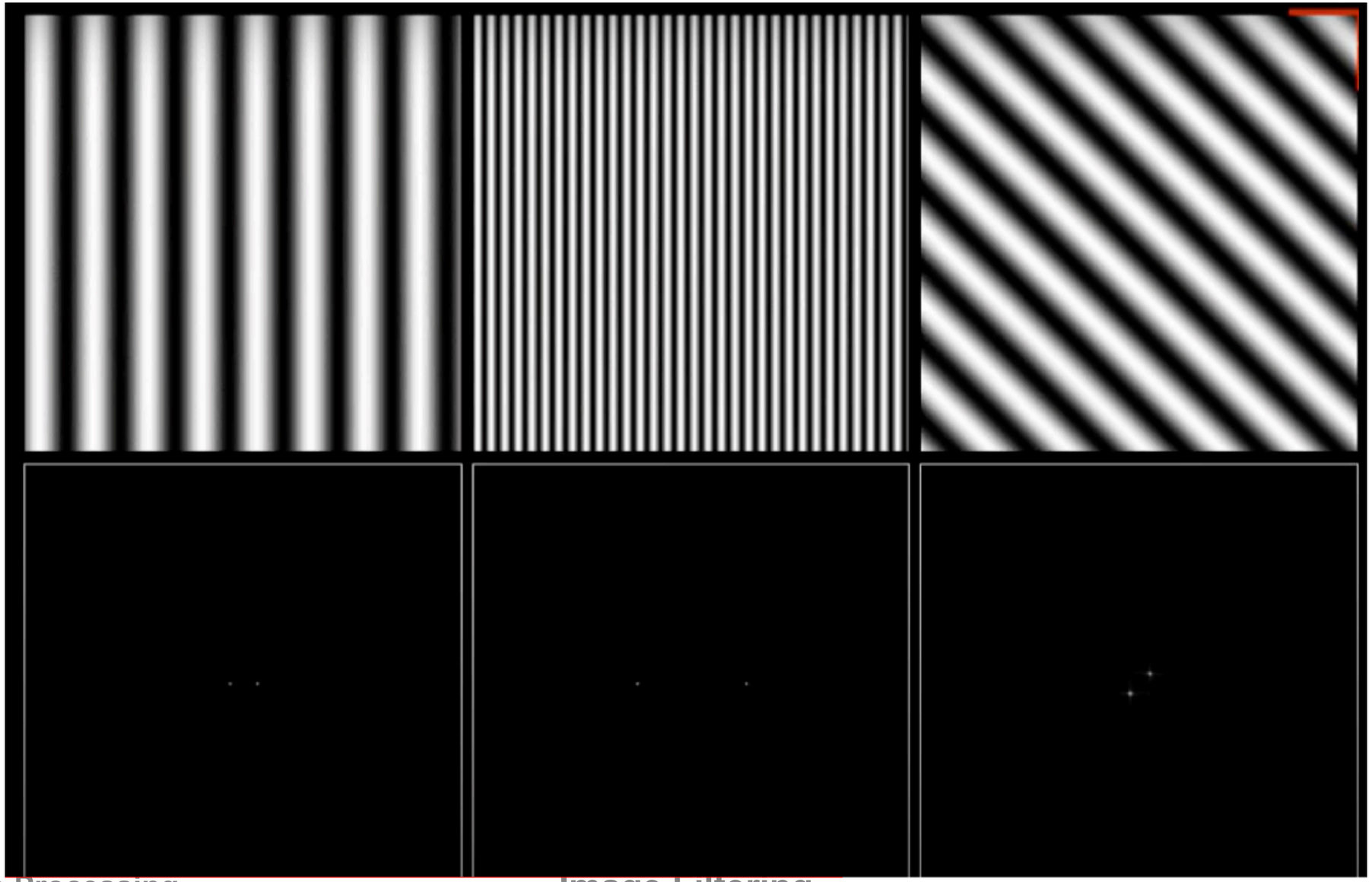
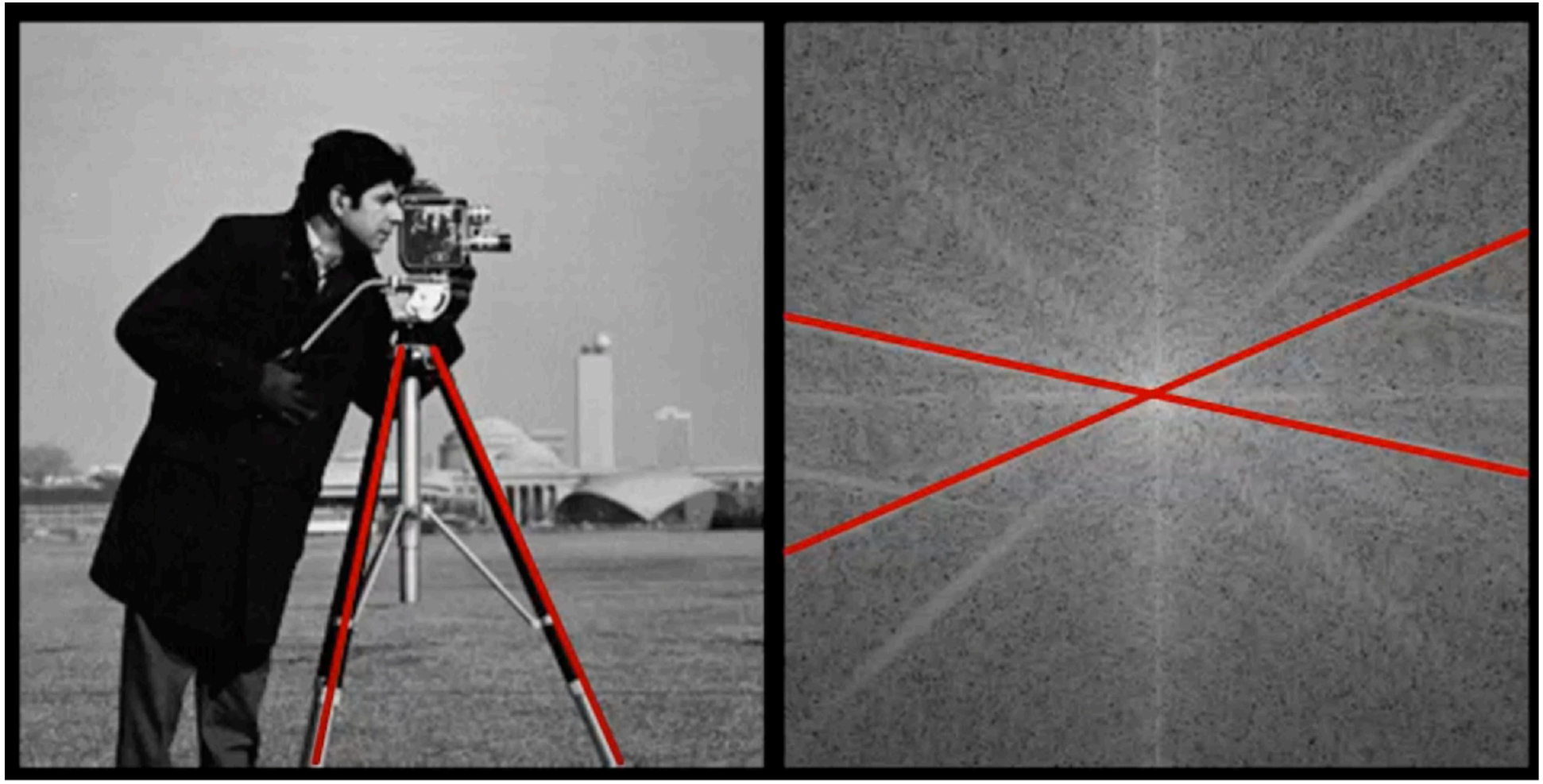
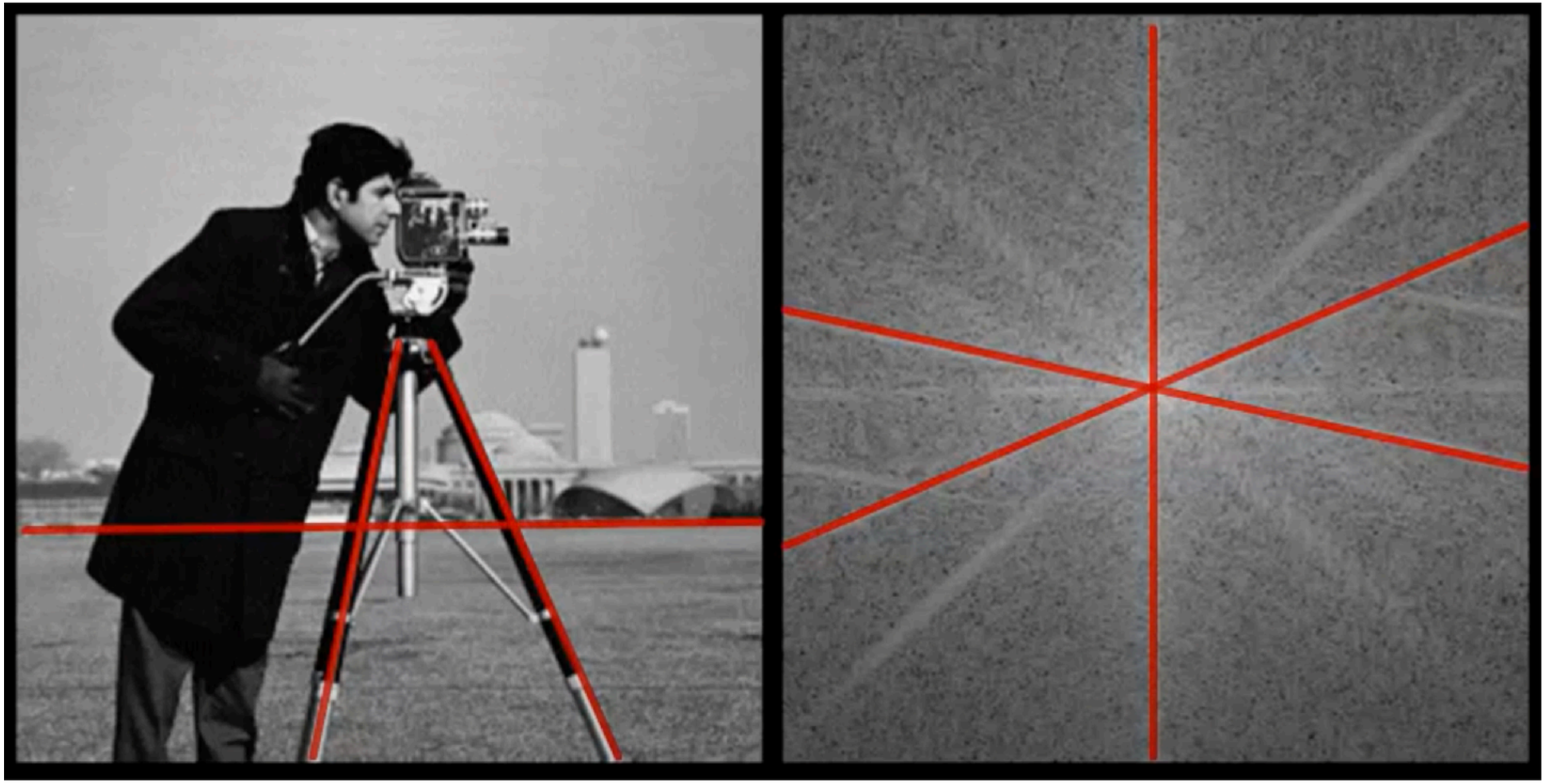
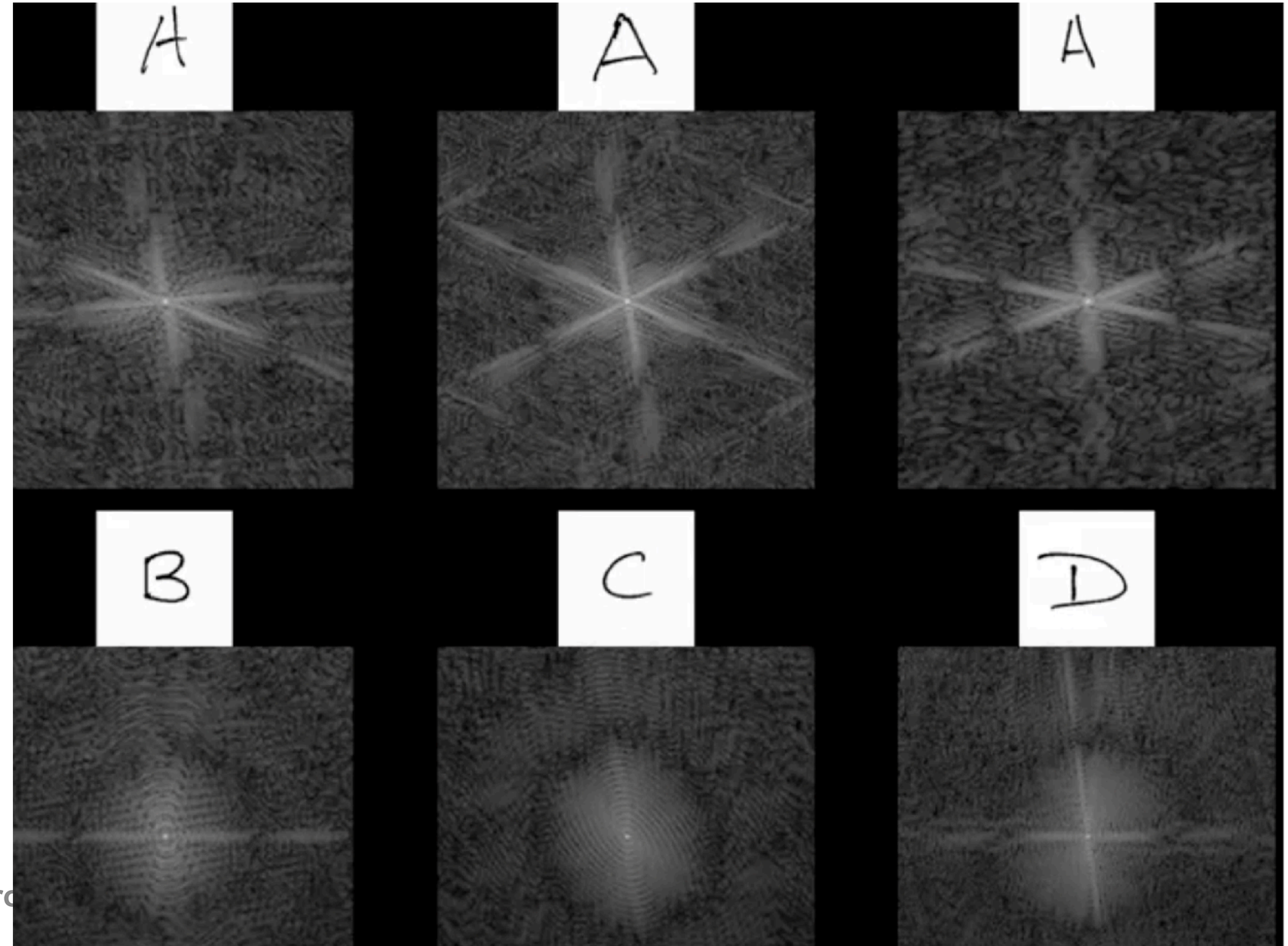


Image: ‘cameraman’

Copyright owned by MIT. Used with permission.






A
A
A
B
C
D