universität innsbruck

Lecture 3. Image processing

703142. Computer Vision

Assoz.Prof. Antonio Rodríguez-Sánchez, PhD.

Image processing

- Introduction
- Point operators
- Linear filtering
- Non linear filtering
- The Fourier transform
- Pyramids and scales

Introduction

- Many times in Computer Vision it is useful to preprocess an image
- Examples include
  - Reduction of noise.
  - Increasing sharpness.
  - Color balancing.
  - Straightening.
  - Rotating …

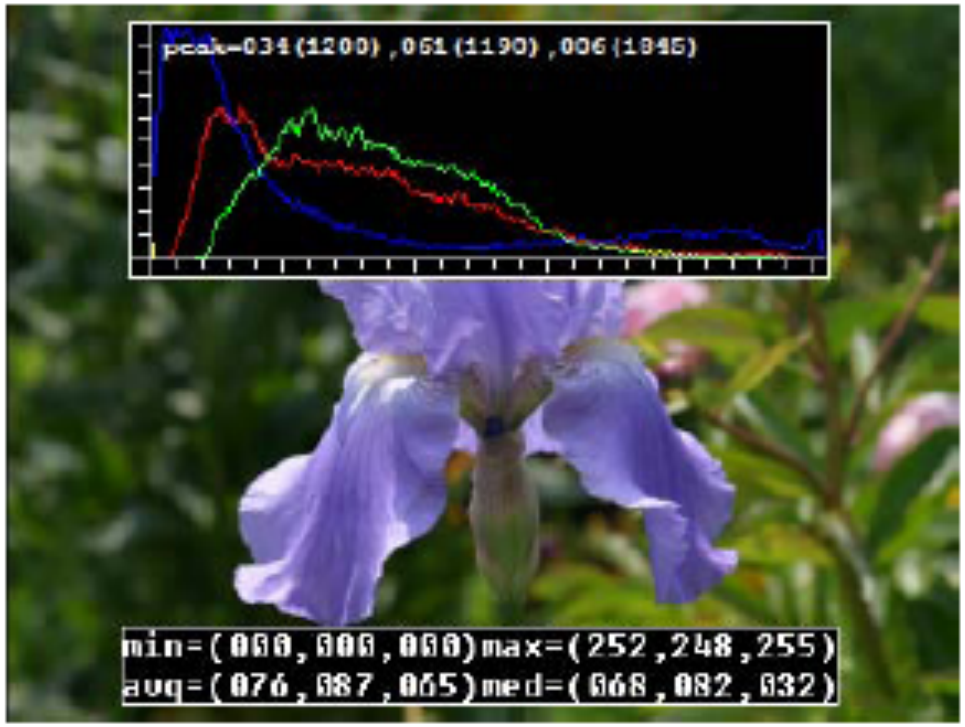
Image processing

- Introduction
- Point operators
- Linear filtering
- Non linear filtering
- The Fourier transform
- Pyramids and scales

Point operators

- Each output pixel depends on only the corresponding input pixel plus some globally corrected information or parameters
  - Brightness adjustments.
  - Contrast adjustments.
  - Color corrections.
  - Color transformations.
  - Image addition.

Point operators



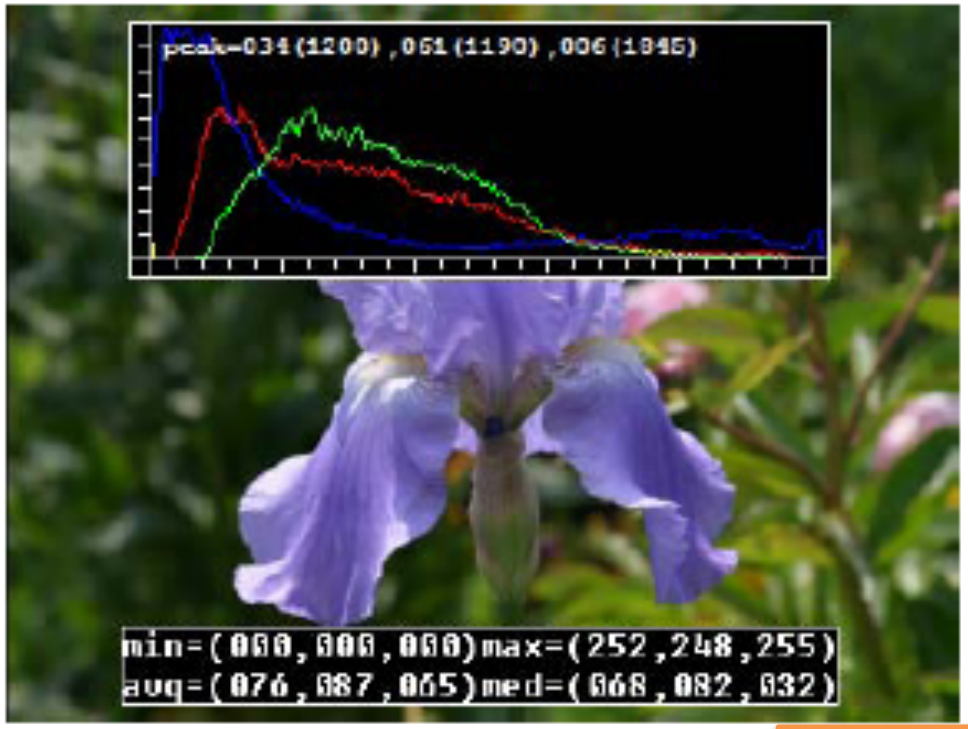
- Operator
  - Take one or more input images and produce an output image

$$g(x)=h(f(x)),\ or$$
$$g(x)=h(f_0(x),\ldots,f_n(x))$$
$$x=(i,j)$$

Point operators

- Operator
  - Take one or more input images and produce an output image

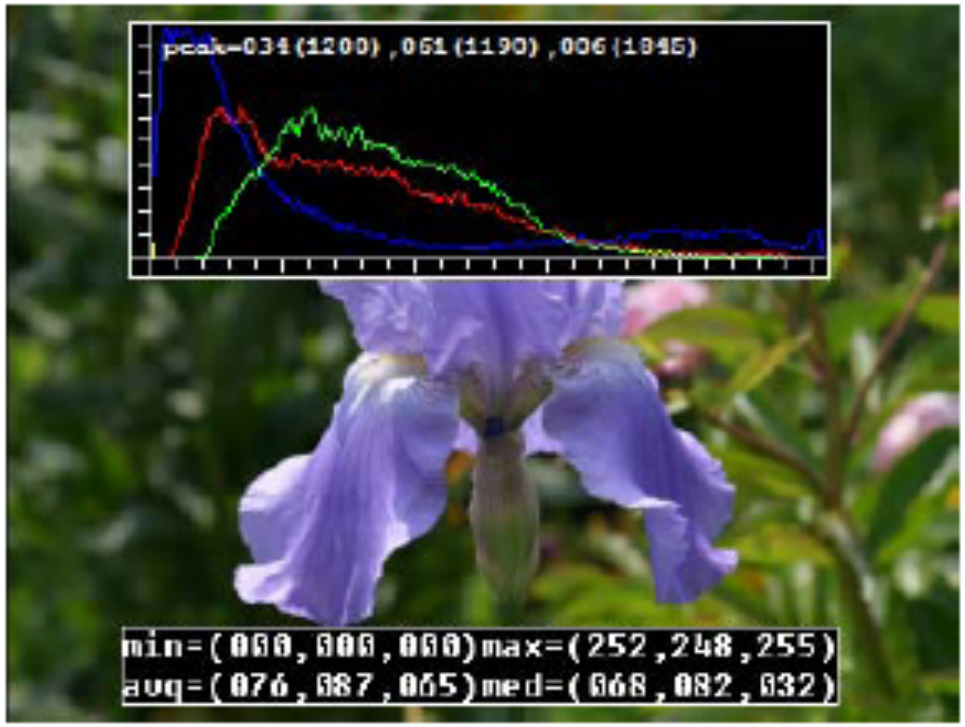
$$g(x)=h(f(x)),\ or$$
$$g(x)=h(f_0(x),\ldots,f_n(x))$$
$$x=(i,j)$$



$$g(x)=af(x)+b$$
$$g(x)=[f(x)]^{1/\gamma}$$
$$g(x)=(1-\alpha)f_0(x)+\alpha f_1(x)$$

Point operators

- Operator
  - Take one or more input images and produce an output image

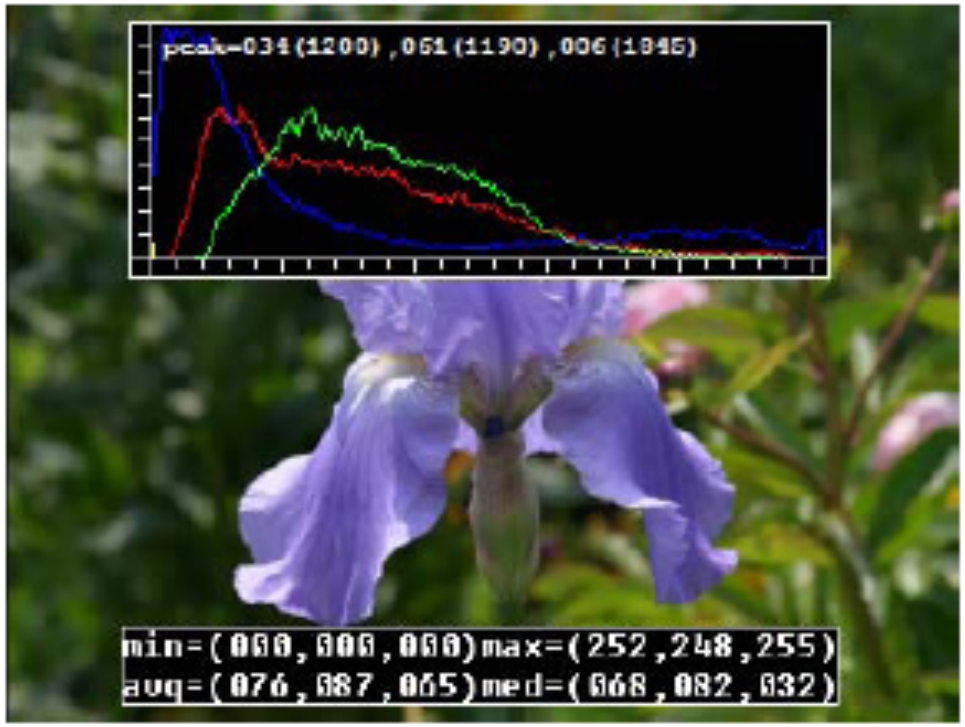


$$g(x)=h(f(x)),\ or$$
$$g(x)=h(f_0(x),\ldots,f_n(x))$$
$$x=(i,j)$$

$$g(x)=af(x)+b$$
$$g(x)=[f(x)]^{1/\gamma}$$
$$g(x)=(1-\alpha)f_0(x)+\alpha f_1(x)$$

Point operators

- Operator
  - Take one or more input images and produce an output image

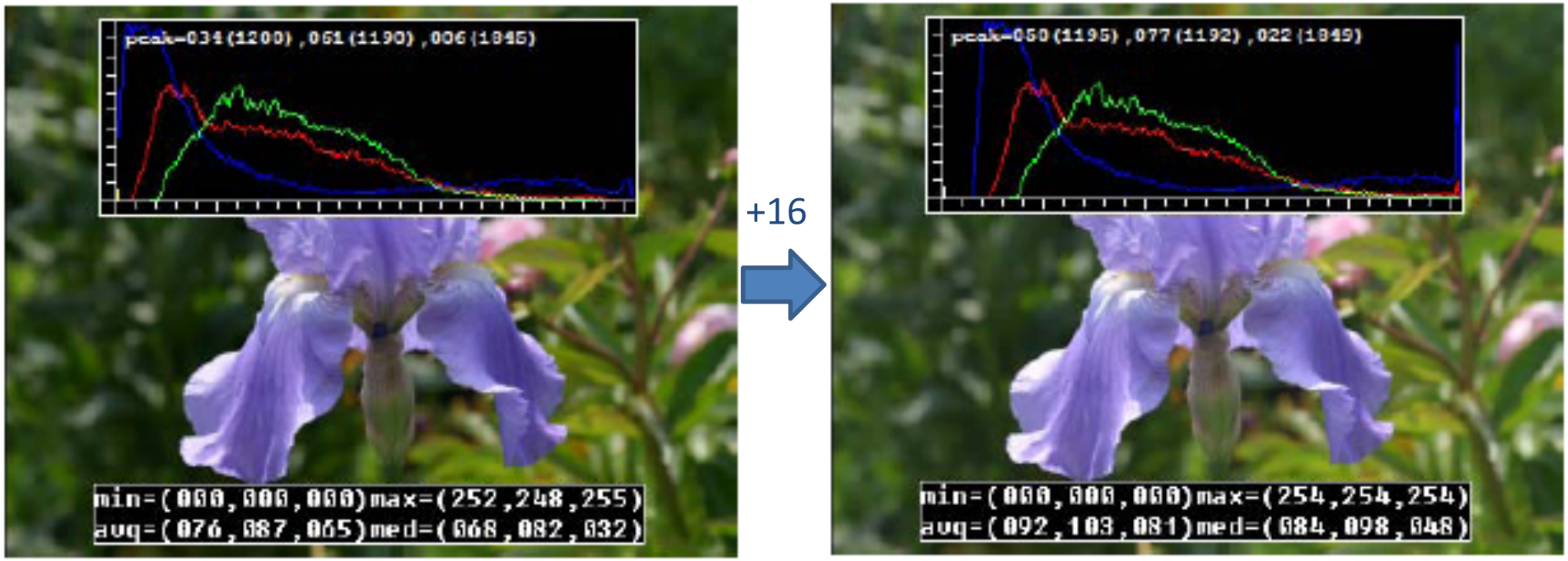


$$g(x)=h(f(x)),\ or$$
$$g(x)=h(f_0(x),\ldots,f_n(x))$$
$$x=(i,j)$$

$$g(x)=af(x)+b$$
$$g(x)=[f(x)]^{1/\gamma}$$
$$g(x)=(1-\alpha)f_0(x)+\alpha f_1(x)$$

Point operators

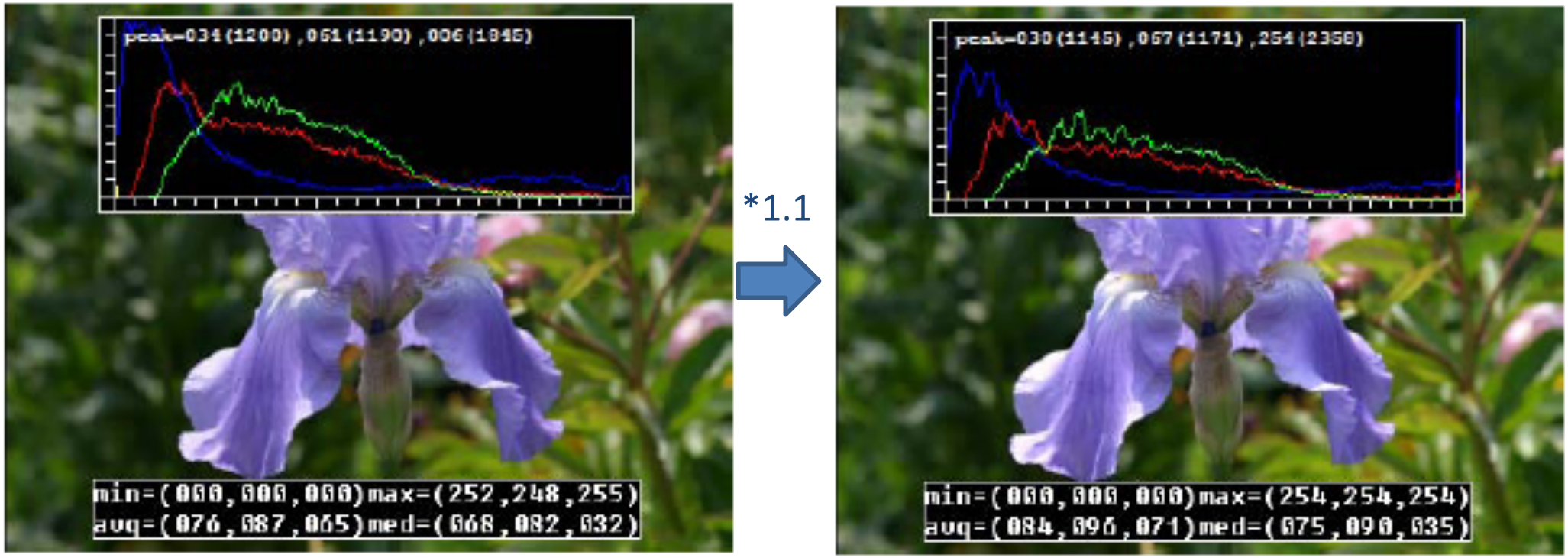
Increased brightness



$$g(x)=f(x)+16$$

Point operators

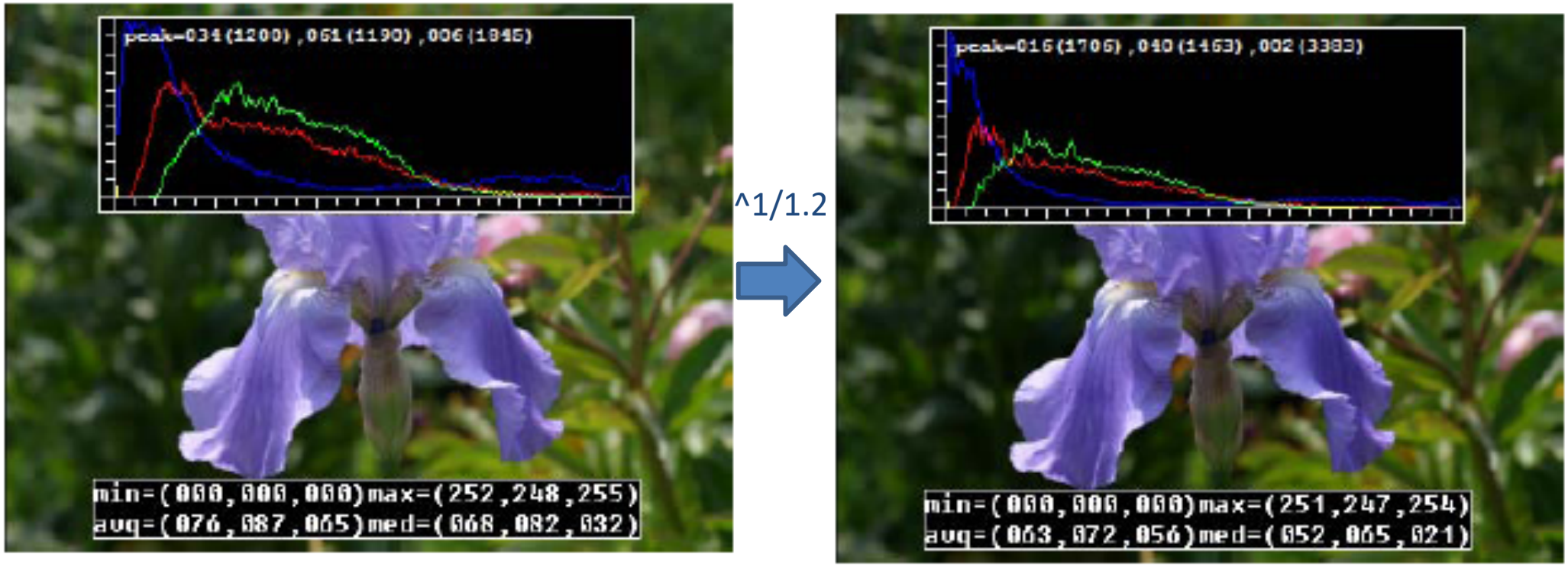
Increased contrast



$$g(x)=1.1f(x)$$

Point operators

Gamma correction



$$g(x)=[f(x)]^{1/1.2}$$

Point operators

- Color transforms
  - Color balancing: Multiply each channel with different factor

Point operators

- Matting
  - Extracting an object from an image.

Point operators

- Matting
  - Extracting an object from an image.

- Compositing
  - Inserting an object into another image.

Point operators

- Histogram equalization
  - Make the histogram flat (more or less)

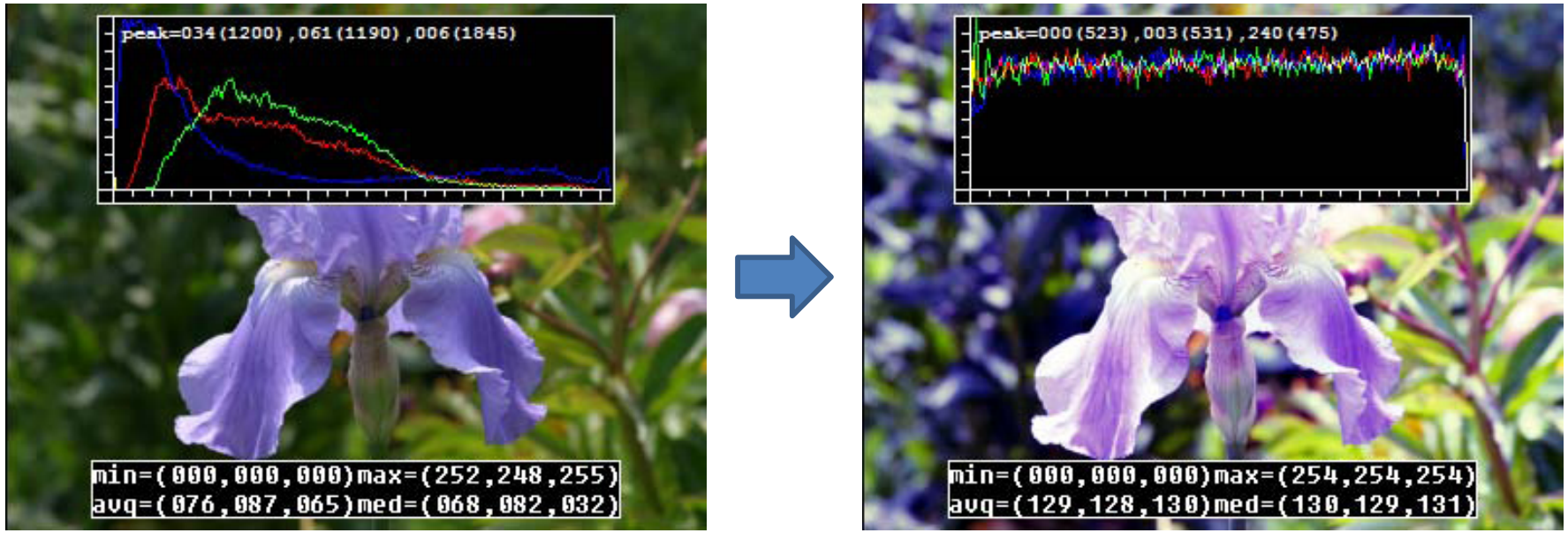
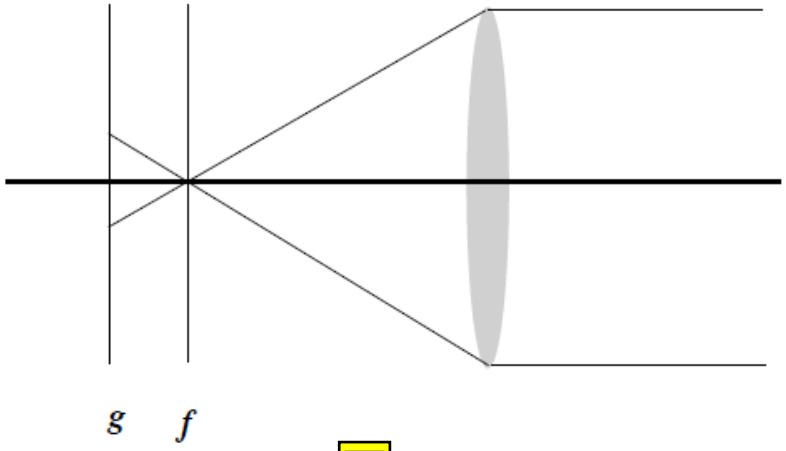


Image processing

- Introduction
- Point operators
- Linear filtering
- Non linear filtering
- The Fourier transform
- Pyramids and scales

Linear filtering

- Linear, shift invariant (LSI) systems
  - Consider
    - An ideal, in focus image, f
    - Its out of focus counterpart, g
  - Suppose the light is changed so that
    - the irradiance in f is doubled,
    - then so is the irradiance in g doubled.
  - Suppose the imaging system is moved so that
    - the pattern in f is shifted,
    - then the pattern in g is similarly shifted.
  - The transformation from the ideal to the out of focus system is said to be a linear, shift invariant operation.



Source: Richard Wildes slides

Linear filtering

- **Consider a 2D system**
  - Let the system produce output $g1(x,y)$ when given $f1(x,y)$
  - Let the system produce output $g2(x,y)$ when given $f2(x,y)$

Source: Richard Wildes slides

Linear filtering

- **Consider a 2D system**
  - Let the system produce output $g1(x,y)$ when given $f1(x,y)$
  - Let the system produce output $g2(x,y)$ when given $f2(x,y)$

- **A system is linear**
  - if the output ($a\ g1(x,y) + b\ g2(x,y)$) is produced when the input is ($a\ f1(x,y) + b\ f2(x,y)$).

- Remarks
  - Most real systems are limited in their maximum response and thus cannot be strictly linear.
  - Irradiance, power/unit area, cannot be negative; so, visual input is restricted to be nonnegative.

Source: Richard Wildes slides

Linear filtering

- **Consider a 2D system**
  - Let the system produce output $g1(x,y)$ when given $f1(x,y)$
  - Let the system produce output $g2(x,y)$ when given $f2(x,y)$

- **A system is invariant**
  - If it produces the shifted output ***g1(x-a,y-b)*** *when given the input* ***f1(x-a,y-b)***

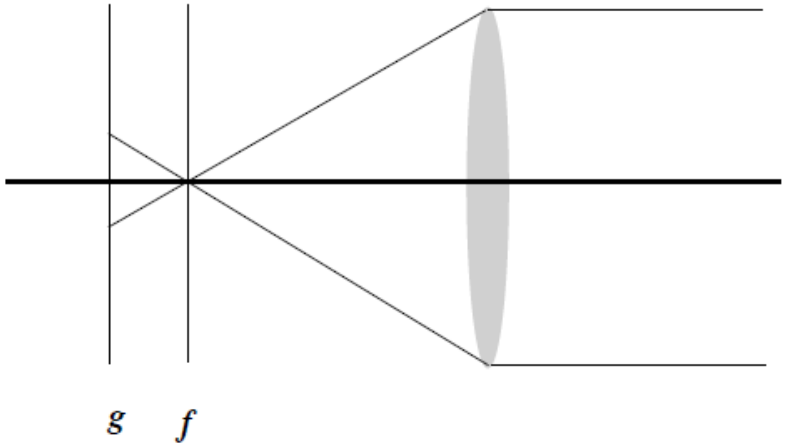
- Remarks

  - In practice, images are limited in area; so, shift invariance only holds for limited displacements.
  - Aberrations in optical imaging vary spatially, yielding further departure from perfect shift invariance.

Source: Richard Wildes slides

Linear filtering

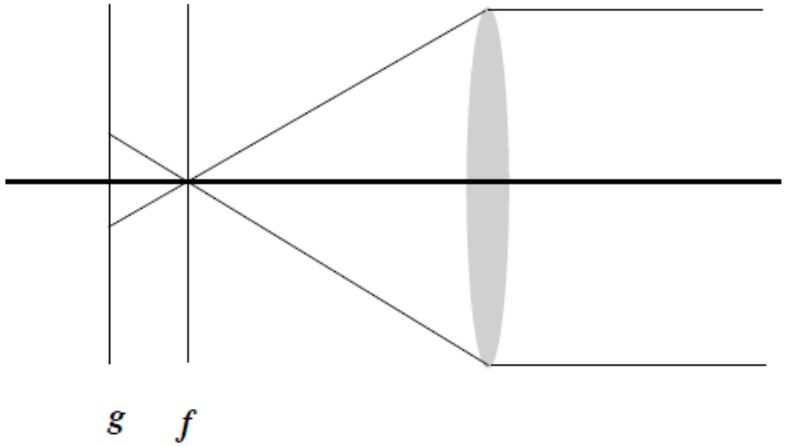
- Linear, shift invariant (LSI) systems
  - Consider
    - An ideal, in focus image, $f(x,y)$
    - Its out of focus counterpart, $g(x,y)$
  - Suppose the light is changed so that
    - the irradiance in f is doubled,
      - *The system input is*
        $1\ f(x,y) + 1\ f(x,y)$
    - then so is the irradiance in g doubled.
      - *The system output is*
        $1\ g(x,y) + 1\ g(x,y)$
    - **This system is linear**



Source: Richard Wildes slides

Linear filtering

- Linear, shift invariant (LSI) systems
  - Consider
    - An ideal, in focus image, $f(x,y)$
    - Its out of focus counterpart, $g(x,y)$
  - Suppose the imaging system is moved so that
    - the pattern in f is shifted,
      - *The system input is* $f(x-a, y-b)$
    - then the pattern in g is similarly shifted.
      - *The system output is* $g(x-a, y-b)$
    - **This system is shift invariant**



Source: Richard Wildes slides

Linear filtering

- Neighborhood operators
  - The output pixel's value is determined as a weighted sum of input pixel values

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l)$$

Linear filtering

- Neighborhood operators
  - The output pixel's value is determined as a weighted sum of input pixel values

45	60	98	127	132	133	137	133
46	65	98	123	126	128	131	133
47	65	96	115	119	123	135	137
47	63	91	107	113	122	138	134
50	59	80	97	110	123	133	134
49	53	68	83	97	113	128	133
50	50	58	70	84	102	116	126
50	50	52	58	69	86	101	120

$f(x,y)$

*

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1

$h(x,y)$

=

69	95	116	125	129	132
68	92	110	120	126	132
66	86	104	114	124	132
62	78	94	108	120	129
57	69	83	98	112	124
53	60	71	85	100	114

$g(x,y)$

Linear filtering

- Neighborhood operators
  - The output pixel's value is determined as a weighted sum of input pixel values

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l)$$

  - $h(k,l)$ is the filter, kernel or mask

Linear filtering

- Neighborhood operators
  - The output pixel's value is determined as a weighted sum of input pixel values

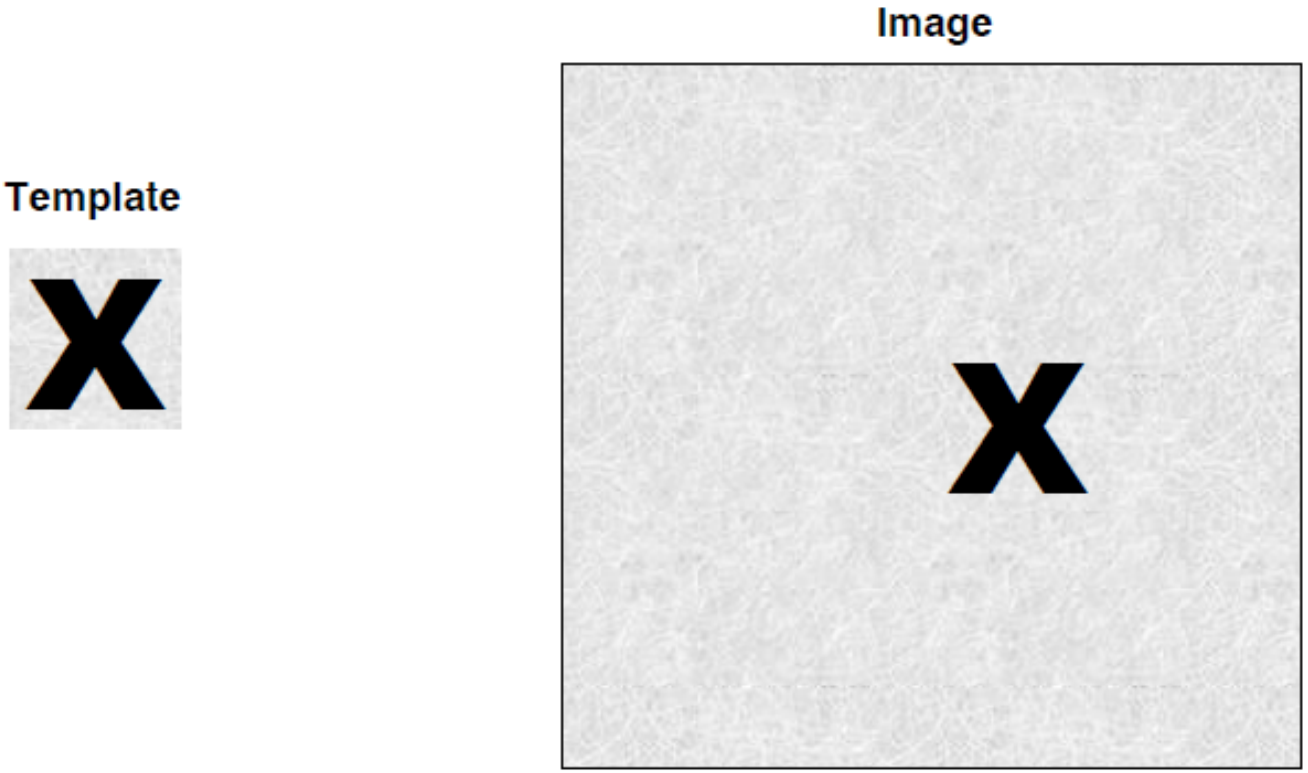
$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l)$$

  - This expression is the cross-correlation, represented as

$$g = f \otimes h$$

Linear filtering

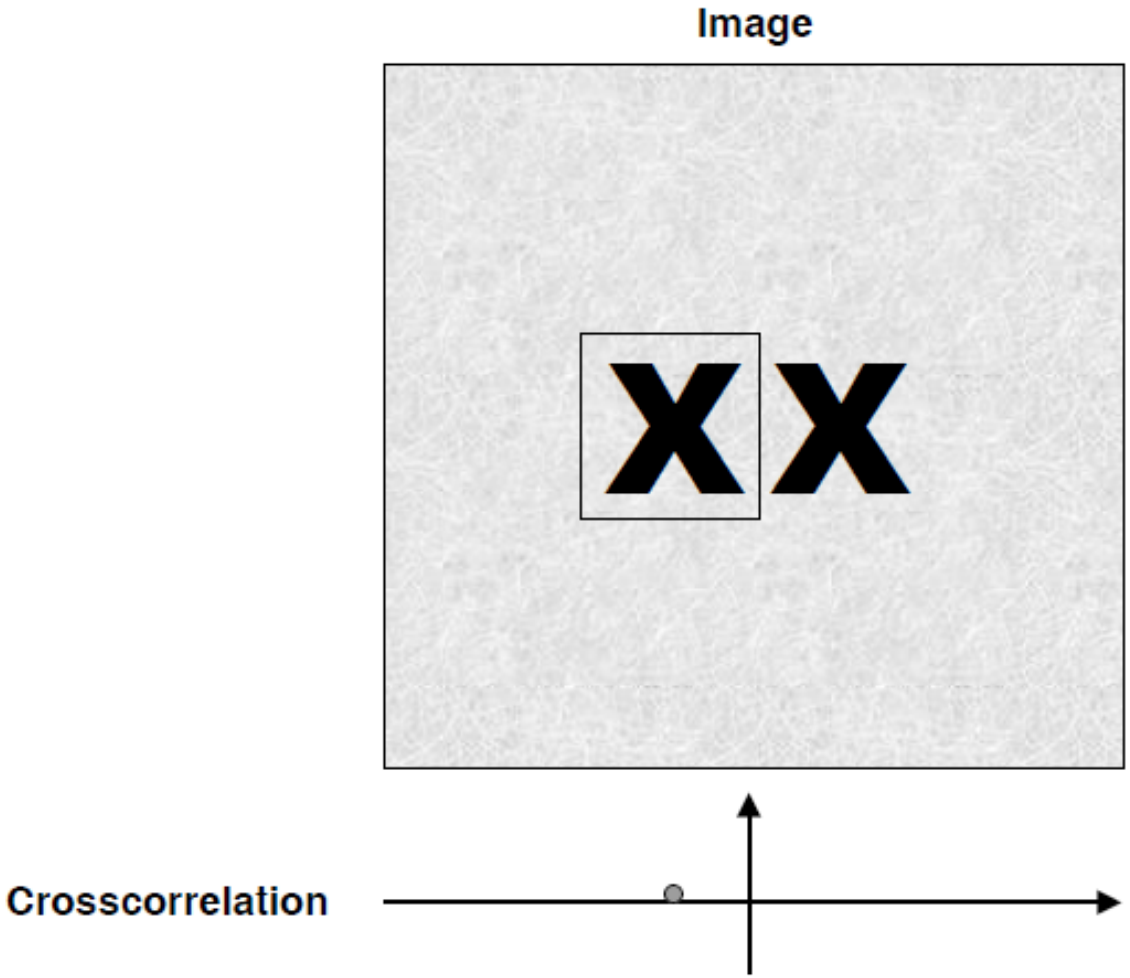
- Cross-correlation



Source: Richard Wildes slides

Linear filtering

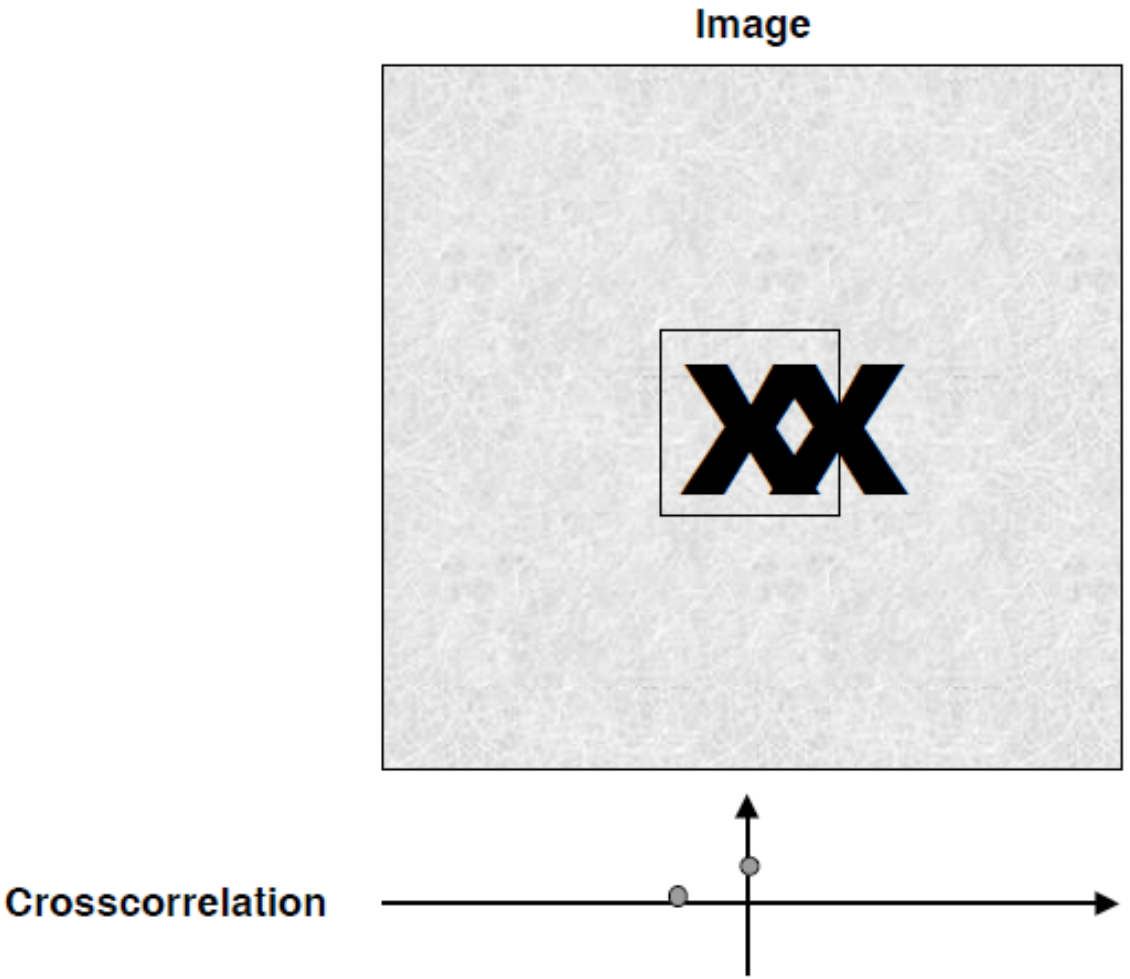
- Cross-correlation



Source: Richard Wildes slides

Linear filtering

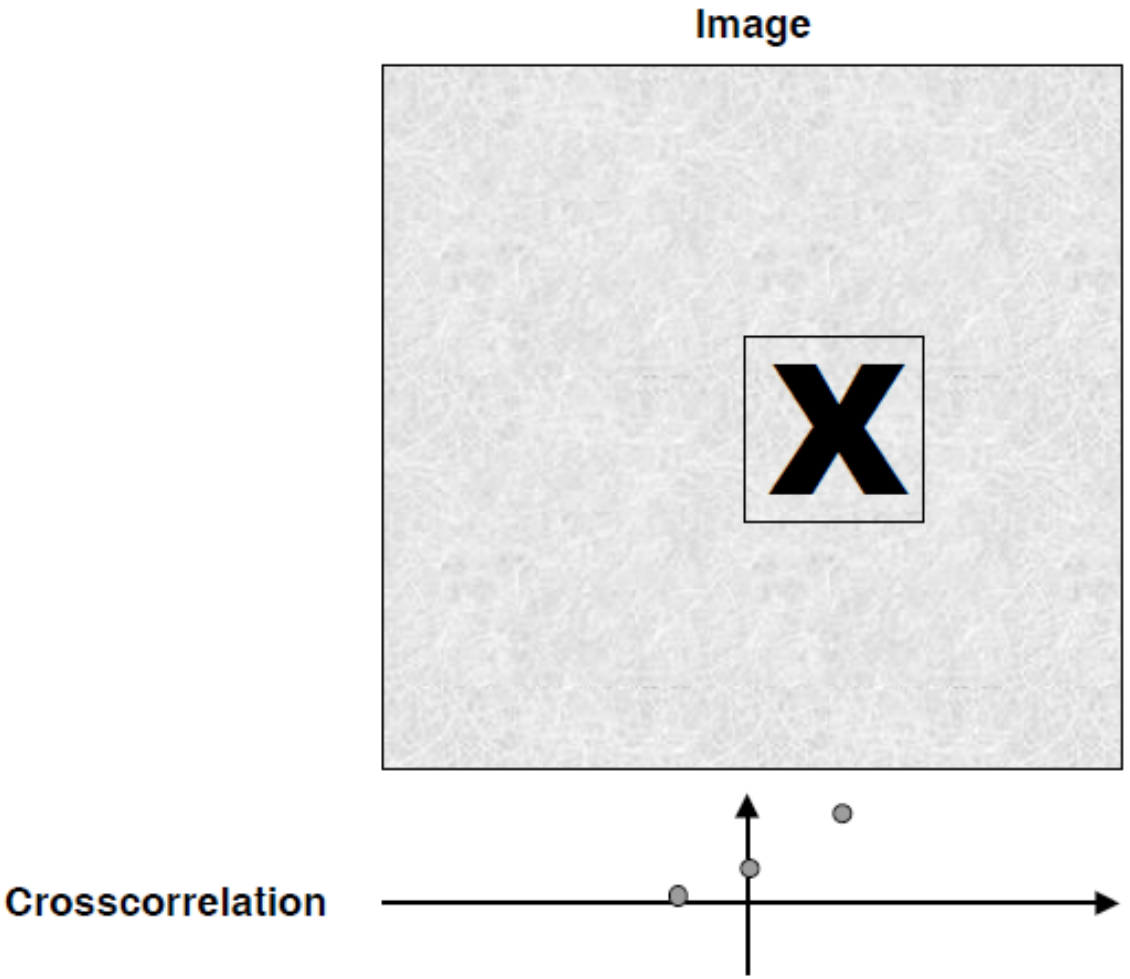
- Cross-correlation



Source: Richard Wildes slides

Linear filtering

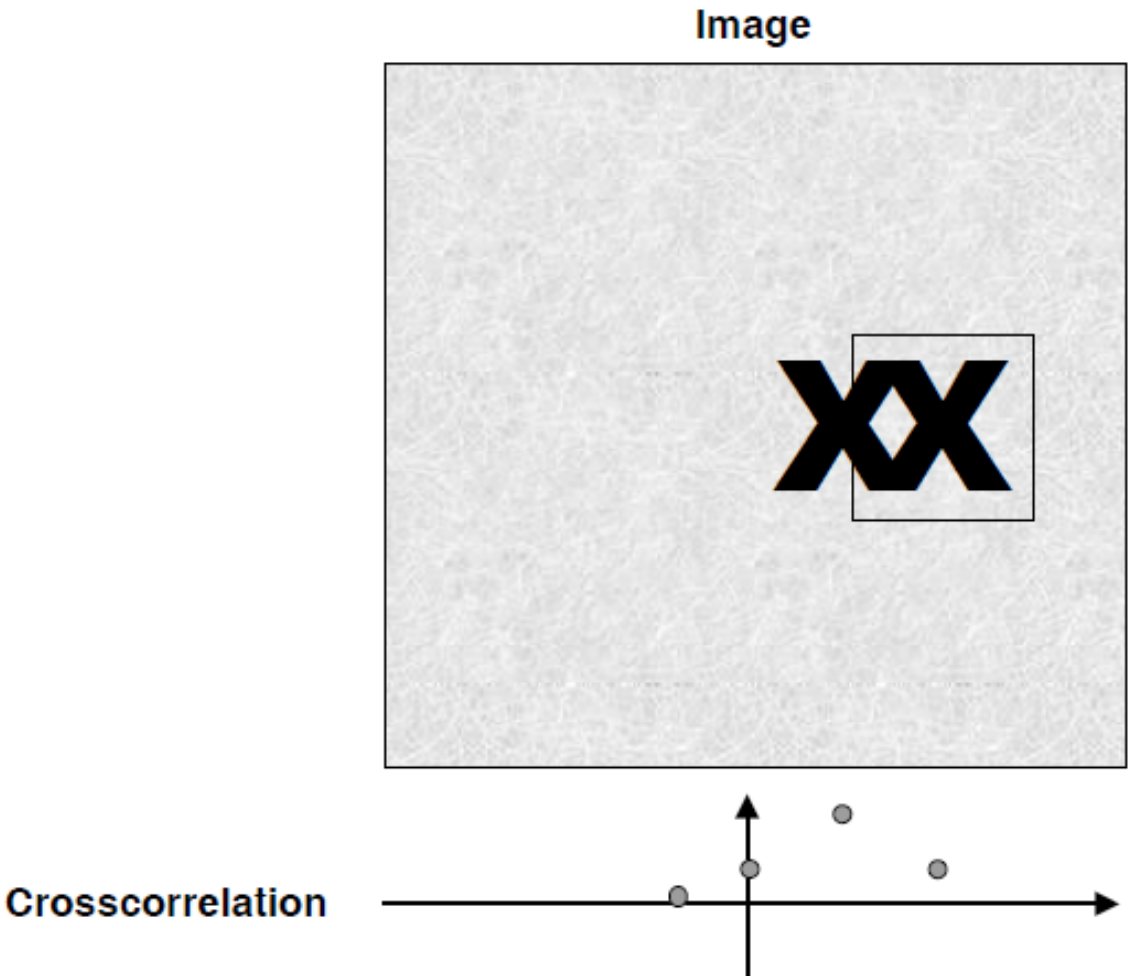
- Cross-correlation



Source: Richard Wildes slides

Linear filtering

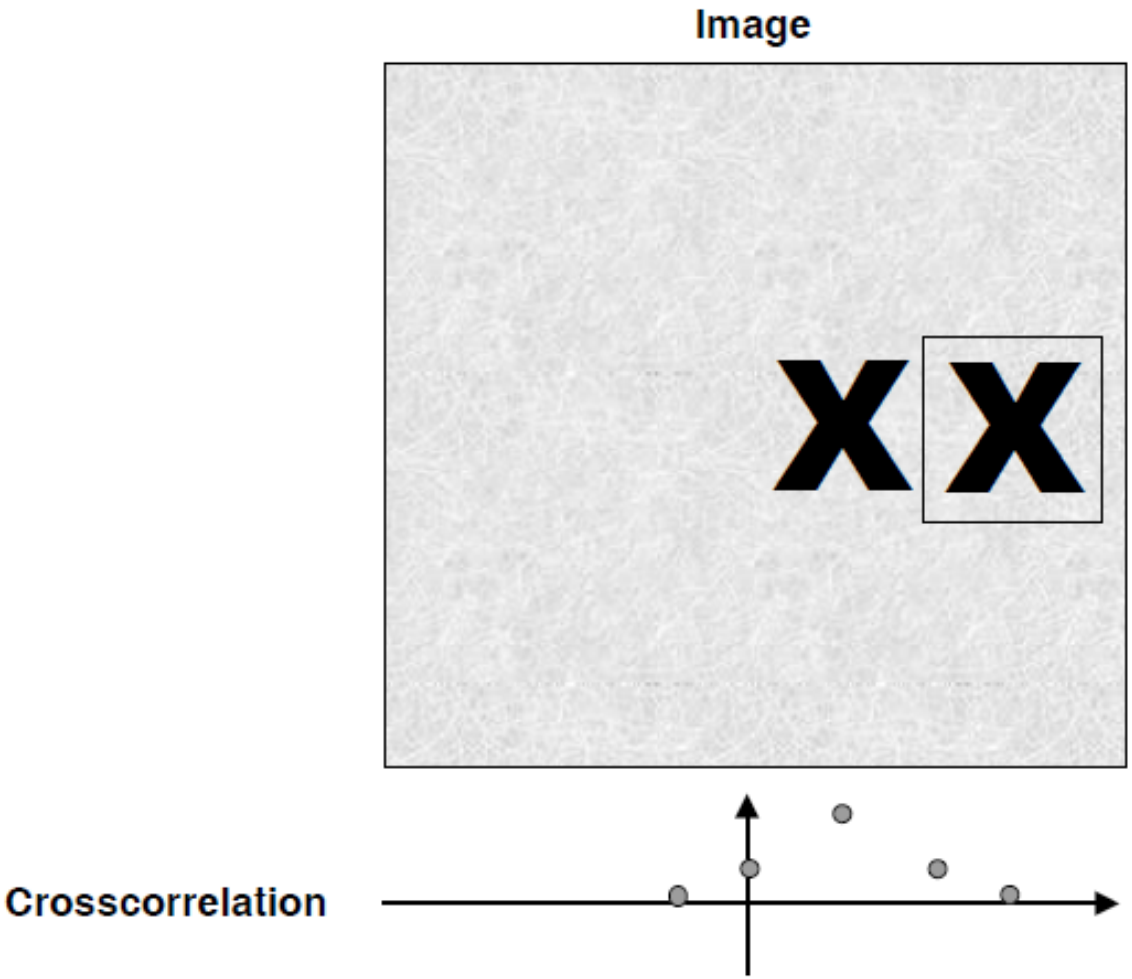
- Cross-correlation



Source: Richard Wildes slides

Linear filtering

- Cross-correlation



Source: Richard Wildes slides

Linear filtering

- Neighborhood operators
  - The output pixel's value is determined as a weighted sum of input pixel values

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l)$$

  - A common variant of this formula is

$$g(i,j) = \sum_{k,l} f(i-k, j-l)h(k,l) = \sum_{k,l} f(k,l)h(i-k, j-l)$$

Linear filtering

- Neighborhood operators
  - A common variant of the cross-correlation is

$$g(i,j) = \sum_{k,l} f(i-k, j-l)h(k,l)$$

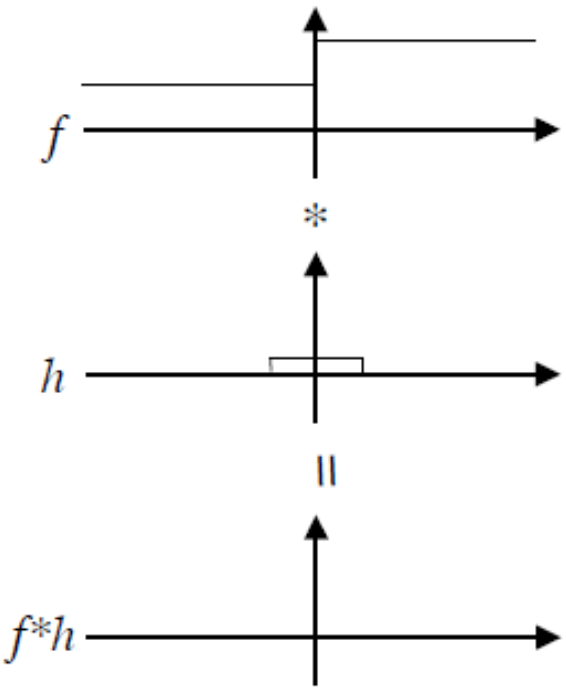
  - This is the **convolution**, represented as

$$g = f * h$$

Linear filtering

- Convolution

Graphic depiction



Formalization

Numerical calculation

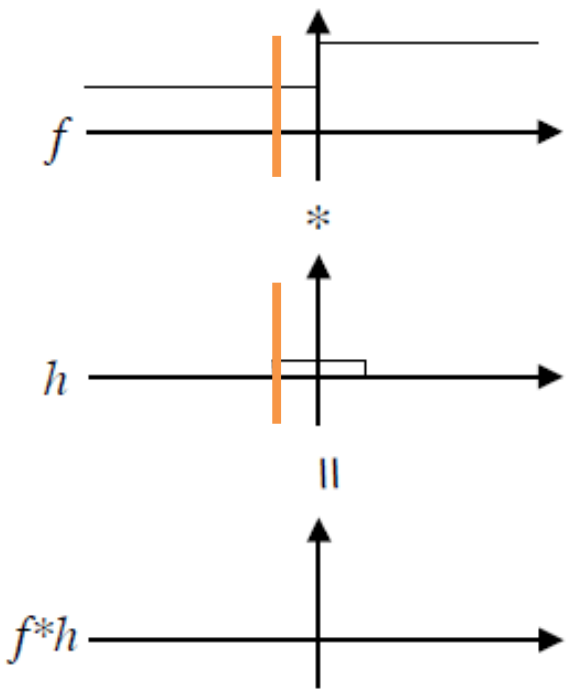
x	s	f(x-s)h(s)	+
-2	-1		
	0		
	1		
-1	-1		
	0		
	1		
0	-1		
	0		
	1		
1	-1		
	0		
	1		
2	-1		
	0		
	1		

Source: Richard Wildes slides

Linear filtering

- Convolution

Graphic depiction



Formalization

Numerical calculation

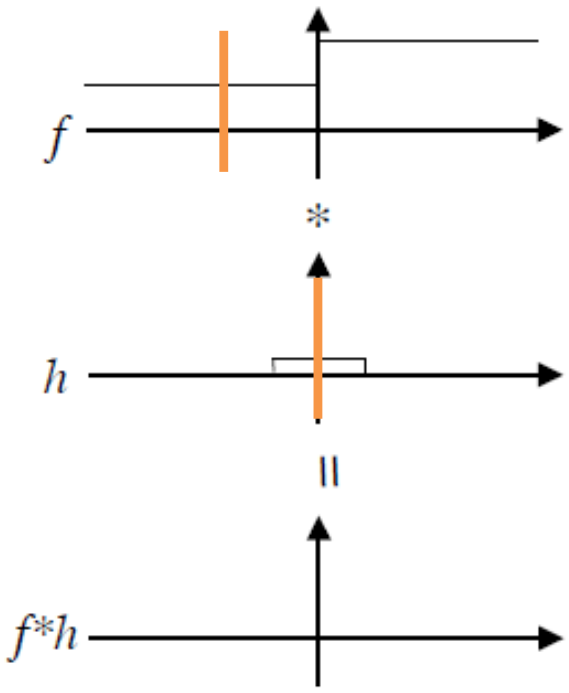
x	s	f(x-s)h(s)	+
-2	-1	(1)(1/3)	
	0		
	1		
-1	-1		
	0		
	1		
0	-1		
	0		
	1		
1	-1		
	0		
	1		
2	-1		
	0		
	1		

Source: Richard Wildes slides

Linear filtering

- Convolution

Graphic depiction



Formalization

Numerical calculation

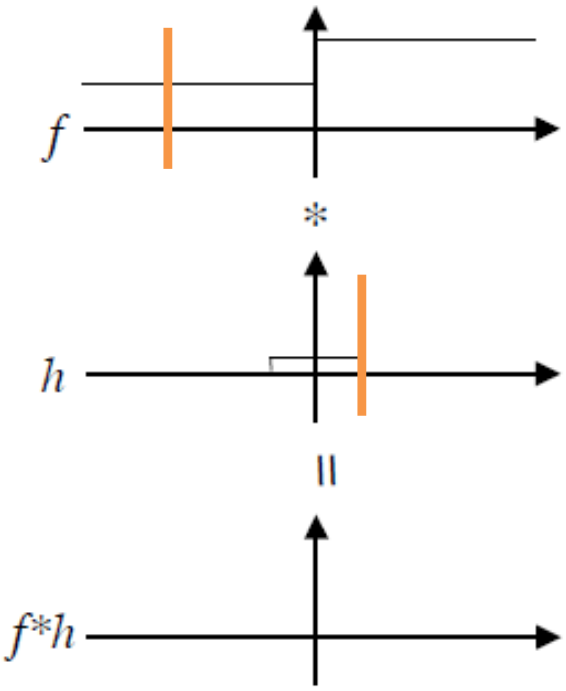
x	s	f(x-s)h(s)	+
-2	-1	(1)(1/3)	
	0	(1)(1/3)	
	1		
-1	-1		
	0		
	1		
0	-1		
	0		
	1		
1	-1		
	0		
	1		
2	-1		
	0		
	1		

Source: Richard Wildes slides

Linear filtering

- Convolution

Graphic depiction



Formalization

Numerical calculation

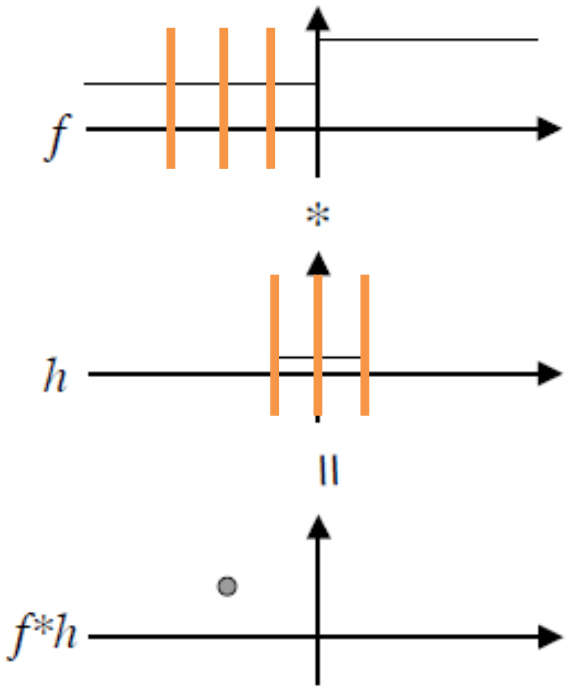
x	s	f(x-s)h(s)	+
-2	-1	(1)(1/3)	
	0	(1)(1/3)	
	1	(1)(1/3)	
-1	-1		
	0		
	1		
0	-1		
	0		
	1		
1	-1		
	0		
	1		
2	-1		
	0		
	1		

Source: Richard Wildes slides

Linear filtering

- Convolution

Graphic depiction



Numerical calculation

x	s	f(x-s)h(s)	+
-2	-1	(1)(1/3)	
	0	(1)(1/3)	
	1	(1)(1/3)	1
-1	-1		
	0		
	1		
0	-1		
	0		
	1		
1	-1		
	0		
	1		
2	-1		
	0		
	1		

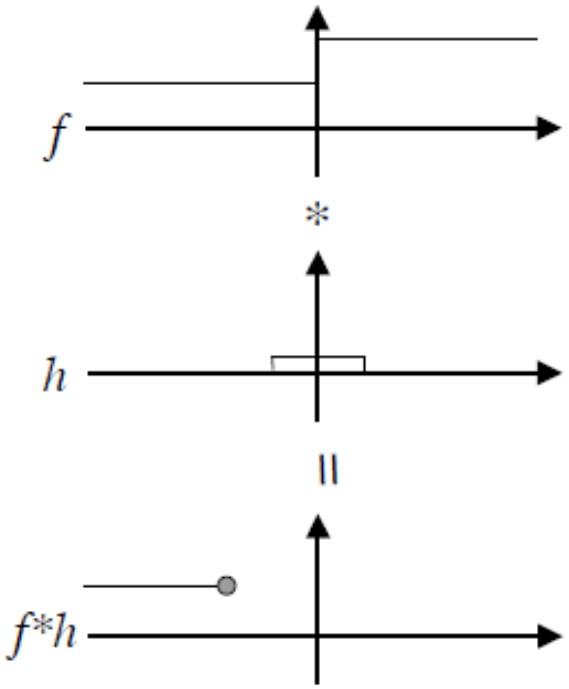
Formalization

Source: Richard Wildes slides

Linear filtering

- Convolution

Graphic depiction



Numerical calculation

x	s	f(x-s)h(s)	+
-2	-1	(1)(1/3)	
	0	(1)(1/3)	
	1	(1)(1/3)	1
-1	-1		
	0		
	1		
0	-1		
	0		
	1		
1	-1		
	0		
	1		
2	-1		
	0		
	1		

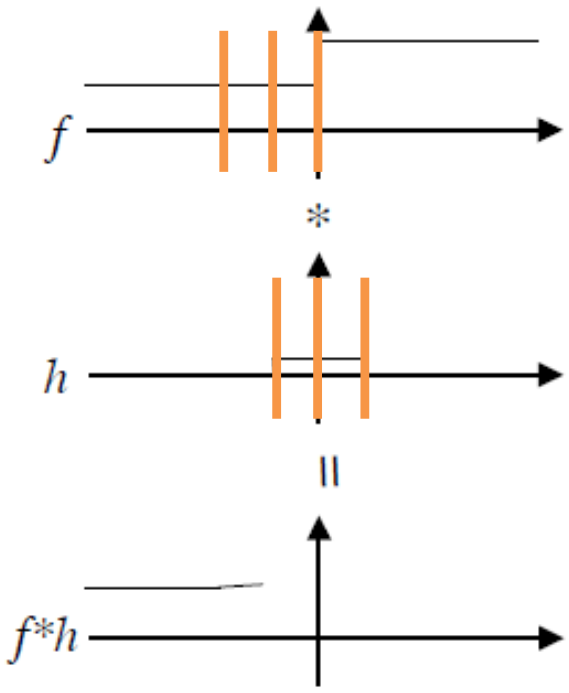
Formalization

Source: Richard Wildes slides

Linear filtering

- Convolution

Graphic depiction



Numerical calculation

x	s	f(x-s)h(s)	+
-2	-1	(1)(1/3)	
	0	(1)(1/3)	
	1	(1)(1/3)	1
-1	-1	(3/2)(1/3)	
	0	(1)(1/3)	
	1	(1)(1/3)	7.0/6.0
0	-1		
	0		
	1		
1	-1		
	0		
	1		
2	-1		
	0		
	1		

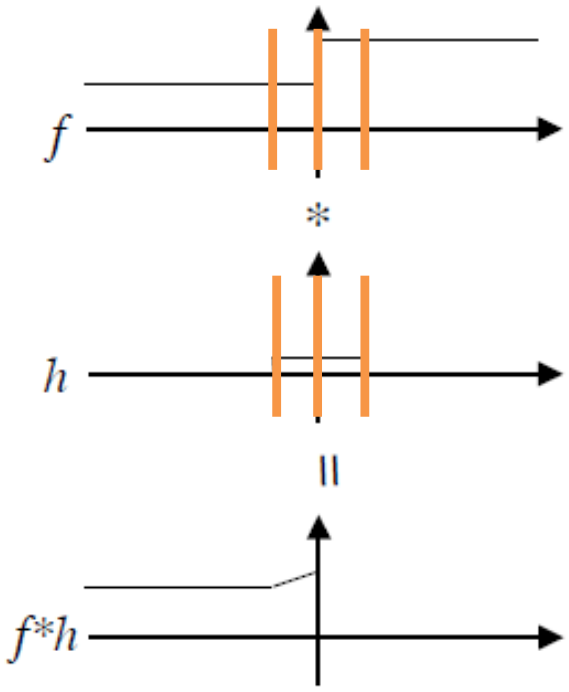
Formalization

Source: Richard Wildes slides

Linear filtering

- Convolution

Graphic depiction



Formalization

Numerical calculation

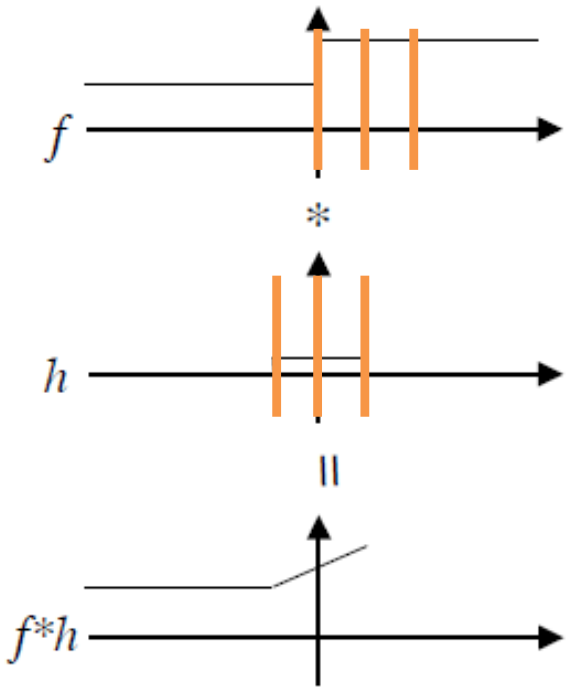
x	s	f(x-s)h(s)	+
-2	-1	(1)(1/3)	
	0	(1)(1/3)	
	1	(1)(1/3)	1
-1	-1	(3/2)(1/3)	
	0	(1)(1/3)	
	1	(1)(1/3)	7.0/6.0
0	-1	(2)(1/3)	
	0	(3/2)(1/3)	
	1	(1)(1/3)	3.0/2.0
1	-1		
	0		
	1		
2	-1		
	0		
	1		

Source: Richard Wildes slides

Linear filtering

- Convolution

Graphic depiction



Formalization

Numerical calculation

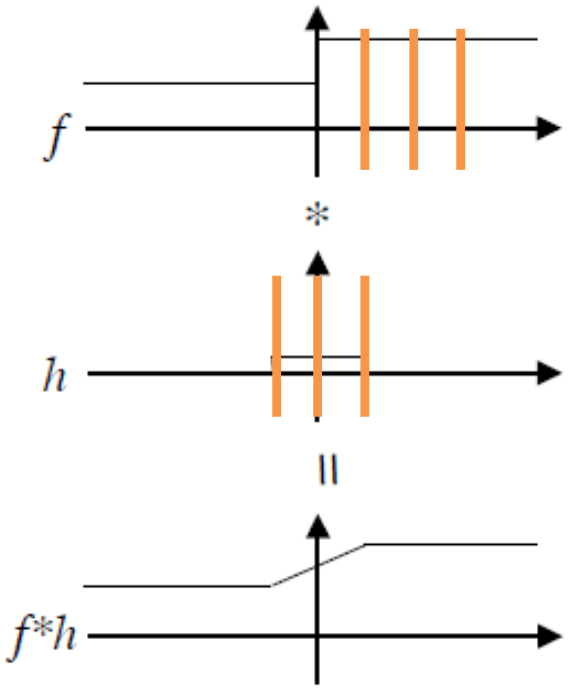
x	s	f(x-s)h(s)	+
-2	-1	(1)(1/3)	
	0	(1)(1/3)	
	1	(1)(1/3)	1
-1	-1	(3/2)(1/3)	
	0	(1)(1/3)	
	1	(1)(1/3)	7.0/6.0
0	-1	(2)(1/3)	
	0	(3/2)(1/3)	
	1	(1)(1/3)	3.0/2.0
1	-1	(2)(1/3)	
	0	(2)(1/3)	
	1	(3/2)(1/3)	11.0/6.0
2	-1		
	0		
	1		

Source: Richard Wildes slides

Linear filtering

- Convolution

Graphic depiction



Numerical calculation

x	s	f(x-s)h(s)	+
-2	-1	(1)(1/3)	
	0	(1)(1/3)	
	1	(1)(1/3)	1
-1	-1	(3/2)(1/3)	
	0	(1)(1/3)	
	1	(1)(1/3)	7.0/6.0
0	-1	(2)(1/3)	
	0	(3/2)(1/3)	
	1	(1)(1/3)	3.0/2.0
1	-1	(2)(1/3)	
	0	(2)(1/3)	
	1	(3/2)(1/3)	11.0/6.0
2	-1	(2)(1/3)	
	0	(2)(1/3)	
	1	(2)(1/3)	2

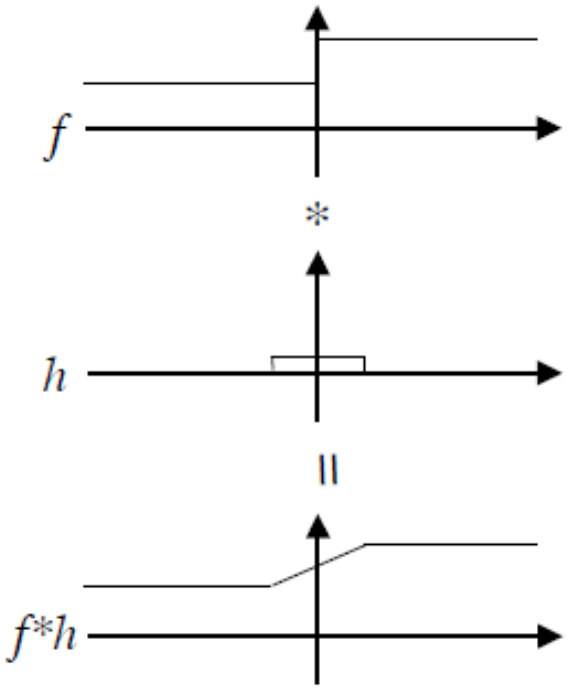
Formalization

Source: Richard Wildes slides

Linear filtering

- Convolution

Graphic depiction



Formalization

$$f(x)$$

Numerical calculation

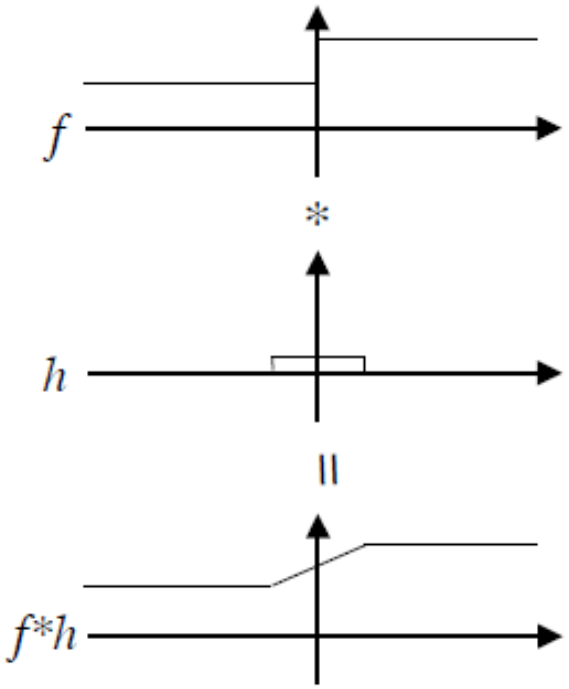
x	s	f(x-s)h(s)	+
-2	-1	(1)(1/3)	
	0	(1)(1/3)	
	1	(1)(1/3)	1
-1	-1	(3/2)(1/3)	
	0	(1)(1/3)	
	1	(1)(1/3)	7.0/6.0
0	-1	(2)(1/3)	
	0	(3/2)(1/3)	
	1	(1)(1/3)	3.0/2.0
1	-1	(2)(1/3)	
	0	(2)(1/3)	
	1	(3/2)(1/3)	11.0/6.0
2	-1	(2)(1/3)	
	0	(2)(1/3)	
	1	(2)(1/3)	2

Source: Richard Wildes slides

Linear filtering

- Convolution

Graphic depiction



Numerical calculation

x	s	f(x-s)h(s)	+
-2	-1	(1)(1/3)	
	0	(1)(1/3)	
	1	(1)(1/3)	1
-1	-1	(3/2)(1/3)	
	0	(1)(1/3)	
	1	(1)(1/3)	7.0/6.0
0	-1	(2)(1/3)	
	0	(3/2)(1/3)	
	1	(1)(1/3)	3.0/2.0
1	-1	(2)(1/3)	
	0	(2)(1/3)	
	1	(3/2)(1/3)	11.0/6.0
2	-1	(2)(1/3)	
	0	(2)(1/3)	
	1	(2)(1/3)	2

Formalization

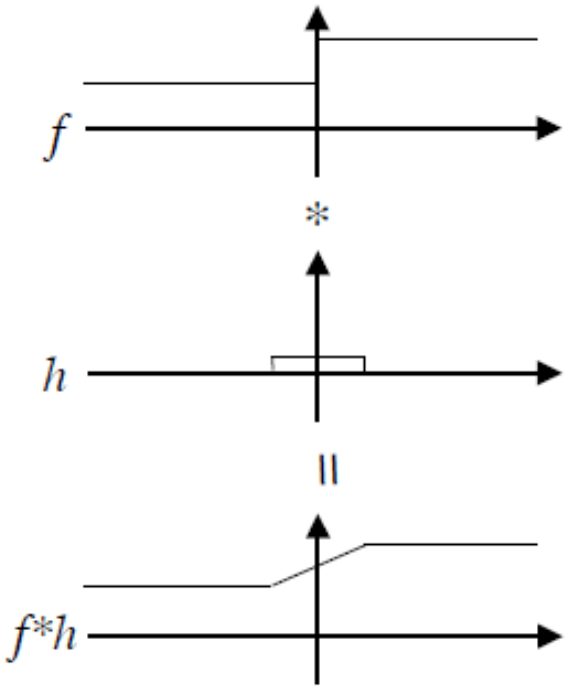
$$f(x)h$$

Source: Richard Wildes slides

Linear filtering

- Convolution

Graphic depiction



Numerical calculation

x	s	f(x-s)h(s)	+
-2	-1	(1)(1/3)	
	0	(1)(1/3)	
	1	(1)(1/3)	1
-1	-1	(3/2)(1/3)	
	0	(1)(1/3)	
	1	(1)(1/3)	7.0/6.0
0	-1	(2)(1/3)	
	0	(3/2)(1/3)	
	1	(1)(1/3)	3.0/2.0
1	-1	(2)(1/3)	
	0	(2)(1/3)	
	1	(3/2)(1/3)	11.0/6.0
2	-1	(2)(1/3)	
	0	(2)(1/3)	
	1	(2)(1/3)	2

Formalization

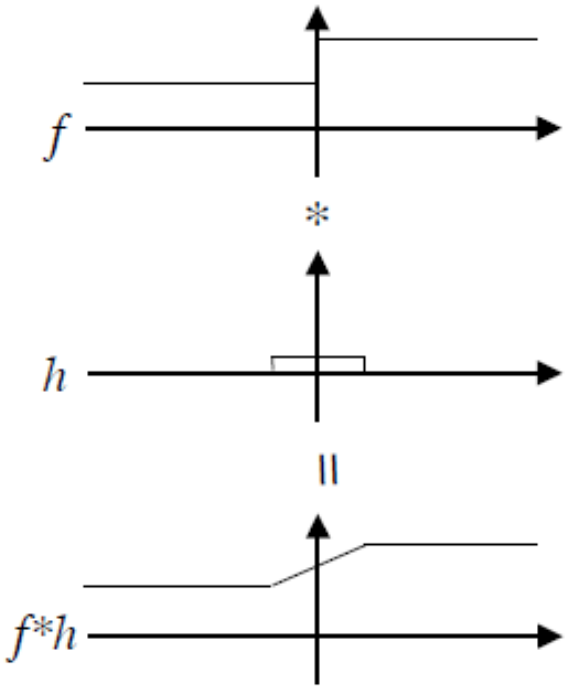
$$f(x-s)h(s)$$

Source: Richard Wildes slides

Linear filtering

- Convolution

Graphic depiction



Numerical calculation

x	s	f(x-s)h(s)	+
-2	-1	(1)(1/3)	
	0	(1)(1/3)	
	1	(1)(1/3)	1
-1	-1	(3/2)(1/3)	
	0	(1)(1/3)	
	1	(1)(1/3)	7.0/6.0
0	-1	(2)(1/3)	
	0	(3/2)(1/3)	
	1	(1)(1/3)	3.0/2.0
1	-1	(2)(1/3)	
	0	(2)(1/3)	
	1	(3/2)(1/3)	11.0/6.0
2	-1	(2)(1/3)	
	0	(2)(1/3)	
	1	(2)(1/3)	2

Formalization

$$\int f(x-s)h(s)ds$$

Source: Richard Wildes slides

Linear filtering

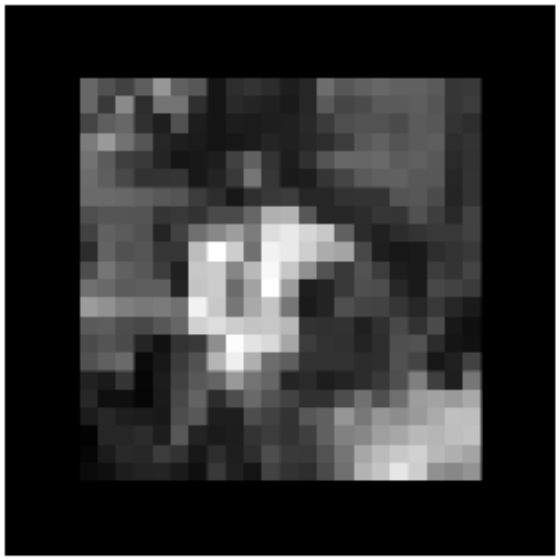
- Neighborhood operators
  - Convolution and correlation are Linear Shift Invariant (LSI) systems

$$h \circ (f_0 + f_1) = h \circ f_0 + h \circ f_1,$$

$$g(i,j) = f(i+k, j+l) \quad \Leftrightarrow \quad (h \circ g)(i,j) = (h \circ f)(i+k, j+l),$$

Linear filtering

- Neighborhood operators
  - Convolution and correlation are Linear Shift Invariant (LSI) systems
  - Padding (border effects)
    - zero

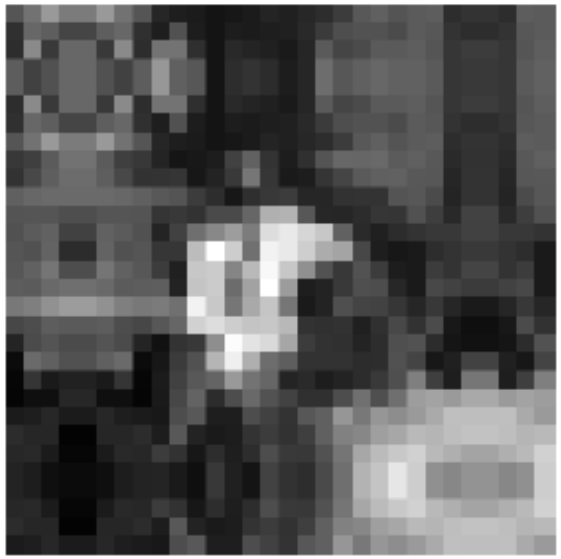


Linear filtering

- Neighborhood operators
  - Convolution and correlation are Linear Shift Invariant (LSI) systems
  - Padding (border effects)
    - clamp

Linear filtering

- Neighborhood operators
  - Convolution and correlation are Linear Shift Invariant (LSI) systems
  - Padding (border effects)
    - mirror



Linear filtering

- Separable filtering
  - Filtering in 2D: K^2 operations per pixel

    K is the size of the kernel

Linear filtering

- Separable filtering
  - Filtering in 2D: K^2 operations per pixel

    K is the size of the kernel
  - In a separable filter we can do it
    - Horizontally
    - Vertically
    - Requires $2K$ operations per pixel
    - We can obtain the 2D filter by applying

$$K = vh^T$$

Linear filtering

- Separable filtering

$$\frac{1}{K^2}\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & 1 & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}$$

$$\frac{1}{K}\begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}$$

Linear filtering

- Separable filtering

$$\frac{1}{K^2}\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & 1 & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}$$

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

$$\frac{1}{K}\begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}$$

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

Linear filtering

- Separable filtering

$$\frac{1}{256}\begin{array}{|c|c|c|c|c|}\hline 1 & 4 & 6 & 4 & 1 \\ \hline 4 & 16 & 24 & 16 & 4 \\ \hline 6 & 24 & 36 & 24 & 6 \\ \hline 4 & 16 & 24 & 16 & 4 \\ \hline 1 & 4 & 6 & 4 & 1 \\ \hline \end{array}$$

$$\frac{1}{16}\begin{array}{|c|c|c|c|c|}\hline 1 & 4 & 6 & 4 & 1 \\ \hline \end{array}$$

Linear filtering

- Separable filtering

$$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

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

$$\frac{1}{16}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

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

Linear filtering

- Low-pass filtering
  - Pass-through lower frequencies and attenuates higher frequencies
  - Blurs the image
  - Removes noise (usually high-frequencies)

Linear filtering

- Low-pass filtering
  - Pass-through lower frequencies and attenuates higher frequencies
  - Blurs the image
  - Removes noise (usually high-frequencies)
  - Examples
    - Gaussian filter

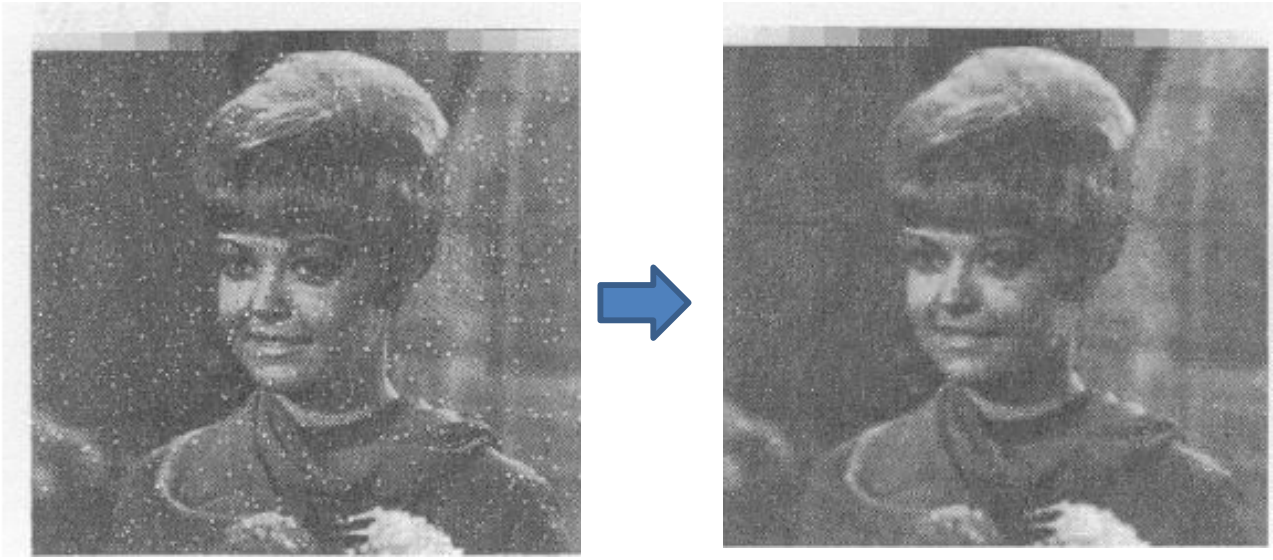
$$G(x, y; \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}},$$

Linear filtering

- Low-pass filtering
  - Pass-through lower frequencies and attenuates higher frequencies
  - Blurs the image
  - Removes noise (usually high-frequencies)
  - Examples
    - Gaussian filter

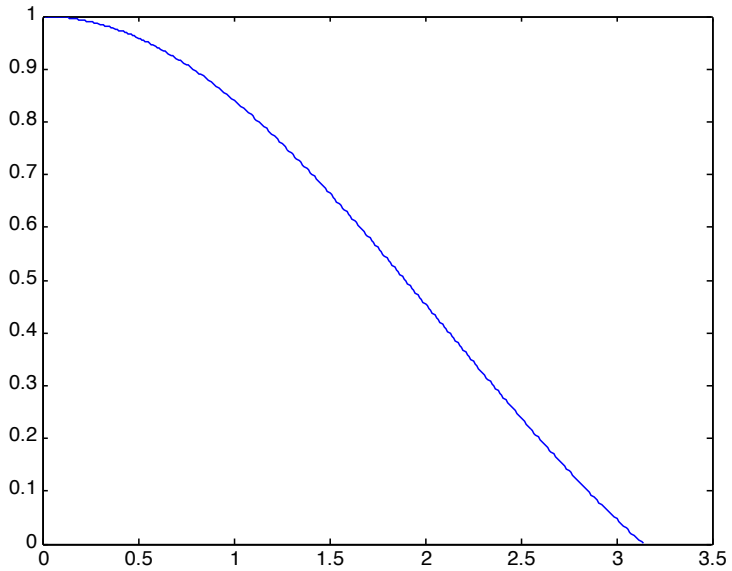
Linear filtering

- Low-pass filtering
  - Pass-through lower frequencies and attenuates higher frequencies
  - Blurs the image
  - Removes noise (usually high-frequencies)
  - Examples
    - Gaussian filter



Linear filtering

- Low-pass filtering
  - Pass-through lower frequencies and attenuates higher frequencies
  - Blurs the image
  - Removes noise (usually high-frequencies)
  - Examples
    - Gaussian filter
    - $(\sin x)/x$



Linear filtering

- Low-pass filtering
  - Pass-through lower frequencies and attenuates higher frequencies
  - Blurs the image
  - Removes noise (usually high-frequencies)
  - Unsharp mask
    - Adding some of the difference between the original and the blurred image makes it sharper

$$g_{\text{sharp}} = f + \gamma(f - h_{\text{blur}} * f).$$

Linear filtering

- Low-pass filtering
  - Pass-through lower frequencies and attenuates higher frequencies
  - Blurs the image
  - Removes noise (usually high-frequencies)
  - Unsharp mask
    - Adding some of the difference between the original and the blurred image makes it sharper

$$g_{\text{sharp}} = f + \gamma(f - h_{\text{blur}} * f).$$

Linear filtering

- Low-pass filtering
  - Pass-through lower frequencies and attenuates higher frequencies
  - Blurs the image
  - Removes noise (usually high-frequencies)
  - Unsharp mask
    - Adding some of the difference between the original and the blurred image makes it sharper

$$g_{\text{sharp}} = f + \gamma(f - h_{\text{blur}} * f).$$

Linear filtering

- Band-pass filtering
  - Filter low and high frequencies
  - Remember the Gaussian equation

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

Linear filtering

- Band-pass filtering
  - Filter low and high frequencies
  - Remember the Gaussian equation

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

  - If we take the Laplacian

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Note: there is a typo in Szeliski book

Linear filtering

- Band-pass filtering
  - Filter low and high frequencies
  - Remember the Gaussian equation

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

  - If we take the Laplacian

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Note: there is a typo in Szeliski book

  - We obtain the Laplacian of Gaussian filter

$$\nabla^2 G(x, y; \sigma) = \left(\frac{x^2 + y^2}{\sigma^4} - \frac{2}{\sigma^2} \right) G(x, y; \sigma),$$

Linear filtering

- Band-pass filtering
  - Filter low and high frequencies
  - Remember the Gaussian equation

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

  - If we take the Laplacian

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

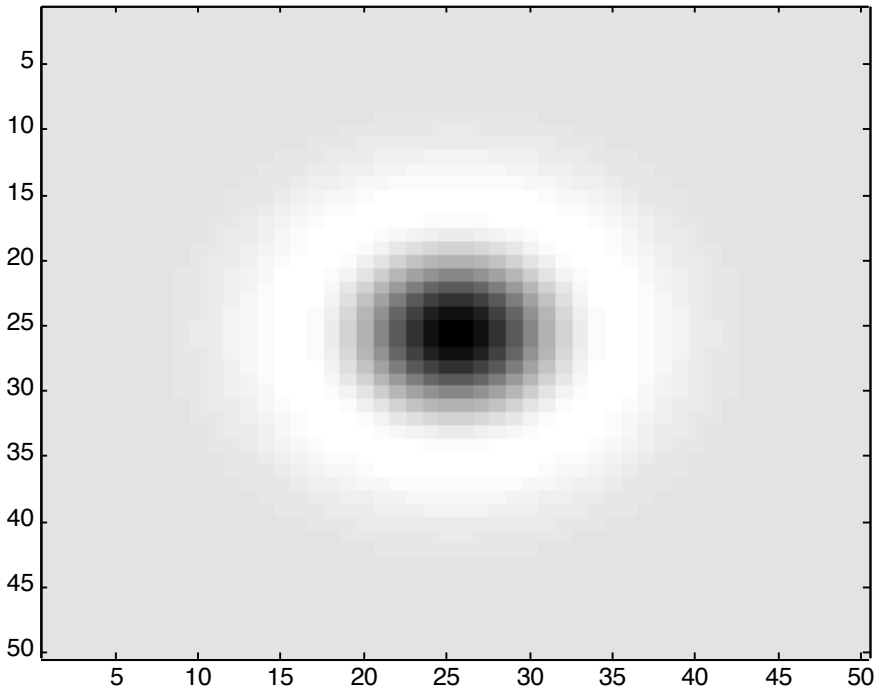
Note: there is a typo in Szeliski book

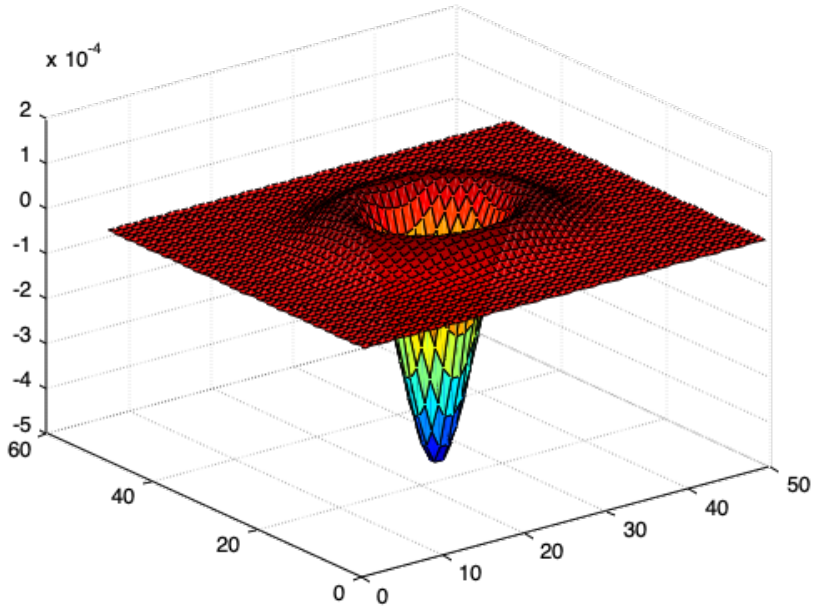
  - We obtain the Laplacian of Gaussian filter

$$\nabla^2 G(x, y; \sigma) = \left(\frac{x^2 + y^2}{\sigma^4} - \frac{2}{\sigma^2} \right) G(x, y; \sigma) = -\frac{1}{\pi\sigma^4} \left[1 - \frac{x^2 + y^2}{2\sigma^2} \right] e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Linear filtering

- Band-pass filtering





Linear filtering

- Band-pass filtering

0	1	1	2	2	2	1	1	0
1	2	4	5	5	5	4	2	1
1	4	5	3	0	3	5	4	1
2	5	3	-12	-24	-12	3	5	2
2	5	0	-24	-40	-24	0	5	2
2	5	3	-12	-24	-12	3	5	2
1	4	5	3	0	3	5	4	1
1	2	4	5	5	5	4	2	1
0	1	1	2	2	2	1	1	0

Note: undersampled

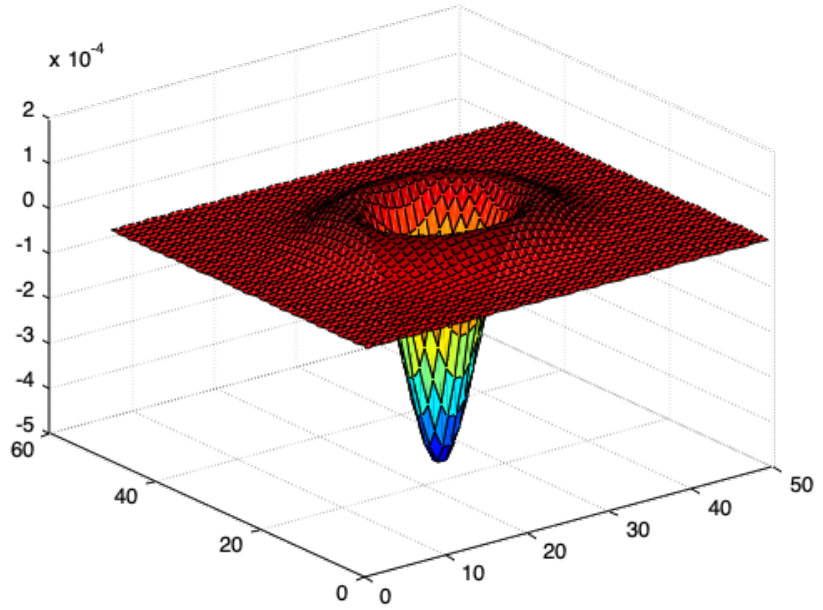


Image processing

- Introduction
- Point operators
- Linear filtering
- Non linear filtering
- The Fourier transform
- Pyramids and scales

Non linear filtering

- Median filtering
  - Select the median value from a neighborhood

1	2	1	2	4
2	1	3	5	8
1	3	7	6	9
3	4	8	6	7
4	5	7	8	9

Non linear filtering

- α-mean
  - Averages all the pixels except for the largest and smallest

1	2	1	2	4
2	1	3	5	8
1	3	7	6	9
3	4	8	6	7
4	5	7	8	9

  - A variation is the weighted mean

Non linear filtering

(a)

(b)

Non linear filtering

(a)

(b)

(c)

Non linear filtering

(a) (b) (c)

(e) (f) (g)

Image processing

- Introduction
- Point operators
- Linear filtering
- Non linear filtering
- The Fourier transform
- Pyramids and scales

The Fourier transform

- Helps us analyze the frequency characteristics

The Fourier transform

- Helps us analyze the frequency characteristics
- Consider a (periodic) signal with fundamental frequency 2π

$$f(x) = \sum_{k=-3}^{3} a_k \exp(ik2\pi x)$$

With

$a_0 = 1$

$a_1 = a_{-1} = 1/4$

$a_2 = a_{-2} = 1/2$

$a_3 = a_{-3} = 1/3$

Source: Richard Wildes slides

The Fourier transform

- Helps us analyze the frequency characteristics
- Consider a (periodic) signal with fundamental frequency 2π

$$f(x) = \sum_{k=-3}^{3} a_k \exp(ik2\pi x)$$

With

$a_0=1$

$a_1=a_{-1}=1/4$

$a_2=a_{-2}=1/2$

$a_3=a_{-3}=1/3$

- A form likes this, “begs” to have common terms collected together $= 1 +$

Source: Richard Wildes slides

The Fourier transform

- Helps us analyze the frequency characteristics
- Consider a (periodic) signal with fundamental frequency 2π

$$f(x) = \sum_{k=-3}^{3} a_k \exp(ik2\pi x)$$

With

$a_0 = 1$

$a_1 = a_{-1} = 1/4$

$a_2 = a_{-2} = 1/2$

$a_3 = a_{-3} = 1/3$

- A form likes this, "begs" to have common terms collected together

$$= 1 + \frac{1}{4}[\exp(i2\pi x) + \exp(-i2\pi x)]$$

Source: Richard Wildes slides

The Fourier transform

- Helps us analyze the frequency characteristics
- Consider a (periodic) signal with fundamental frequency 2π

$$f(x) = \sum_{k=-3}^{3} a_k \exp(ik2\pi x)$$

With

$a_0=1$

$a_1=a_{-1}=1/4$

$a_2=a_{-2}=1/2$

$a_3=a_{-3}=1/3$

- A form likes this, "begs" to have common terms collected together

$$= 1 + \frac{1}{4}[\exp(i2\pi x) + \exp(-i2\pi x)]$$

$$+ \frac{1}{2}[\exp(i4\pi x) + \exp(-i4\pi x)]$$

$$+ \frac{1}{3}[\exp(i6\pi x) + \exp(-i6\pi x)]$$

Source: Richard Wildes slides

The Fourier transform

- We further examine our expansion

$$= 1 + \frac{1}{4}[\exp(i2\pi x) + \exp(-i2\pi x)]$$
$$+ \frac{1}{2}[\exp(i4\pi x) + \exp(-i4\pi x)]$$
$$+ \frac{1}{3}[\exp(i6\pi x) + \exp(-i6\pi x)]$$

- And note that we can cancel terms inside the grouped exponents via Euler's relation to yield

$$e^{ix} = cos(x) + isin(x)$$

Source: Richard Wildes slides

The Fourier transform

- We further examine our expansion

$$= 1 + \frac{1}{4}[\exp(i2\pi x) + \exp(-i2\pi x)]$$
$$+ \frac{1}{2}[\exp(i4\pi x) + \exp(-i4\pi x)]$$
$$+ \frac{1}{3}[\exp(i6\pi x) + \exp(-i6\pi x)]$$

- And note that we can cancel terms inside the grouped exponents via Euler's relation to yield

$$= 1$$
$$+ \frac{1}{2}\cos 2\pi x$$

$$e^{ix} = cos(x) + isin(x)$$

Source: Richard Wildes slides

The Fourier transform

- We further examine our expansion

$$= 1 + \frac{1}{4}[\exp(i2\pi x) + \exp(-i2\pi x)]$$
$$+ \frac{1}{2}[\exp(i4\pi x) + \exp(-i4\pi x)]$$
$$+ \frac{1}{3}[\exp(i6\pi x) + \exp(-i6\pi x)]$$

- And note that we can cancel terms inside the grouped exponents via Euler's relation to yield

$$= 1$$
$$+ \frac{1}{2}\cos 2\pi x$$
$$+ \cos 4\pi x$$
$$+ \frac{2}{3}\cos 6\pi x$$

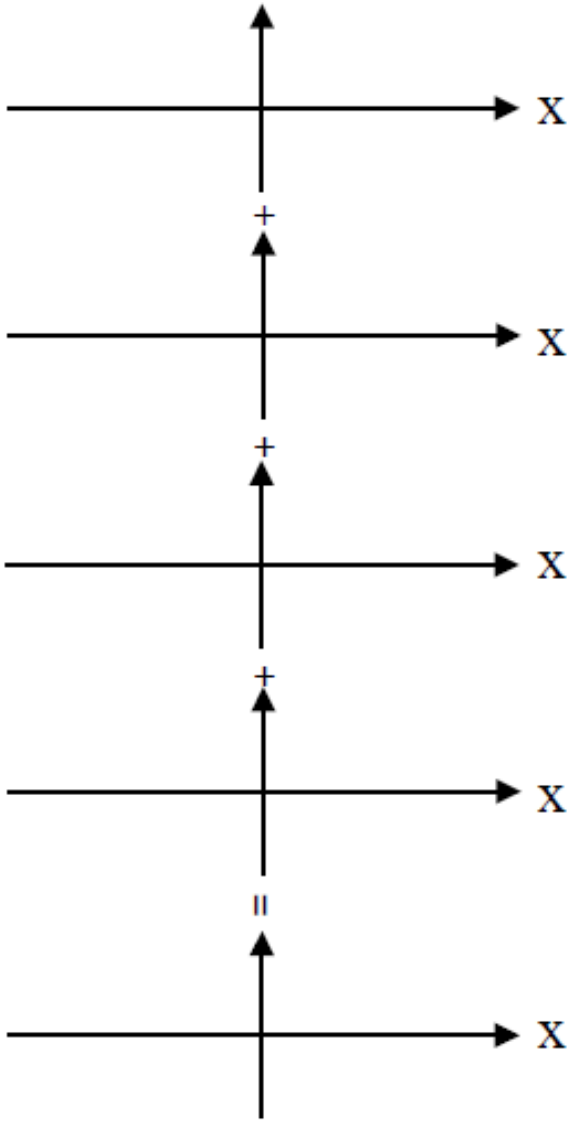
$$e^{ix} = cos(x) + isin(x)$$

Source: Richard Wildes slides

The Fourier transform

- Let's look at a graphical interpretation

$$
\begin{aligned}
& 1 \\
& + \frac{1}{2}\cos 2\pi x \\
& + \cos 4\pi x \\
& + \frac{2}{3}\cos 6\pi x \\
& = f(x)
\end{aligned}
$$



Source: Richard Wildes slides

The Fourier transform

- Let's look at a graphical interpretation

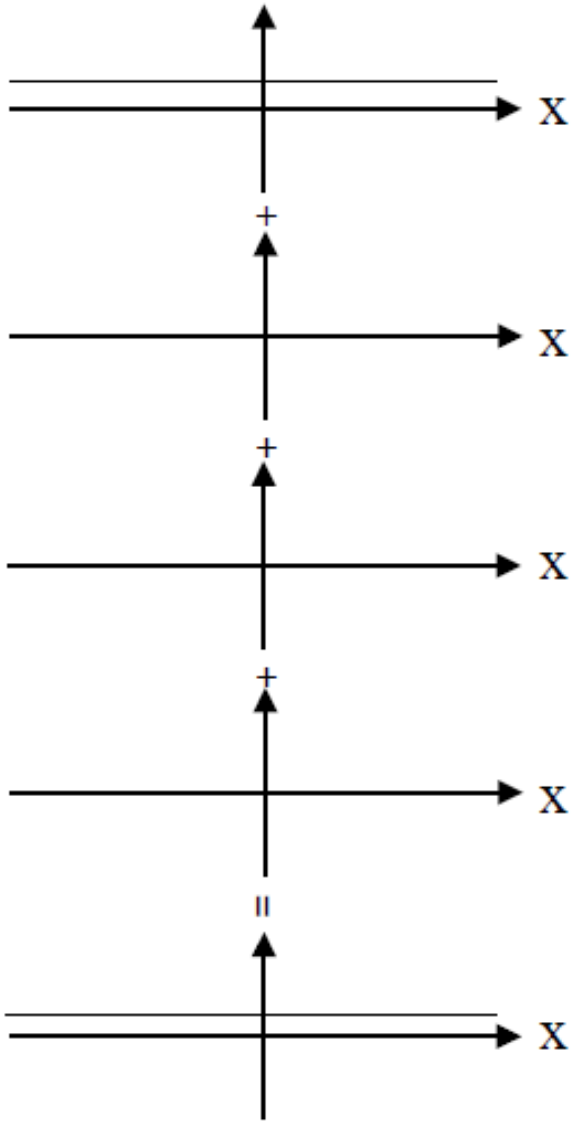
$$1$$

$$+\frac{1}{2}\cos 2\pi x$$

$$+\cos 4\pi x$$

$$+\frac{2}{3}\cos 6\pi x$$

$$= f(x)$$



Source: Richard Wildes slides

The Fourier transform

- Let's look at a graphical interpretation

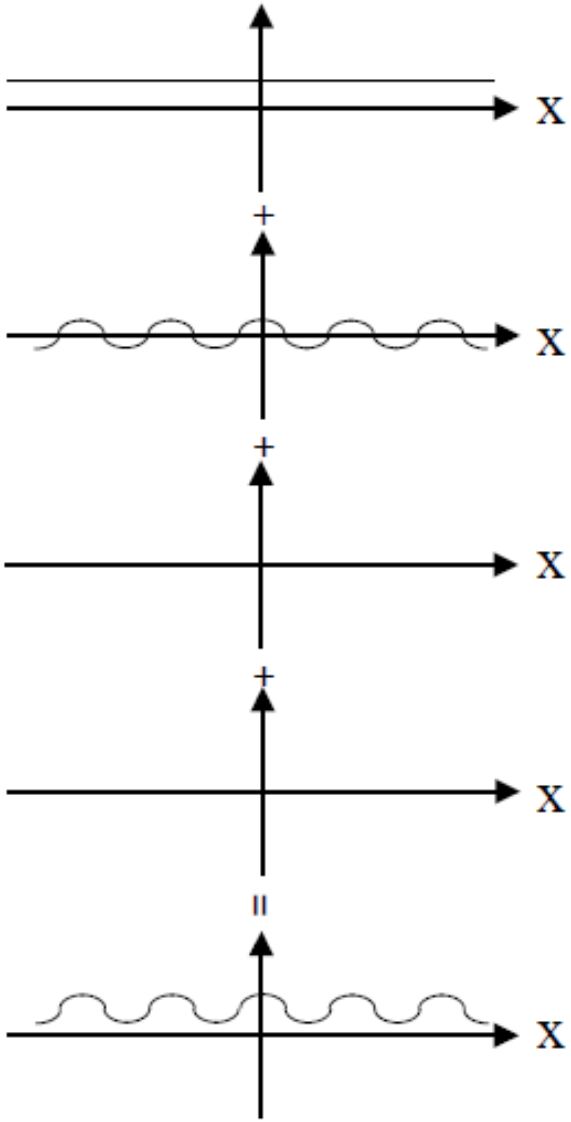
$$1$$

$$+\frac{1}{2}\cos 2\pi x$$

$$+\cos 4\pi x$$

$$+\frac{2}{3}\cos 6\pi x$$

$$= f(x)$$



Source: Richard Wildes slides

The Fourier transform

- Let's look at a graphical interpretation

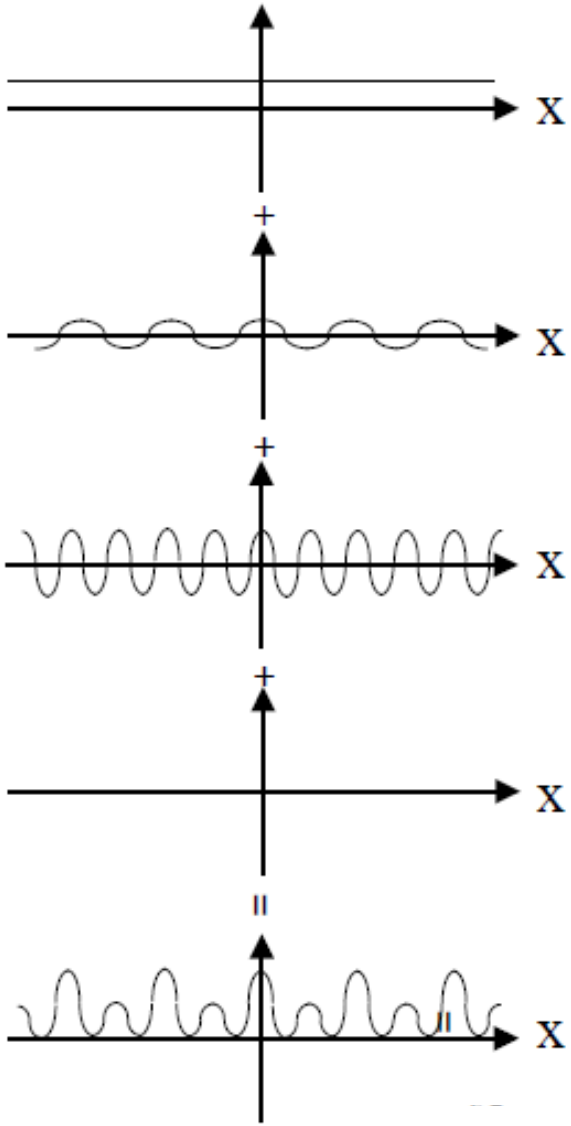
$$1$$

$$+\frac{1}{2}\cos 2\pi x$$

$$+\cos 4\pi x$$

$$+\frac{2}{3}\cos 6\pi x$$

$$= f(x)$$



Source: Richard Wildes slides

The Fourier transform

- Let's look at a graphical interpretation

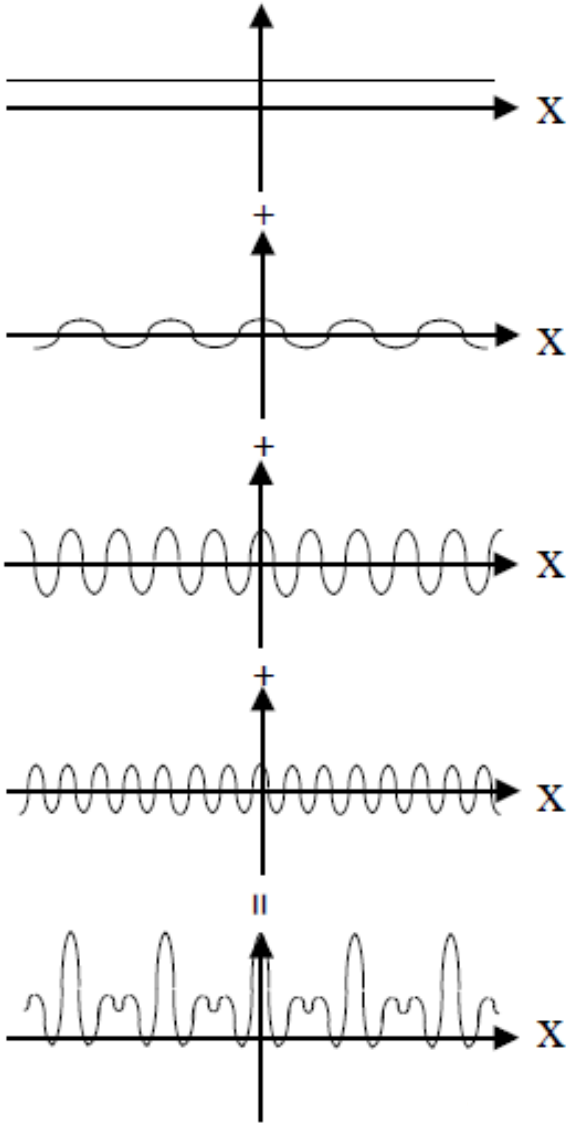
$$1$$

$$+\frac{1}{2}\cos 2\pi x$$

$$+\cos 4\pi x$$

$$+\frac{2}{3}\cos 6\pi x$$

$$= f(x)$$



Source: Richard Wildes slides

The Fourier transform

- Let's look at a graphical interpretation

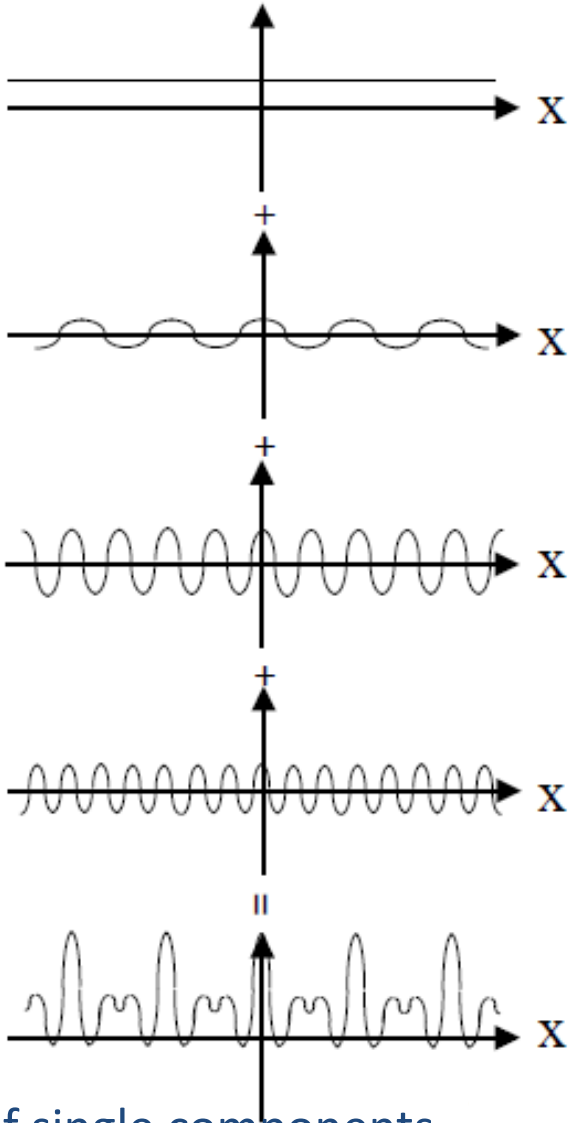
$$1$$

$$+\frac{1}{2}\cos 2\pi x$$

$$+\cos 4\pi x$$

$$+\frac{2}{3}\cos 6\pi x$$

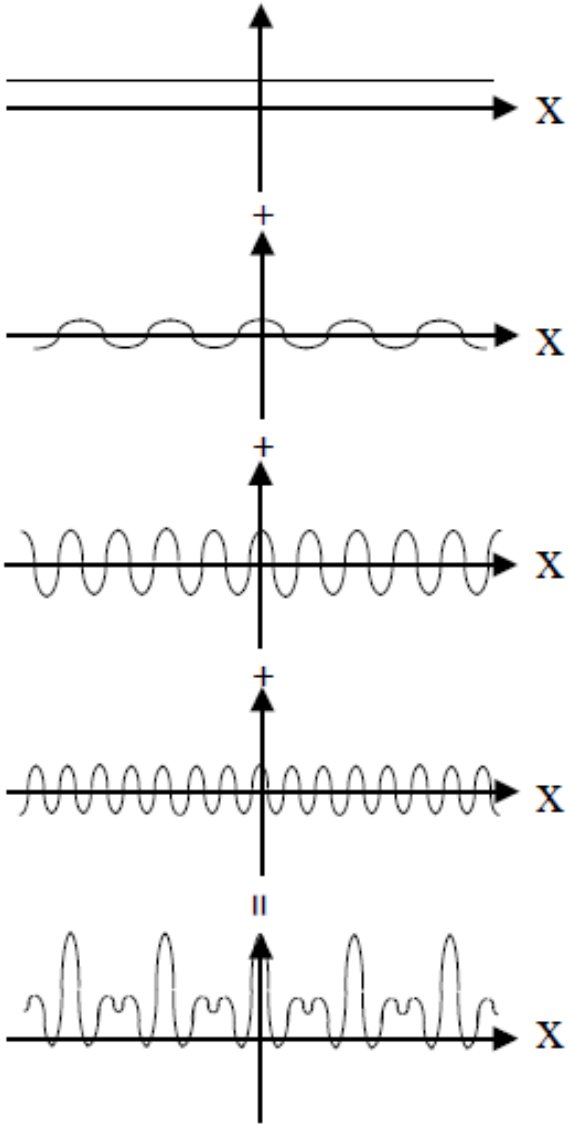
$$= f(x)$$



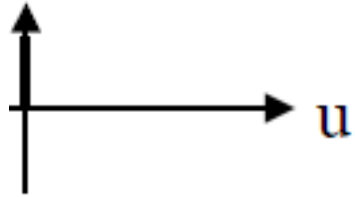
Complicated signals can be represented as the sum of single components

Source: Richard Wildes slides

The Fourier transform

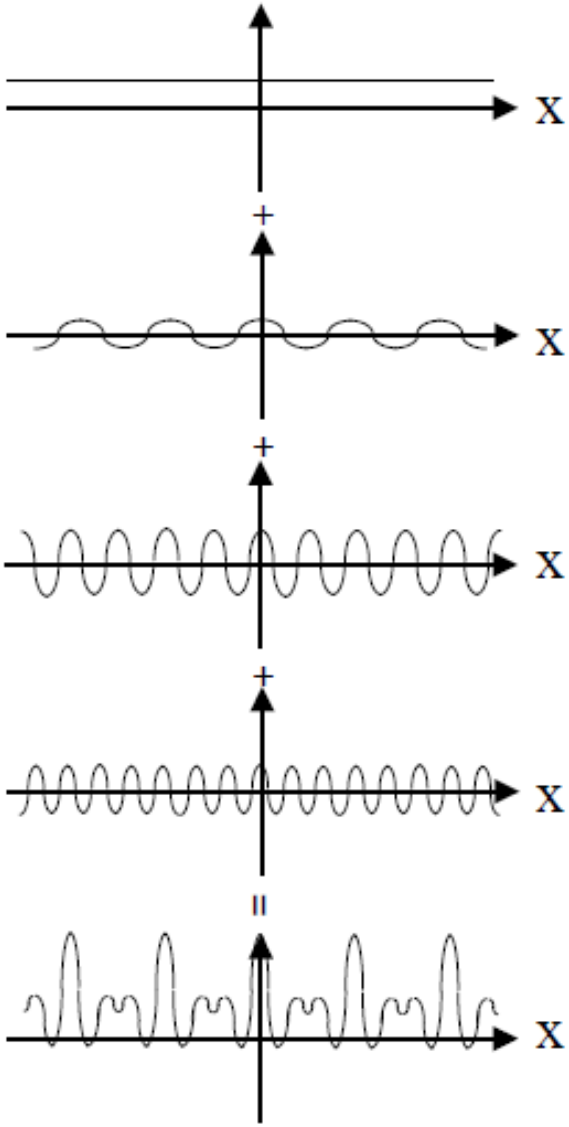


$$1\cos 0\pi x \longrightarrow 1\cos ux, u = 0$$



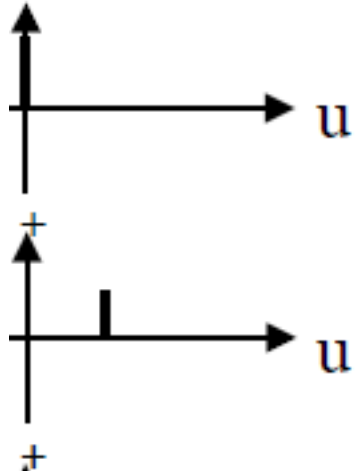
Source: Richard Wildes slides

The Fourier transform



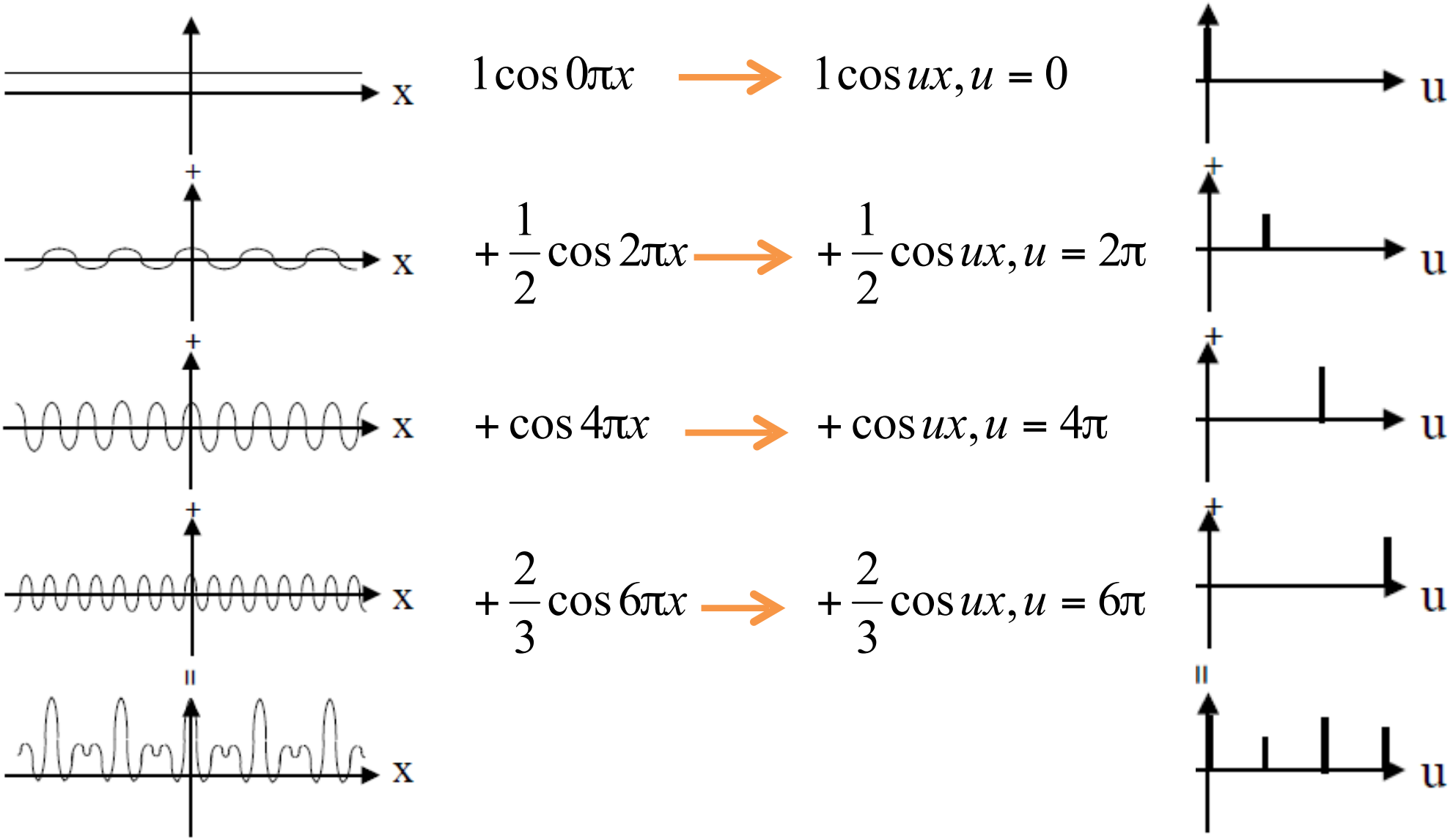
$$1\cos 0\pi x \rightarrow 1\cos ux, u = 0$$

$$+\frac{1}{2}\cos 2\pi x \rightarrow +\frac{1}{2}\cos ux, u = 2\pi$$



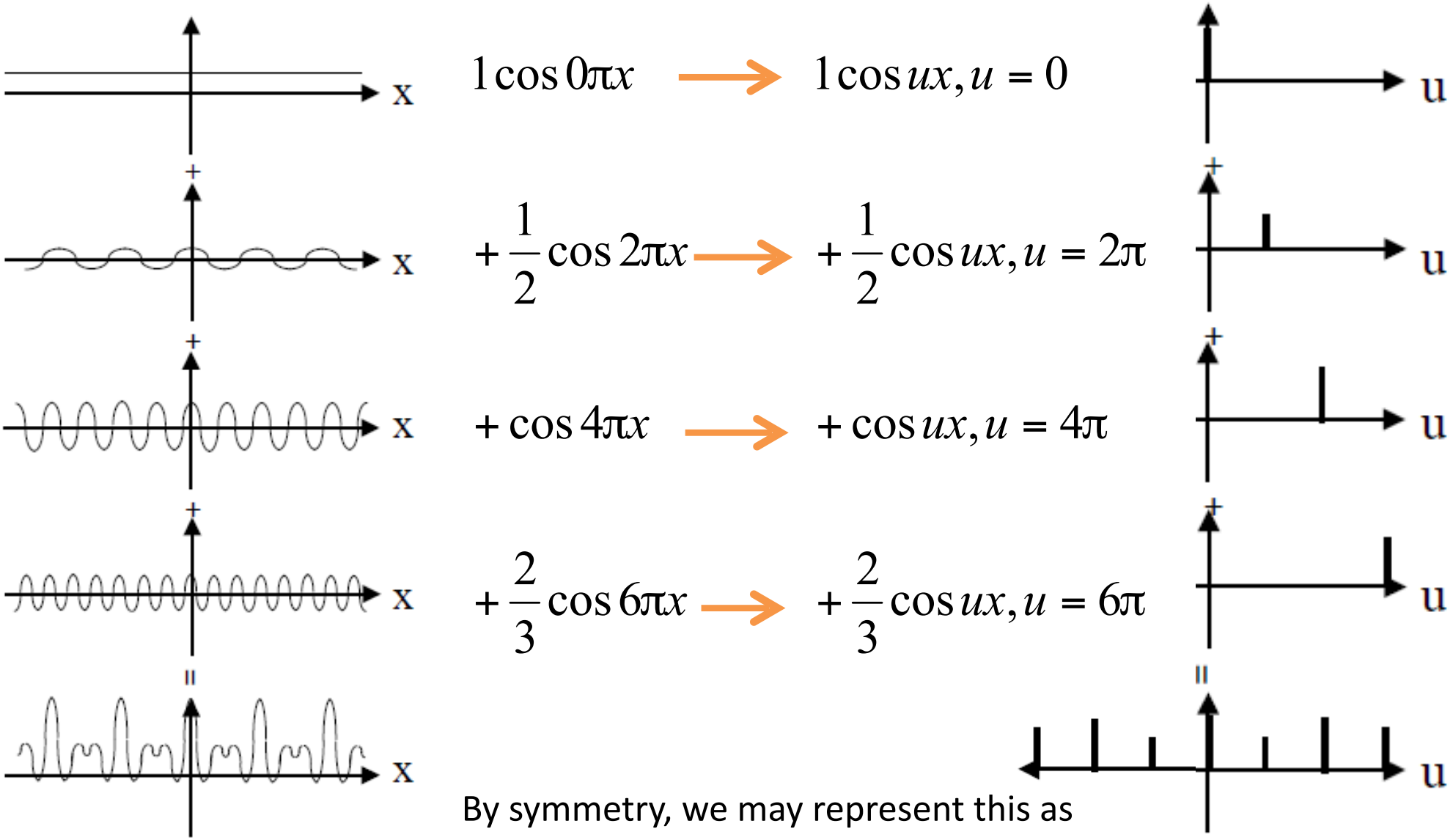
Source: Richard Wildes slides

The Fourier transform



Source: Richard Wildes slides

The Fourier transform



By symmetry, we may represent this as

Source: Richard Wildes slides

The Fourier transform

- An input, ***f(x,y)***, can be considered as the sum of an infinite number of sinusoidal waves.

$$f(x,y) = \frac{1}{4\pi^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u,v)\exp[i(ux+vy)]dudv$$

- We can obtain ***F(u,v)*** given ***f(x,y)***

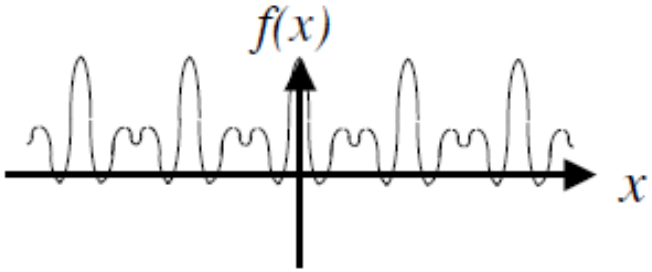
$$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\exp[-i(ux+vy)]dxdy$$

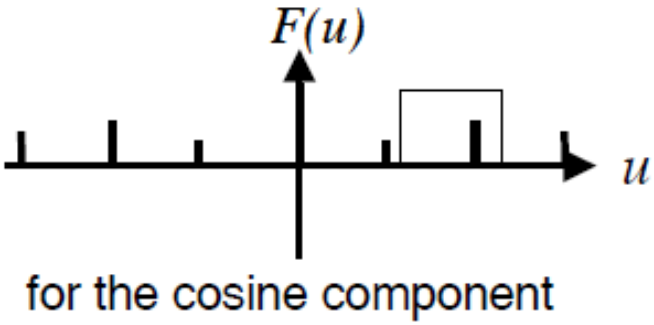
- We call ***F(u,v)*** the **Fourier Transform** of ***f(x,y)***

Source: Richard Wildes slides

The Fourier transform

- Images are 2D
- Recall the 1D example

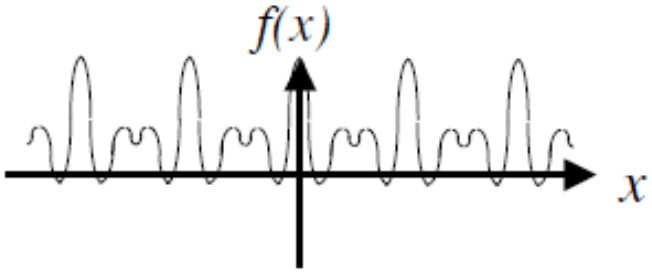


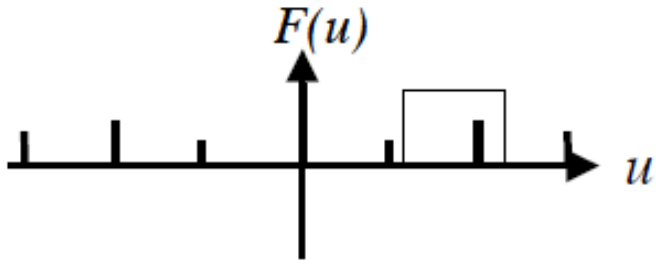


Source: Richard Wildes slides

The Fourier transform

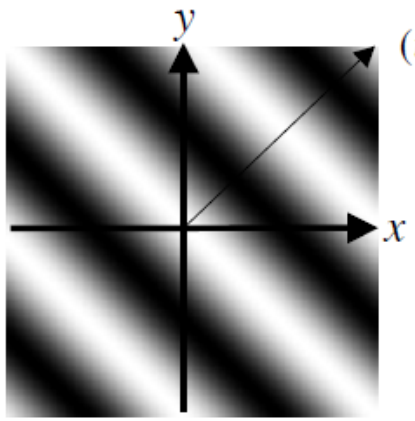
- Images are 2D
- Recall the 1D example





for the cosine component

- The interpretation in 2D spatial frequency



$(u,v)/|(u,v)|$

The maxima and minima of the cosinusoids lie along parallel equidistant lines $ux+vy=k\pi$ for k an integer.

Source: Richard Wildes slides

The Fourier transform

- The interpretation in 2D spatial frequency
  - To get some sense of what basis elements look like, we plot a basis element --- or rather, its real part as a function of x,y for some fixed u,v. We get a function that is constant when $(ux+vy)$ is constant. The magnitude of the vector (u,v) gives a frequency, and its direction gives an orientation. The function is a sinusoid with this frequency along the direction, and constant perpendicular to the direction.

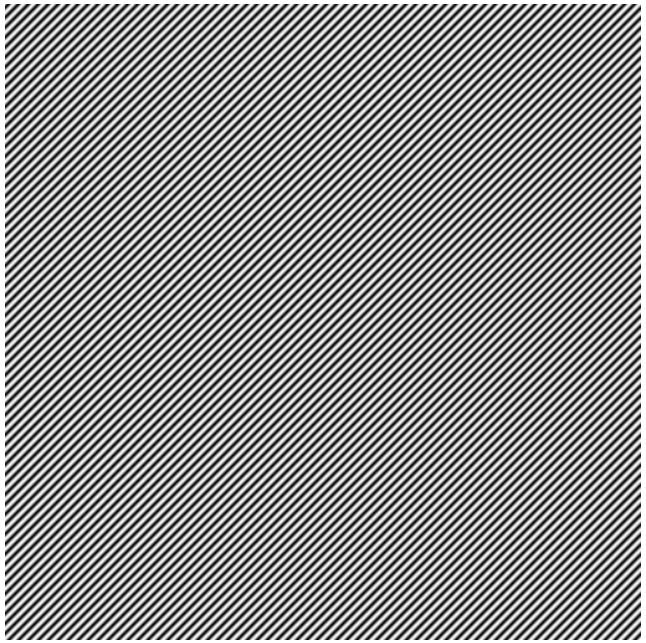
The Fourier transform

- The interpretation in 2D spatial frequency
  - To get some sense of what basis elements look like, we plot a basis element --- or rather, its real part as a function of x,y for some fixed u,v. We get a function that is constant when $(ux+vy)$ is constant. The magnitude of the vector (u,v) gives a frequency, and its direction gives an orientation. The function is a sinusoid with this frequency along the direction, and constant perpendicular to the direction.

Which (u,v)
Is larger?

The Fourier transform

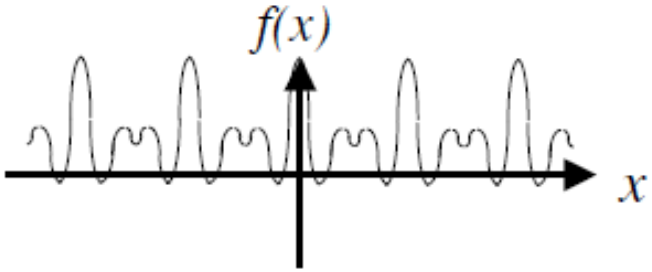
- The interpretation in 2D spatial frequency
  - To get some sense of what basis elements look like, we plot a basis element --- or rather, its real part as a function of x,y for some fixed u,v. We get a function that is constant when $(ux+vy)$ is constant. The magnitude of the vector (u,v) gives a frequency, and its direction gives an orientation. The function is a sinusoid with this frequency along the direction, and constant perpendicular to the direction.

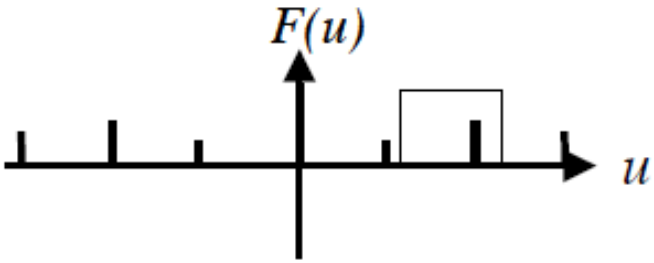


(u,v) even larger

The Fourier transform

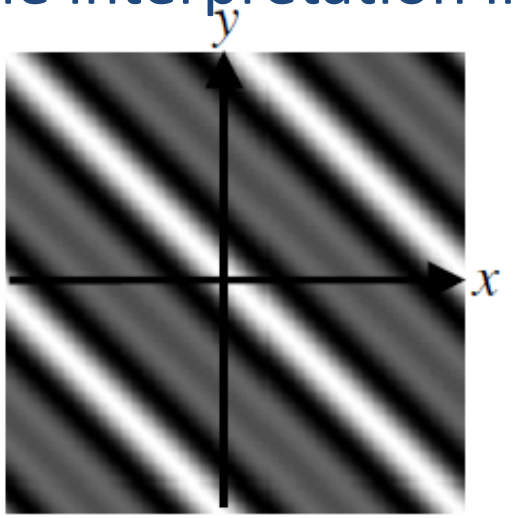
- Images are 2D
- Recall the 1D example





for the cosine component

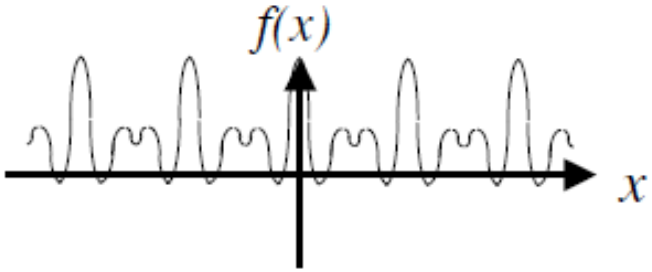
- The interpretation in 2D spatial frequency

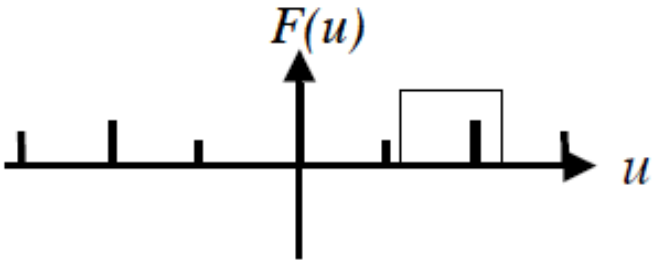


Source: Richard Wildes slides

The Fourier transform

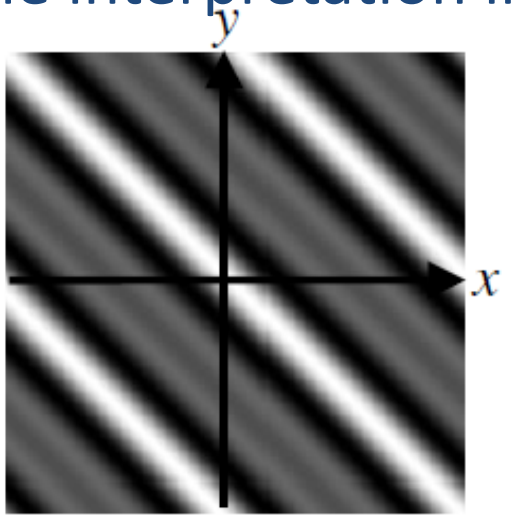
- Images are 2D
- Recall the 1D example

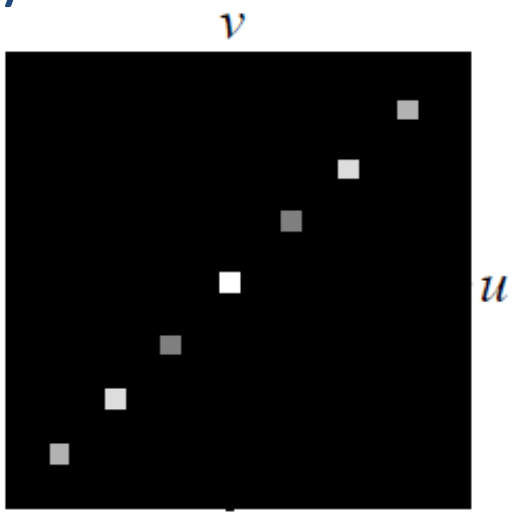




for the cosine component

- The interpretation in 2D spatial frequency





Source: Richard Wildes slides

The Fourier transform

- Phase and amplitude

The Fourier transform

- Phase and amplitude
  - *f(x,y)* is real
  - But *F(u,v)* is complex

$$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\exp[-i(ux+vy)]dxdy$$

The Fourier transform

- Phase and amplitude
  - *f(x,y)* is real
  - But *F(u,v)* is complex

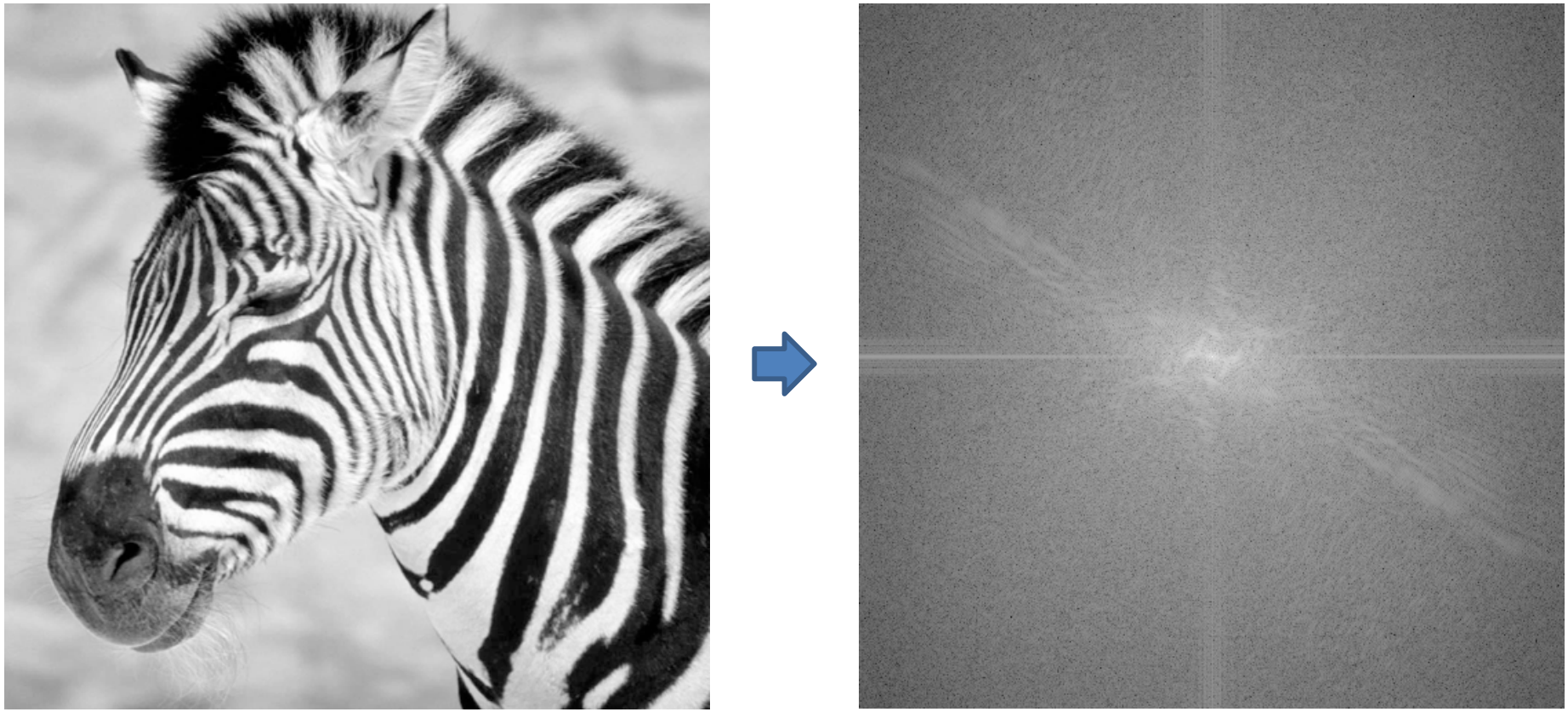
$$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\exp[-i(ux+vy)]dxdy$$

$$Magnitude(F) = \sqrt{\mathrm{Re}(F)^2 + \mathrm{Im}(F)^2}$$

$$Phase(F) = \tan^{-1}\left(\frac{\mathrm{Im}(F)}{\mathrm{Re}(F)}\right)$$

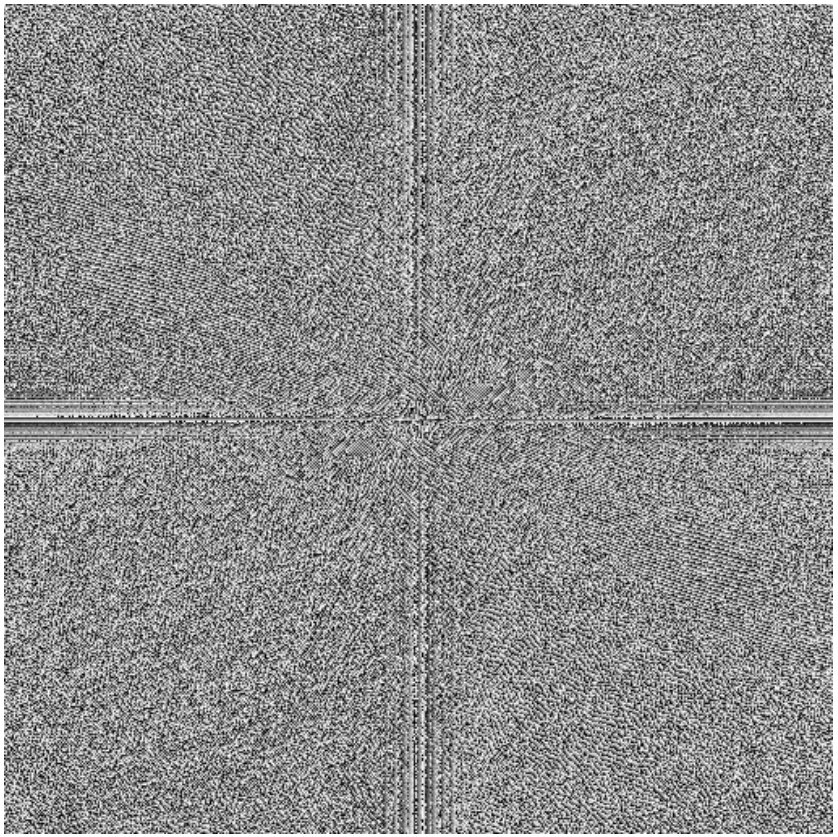
The Fourier transform

- Phase and **amplitude**



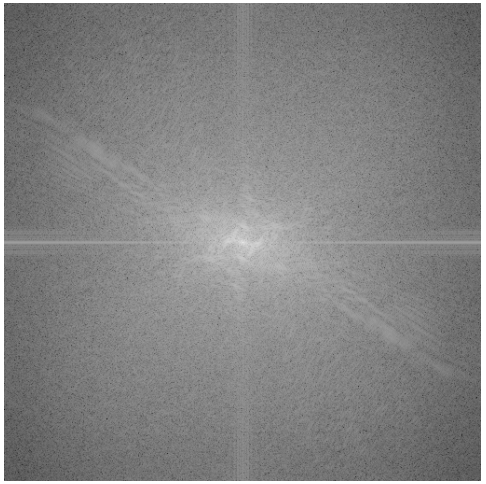
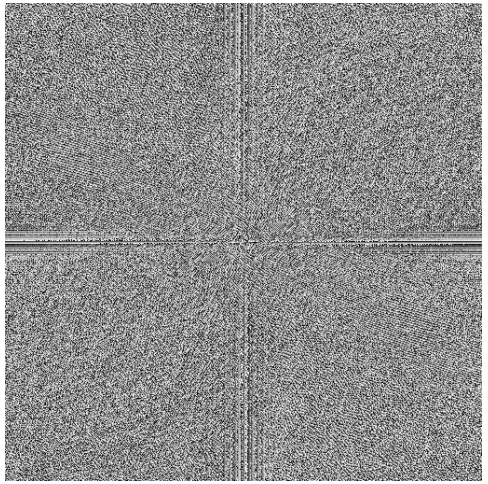
The Fourier transform

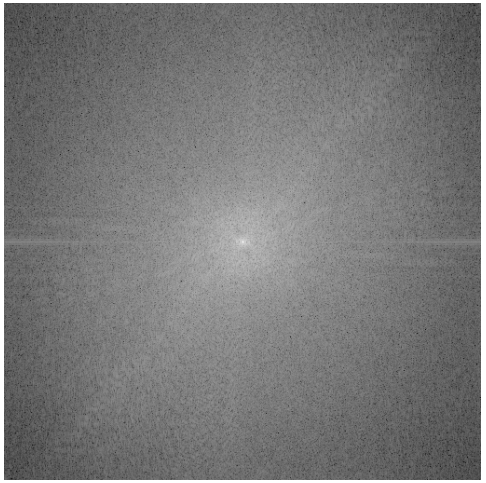
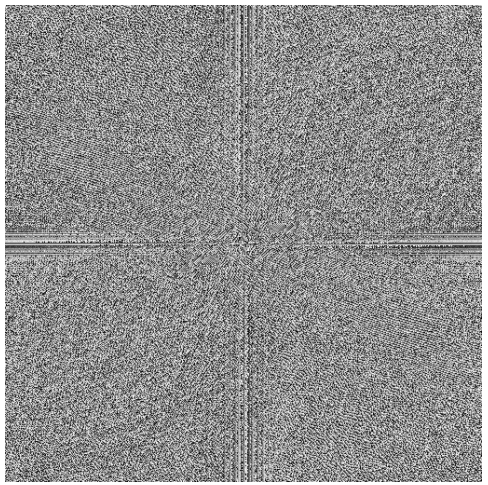
- **Phase** and amplitude



The Fourier Transform

- Phase and amplitude

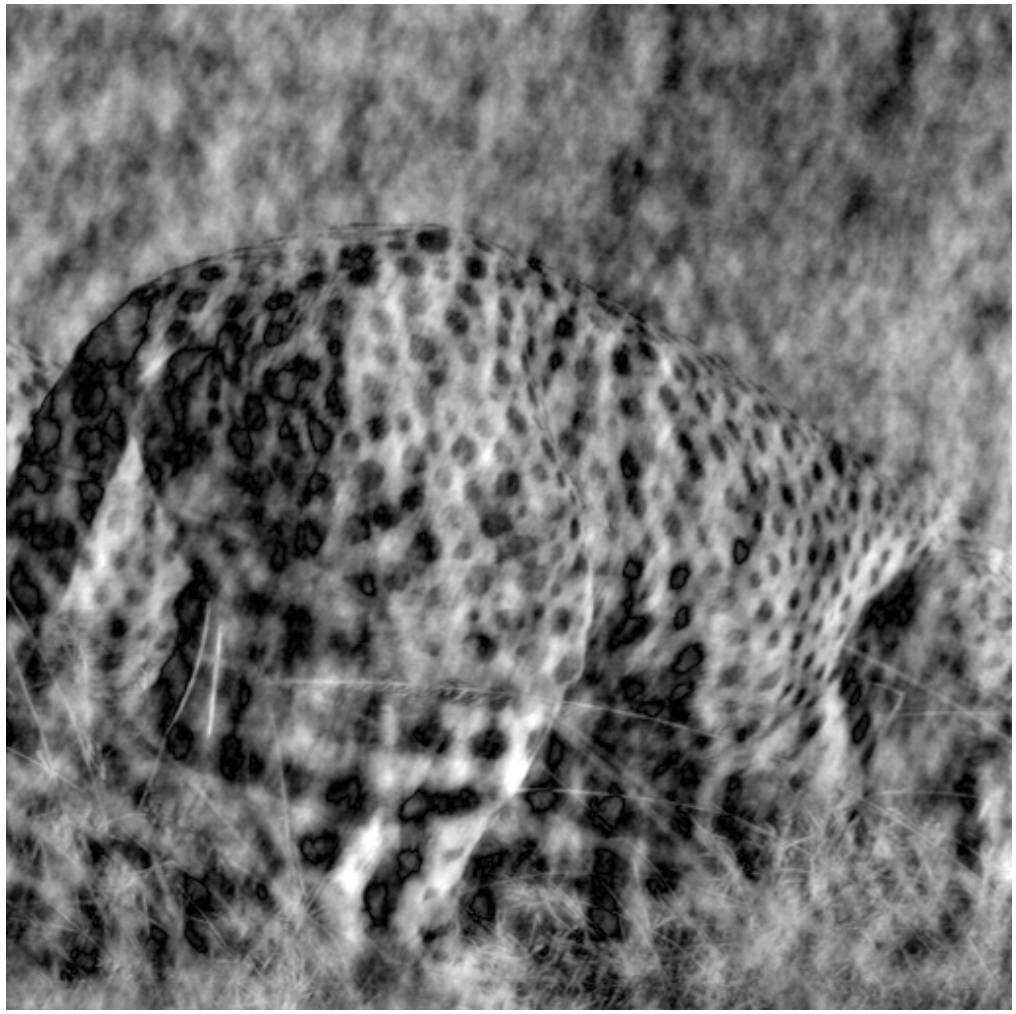



The Fourier Transform

- Reconstruction with zebra phase and cheetah magnitude

The Fourier Transform

- Reconstruction with cheetah phase and zebra magnitude



The Fourier Transform

- Properties

Superposition	$f_1(x)+f_2(x)$	$F_1(u)+F_2(v)$
Shift	$f(x-x_0)$	$F(u)e^{-iux0}$
Reversal	$f(-x)$	$\overline{F(u)}$
Convolution	$f(x) * h(x)$	$F(u)H(u)$
Correlation	$f(x) \otimes h(x)$	$F(u)\overline{H(u)}$
Multiplication	$f(x)h(x)$	$F(u) * H(u)$
Differentiation	$f'(x)$	$iuF(u)$
Domain scaling	$f(ax)$	$1/aF(u/a)$
Conjugation	If $f(x)=\overline{f(x)}$	$F(u)=F(-u)$
Parseval's Theorem	$\sum_x [f(x)]^2$	$\sum_u [F(u)]^2$

The Fourier Transform

- Discrete Fourier Transform (DFT)

$$F(u,v) = \frac{1}{NM} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} f(x,y) \exp[-2\pi i((ux+vy)/NM)]$$

The Fourier Transform

- Discrete Fourier Transform (DFT)

$$F(u,v) = \frac{1}{NM} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} f(x,y) \exp[-2\pi i((ux+vy)/NM)]$$

  - Takes $O(N^2)$
- Normally, for implementation, we use the Fast Fourier Transform (FFT)
  - Takes $O(N\log_2 N)$

The Fourier Transform

- Summary

$$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\exp[-i(ux+vy)]dxdy$$

Fourier transform

Spatial domain

- $f(x,y)$
- Convolution
- Multiplication

Frequency domain

- $F(u,v)$
- Multiplication
- Convolution

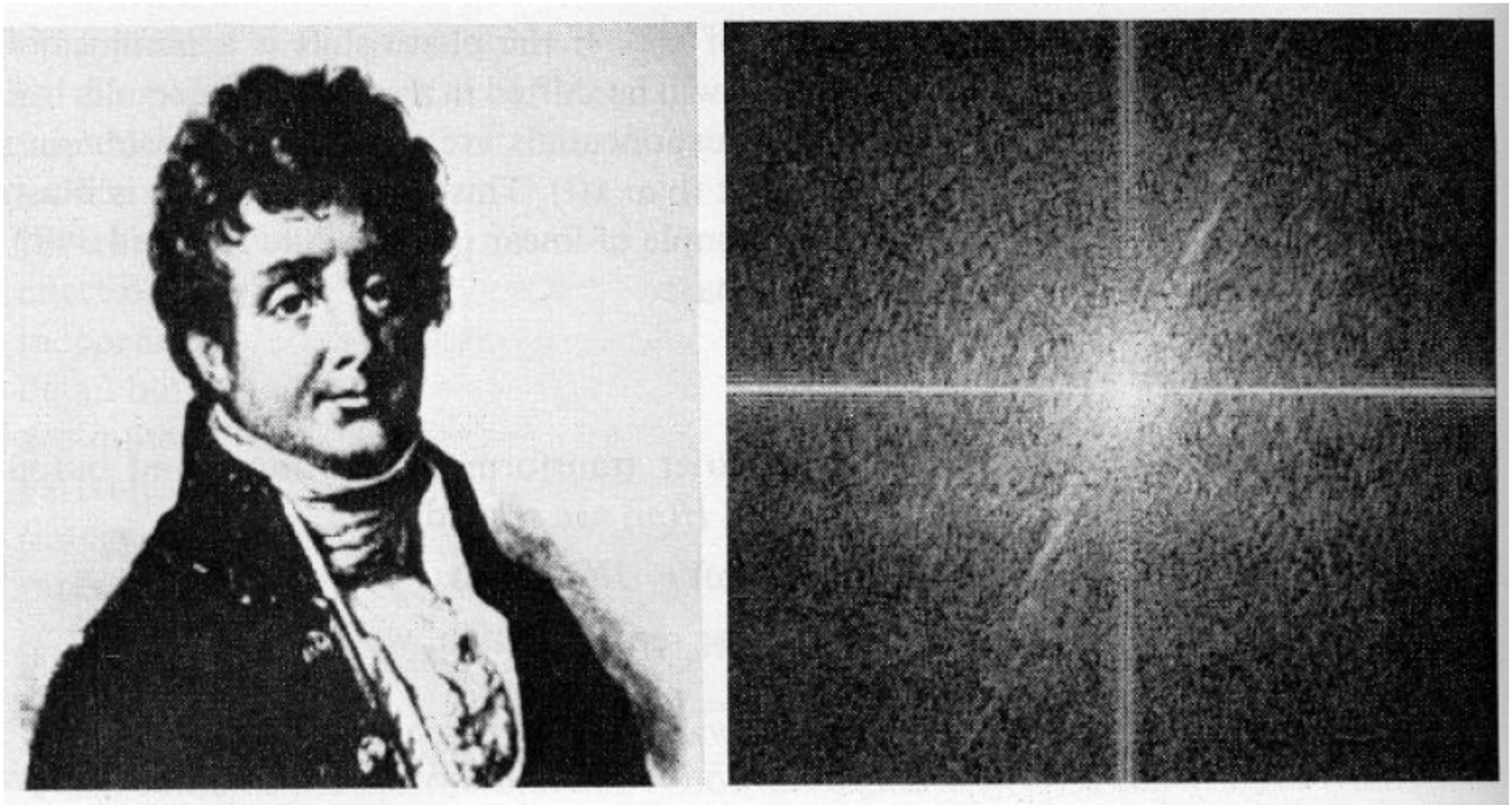
Inverse Fourier transform

$$f(x,y) = \frac{1}{4\pi^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u,v)\exp[i(ux+vy)]dudv$$

Source: Richard Wildes slides

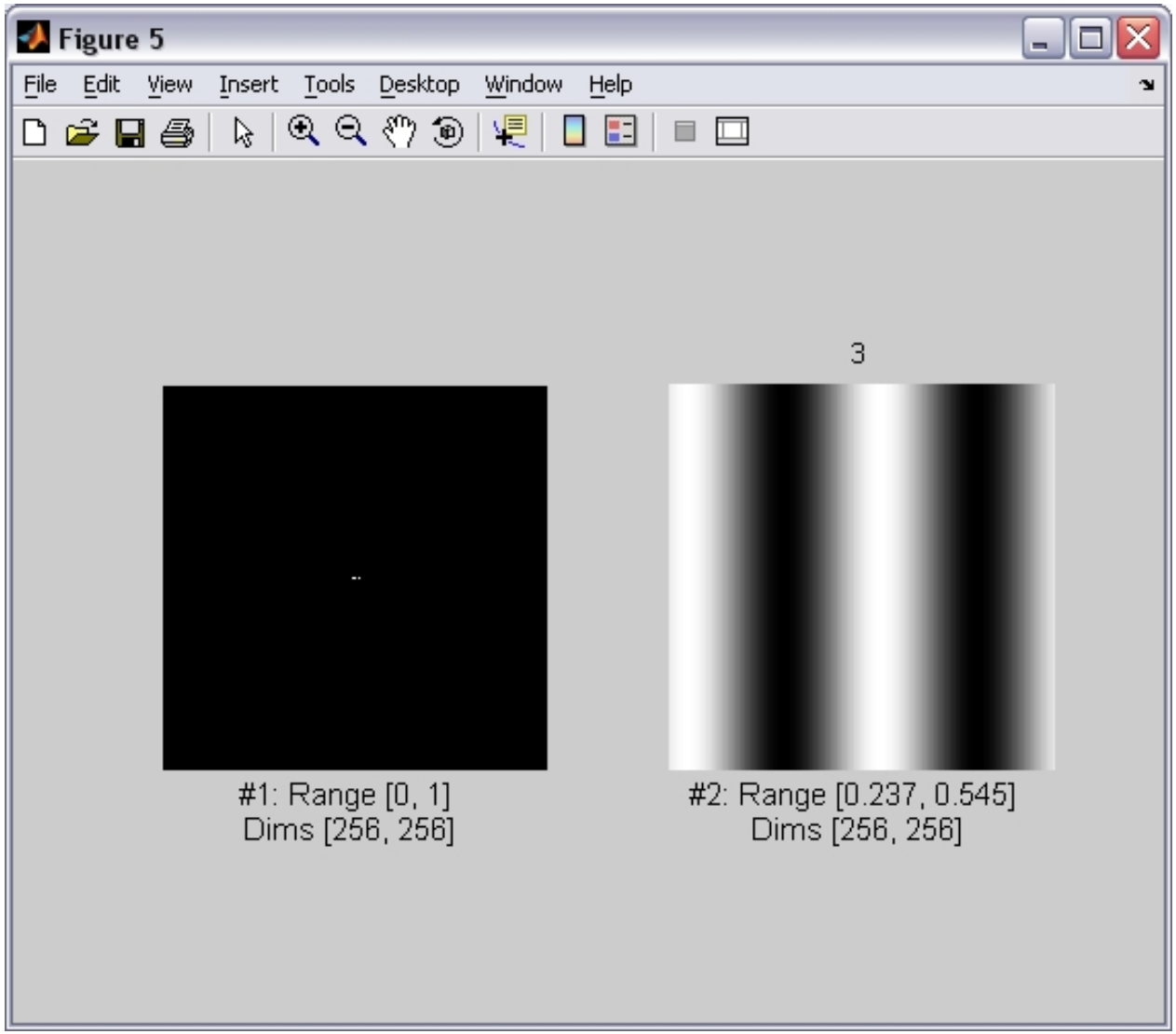
The Fourier Transform

- Summary



Source: Richard Wildes slides

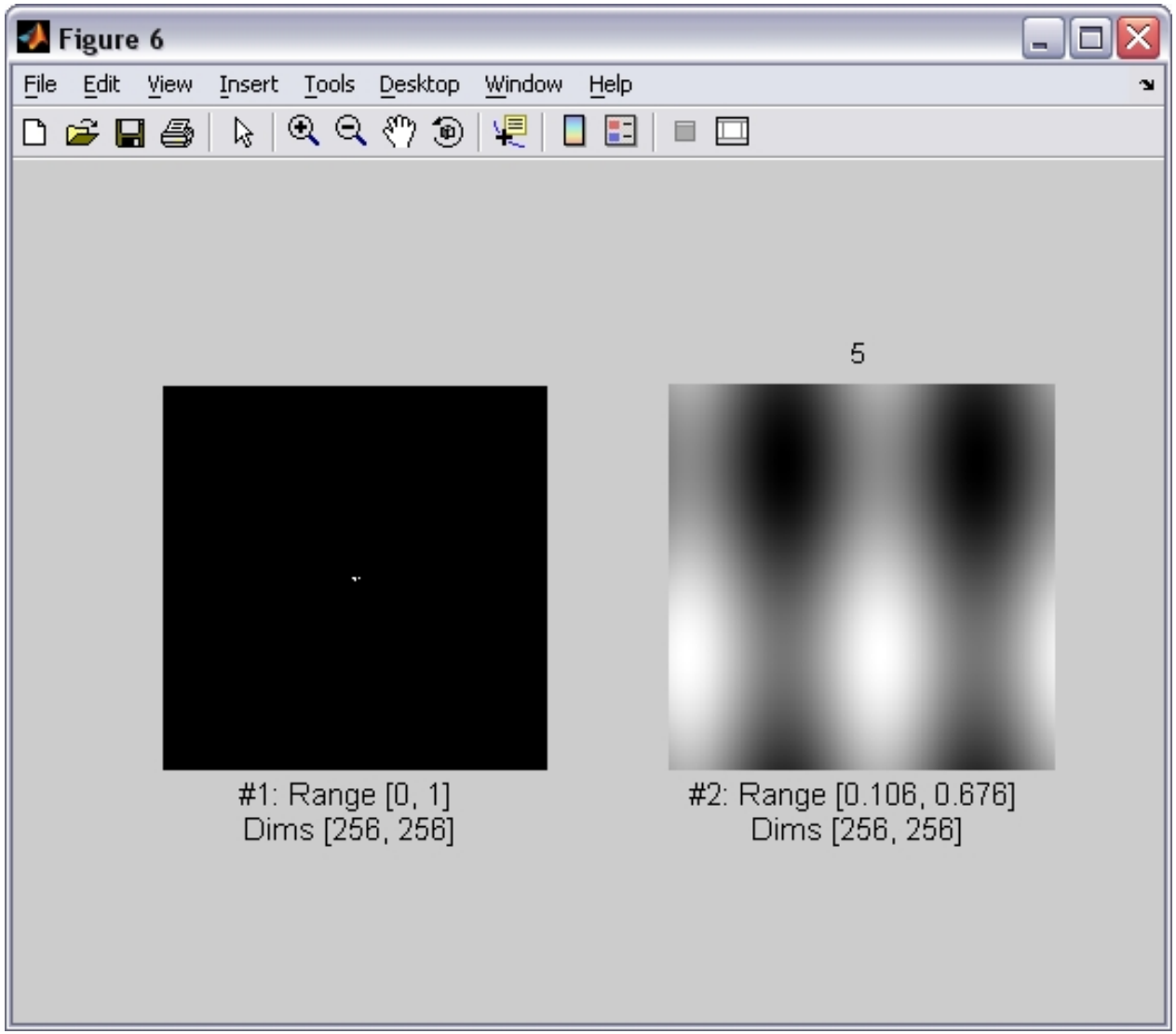
3



Select Fourier components in descending order of magnitude.

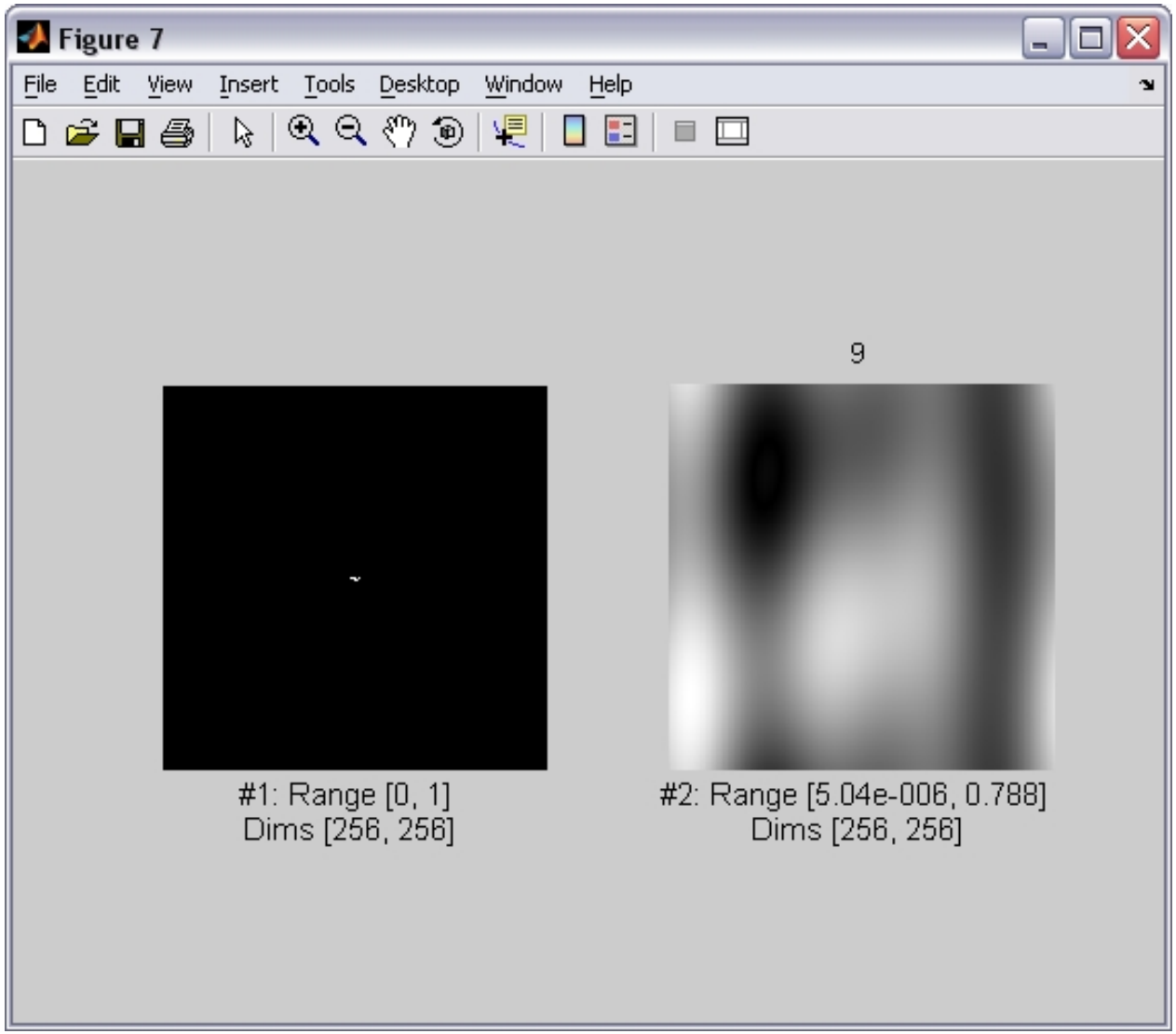
Source: Antonio Torralba slides

5



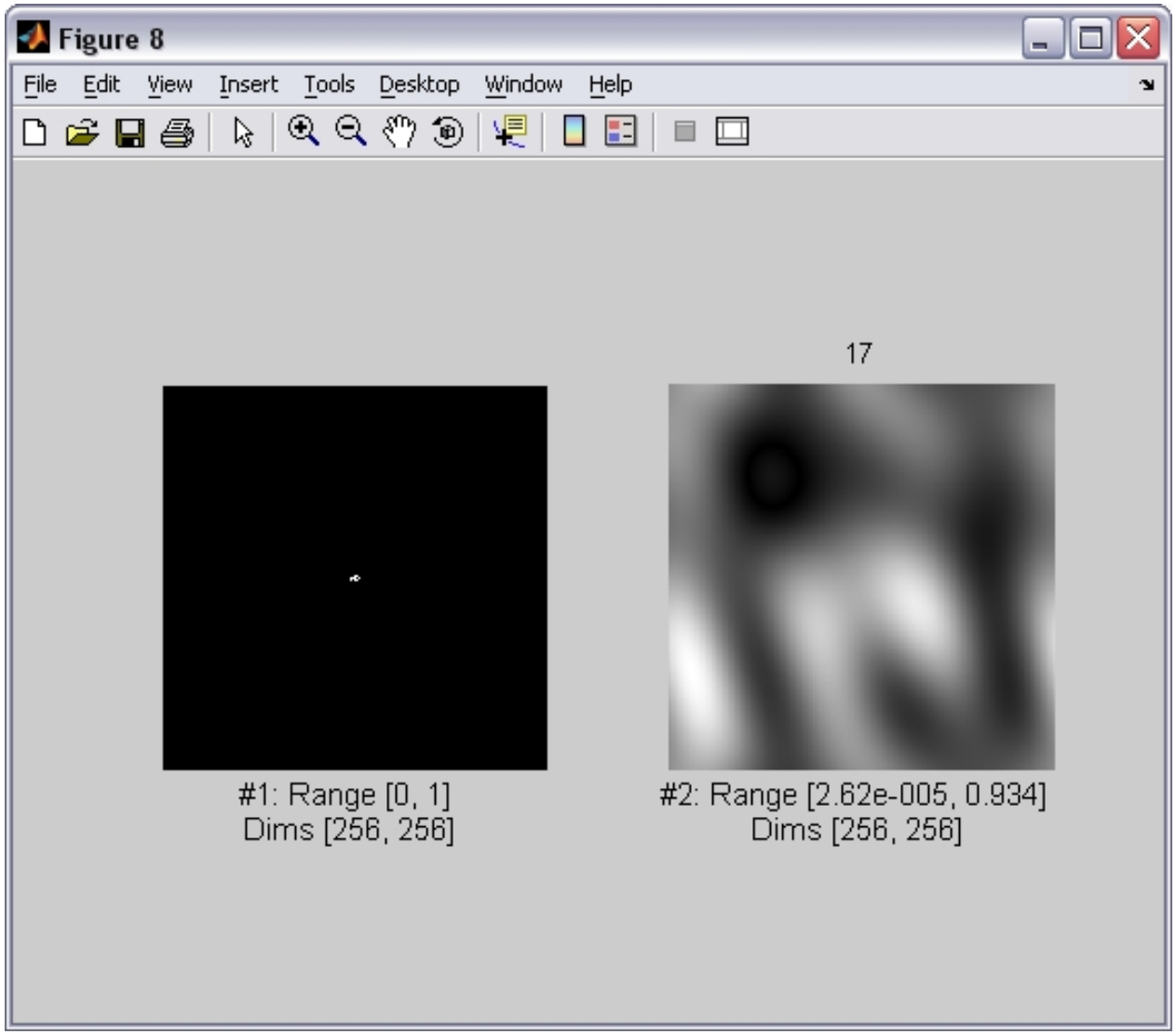
Source: Antonio Torralba slides

9



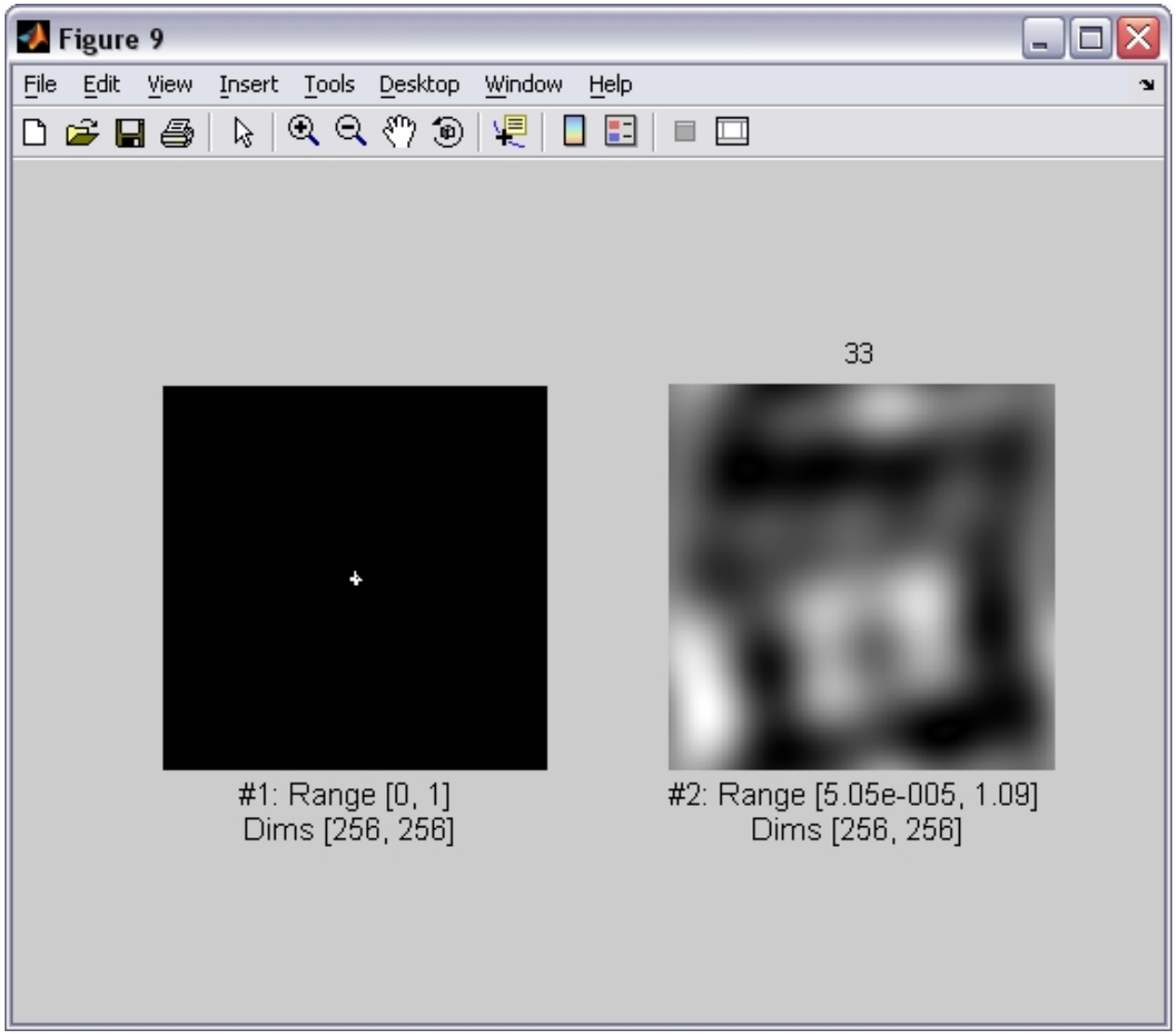
Source: Antonio Torralba slides

17



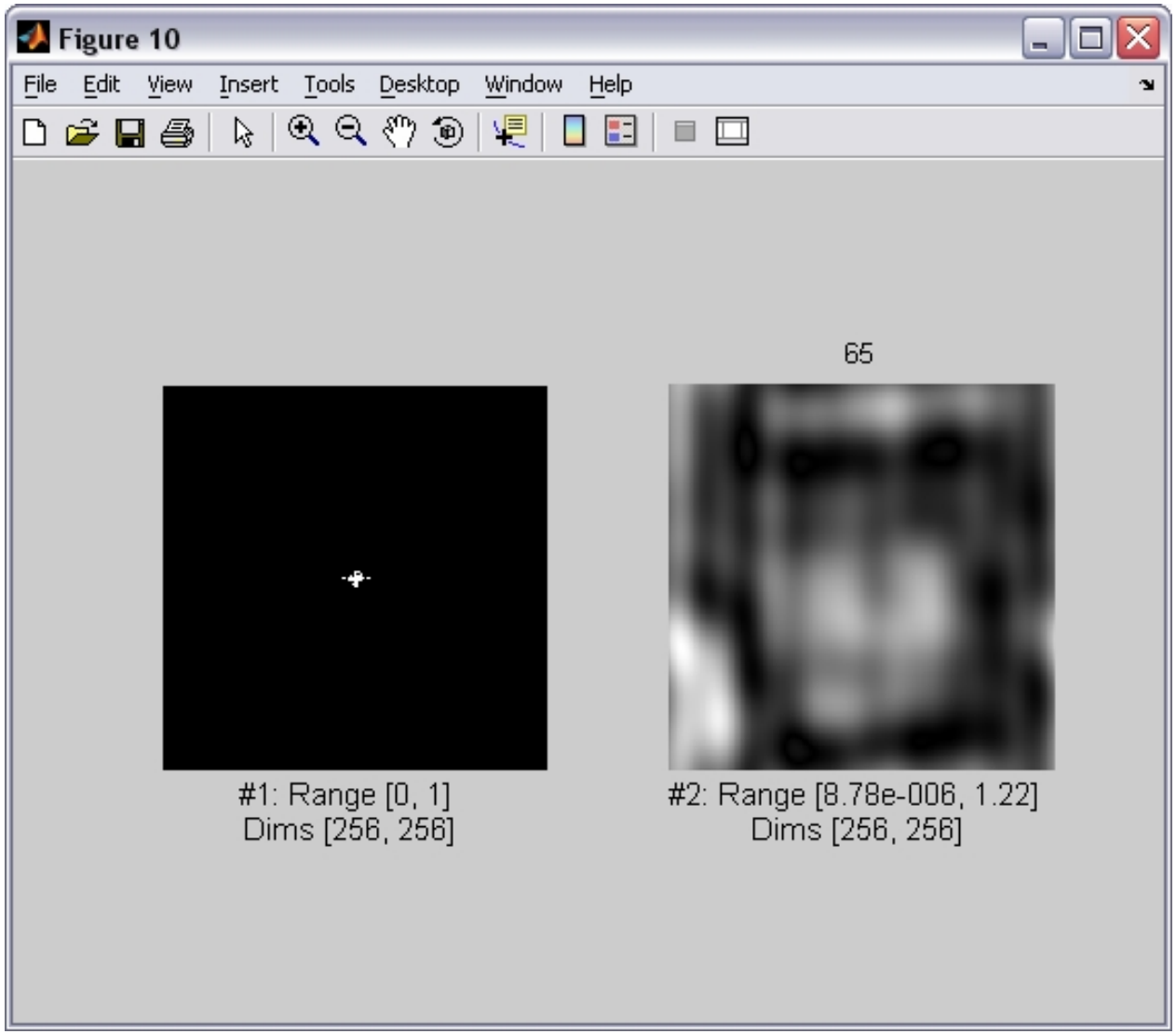
Source: Antonio Torralba slides

33



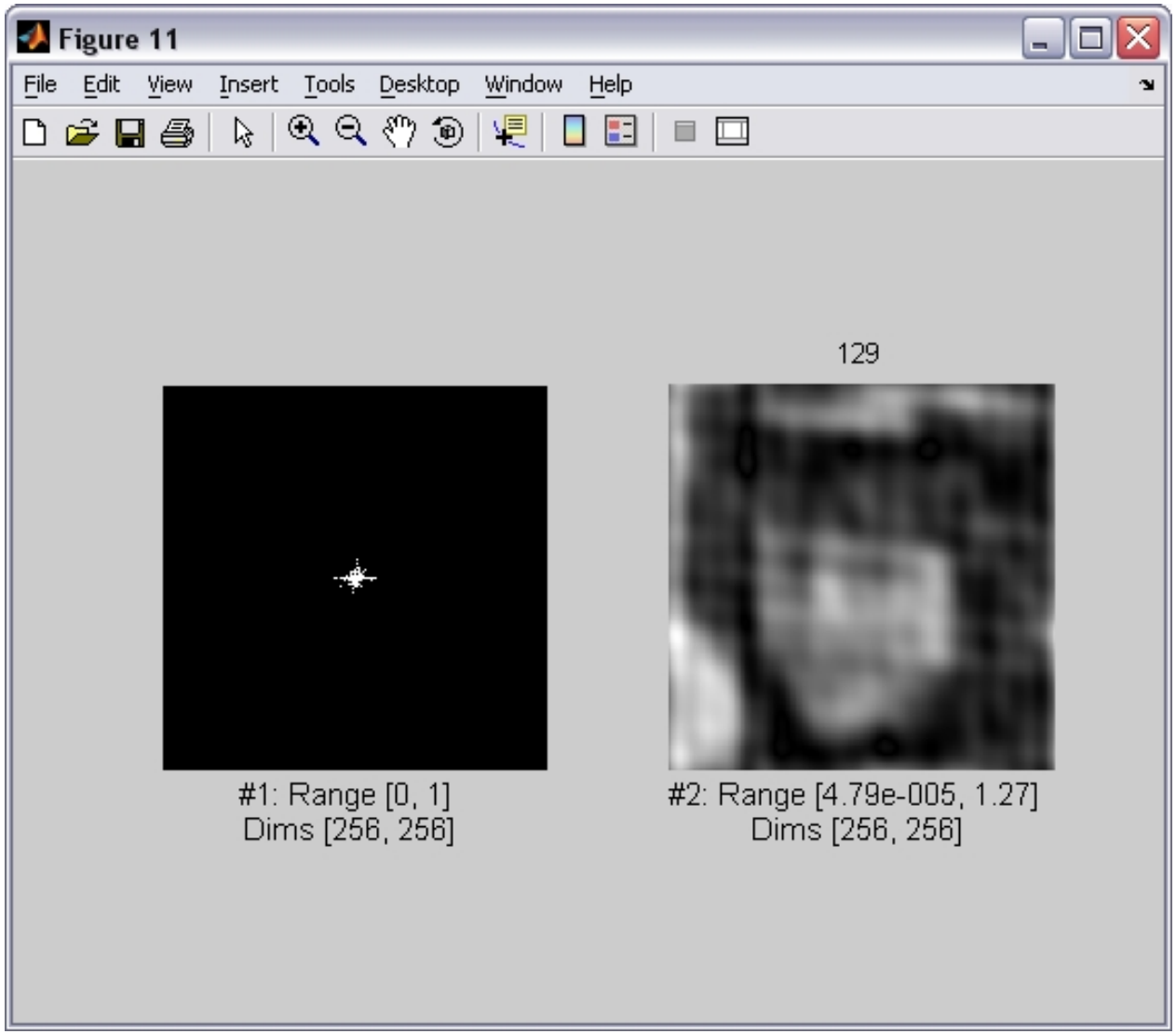
Source: Antonio Torralba slides

65



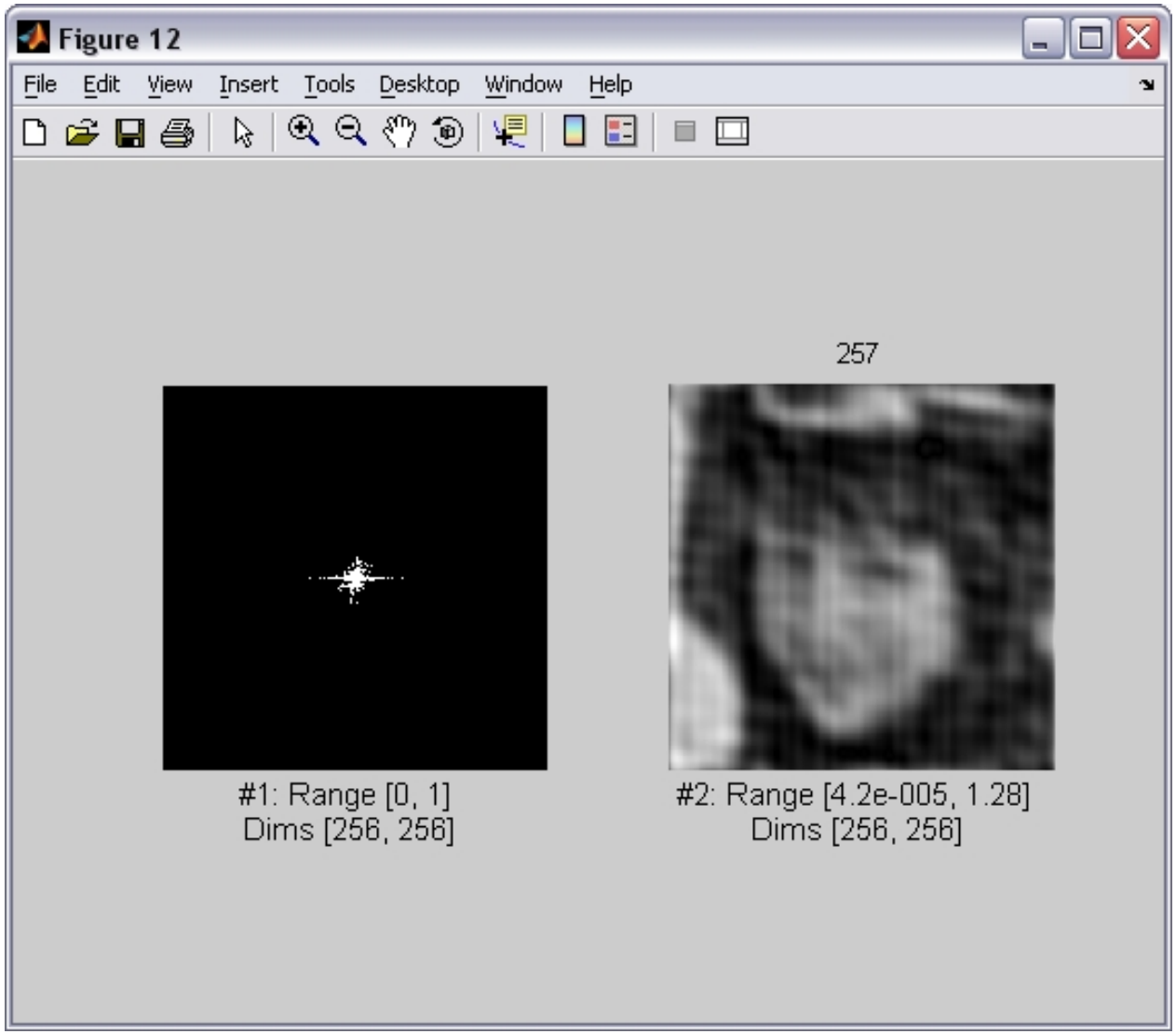
Source: Antonio Torralba slides

129



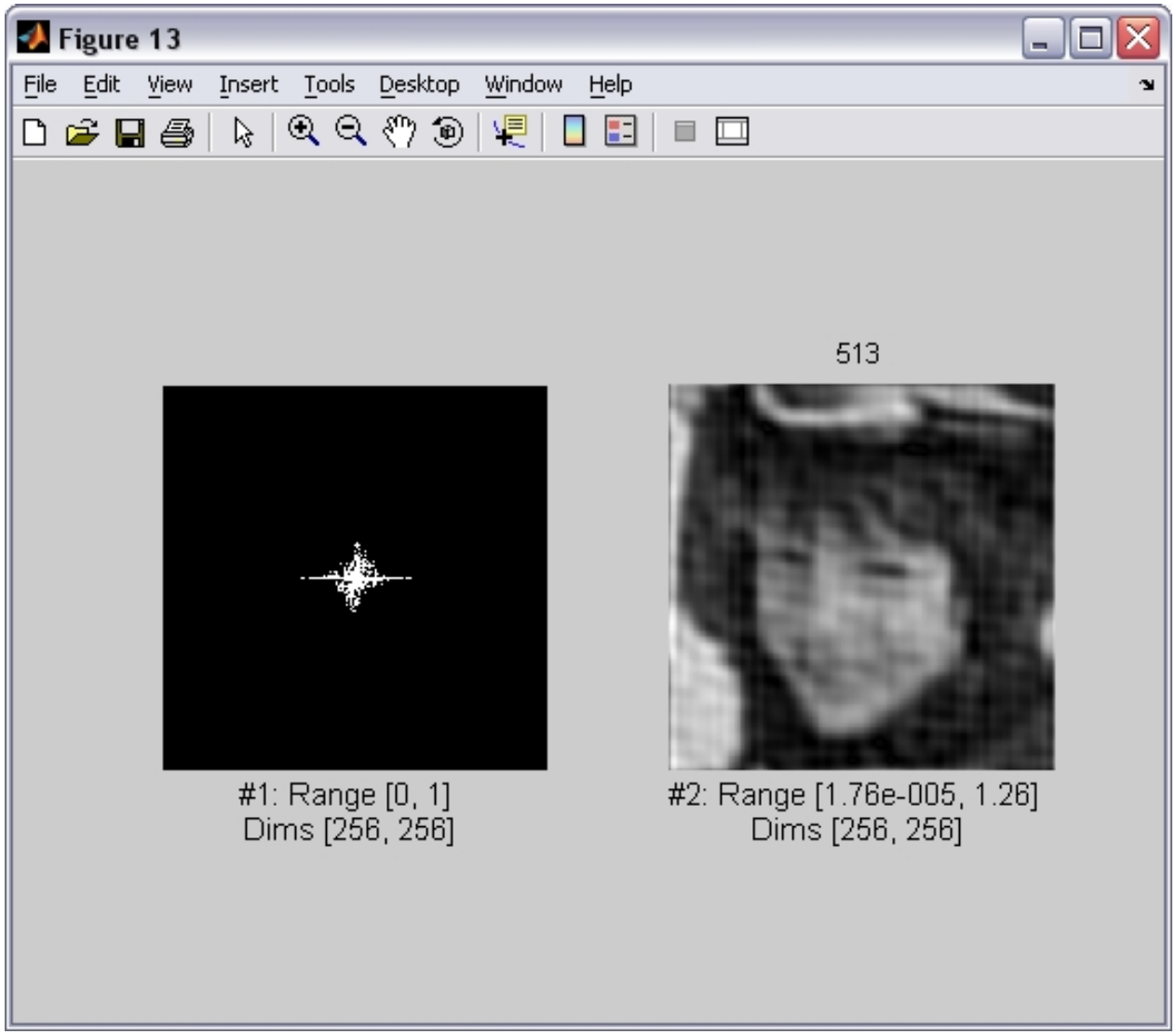
Source: Antonio Torralba slides

257



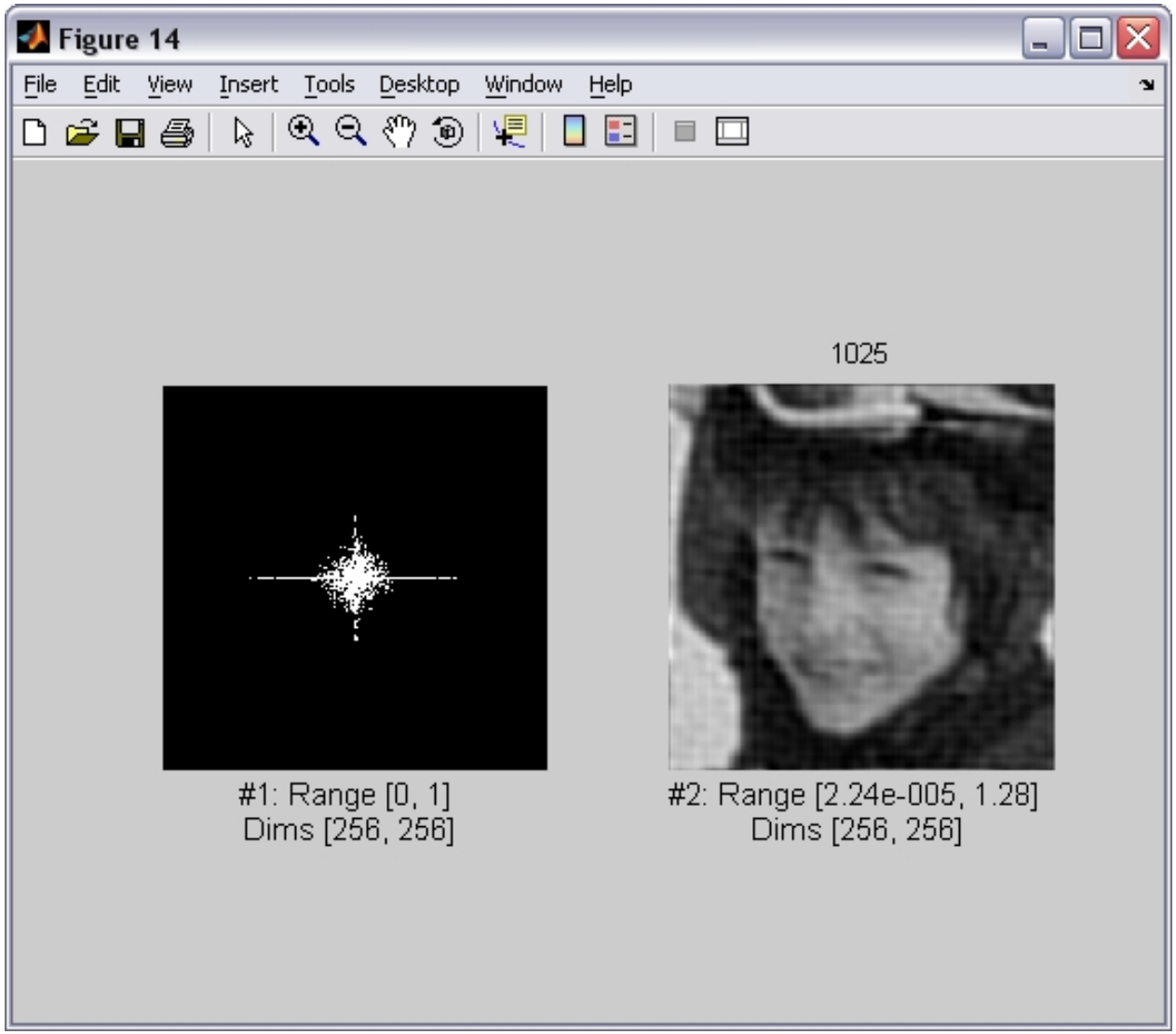
Source: Antonio Torralba slides

513



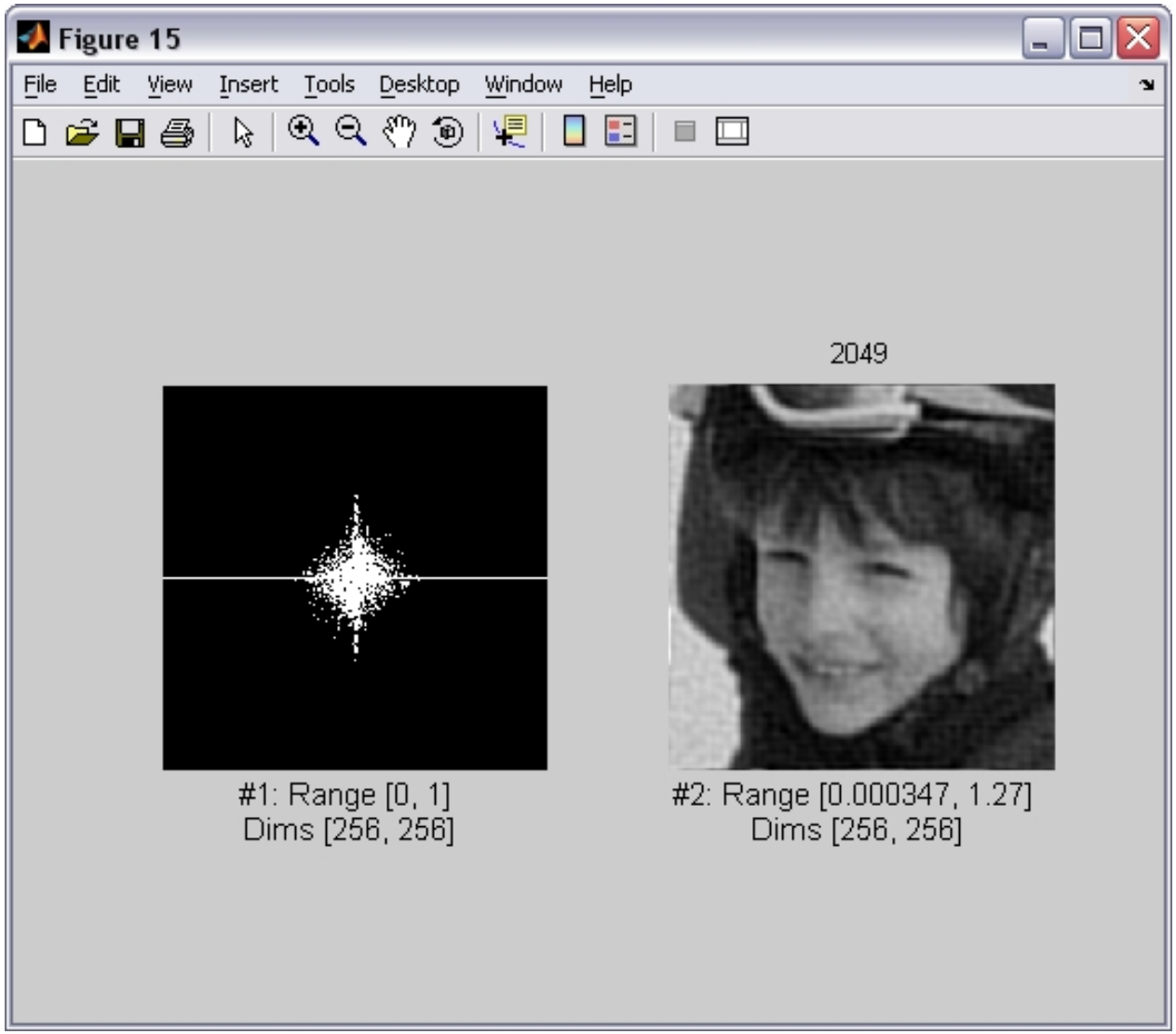
Source: Antonio Torralba slides

1025



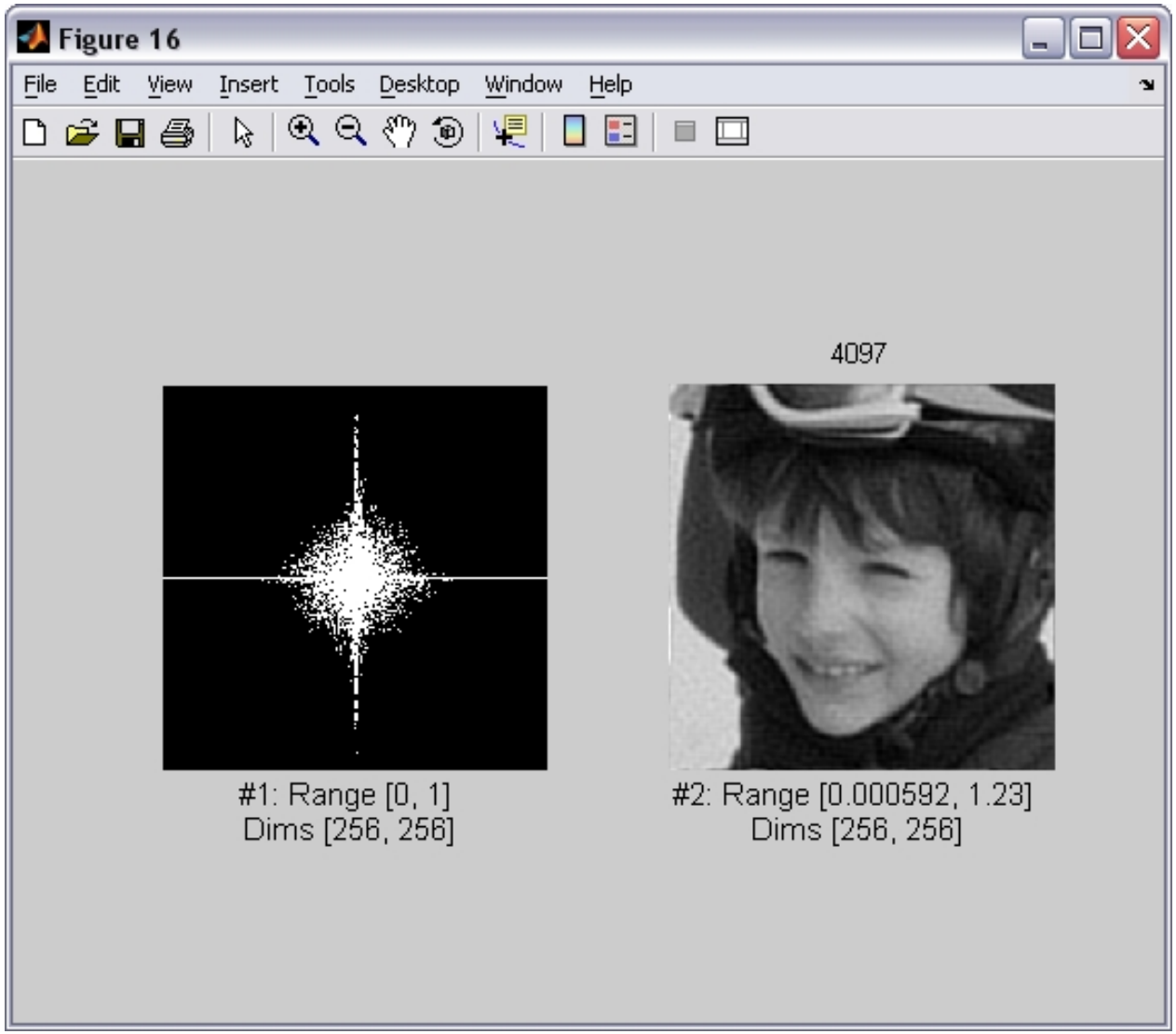
Source: Antonio Torralba slides

2049



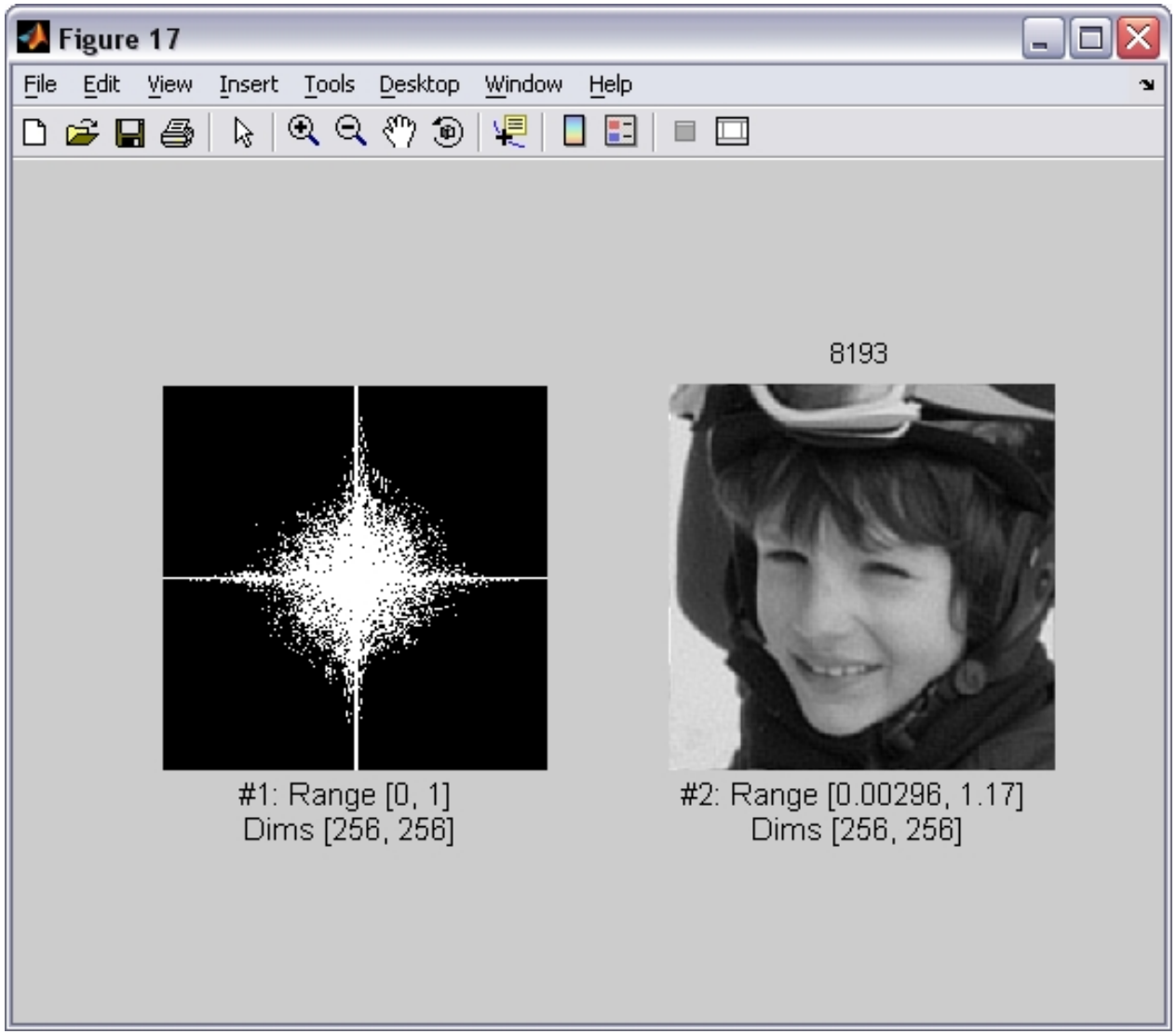
Source: Antonio Torralba slides

4097



Source: Antonio Torralba slides

8193



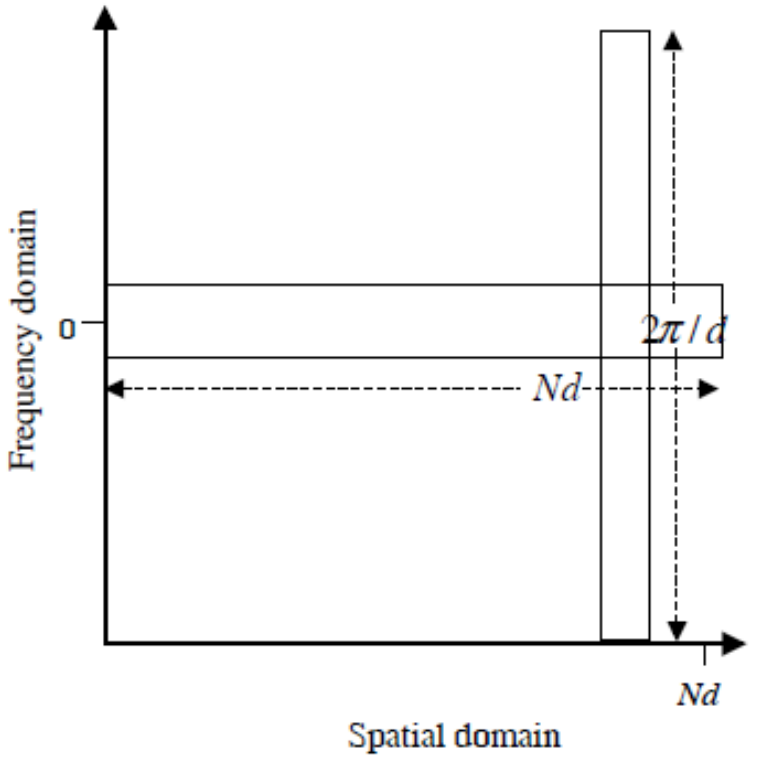
Source: Antonio Torralba slides

Outline

- Introduction
- Point operators
- Linear filtering
- Non linear filtering
- The Fourier transform
- Pyramids and scales

Pyramids

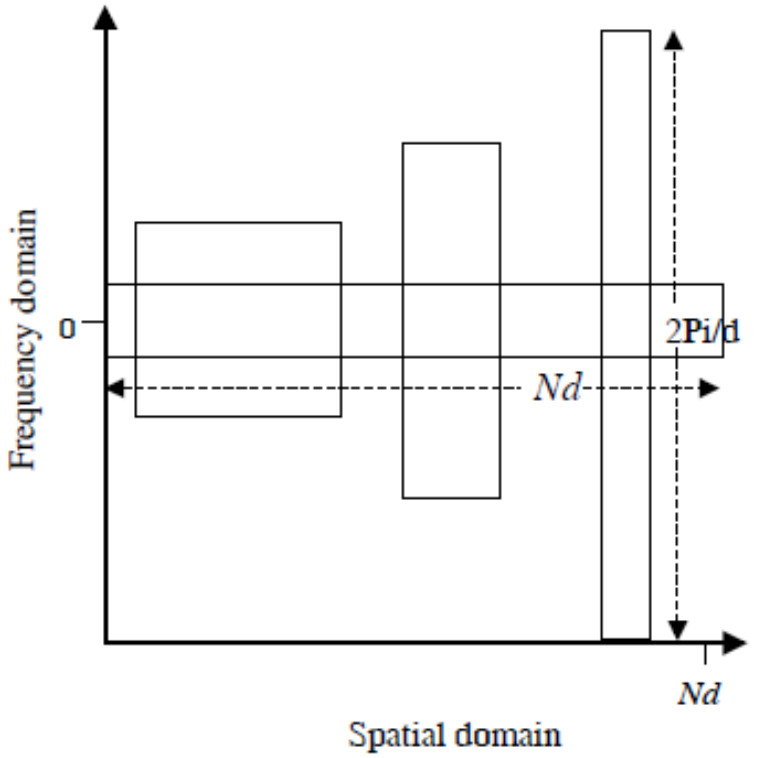
- We have two different ways to represent an image
  - The spatial domain
  - The frequency domain
- Spatial domain
  - We know spatial position
  - Frequency resolution lost
- Frequency domain
  - Frequency content with precision
  - Spatial position lost



Source: Richard Wildes slides

Pyramids

- We have two different ways to represent an image
  - The spatial domain
  - The frequency domain
- We desire a joint representation that allows us to capture the range of scales present locally in the image
- Such representations are known as multiscale or multiresolution



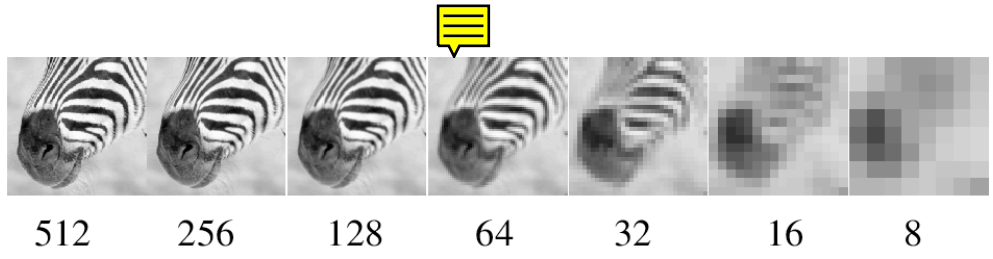
Source: Richard Wildes slides

Pyramids

- Image pyramid
  - Each layer of the pyramid is half the width and half the height of the previous layer
  - Gaussian pyramid: Each layer is smoothed by a symmetric Gaussian kernel and resampled to get the next layer

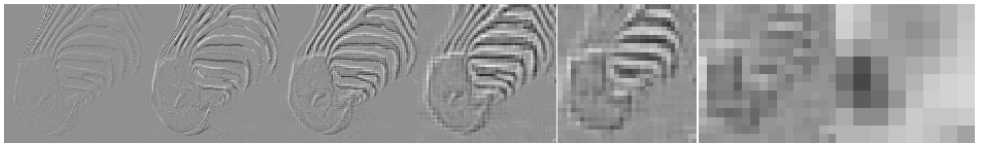
Pyramids

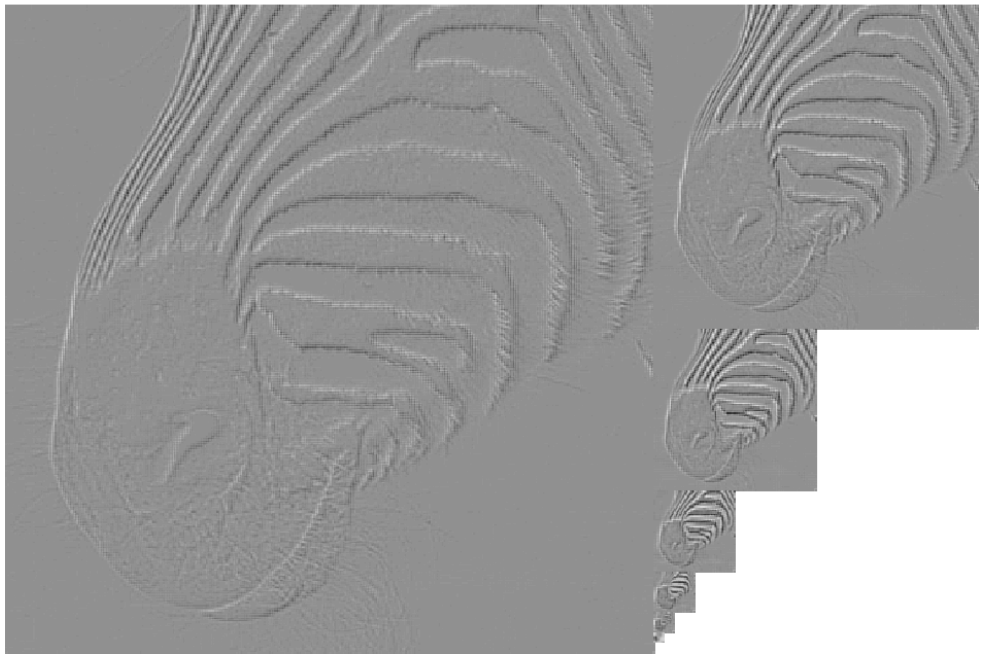
- Image pyramid
  - Each layer of the pyramid is half the width and half the height of the previous layer
  - Gaussian pyramid: Each layer is smoothed by a symmetric Gaussian kernel and resampled to get the next layer



Pyramids

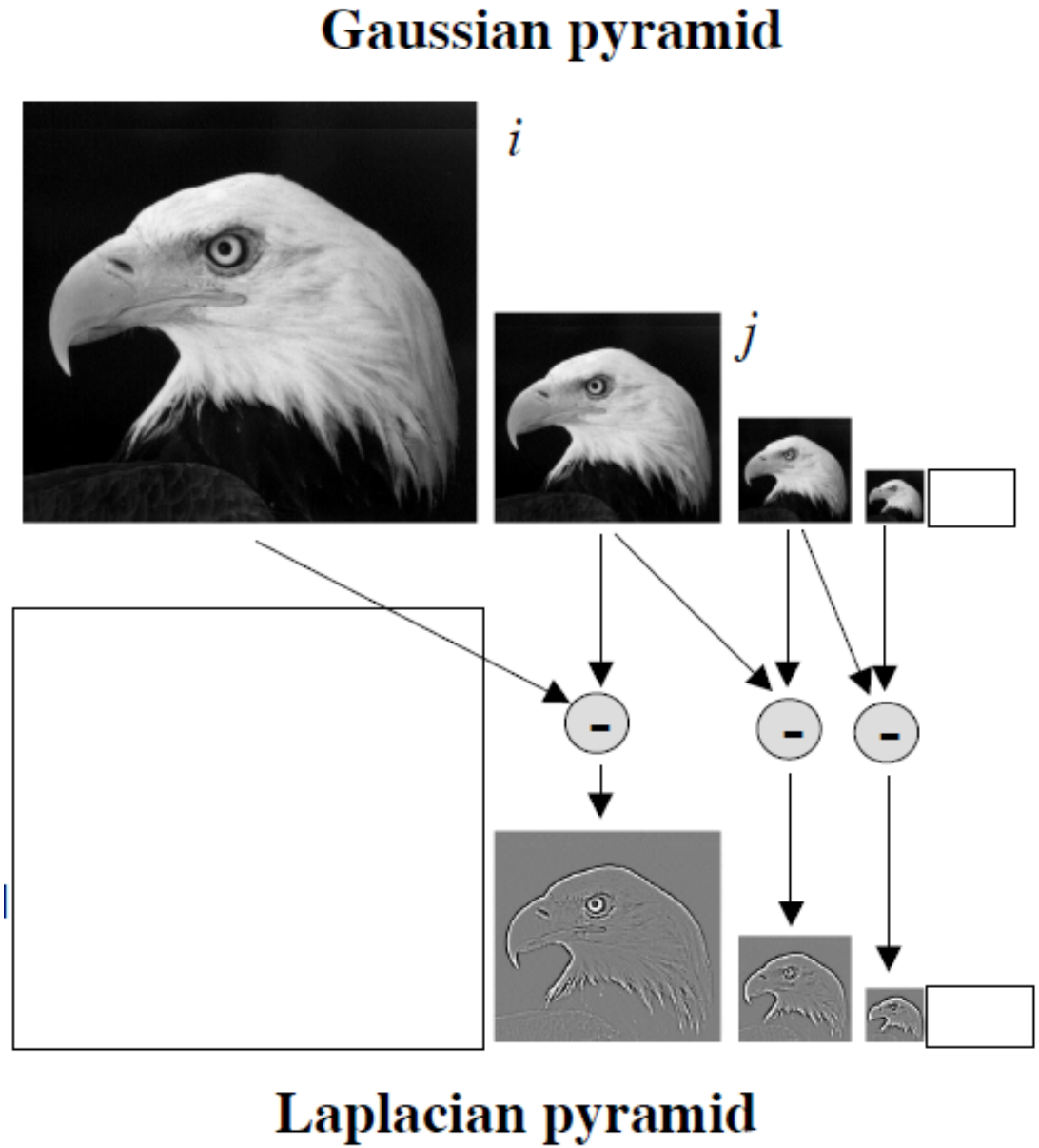
- Image pyramid
  - Each layer of the pyramid is half the width and half the height of the previous layer
  - Laplacian pyramid: preserve difference between upsampled Gaussian pyramid level and Gaussian pyramid level
    - **band pass filter** - each level represents spatial frequencies (largely) unrepresented at other levels





Pyramids

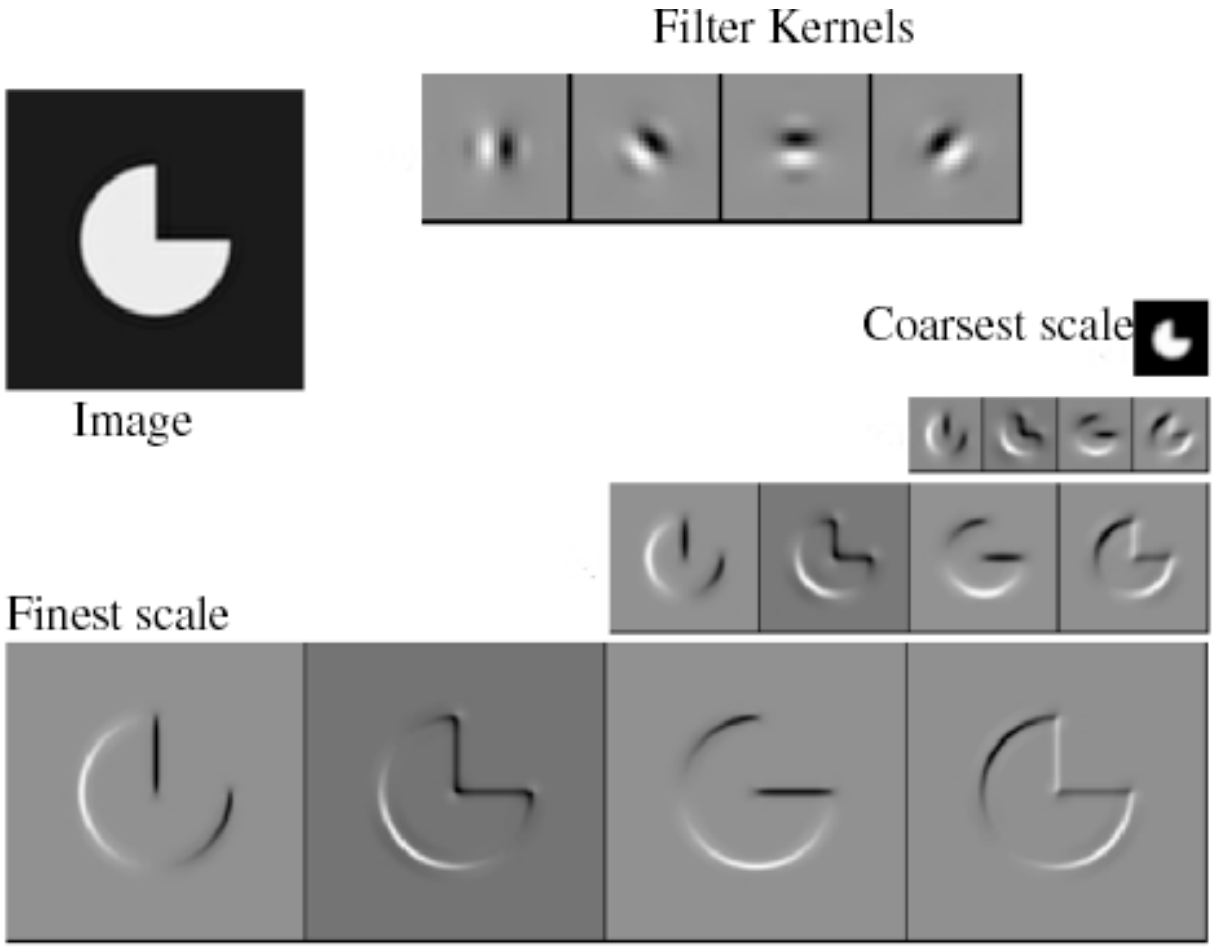
- Comparison Gaussian and Laplacian pyramid



Source: Richard Wildes slides

Pyramids

- Orientation pyramid
  - Apply an oriented filter to determine orientations at each layer



Local scale

- The windowed Fourier transform
  - Break image up into a set of tiles
  - Apply the Fourier transform to each tile individually.
  - Then perhaps we have captured a spatial frequency representation that is local to each tile.

Source: Richard Wildes slides

Local scale

- The windowed Fourier transform
  - Break image up into a set of tiles
  - Apply the Fourier transform to each tile individually.
  - Then perhaps we have captured a spatial frequency representation that is local to each tile.
  - We consider application of the Fourier transform, ***F(u,v)***, within a window, ***w(x,y)***, of the image and move the window across the image.

$$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)w(x-a,y-b)\exp[-i(ux+vy)]dxdy$$

Source: Richard Wildes slides

Local scale

- The windowed Fourier transform
  - Break image up into a set of tiles
  - Apply the Fourier transform to each tile individually.
  - Then perhaps we have captured a spatial frequency representation that is local to each tile.
  - We consider application of the Fourier transform, ***F(u,v)***, within a window, ***w(x,y)***, of the image and move the window across the image.

$$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)w(x-a,y-b)\exp[-i(ux+vy)]dxdy$$

  - Note, this has the form of a convolution

$$f(x,y) * w(x,y)\exp[i(u_0x+v_0y)]$$

Source: Richard Wildes slides

Local scale

- The windowed Fourier transform
  - Break image up into a set of tiles
  - Apply the Fourier transform to each tile individually.
  - Then perhaps we have captured a spatial frequency representation that is local to each tile.
  - We consider application of the Fourier transform, ***F(u,v)***, within a window, ***w(x,y)***, of the image and move the window across the image.

$$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)w(x-a,y-b)\exp[-i(ux+vy)]dxdy$$

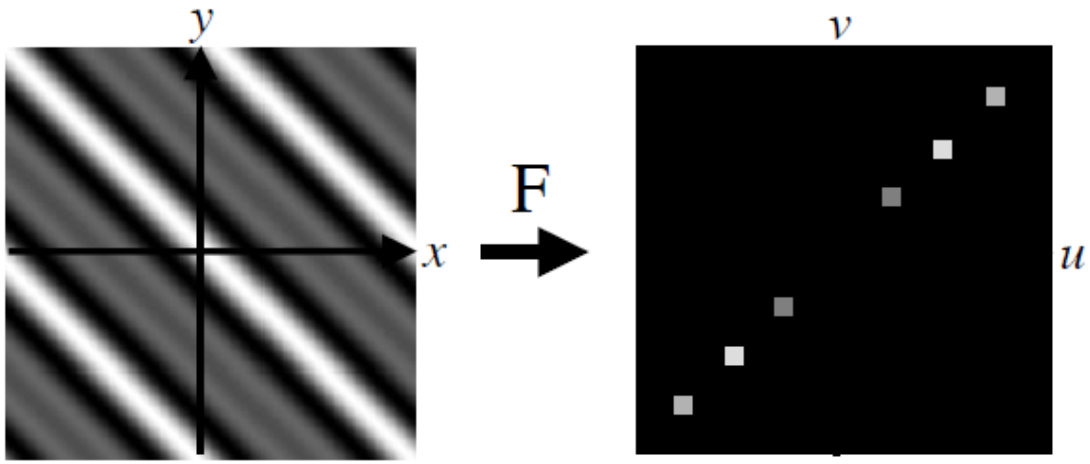
  - Note, this has the form of a convolution

$$f(x,y) * w(x,y)\exp[i(u_0x+v_0y)]$$

This is a band pass filter !

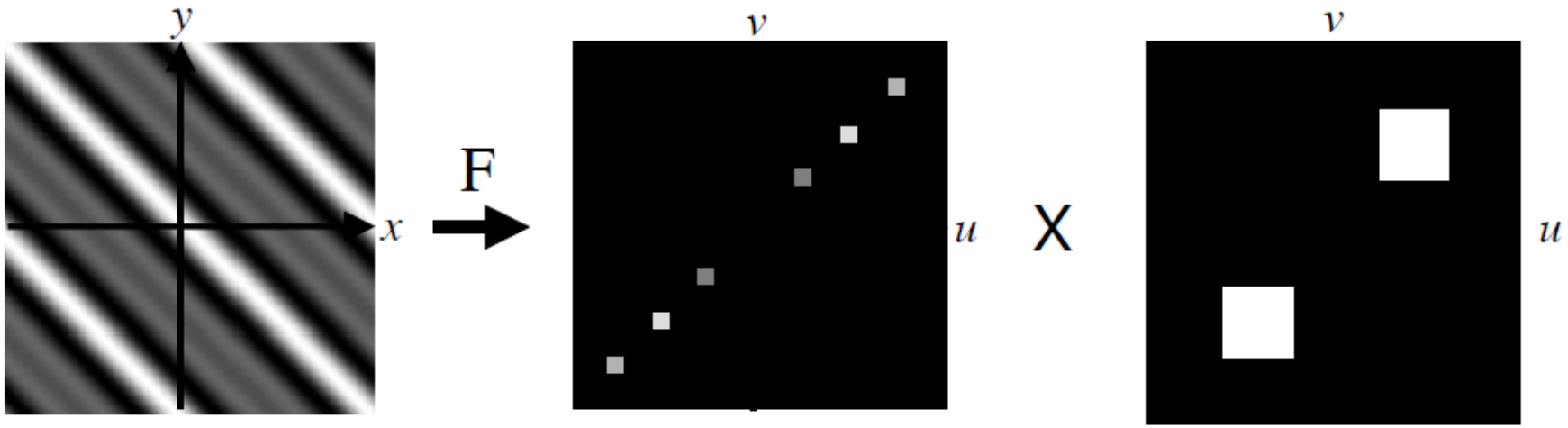
Source: Richard Wildes slides

Local scale



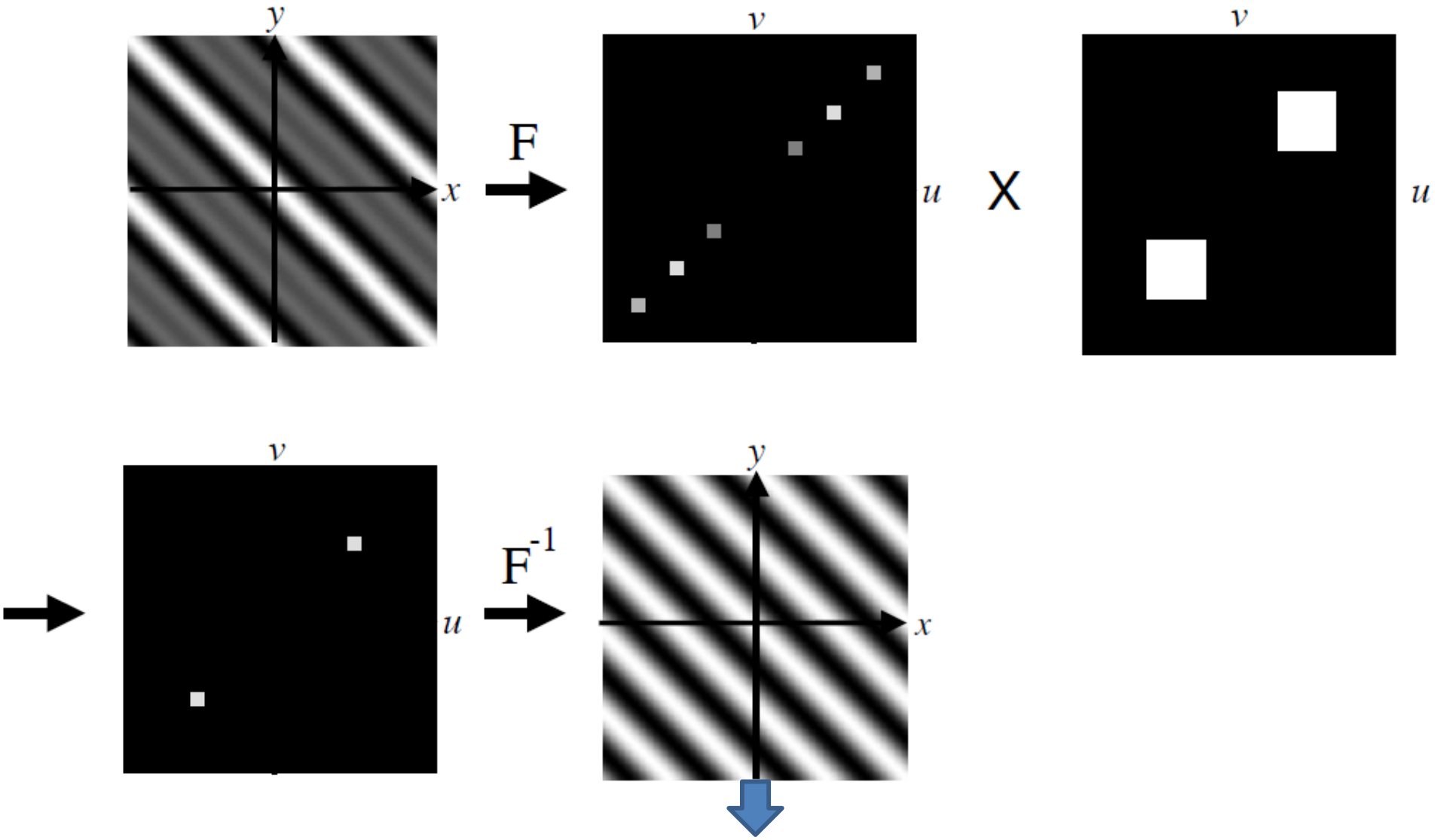
Source: Richard Wildes slides

Local scale



Source: Richard Wildes slides

Local scale



This is a band pass filter !

Source: Richard Wildes slides

Local scale

- The Gabor filter
  - Is a Gaussian kernel function modulated by a sinusoidal plane wave.

$$\frac{1}{2\pi\sigma^2}\exp[i(u_0x+v_0y)]\exp\left[-\frac{1}{2}\left(\frac{x^2+y^2}{\sigma^2}\right)\right]$$

Local scale

- The Gabor filter
  - Is a Gaussian kernel function modulated by a sinusoidal plane wave.

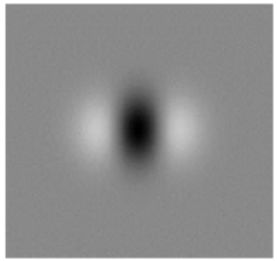
$$\frac{1}{2\pi\sigma^2}\exp[i(u_0x+v_0y)]\exp\left[-\frac{1}{2}\left(\frac{x^2+y^2}{\sigma^2}\right)\right]$$

Local scale

- The Gabor filter
  - Is a Gaussian kernel function modulated by a sinusoidal plane wave.

$$\frac{1}{2\pi\sigma^2}\exp[i(u_0x+v_0y)]\exp\left[-\frac{1}{2}\left(\frac{x^2+y^2}{\sigma^2}\right)\right]$$

$$\frac{1}{2\pi\sigma^2}\cos[(u_0x+v_0y)]\exp\left[-\frac{1}{2}\left(\frac{x^2+y^2}{\sigma^2}\right)\right]$$



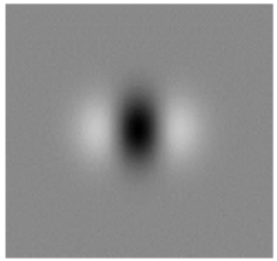
Local scale

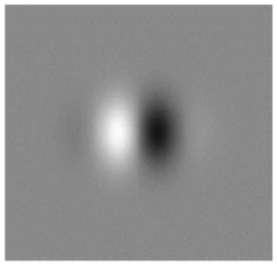
- The Gabor filter
  - Is a Gaussian kernel function modulated by a sinusoidal plane wave.

$$\frac{1}{2\pi\sigma^2}\exp[i(u_0x+v_0y)]\exp\left[-\frac{1}{2}\left(\frac{x^2+y^2}{\sigma^2}\right)\right]$$

$$\frac{1}{2\pi\sigma^2}\cos[(u_0x+v_0y)]\exp\left[-\frac{1}{2}\left(\frac{x^2+y^2}{\sigma^2}\right)\right]$$

$$\frac{1}{2\pi\sigma^2}\sin[(u_0x+v_0y)]\exp\left[-\frac{1}{2}\left(\frac{x^2+y^2}{\sigma^2}\right)\right]$$





Local scale

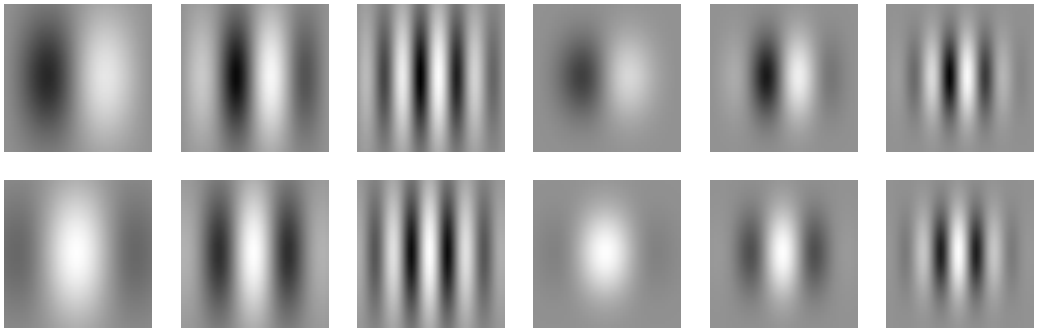
- The Gabor filter
  - Is a Gaussian kernel function modulated by a sinusoidal plane wave.

$$\frac{1}{2\pi\sigma^2}\exp[i(u_0x+v_0y)]\exp\left[-\frac{1}{2}\left(\frac{x^2+y^2}{\sigma^2}\right)\right]$$

  - Gabor filters at different scales and spatial frequencies

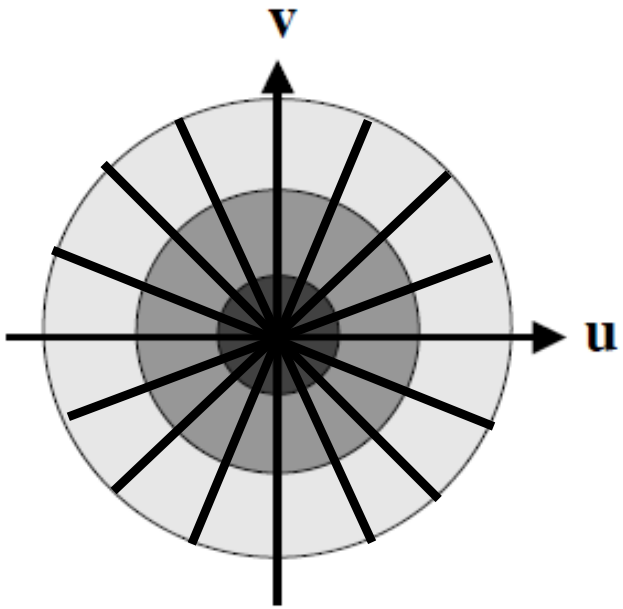
Odd or anti-symmetric

Even or symmetric



Scale and orientation

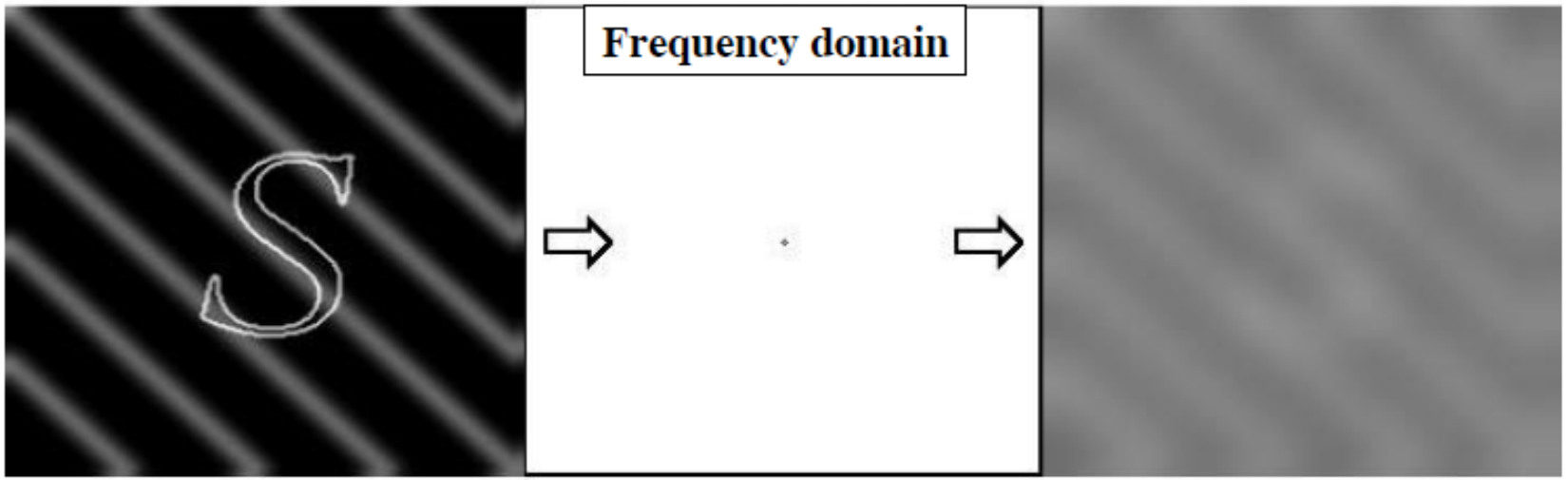
- For given u and v in the frequency domain, there corresponds a (cos)sinusoidal waveforms, $cos(ux+vy)$ and $sin(ux+vy)$, at a particular orientation and periodicity in the spatial domain.
- Through application of the filters that we have just constructed, we can parse the image information according to its local structural components
  - Scale: magnitude of the (center) frequency $|u0,v0|$
  - Orientation: direction of sinusoid $(u0,v0)/|u0,v0|$



Source: Richard Wildes slides

Scale and orientation

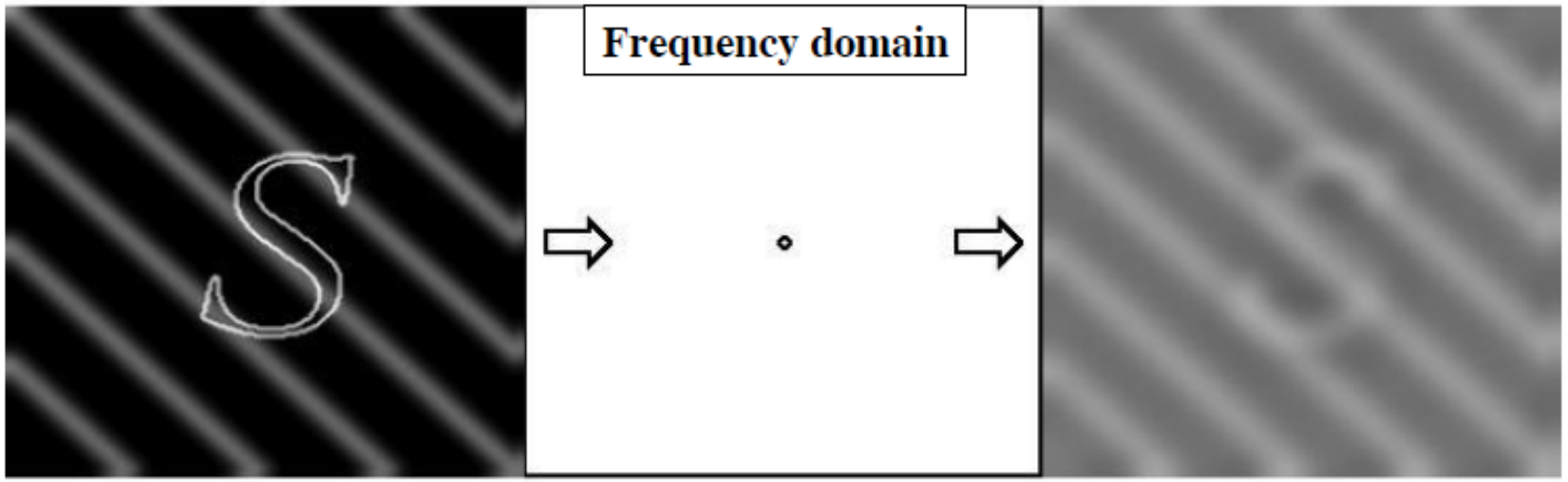
Scale selection



Source: Richard Wildes slides

Scale and orientation

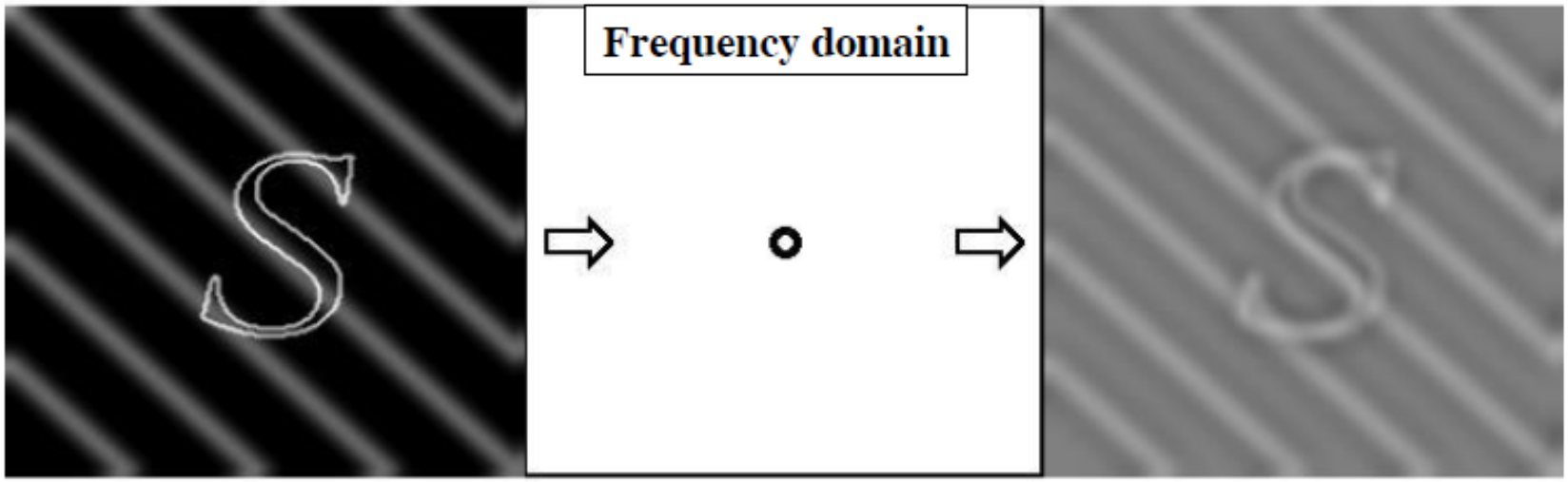
Scale selection



Source: Richard Wildes slides

Scale and orientation

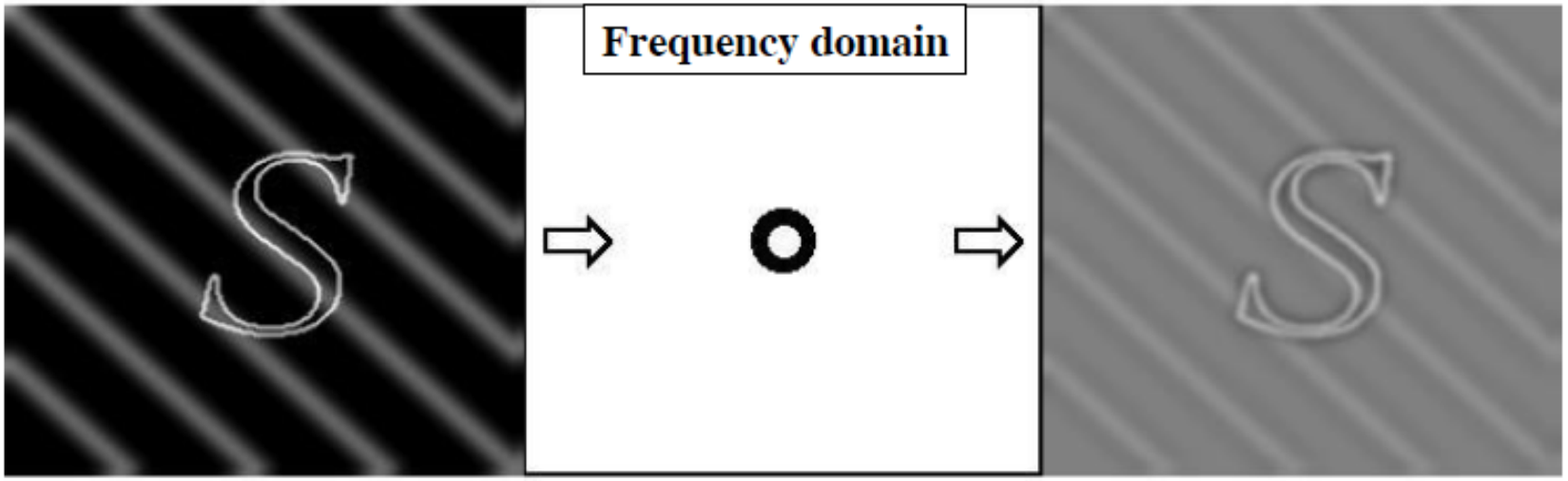
Scale selection



Source: Richard Wildes slides

Scale and orientation

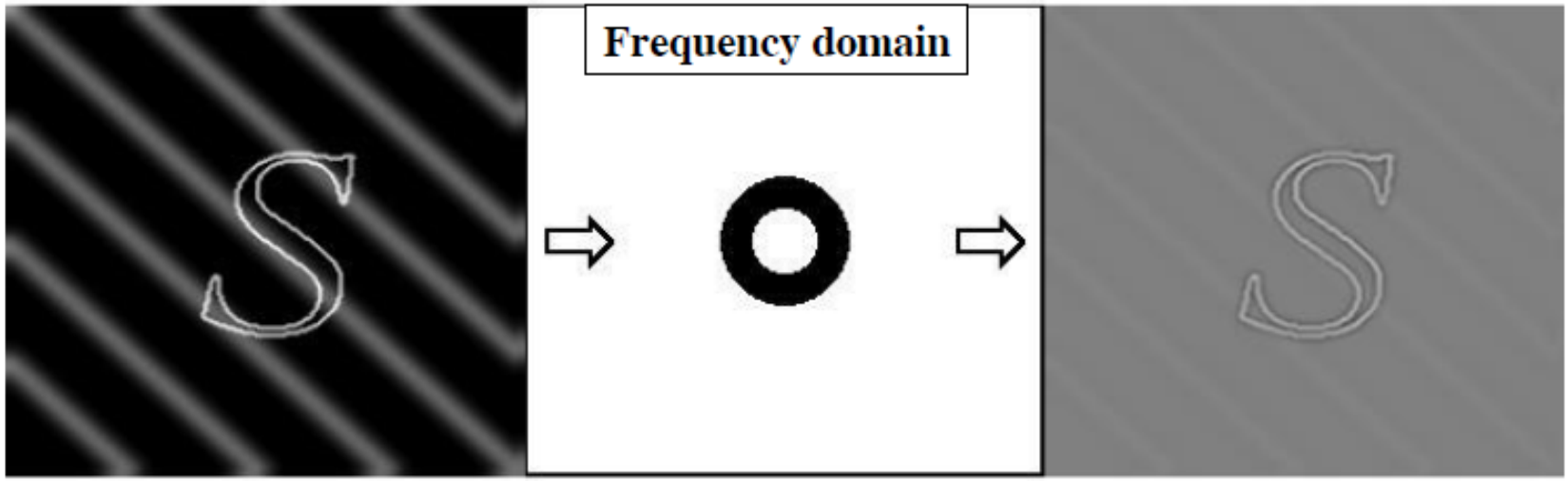
Scale selection



Source: Richard Wildes slides

Scale and orientation

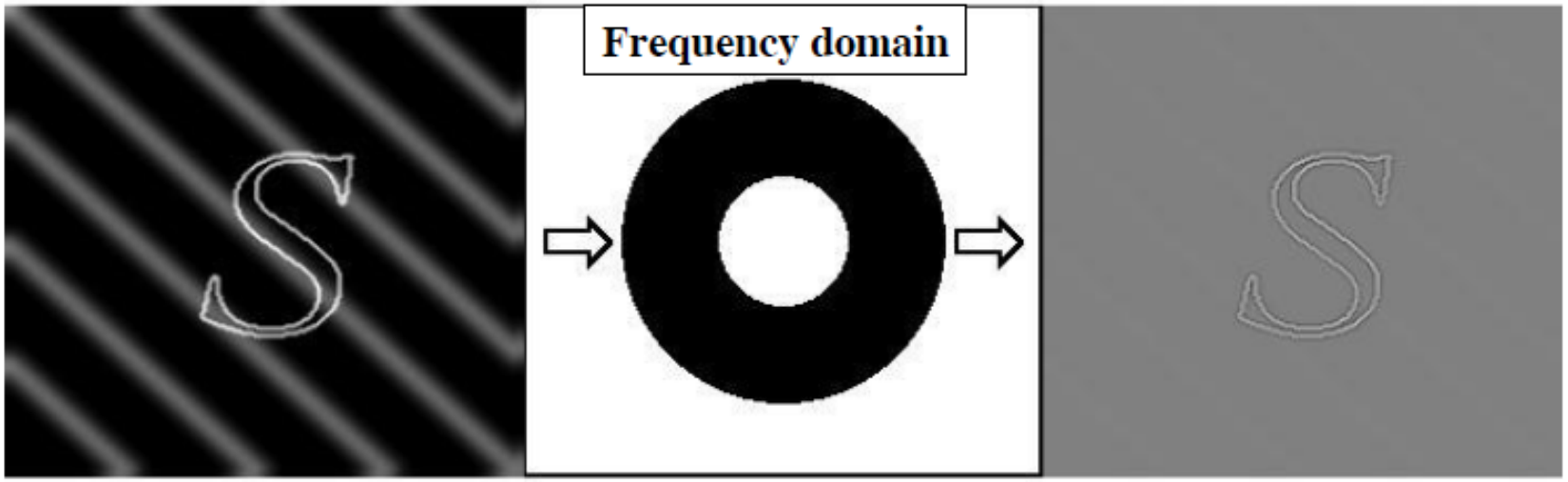
Scale selection



Source: Richard Wildes slides

Scale and orientation

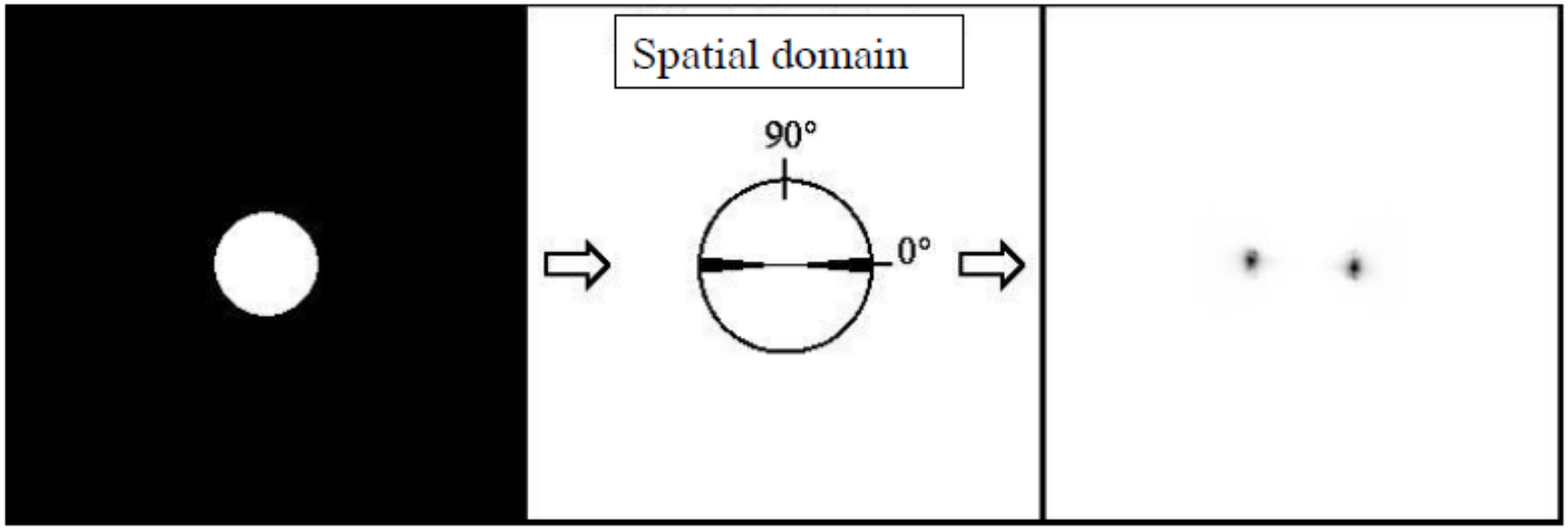
Scale selection



Source: Richard Wildes slides

Scale and **orientation**

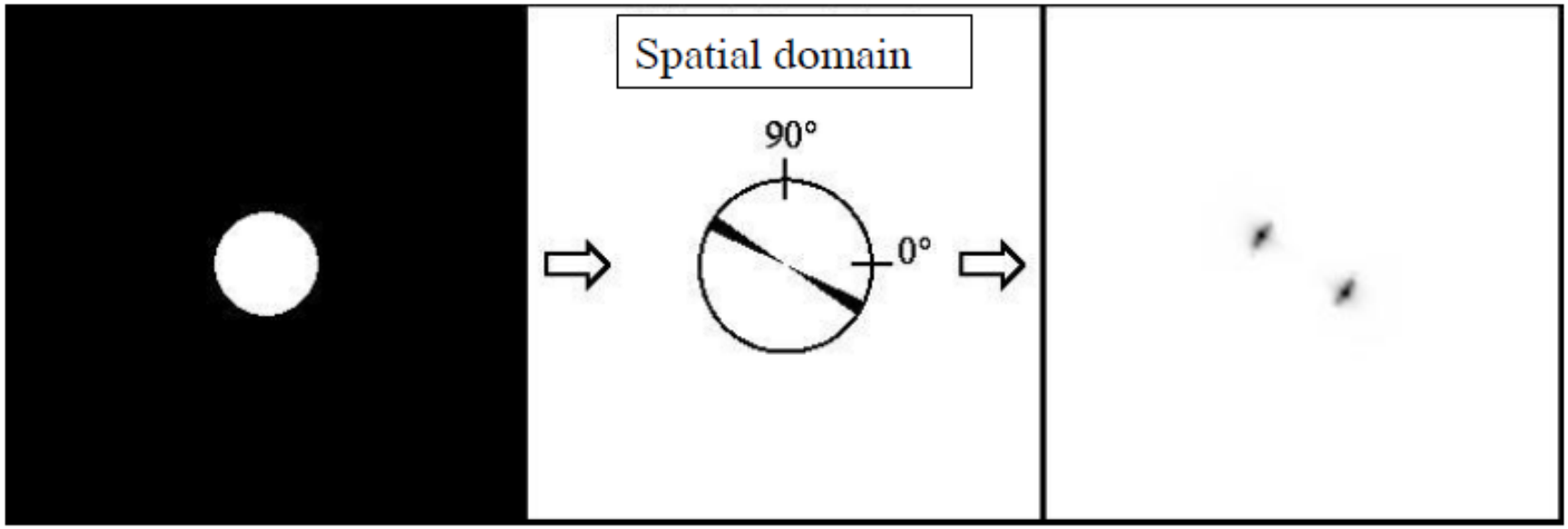
Orientation selection



Source: Richard Wildes slides

Scale and **orientation**

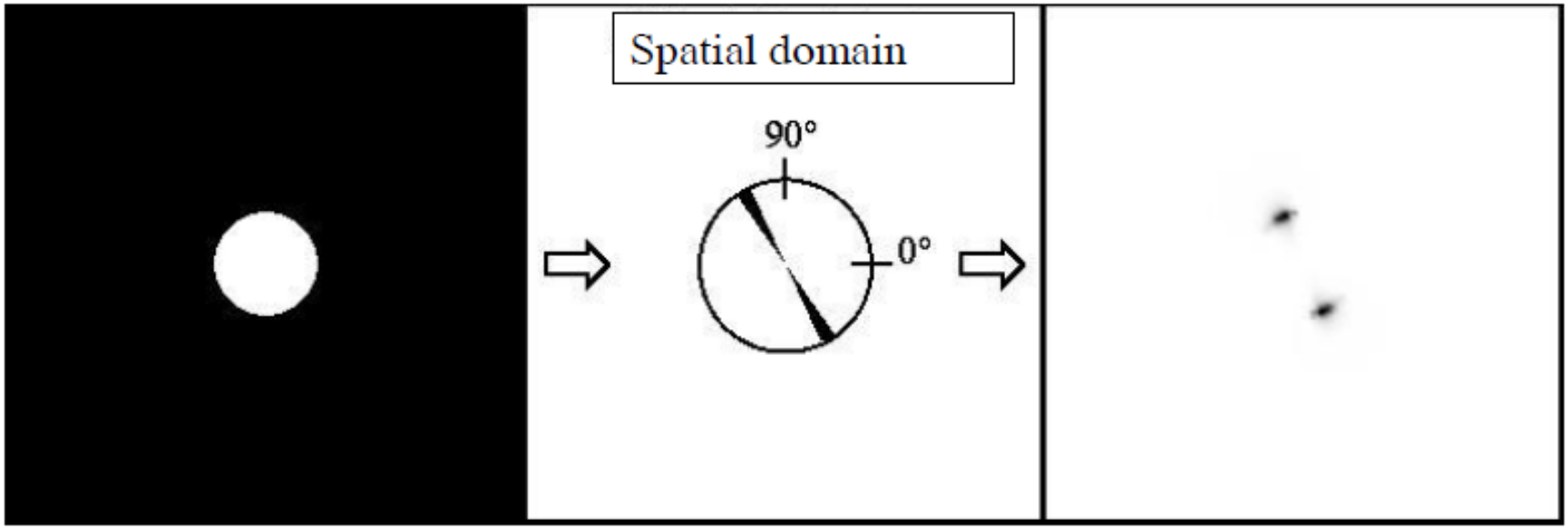
Orientation selection



Source: Richard Wildes slides

Scale and **orientation**

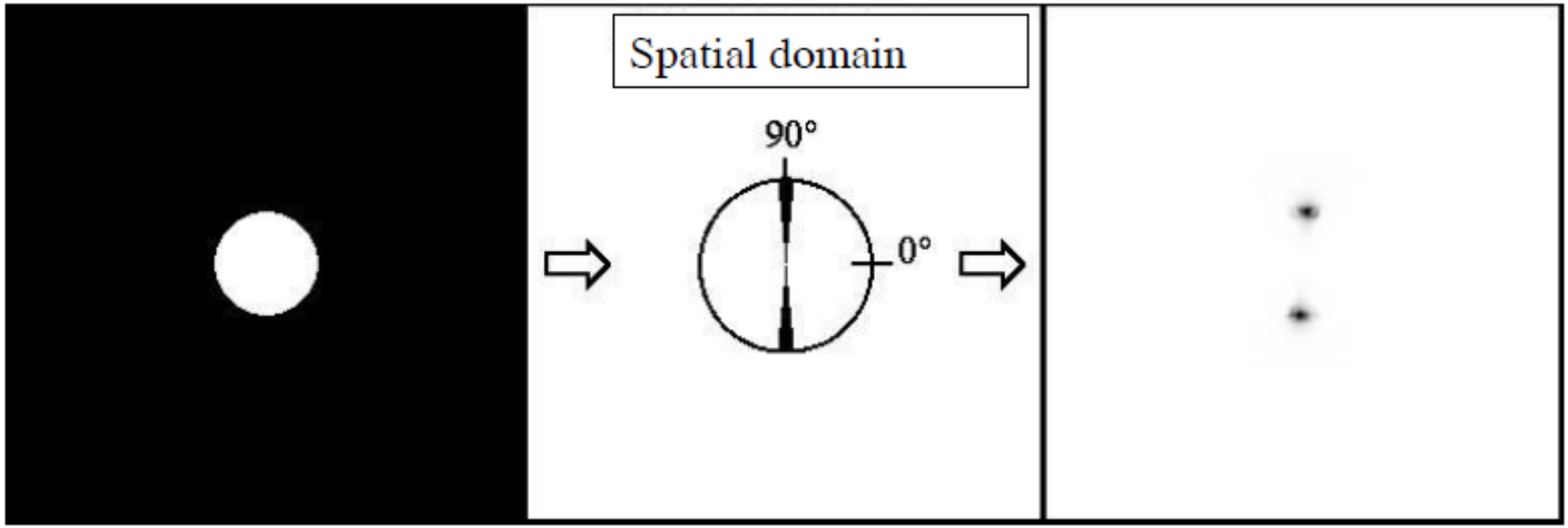
Orientation selection



Source: Richard Wildes slides

Scale and **orientation**

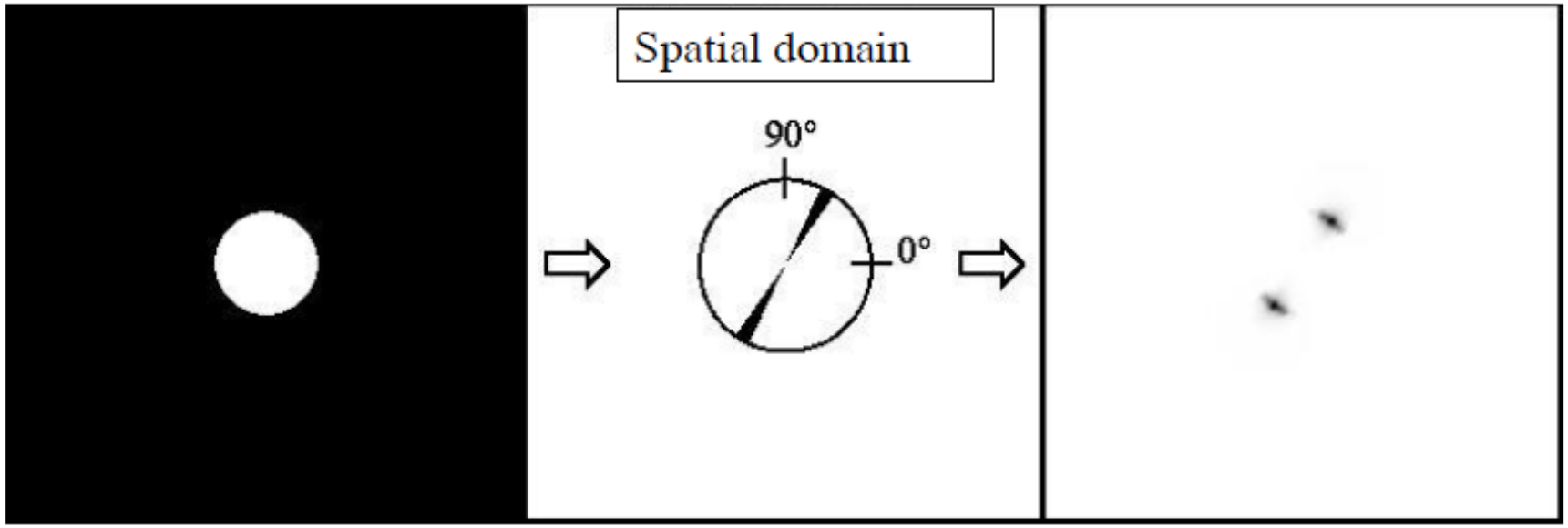
Orientation selection



Source: Richard Wildes slides

Scale and **orientation**

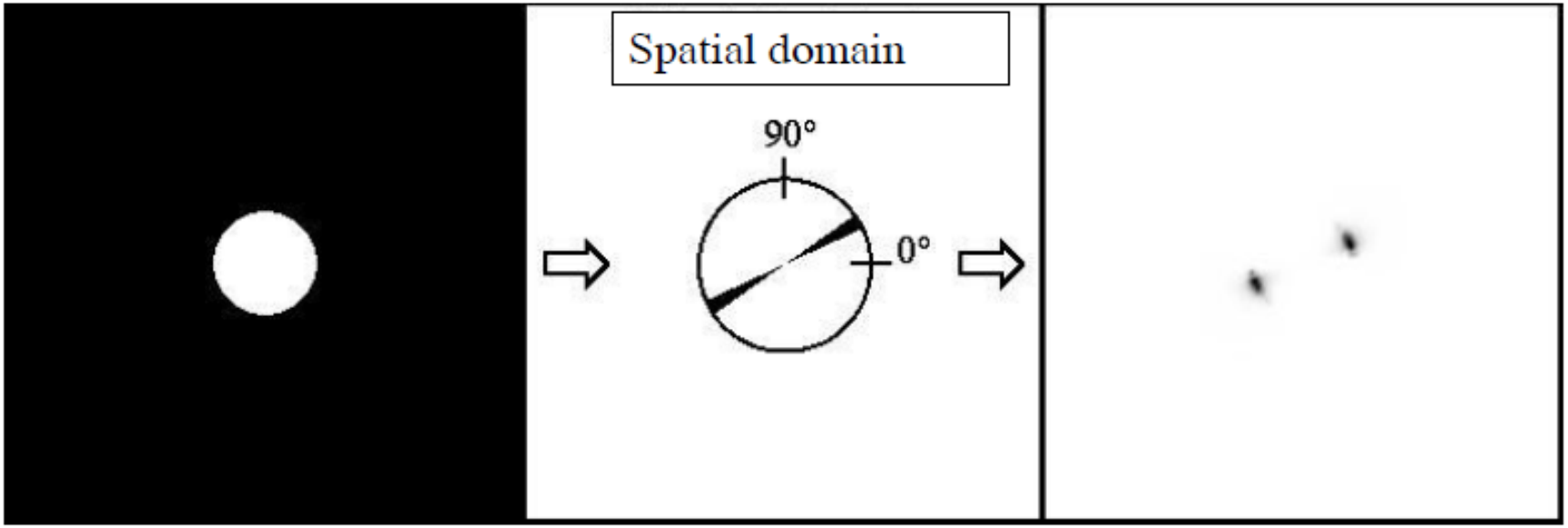
Orientation selection



Source: Richard Wildes slides

Scale and **orientation**

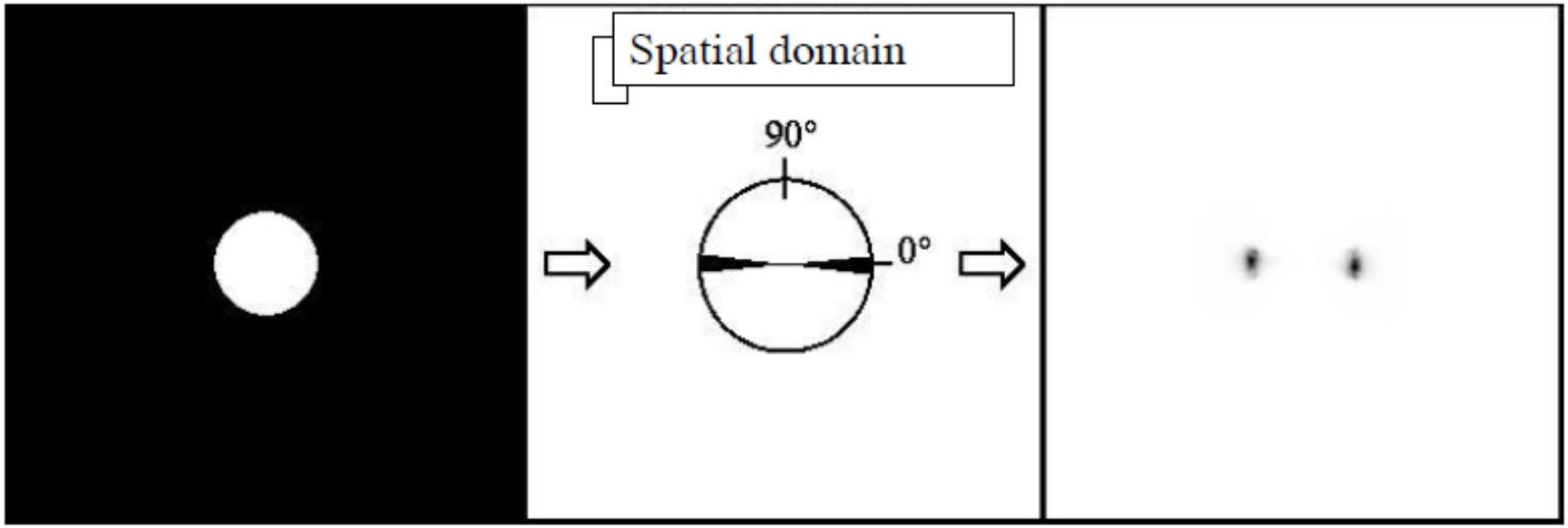
Orientation selection



Source: Richard Wildes slides

Scale and **orientation**

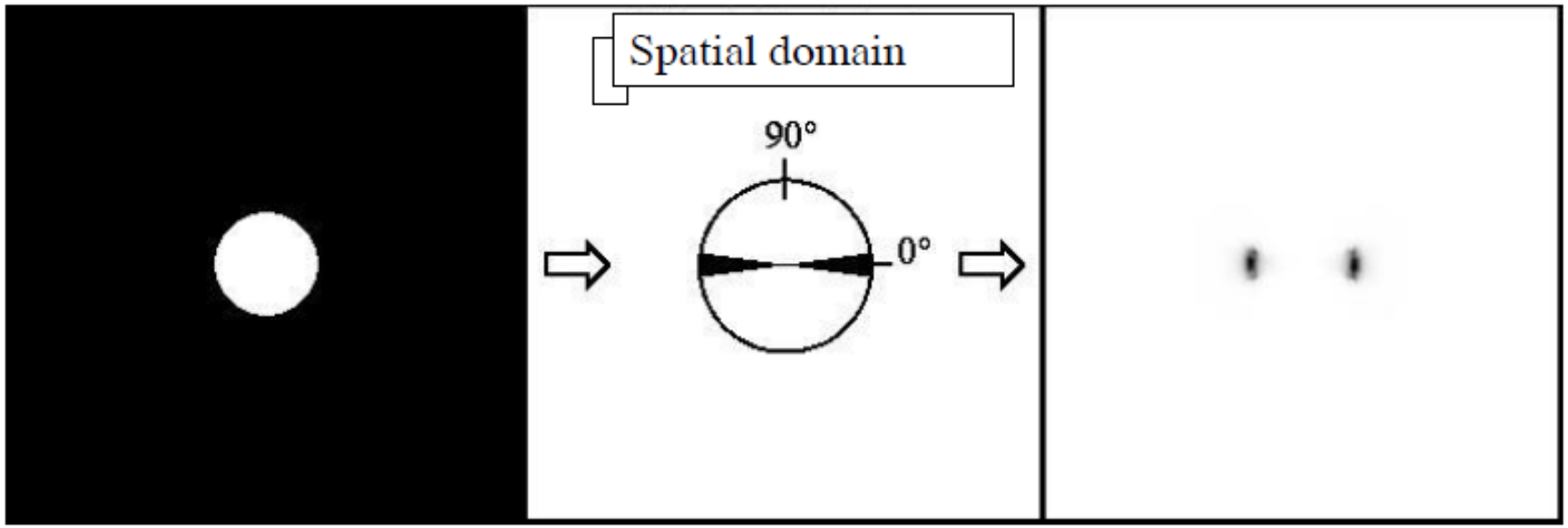
Bandwidth selection



Source: Richard Wildes slides

Scale and **orientation**

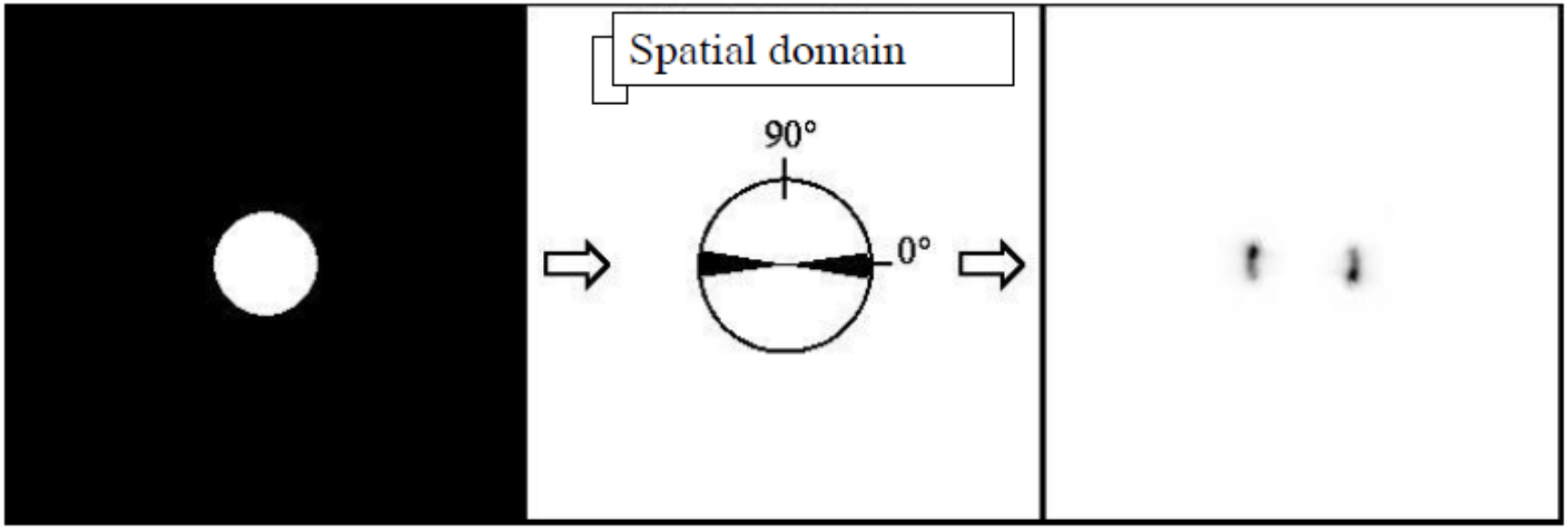
Bandwidth selection



Source: Richard Wildes slides

Scale and **orientation**

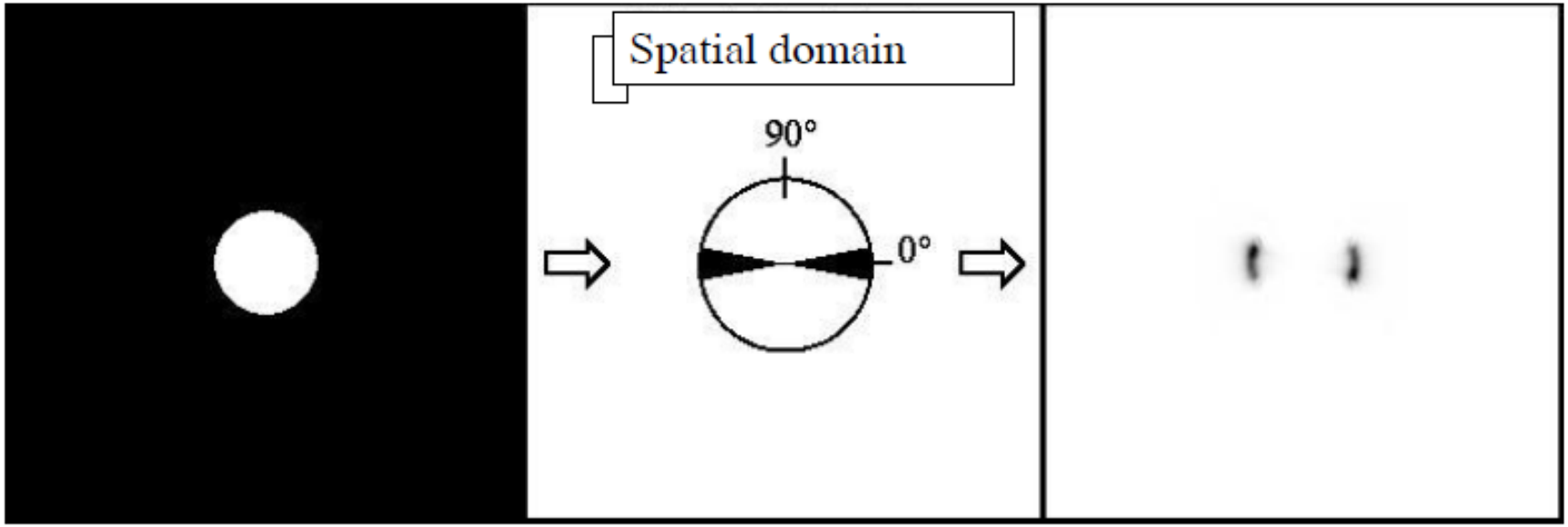
Bandwidth selection



Source: Richard Wildes slides

Scale and **orientation**

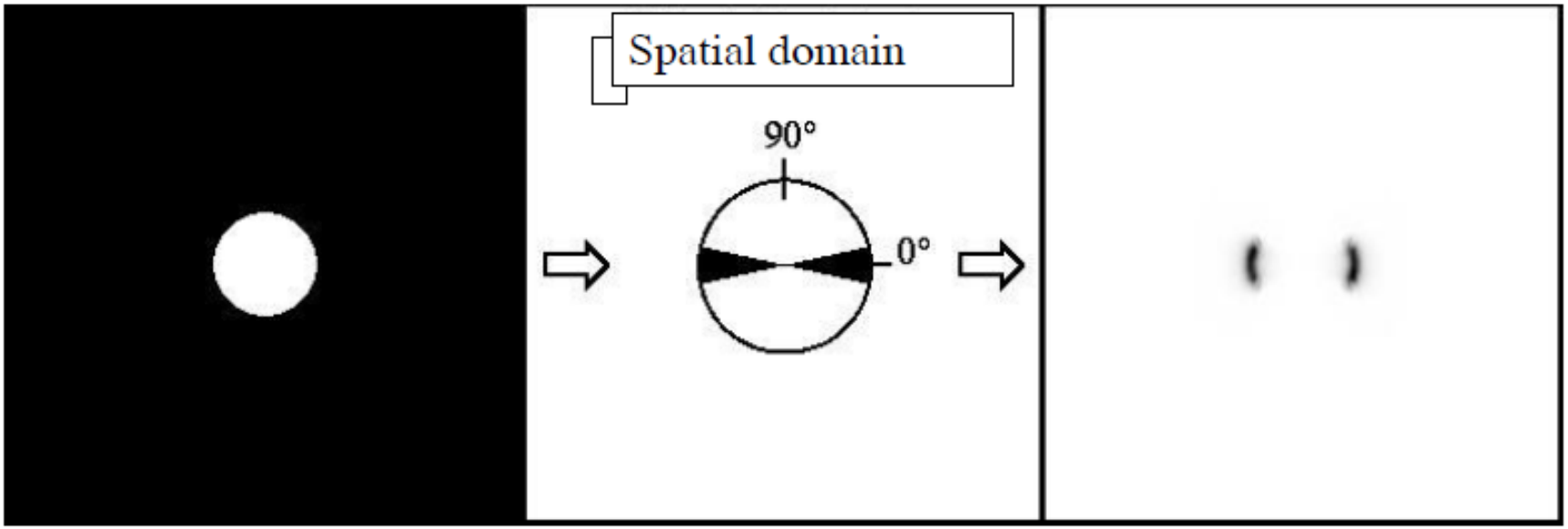
Bandwidth selection



Source: Richard Wildes slides

Scale and **orientation**

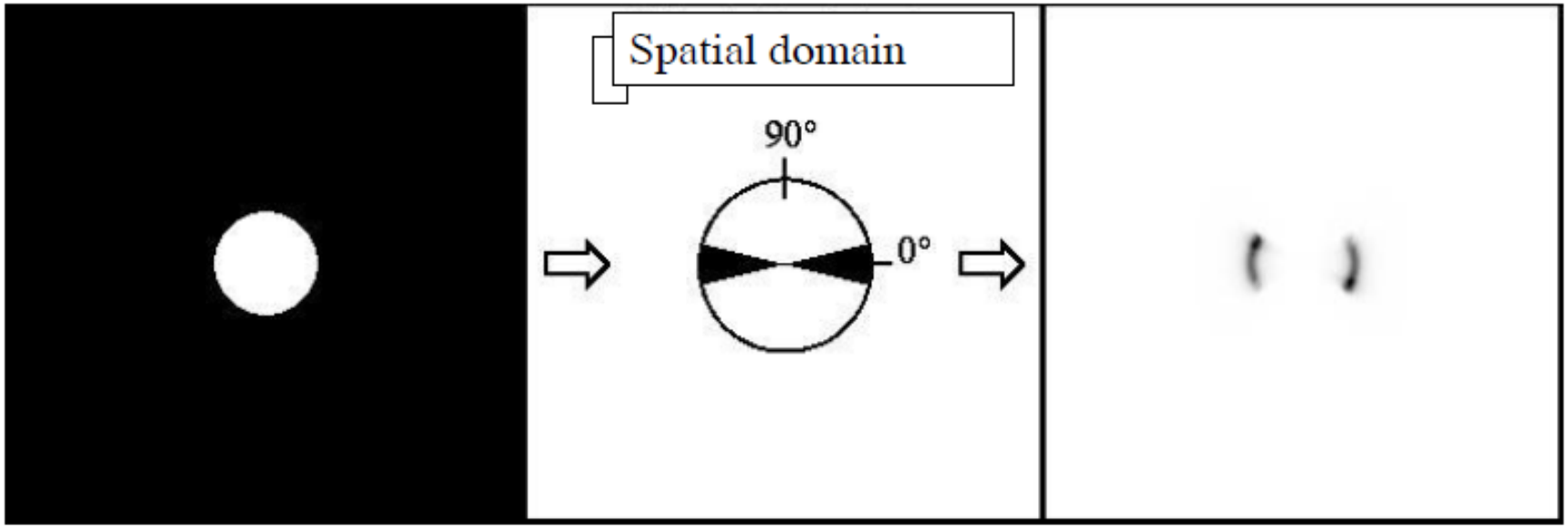
Bandwidth selection



Source: Richard Wildes slides

Scale and **orientation**

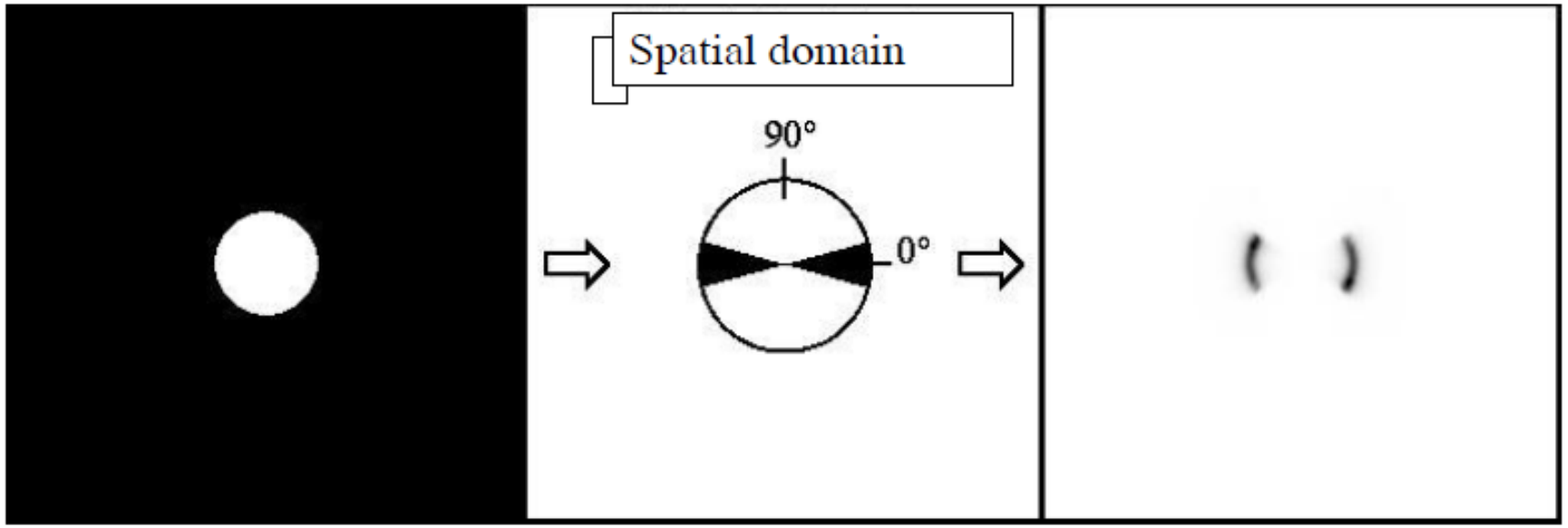
Bandwidth selection



Source: Richard Wildes slides

Scale and **orientation**

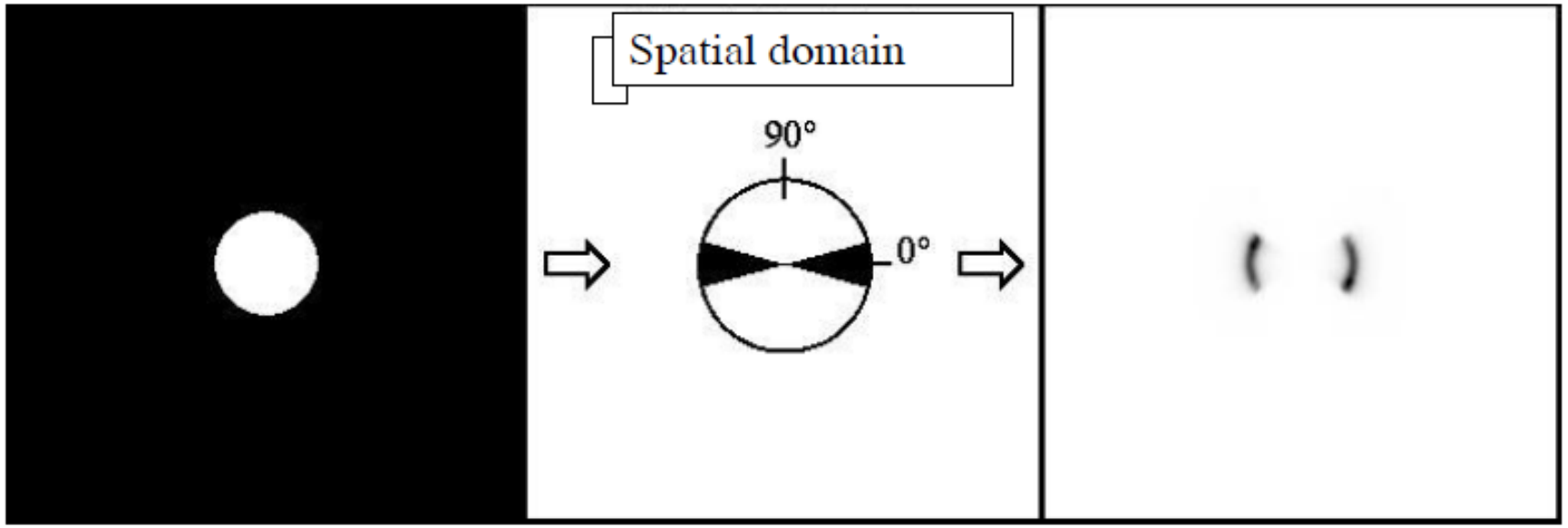
Bandwidth selection



Source: Richard Wildes slides

Scale and **orientation**

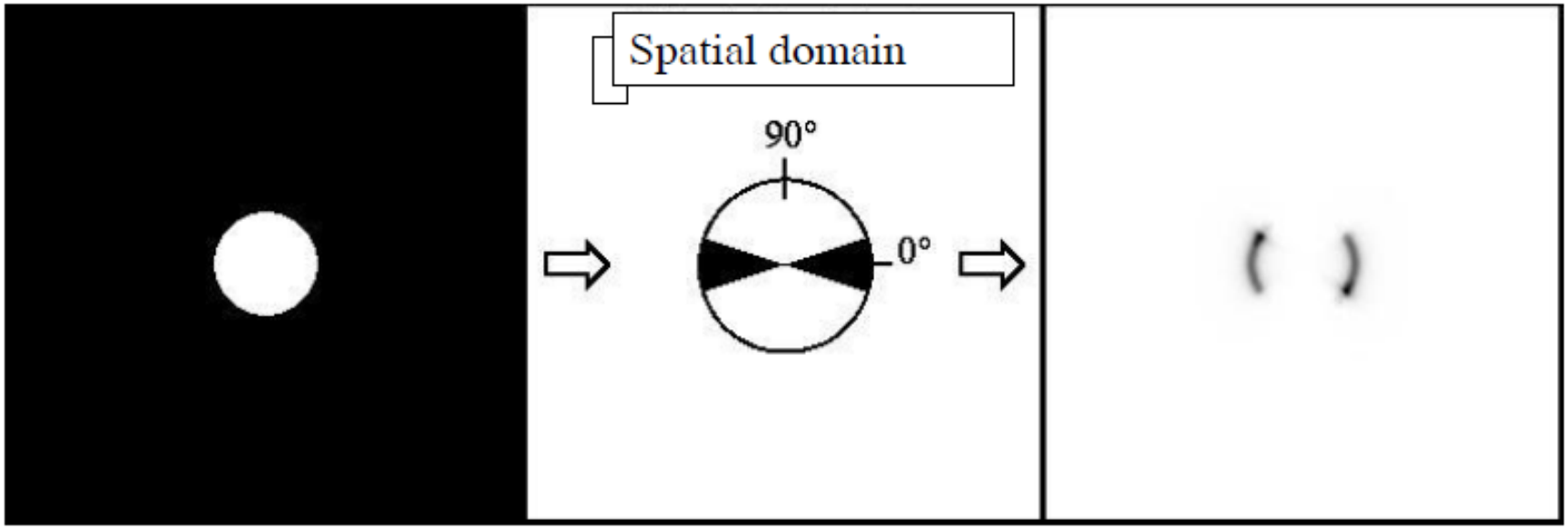
Bandwidth selection



Source: Richard Wildes slides

Scale and **orientation**

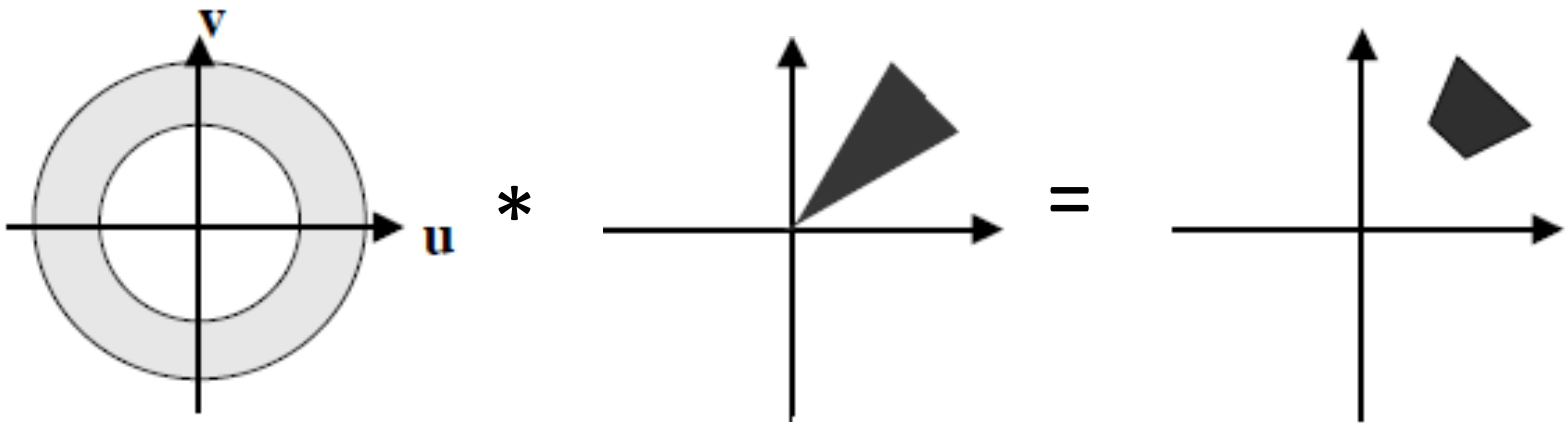
Bandwidth selection



Source: Richard Wildes slides

Scale and orientation

Combined analysis

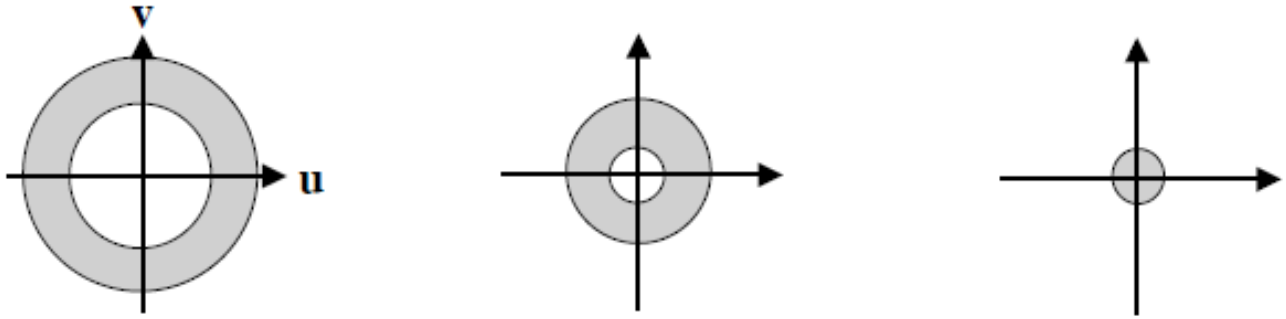


Source: Richard Wildes slides

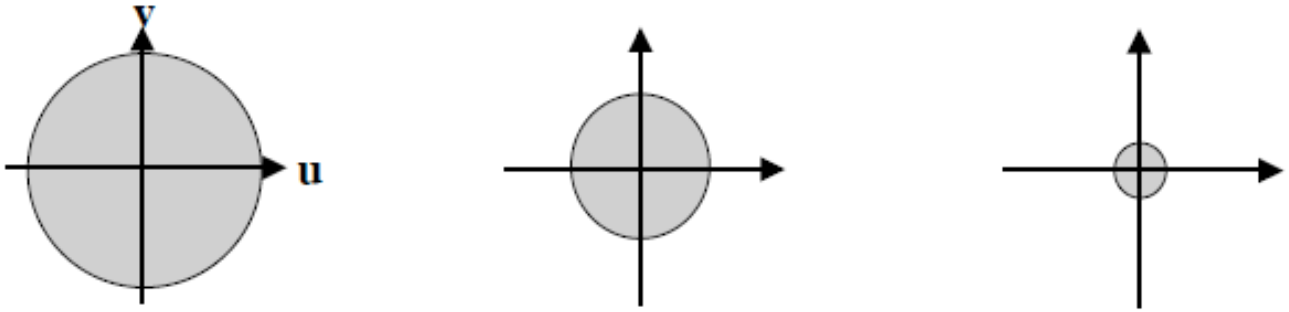
Scale and orientation

- Band and low-pass filters revisited

Band-pass filters



Low-pass filters



Source: Richard Wildes slides

Image processing

- Introduction
- Point operators
- Linear filtering
- Non linear filtering
- The Fourier transform
- Pyramids and scales

universität innsbruck

OpenCV.
Image processing

Assoz.Prof. Antonio Rodríguez-Sánchez, PhD.

Mat: Filtering

```
Mat kern = (Mat_<char>(3,3) <<  0, -1,  0,
                              -1,  5, -1,
                               0, -1,  0);
```

```
filter2D(I, K, I.depth(), kern);
```

Mat: Filtering

```
Mat kern = (Mat_<char>(3,3) <<  0, -1,  0,
                               -1,  5, -1,
                                0, -1,  0);
```

```
filter2D(I, K, I.depth(), kern);
```

Example: Blending

```
#include <cv.h>
#include <highgui.h>
#include <iostream>

using namespace cv;

int main( int argc, char** argv )
{
 double alpha = 0.5; double beta; double input;

 Mat src1, src2, dst;

 /// Ask the user enter alpha
 std::cout<<" Simple Linear Blender "<<std::endl;
 std::cout<<"-----------------------"<<std::endl;
 std::cout<<"* Enter alpha [0-1]: ";
 std::cin>>input;

 /// We use the alpha provided by the user if it is between 0 and 1
 if( input >= 0.0 && input <= 1.0 )
   { alpha = input; }
```

Example: Blending

```
 /// Read image ( same size, same type )
 src1 = imread("../../images/LinuxLogo.jpg");
 src2 = imread("../../images/WindowsLogo.jpg");

 if( !src1.data ) { printf("Error loading src1 \n"); return -1; }
 if( !src2.data ) { printf("Error loading src2 \n"); return -1; }

 /// Create Windows
 namedWindow("Linear Blend", 1);

 beta = ( 1.0 - alpha );
 addWeighted( src1, alpha, src2, beta, 0.0, dst);

 imshow( "Linear Blend", dst );

 waitKey(0);
 return 0;
}
```

Example: Blending

```
 /// Read image ( same size, same type )
 src1 = imread("../../images/LinuxLogo.jpg");
 src2 = imread("../../images/WindowsLogo.jpg");

 if( !src1.data ) { printf("Error loading src1 \n"); return -1; }
 if( !src2.data ) { printf("Error loading src2 \n"); return -1; }

 /// Create Windows
 namedWindow("Linear Blend", 1);

 beta = ( 1.0 - alpha );
 addWeighted( src1, alpha, src2, beta, 0.0, dst);

 imshow( "Linear Blend", dst );

 waitKey(0);
 return 0;
}
```

Example: Brightness and contrast

```
#include <cv.h>
#include <highgui.h>
#include <iostream>

using namespace cv;
double alpha; /**< Simple contrast control */
int beta;  /**< Simple brightness control */

int main( int argc, char** argv )
{
 /// Read image given by user
 Mat image = imread( argv[1] );
 Mat new_image = Mat::zeros( image.size(), image.type() );

 /// Initialize values
 std::cout<<" Basic Linear Transforms "<<std::endl;
 std::cout<<"-------------------------"<<std::endl;
 std::cout<<"* Enter the alpha value [1.0-3.0]: ";std::cin>>alpha;
 std::cout<<"* Enter the beta value [0-100]: "; std::cin>>beta;

 /// Do the operation new_image(i,j) = alpha*image(i,j) + beta
 for( int y = 0; y < image.rows; y++ )
    { for( int x = 0; x < image.cols; x++ )
         { for( int c = 0; c < 3; c++ )
              {
      new_image.at<Vec3b>(y,x)[c] =
         saturate_cast<uchar>( alpha*( image.at<Vec3b>(y,x)[c] ) + beta );
             }
    }
```

Example: Brightness and contrast

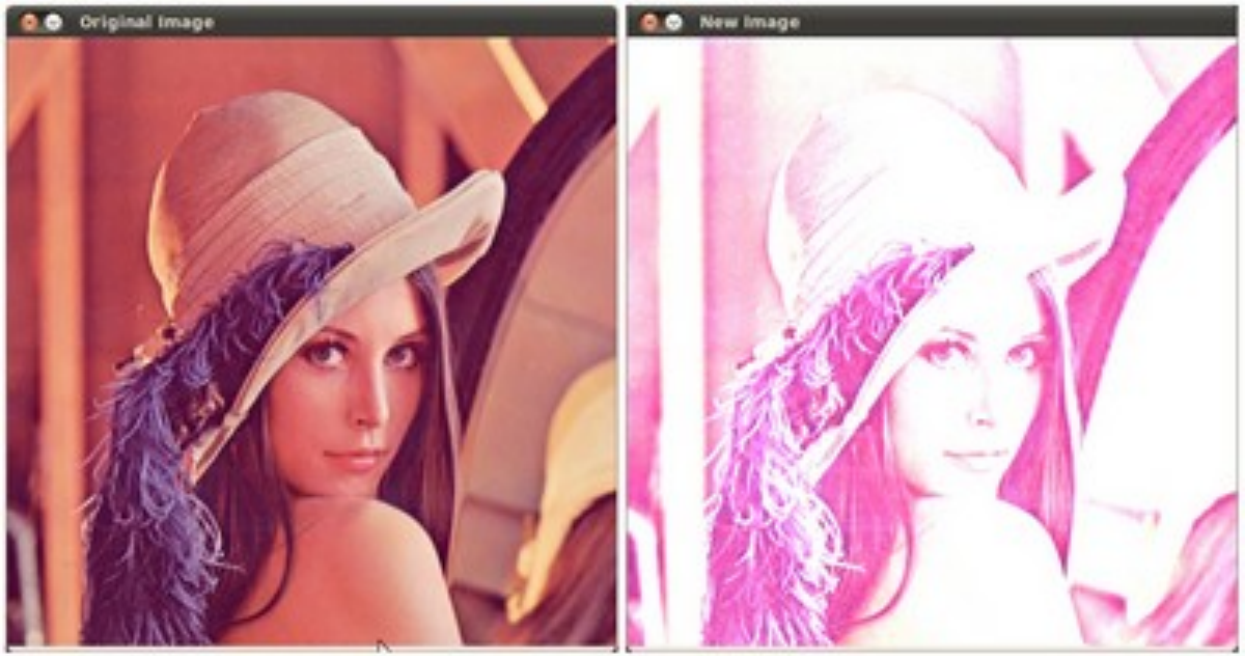
```
#include <cv.h>
#include <highgui.h>
#include <iostream>

using namespace cv;
double alpha; /**< Simple contrast control */
int beta;  /**< Simple brightness control */

int main( int argc, char** argv )
{
 /// Read image given by user
 Mat image = imread( argv[1] );
 Mat new_image = Mat::zeros( image.size(), image.type() );

 /// Initialize values
 std::cout<<" Basic Linear Transforms "<<std::endl;
 std::cout<<"-------------------------"<<std::endl;
 std::cout<<"* Enter the alpha value [1.0-3.0]: ";std::cin>>alpha;
 std::cout<<"* Enter the beta value [0-100]: "; std::cin>>beta;

 /// Do the operation new_image(i,j) = alpha*image(i,j) + beta
 for( int y = 0; y < image.rows; y++ )
    { for( int x = 0; x < image.cols; x++ )
         { for( int c = 0; c < 3; c++ )
              {
    new_image.at<Vec3b>(y,x)[c] =
       saturate_cast<uchar>( alpha*( image.at<Vec3b>(y,x)[c] ) + beta );
              }
    }
```

Example: Brightness and contrast

```
 /// Do the operation new_image(i,j) = alpha*image(i,j) + beta
 for( int y = 0; y < image.rows; y++ )
    { for( int x = 0; x < image.cols; x++ )
         { for( int c = 0; c < 3; c++ )
              {
      new_image.at<Vec3b>(y,x)[c] =
         saturate_cast<uchar>( alpha*( image.at<Vec3b>(y,x)[c] ) + beta );
             }
    }
    }

 /// Create Windows
 namedWindow("Original Image", 1);
 namedWindow("New Image", 1);

 /// Show stuff
 imshow("Original Image", image);
 imshow("New Image", new_image);

 /// Wait until user press some key
 waitKey();
 return 0;
}
```

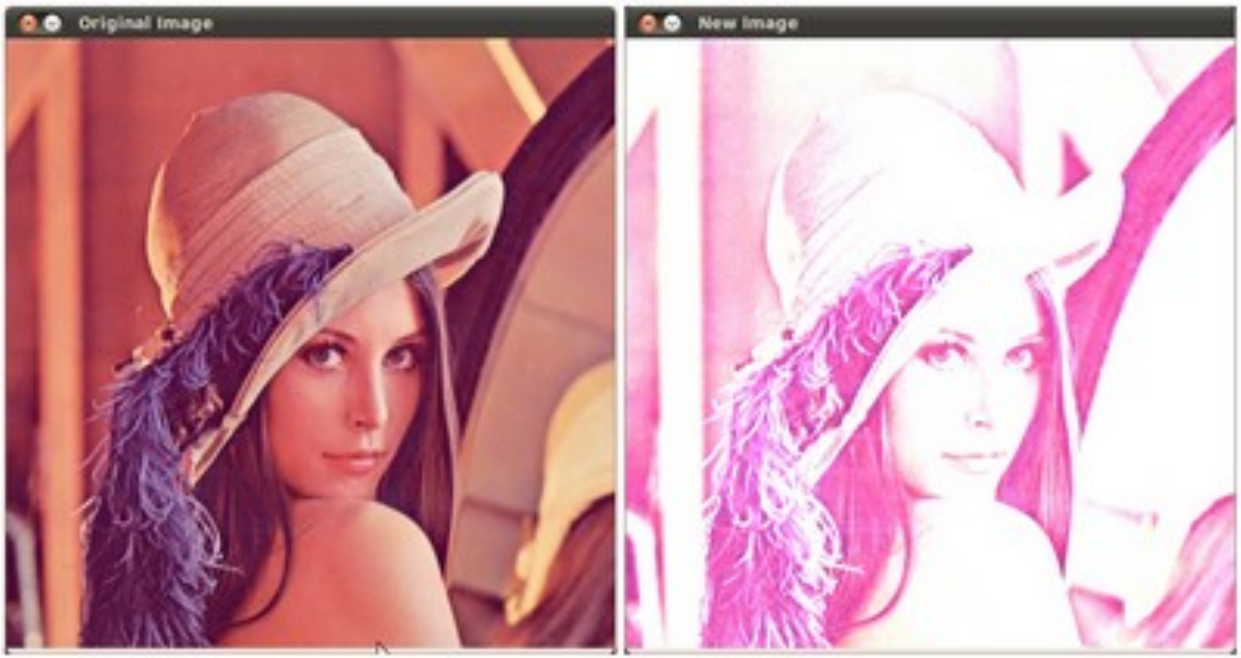
Example: Brightness and contrast

```
$ ./BasicLinearTransforms lena.jpg
Basic Linear Transforms
-------------------------
* Enter the alpha value [1.0-3.0]: 2.2
* Enter the beta value [0-100]: 50
```



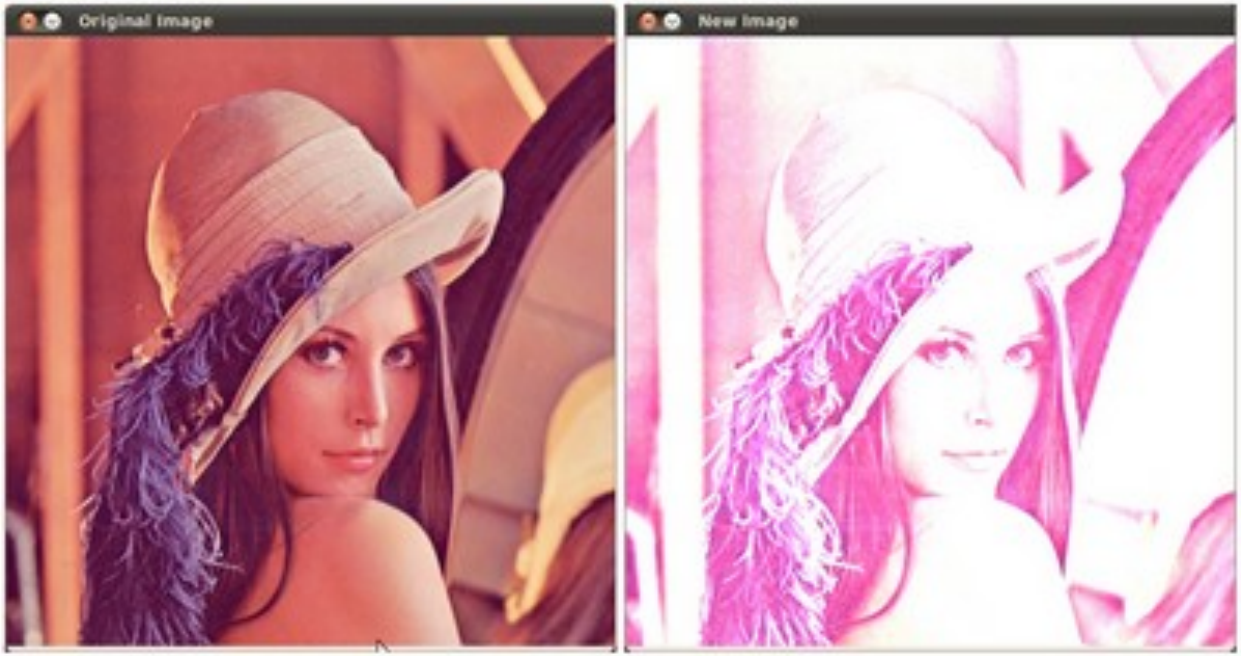
Example: Brightness and contrast

```
/// Do the operation new_image(i,j) = alpha*image(i,j) + beta
for( int y = 0; y < image.rows; y++ )
   { for( int x = 0; x < image.cols; x++ )
        { for( int c = 0; c < 3; c++ )
             {
     new_image.at<Vec3b>(y,x)[c] =
        saturate_cast<uchar>( alpha*( image.at<Vec3b>(y,x)[c] ) + beta );
            }
   }
   }

/// Create Windows
namedWindow("Original Image", 1);
namedWindow("New Image", 1);

/// Show stuff
imshow("Original Image", image);
imshow("New Image", new_image);

/// Wait until user press some key
waitKey();
return 0;
}
```



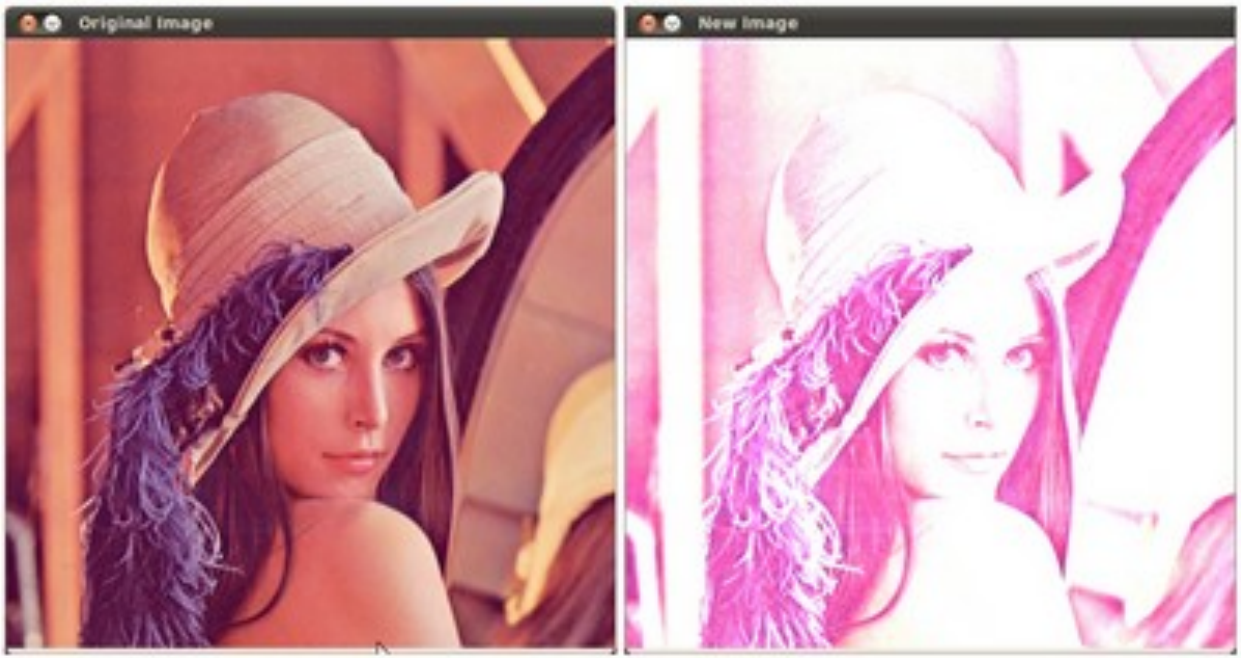
Example: Brightness and contrast

```
 /// Do the operation new_image(i,j) = alpha*image(i,j) + beta
 for( int y = 0; y < image.rows; y++ )
    { for( int x = 0; x < image.cols; x++ )
         { for( int c = 0; c < 3; c++ )
              {
      new_image.at<Vec3b>(y,x)[c] =
         saturate_cast<uchar>( alpha*( image.at<Vec3b>(y,x)[c] ) + beta );
             }
    }
    }

 /// Create Windows
 namedWindow("Original Image", 1);
 namedWindow("New Image", 1);

 /// Show stuff
 imshow("Original Image", image);
 imshow("New Image", new_image);

 /// Wait until user press some key
 waitKey();
 return 0;
}
```



Example: Brightness and contrast

```
 /// Do the operation new_image(i,j) = alpha*image(i,j) + beta
   image.convertTo(new_image, -1, alpha, beta);
 /// Create Windows
 namedWindow("Original Image", 1);
 namedWindow("New Image", 1);

 /// Show stuff
 imshow("Original Image", image);
 imshow("New Image", new_image);

 /// Wait until user press some key
 waitKey();
 return 0;
}
```



Discrete Fourier transform

```
#include "opencv2/core/core.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <iostream>
int main(int argc, char ** argv)
{
    const char* filename = argc >=2 ? argv[1] : "lena.jpg";

    Mat I = imread(filename, CV_LOAD_IMAGE_GRAYSCALE);
    if( I.empty())
        return -1;

    Mat padded;                            //expand input image to optimal size
    int m = getOptimalDFTSize( I.rows );
    int n = getOptimalDFTSize( I.cols ); // on the border add zero values
    copyMakeBorder(I, padded, 0, m - I.rows, 0, n - I.cols, BORDER_CONSTANT, Scalar::all(0));
```

Discrete Fourier transform

```
#include "opencv2/core/core.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <iostream>
int main(int argc, char ** argv)
{
    const char* filename = argc >=2 ? argv[1] : "lena.jpg";

    Mat I = imread(filename, CV_LOAD_IMAGE_GRAYSCALE);
    if( I.empty())
        return -1;

    Mat padded;                            //expand input image to optimal size
    int m = getOptimalDFTSize( I.rows );
    int n = getOptimalDFTSize( I.cols ); // on the border add zero values
    copyMakeBorder(I, padded, 0, m - I.rows, 0, n - I.cols, BORDER_CONSTANT, Scalar::all(0));
```

Discrete Fourier transform

```
#include "opencv2/core/core.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <iostream>
int main(int argc, char ** argv)
{
    const char* filename = argc >=2 ? argv[1] : "lena.jpg";

    Mat I = imread(filename, CV_LOAD_IMAGE_GRAYSCALE);
    if( I.empty())
        return -1;

    Mat padded;                            //expand input image to optimal size
    int m = getOptimalDFTSize( I.rows );
    int n = getOptimalDFTSize( I.cols ); // on the border add zero values
    copyMakeBorder(I, padded, 0, m - I.rows, 0, n - I.cols, BORDER_CONSTANT, Scalar::all(0));
```

Discrete Fourier transform

```
#include "opencv2/core/core.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <iostream>
int main(int argc, char ** argv)
{
    const char* filename = argc >=2 ? argv[1] : "lena.jpg";

    Mat I = imread(filename, CV_LOAD_IMAGE_GRAYSCALE);
    if( I.empty())
        return -1;

    Mat padded;                            //expand input image to optimal size
    int m = getOptimalDFTSize( I.rows );
    int n = getOptimalDFTSize( I.cols ); // on the border add zero values
    copyMakeBorder(I, padded, 0, m - I.rows, 0, n - I.cols, BORDER_CONSTANT, Scalar::all(0));
```

Discrete Fourier transform

```
Mat padded;                            //expand input image to optimal size
int m = getOptimalDFTSize( I.rows );
int n = getOptimalDFTSize( I.cols ); // on the border add zero values
copyMakeBorder(I, padded, 0, m - I.rows, 0, n - I.cols, BORDER_CONSTANT, Scalar::all(0));

Mat planes[] = {Mat_<float>(padded), Mat::zeros(padded.size(), CV_32F)};
Mat complexI;
merge(planes, 2, complexI);         // Add to the expanded another plane with zeros

dft(complexI, complexI);            // this way the result may fit in the source matrix

// compute the magnitude and switch to logarithmic scale
// => log(1 + sqrt(Re(DFT(I))^2 + Im(DFT(I))^2))
split(complexI, planes);                   // planes[0] = Re(DFT(I), planes[1] = Im(DFT(I))
magnitude(planes[0], planes[1], planes[0]);// planes[0] = magnitude
Mat magI = planes[0];

magI += Scalar::all(1);                    // switch to logarithmic scale
log(magI, magI);

// crop the spectrum, if it has an odd number of rows or columns
magI = magI(Rect(0, 0, magI.cols & -2, magI.rows & -2));
```

Discrete Fourier transform

```
Mat padded;                            //expand input image to optimal size
int m = getOptimalDFTSize( I.rows );
int n = getOptimalDFTSize( I.cols ); // on the border add zero values
copyMakeBorder(I, padded, 0, m - I.rows, 0, n - I.cols, BORDER_CONSTANT, Scalar::all(0));

Mat planes[] = {Mat_<float>(padded), Mat::zeros(padded.size(), CV_32F)};
Mat complexI;
merge(planes, 2, complexI);         // Add to the expanded another plane with zeros

dft(complexI, complexI);            // this way the result may fit in the source matrix

// compute the magnitude and switch to logarithmic scale
// => log(1 + sqrt(Re(DFT(I))^2 + Im(DFT(I))^2))
split(complexI, planes);                   // planes[0] = Re(DFT(I), planes[1] = Im(DFT(I))
magnitude(planes[0], planes[1], planes[0]);// planes[0] = magnitude
Mat magI = planes[0];

magI += Scalar::all(1);                    // switch to logarithmic scale
log(magI, magI);

// crop the spectrum, if it has an odd number of rows or columns
magI = magI(Rect(0, 0, magI.cols & -2, magI.rows & -2));
```

Discrete Fourier transform

```
Mat padded;                            //expand input image to optimal size
int m = getOptimalDFTSize( I.rows );
int n = getOptimalDFTSize( I.cols ); // on the border add zero values
copyMakeBorder(I, padded, 0, m - I.rows, 0, n - I.cols, BORDER_CONSTANT, Scalar::all(0));

Mat planes[] = {Mat_<float>(padded), Mat::zeros(padded.size(), CV_32F)};
Mat complexI;
merge(planes, 2, complexI);         // Add to the expanded another plane with zeros

dft(complexI, complexI);            // this way the result may fit in the source matrix

// compute the magnitude and switch to logarithmic scale
// => log(1 + sqrt(Re(DFT(I))^2 + Im(DFT(I))^2))
split(complexI, planes);                   // planes[0] = Re(DFT(I), planes[1] = Im(DFT(I))
magnitude(planes[0], planes[1], planes[0]);// planes[0] = magnitude
Mat magI = planes[0];

magI += Scalar::all(1);                    // switch to logarithmic scale
log(magI, magI);

// crop the spectrum, if it has an odd number of rows or columns
magI = magI(Rect(0, 0, magI.cols & -2, magI.rows & -2));
```

Discrete Fourier transform

```
// crop the spectrum, if it has an odd number of rows or columns
magI = magI(Rect(0, 0, magI.cols & -2, magI.rows & -2));

// rearrange the quadrants of Fourier image  so that the origin is at the image center
int cx = magI.cols/2;
int cy = magI.rows/2;

Mat q0(magI, Rect(0, 0, cx, cy));   // Top-Left - Create a ROI per quadrant
Mat q1(magI, Rect(cx, 0, cx, cy));  // Top-Right
Mat q2(magI, Rect(0, cy, cx, cy));  // Bottom-Left
Mat q3(magI, Rect(cx, cy, cx, cy)); // Bottom-Right

Mat tmp;                            // swap quadrants (Top-Left with Bottom-Right)
q0.copyTo(tmp);
q3.copyTo(q0);
tmp.copyTo(q3);

q1.copyTo(tmp);                     // swap quadrant (Top-Right with Bottom-Left)
q2.copyTo(q1);
tmp.copyTo(q2);

normalize(magI, magI, 0, 1, CV_MINMAX); // Transform the matrix with float values into a
                                        // viewable image form (float between values 0 and 1).

imshow("Input Image"       , I   );    // Show the result
imshow("spectrum magnitude", magI);
waitKey();

return 0;
```

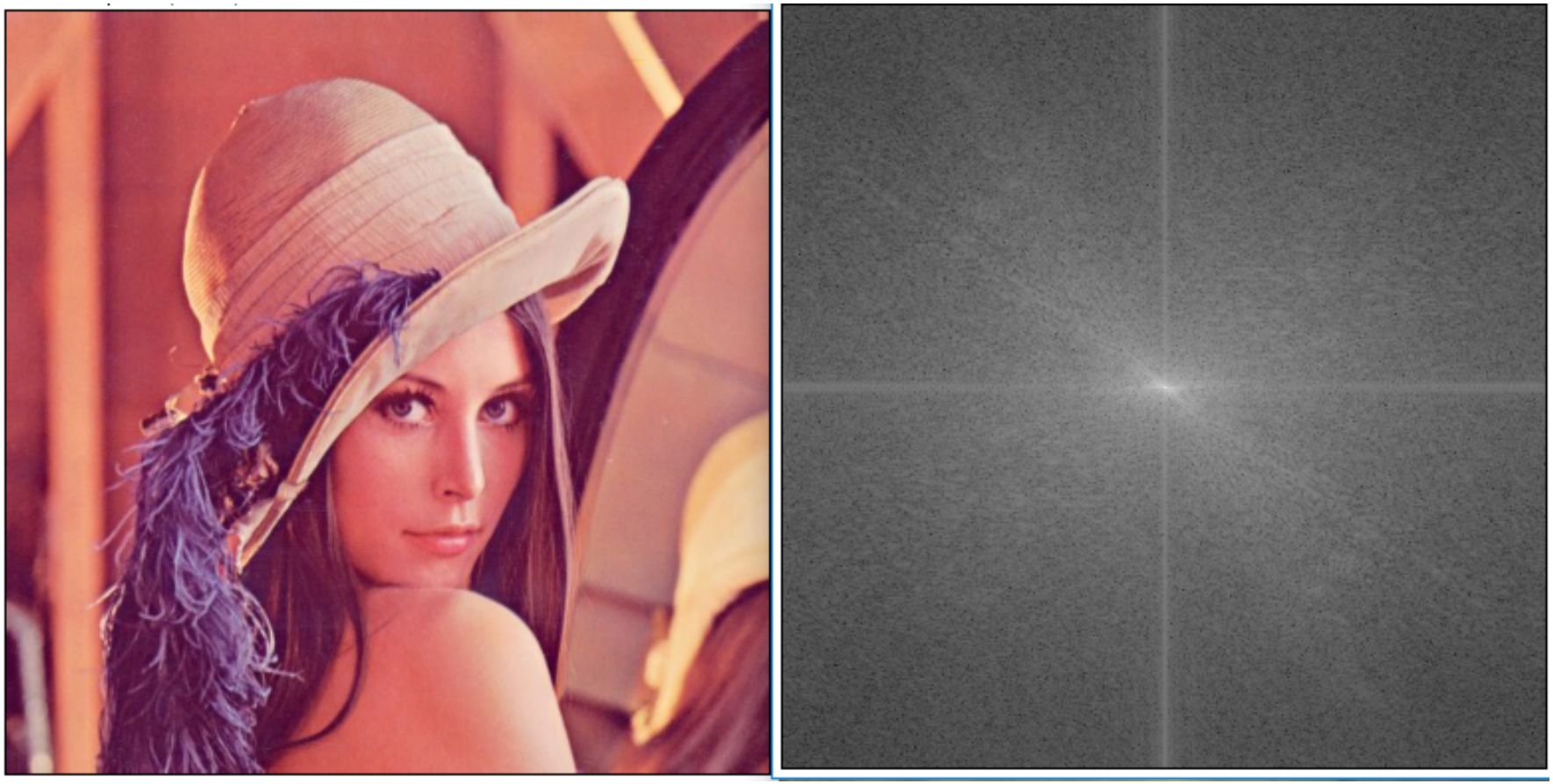
Discrete Fourier transform

```
// crop the spectrum, if it has an odd number of rows or columns
magI = magI(Rect(0, 0, magI.cols & -2, magI.rows & -2));

// rearrange the quadrants of Fourier image  so that the origin is at the image center
int cx = magI.cols/2;
int cy = magI.rows/2;

Mat q0(magI, Rect(0, 0, cx, cy));   // Top-Left - Create a ROI per quadrant
Mat q1(magI, Rect(cx, 0, cx, cy));  // Top-Right
Mat q2(magI, Rect(0, cy, cx, cy));  // Bottom-Left
Mat q3(magI, Rect(cx, cy, cx, cy)); // Bottom-Right

Mat tmp;                            // swap quadrants (Top-Left with Bottom-Right)
q0.copyTo(tmp);
q3.copyTo(q0);
tmp.copyTo(q3);

q1.copyTo(tmp);                     // swap quadrant (Top-Right with Bottom-Left)
q2.copyTo(q1);
tmp.copyTo(q2);

normalize(magI, magI, 0, 1, CV_MINMAX); // Transform the matrix with float values into a
                                        // viewable image form (float between values 0 and 1).

imshow("Input Image"       , I   );    // Show the result
imshow("spectrum magnitude", magI);
waitKey();

return 0;
```

Discrete Fourier transform

```
// crop the spectrum, if it has an odd number of rows or columns
magI = magI(Rect(0, 0, magI.cols & -2, magI.rows & -2));

// rearrange the quadrants of Fourier image  so that the origin is at the image center
int cx = magI.cols/2;
int cy = magI.rows/2;

Mat q0(magI, Rect(0, 0, cx, cy));   // Top-Left - Create a ROI per quadrant
Mat q1(magI, Rect(cx, 0, cx, cy));  // Top-Right
Mat q2(magI, Rect(0, cy, cx, cy));  // Bottom-Left
Mat q3(magI, Rect(cx, cy, cx, cy)); // Bottom-Right

Mat tmp;                            // swap quadrants (Top-Left with Bottom-Right)
q0.copyTo(tmp);
q3.copyTo(q0);
tmp.copyTo(q3);

q1.copyTo(tmp);                     // swap quadrant (Top-Right with Bottom-Left)
q2.copyTo(q1);
tmp.copyTo(q2);

normalize(magI, magI, 0, 1, CV_MINMAX); // Transform the matrix with float values into a
                                        // viewable image form (float between values 0 and 1).

imshow("Input Image"       , I   );    // Show the result
imshow("spectrum magnitude", magI);
waitKey();

return 0;
```

Discrete Fourier transform



Histogram equalization

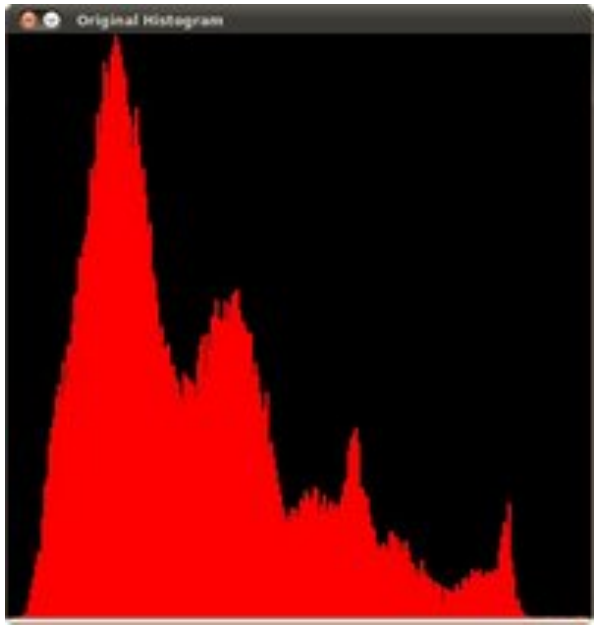
```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
#include <stdio.h>

using namespace cv;
using namespace std;

/** @function main */
int main( int argc, char** argv )
{
  Mat src, dst;

  char* source_window = "Source image";
  char* equalized_window = "Equalized Image";

  /// Load image
  src = imread( argv[1], 1 );

  if( !src.data )
    { cout<<"Usage: ./Histogram_Demo <path_to_image>"<<endl;
      return -1;}

  /// Convert to grayscale
  cvtColor( src, src, CV_BGR2GRAY );
```

Histogram equalization

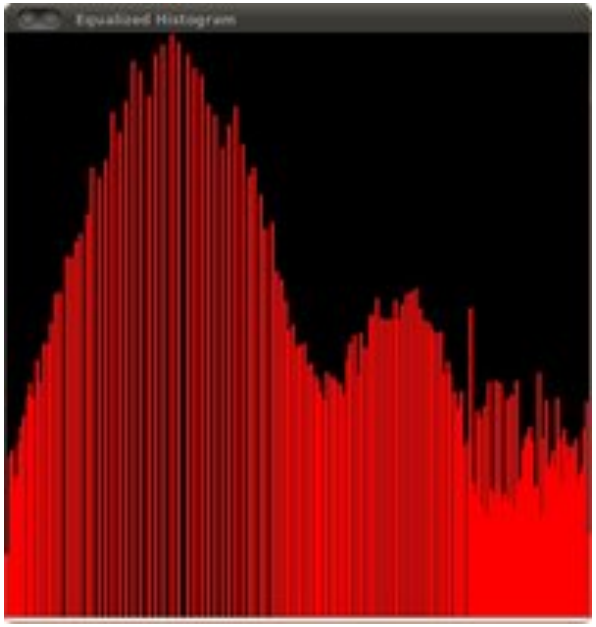
```
  /// Convert to grayscale
  cvtColor( src, src, CV_BGR2GRAY );

  /// Apply Histogram Equalization
  equalizeHist( src, dst );

  /// Display results
  namedWindow( source_window, CV_WINDOW_AUTOSIZE );
  namedWindow( equalized_window, CV_WINDOW_AUTOSIZE );

  imshow( source_window, src );
  imshow( equalized_window, dst );

  /// Wait until user exits the program
  waitKey(0);

  return 0;
}
```

Histogram equalization

Histogram equalization



Histogram equalization



Histogram equalization