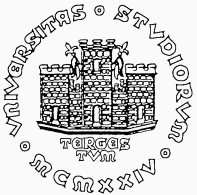
Computer Vision and Pattern Recognition

Course ID: 554SM – Fall 2018

Felice Andrea Pellegrino

University of Trieste
Department of Engineering and Architecture



554SM –Fall 2018

Lecture 3: Image Processing

Linear filtering

Images as functions

We can think of an *image* as a function

$$f : \mathbb{R}^2 \longrightarrow \mathbb{R}$$

- $f(x, y)$ gives the *intensity* at location (x, y)
- Usually, the domain is a rectangle and the intensity range is bounded:

$$f : [a, b] \times [c, d] \longrightarrow \left[I^-, I^+\right]$$

A *color image* is a vector-valued function where the intensity of (usually three) different *channels*, for instance red, green and blue, is represented:

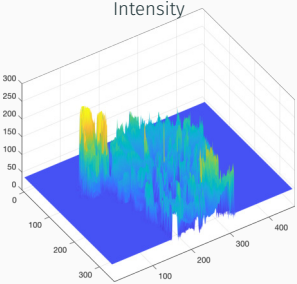
$$f(x, y) = \begin{bmatrix} r(x, y) \\ g(x, y) \\ b(x, y) \end{bmatrix}.$$

Images as functions (cont.)

Color image

Red channel

Intensity



Intensity as a surface

Digital images

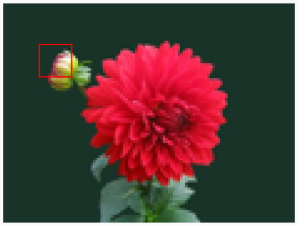
A *digital* image is obtained by

- *Sampling* the 2D space on a regular grid
- *Quantizing* each sample (rounding to the nearest value of a set of discrete values)

If the samples are Δ apart, we obtain (with a slight abuse of notation):

$$f(i,j) = \text{Quantize}\left(f(i\Delta, j\Delta)\right).$$

Thus the image may be represented as a matrix and i, j are row and column indices, respectively. For instance



25	25	25	25	25	25	25	25	25	25	27	25	24
25	25	25	25	25	25	25	25	25	26	24	22	23	25
25	25	25	25	25	25	25	25	24	24	56	40	18	25
25	25	25	25	25	25	24	22	20	39	111	190	128	28
25	25	25	25	25	26	24	31	132	195	178	189	227	182
25	25	25	25	25	23	27	114	172	169	211	230	217	232
25	25	25	25	25	21	55	173	212	234	247	251	255	240
25	25	25	24	25	26	111	193	203	218	229	240	248	228
25	24	24	23	24	42	178	220	231	242	240	242	224	219
25	24	24	26	23	48	156	192	209	221	244	247	219	186
25	25	26	24	22	89	172	161	165	179	194	217	228	182
25	25	25	26	22	101	142	149	142	155	163	168	186	171
25	25	24	23	68	138	136	139	156	154	153	143	134	131
25	24	23	34	136	194	175	153	141	150	162	146	109	80

Red channel of a portion of the image

Local operators

- An *operator* $h(\cdot)$ is a function that takes one or more input images and produces an output image:

$$g(i,j) = (h \circ f)(i,j) = h(f(i,j))$$

- An operator is said to be *local* when a value of a given pixel depends on a collection of pixel values in its neighborhood, for instance the averaging of a box of size $2W+1$ centered in the pixel

$$g(i,j) = \frac{1}{(2W+1)^2} \sum_{k=-W}^{W} \sum_{l=-W}^{W} f(i+k, j+l). \tag{1}$$

199	117	74	88	118	45	62	175
108	224	172	200	109	185	234	140
24	133	178	173	118	121	69	109
68	241	18	02	197	39	196	165
40	163	65	154	83	87	49	166
72	245	58	99	201	155	74	174
113	62	171	234	121	49	24	163
135	173	216	1	10	189	147	242

$*$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$=$

137	151	137	129	118	127
130	149	130	127	141	140
103	125	110	108	107	111
108	116	97	113	120	123
110	139	132	131	94	105
138	140	123	118	108	135

- (1) is an example of *linear filter*, where the output pixel's value is a weighted sum of input pixel's values

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l). \tag{2}$$

  The entries in the weight *kernel* $h(\cdot)$ are called the *filter coefficients*.

- The operator (2) is called *correlation* and is denoted as

$$g = f \otimes h.$$

- A common variant of (2) is

$$g(i,j) = \sum_{k,l} f(i-k, j-l)h(k,l) = \sum_{k,l} f(k,l)h(i-k, j-l) \tag{3}$$

  that is the *convolution* operator, denoted by

$$g = f * h.$$

- For radially symmetric kernels (i.e. $h(k,l) = h(-k,-l)$) convolution and correlation are the same.

Properties

Both correlation and convolution are

- *shift-invariant*: the value of the output depends on the pattern in an image neighborhood rather than the position of the neighborhood:

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

  where $\circ$ stands for a general operator. Shift-invariance implies that the operator "behaves the same everywhere";

- *linear*: they obey the superimposition principle:

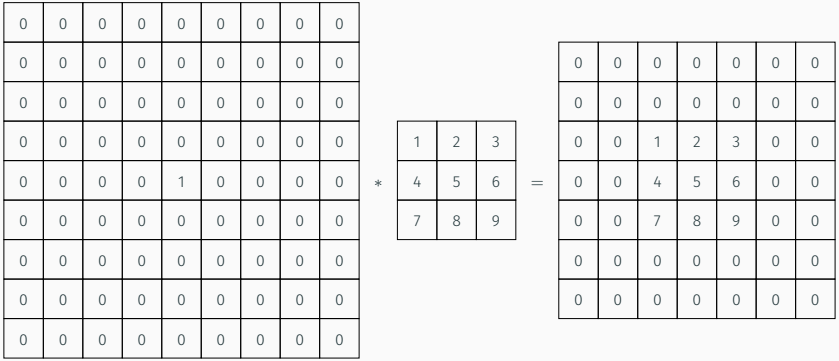
$$h \circ (\alpha f_0 + \beta f_1) = \alpha(h \circ f_0) + \beta(h \circ f_1) \qquad \forall \alpha, \beta \in \mathbb{R}$$

  i.e. "the filtered linear combination of two images is the linear combination of the filtered images, with same coefficients."

Impulse response

- The kernel h of a convolution operator is called the *impulse response function*, because h, convolved with an impulse signal $\delta(i, j)$ (an image that is zero everywhere except at the origin), reproduces itself:

$$\delta * h = h.$$



Convolution vs correlation

Convolution and correlation are useful for, e.g.

- Blurring
- Sharpening
- Edge Detection
- Interpolation

Convolution has a number of nice properties

- Commutative

$$f_1 * f_2 = f_2 * f_1$$

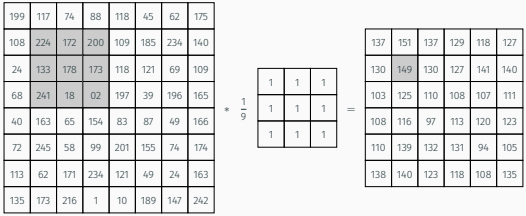
- Associative

$$f_1 * (f_2 * f_3) = (f_1 * f_2) * f_3$$

- Convolution corresponds to product in the Fourier domain

$$\mathcal{F}[f_1 * f_2] = \mathcal{F}[f_1]\mathcal{F}[f_2]$$

Border effects

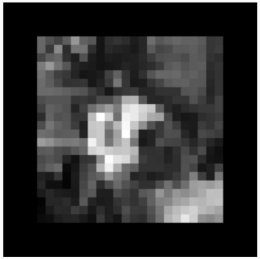


In the previous examples of linear filtering, the result was *smaller* then the original image. The reason is that the pixels in the border do not have a complete neighborhood to compute the weighted sum. Sometimes it is desirable to get a filtered image of the same dimension of the original. This can be achieved by *extending the original image* (*padding*).

Some common padding modes:

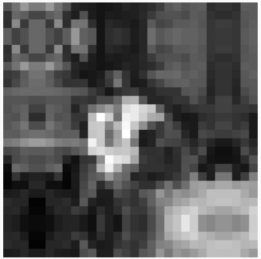
- *zero* (set all pixels outside the border to 0)
- *constant border* (set all pixels outside the border to a specified border value)
- *replicate* (or *clamp*, repeat border pixel indefinitely)
- *wrap* (loop around the image in a toroidal configuration)
- *mirror* (reflect pixels across border)

Border effects (cont.)

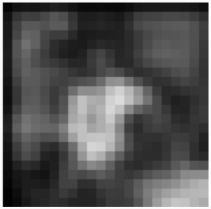


zero

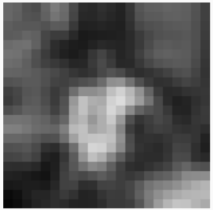
clamp



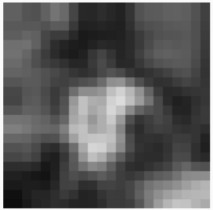
mirror



blurred zero



blurred clamp



blurred mirror

Examples of linear filters

`Box filter`

The *box* filter averages the pixel values in a square window, for instance

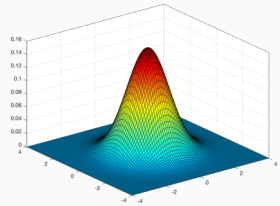
$$h = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

`Gaussian filter`

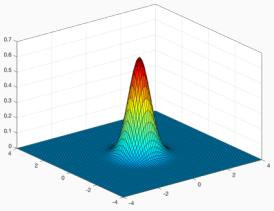
The *Gaussian* filter is obtained from:

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

where $\sigma > 0$ is referred to as the standard deviation of the Gaussian.



$\sigma = 1$



$\sigma = 1/2$

Examples of linear filters (cont.)

The standard deviation σ controls the size of the neighborhood that actually affects the center pixel in the filtered image:

- if σ is very small, the filtered image will be very close to the original (neighboring pixels will have very small weights);
- for larger σ a smoothing effect will occur: noise will be reduced at the cost of some blurring;
- for very large σ much of the image detail will disappear.

original

$\sigma = 1$

$\sigma = 2$

In applications, a kernel may be obtained by constructing a $(2W+1) \times (2W+1)$ array:

$$h(i,j) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(i-W-1)^2 + (j-W-1)^2}{2\sigma^2}\right)$$

Examples of linear filters (cont.)

Bilinear filter

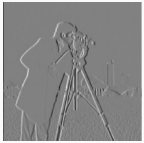
The outer product of two piecewise linear "tent" functions:

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

Sobel filter

A separable combination of a central difference and a tent filter:

$$\frac{1}{8}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \frac{1}{2}\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}.$$



Corner filter

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

Frequency and orientation

Filters seen so far may be categorized according to the spatial frequency behavior and orientation.

filter	frequency behavior	orientation
box	low-pass	not oriented
Gaussian	low-pass	not oriented
bilinear	low-pass	not oriented
Sobel	band-pass	oriented
corner	band-pass	not oriented

Band-pass filters

More sophisticated band-pass filters can be created in two steps as follows

1. Smooth the image with a Gaussian $G(x, y; \sigma)$
2. Take the first or second derivatives

Laplacian of Gaussian

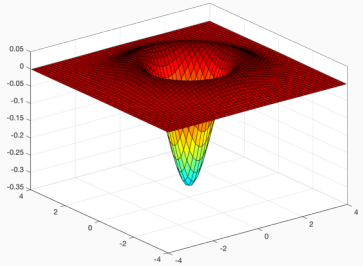
When in step 2 we use the *Laplacian* operator, i.e. the undirected second derivative

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

the result is equivalent to convolving the image with the *Laplacian of Gaussian* (LoG) filter:

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

The LoG filter is not oriented, and has a center-surround response.



LoG filter ($\sigma = 1$)

`Directional derivative`

Define the gradient operator as

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

The directional derivative along the direction $d = (u, v) = (\cos\vartheta, \sin\vartheta)$ is the dot product between d and the gradient ∇f:

$$\nabla_d f = d \cdot \nabla f.$$

When in step 2 we use the directional derivative we get

$$\nabla_d (G * f) = (\nabla_d G) * f = (d \cdot \nabla G) * f.$$

The *smoothed directional derivative* filter

$$d \cdot \nabla G = u\frac{\partial G}{\partial x} + v\frac{\partial G}{\partial y}$$

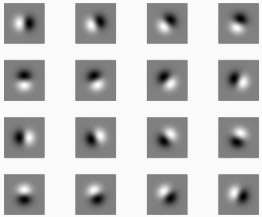
is an example of *steerable filter*.

Steerable filters

It is often needed to convolve the image with the same filter at different orientations.

Steerable filters (Freeman et al., 1991) are filters that can be applied to the image at different orientations, with few convolution operations.

Indeed, it is sufficient to convolve the image with a limited number of *basis filters* and then *steer* the filter along the desired direction.



Smoothed directional derivative filter at 16 orientations

The directional derivative is steerable because

$$d \cdot \nabla G = u\frac{\partial G}{\partial x} + v\frac{\partial G}{\partial y} = uG_x + vG_y.$$

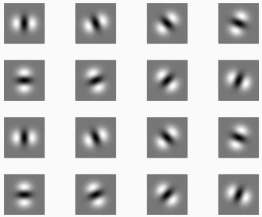
Thus, the directional derivative at any orientation may be obtained by combining the two basis filters G_x and G_y, instead of convolving with a rotated filter.

Steerable filters (cont.)

In (Freeman et al., 1991), a steerable basis set for the second derivative of a Gaussian

$$G_{xx} = \left(4x^2 - 2\right) \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

is proposed.



Smoothed directional second derivative filter at 16 orientations

If $u = (\cos\vartheta, \sin\vartheta)$ is the desired direction, it turns out that

$$G_{uu} = \sum_{i=1}^{3} k_i(\vartheta) G_{u_i u_i}$$

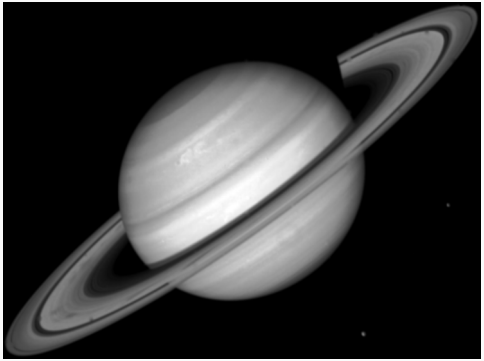
where

- $G_{u_i u_i}$ is the second derivative along the direction $u_i = (\cos\vartheta_i, \sin\vartheta_i)$
- $\vartheta_1 = 0$, $\vartheta_2 = \frac{\pi}{3}$, $\vartheta_3 = 2\frac{\pi}{3}$
- $k_i(\vartheta) = \frac{1}{3}\left[1 + 2\cos\left(2\left(\vartheta - \vartheta_i\right)\right)\right]$

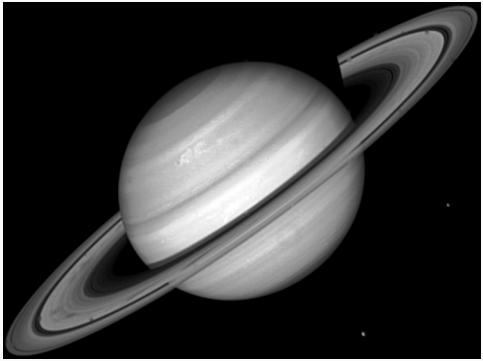
Unsharp masking

Blurring an image reduces high frequency, thus the difference between the original image and its blurred version represents high frequency content of the original image.
Adding some of the difference, will result in a sharper image. The process is called *unsharp masking*:

$$f_{\text{sharp}} = f + \gamma(f - f * h_{\text{blur}}) = f * [(\gamma + 1)\delta - \gamma h_{\text{blur}}]$$



Original image f (of size 1747×1302).



Sharpened image $f + \gamma(f - G(\sigma) * f)$, where $\gamma = 2$ and $\sigma = 3$.

Separable filtering

- Performing a convolution requires W^2 operations (multiply-add) per pixel where W is the size (width or height) of the kernel.
- In some cases a significant speed-up can be achieved by performing *a one-dimensional horizontal convolution followed by a one-dimensional horizontal convolution* (requiring a $2W$ operations per pixel).
- A convolution kernel K for which this is possible is said to be *separable*.
- It is easy to show[1] that the two-dimensional kernel K corresponding to successive convolution with a horizontal kernel $h^\top$ and a vertical kernel v is

$$K = vh^\top, \tag{4}$$

for instance

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}.$$

[1] by computing $v * h^\top * \delta$

Separable filtering (cont.)

How can we tell if a given K is separable?
Notice that (4) implies that a necessary condition is that

$$\operatorname{rank} K = 1.$$

The condition is also sufficient because any rank-one K may be written as

$$K = \sum_{i=1}^{n} \sigma_i u_i v_i^\top = \sigma_1 u_1 v_1^\top,$$

since only the first singular value σ_1 is non-zero. Then, the horizontal and vertical kernels may be taken as $\sqrt{\sigma_1} v_1^\top$ and $\sqrt{\sigma_1} u_1$ respectively.

When K is not separable, a rank-r approximation of K is

$$K \approx \sum_{i=1}^{r} \sigma_i u_i v_i^\top$$

thus approximate filtering may be obtained by applying r pairs of one-dimensional convolutions.

Separable filtering (cont.)

How can we tell if a given filter $f(x, y)$ is separable?

The condition to be checked is the existence of two functions $g(x)$ and $h(y)$ such that $f(x, y)$ factors as

$$f(x, y) = g(x)h(y).$$

For example, the Gaussian filter is separable:

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

Non-linear filters and other neighborhood operators

Median filter

The median filter is a non-linear filter that substitutes each pixel value with the *median* of the distribution of the neighborhood. Since the median is robust to outliers, median filtering helps in filtering out the *shot noise*, where linear filters, for instance Gaussian smoothing perform poorly.

original image with shot noise

Gaussian filtered

median filtered

`$\alpha$-trimmed mean`

A non-linear filter that averages together all the pixels except for the α fraction that are the smallest and the largest.

`Weighted median`

The *weighted median* corresponds to the median of a weighted distribution, where each pixel appears a number of times depending on its distance from the center of the neighborhood.
This is equivalent to minimizing the following objective function:

$$\sum_{k,l} w(k,l)|f(i+k,j+l) - g(i,j)|^p \tag{5}$$

where $-W \leq k, l \leq W$, the desired output value is $g(i,j)$ and $p = 1$.
For $p = 2$ the *weighted mean* (which is a linear filter) is obtained.

Bilateral filtering

Bilateral filtering (Tomasi and Manduchi, 1998) combines

- a weighting filter kernel
- a (soft) outlier rejection

The output pixel value is a weighted combination of neighboring pixel values $f(k,l)$:

$$g(i,j) = \frac{\sum_{k,l} f(k,l) w(i,j,k,l)}{\sum_{k,l} w(i,j,k,l)}. \tag{6}$$

where $(k,l) \in \mathcal{W}$, a neighborhood of pixel (i,j). For instance, a square neighborhood is obtained for $i - W \leq k \leq i + W$ and $j - W \leq l \leq j + W$, or equivalently $\max\{|i-k|, |j-l|\} \leq W$.

The weighting coefficient $w(i,j,k,l)$ is the product of a *domain kernel*

$$d(i,j,k,l) = \exp\left(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2}\right) \tag{7}$$

and a data-dependent *range kernel*

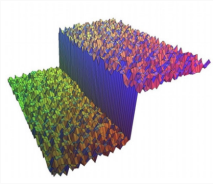
$$r(i,j,k,l) = \exp\left(-\frac{||f(i,j) - f(k,l)||^2}{2\sigma_r^2}\right) \tag{8}$$

Bilateral filtering (cont.)

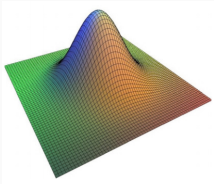
The data-dependent bilateral weight function is

$$w(i,j,k,l) = \exp\left(-\frac{(i-k)^2+(j-l)^2}{2\sigma_d^2} - \frac{||f(i,j)-f(k,l)||^2}{2\sigma_r^2}\right) \tag{9}$$

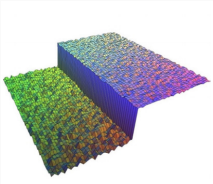
As a result, the weight is high only for those pixels that are spatially close to the center pixel AND whose value is similar to the value of the center pixel.



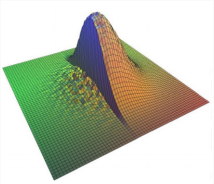
input



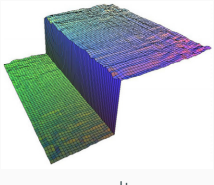
domain kernel



range kernel



weighting function



result

Figures from (Durand and Dorsey, 2002).

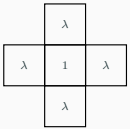
Anisotropic diffusion

Suppose to apply bilateral filtering:

- in an iterative fashion
- with a small neighborhood (using only the four nearest neighbors such that $|k-i|+|l-j| \leq 1$)

Then:

$$d(i,j,k,l) = \exp\left(-\frac{(i-k)^2+(j-l)^2}{2\sigma_d^2}\right)$$
$$= \begin{cases} 1, & |k-i|+|l-j| = 0 \\ \lambda = e^{-1/2\sigma_d^2}, & |k-i|+|l-j| = 1 \end{cases}$$



Thus, (6) becomes

$$f^{(t+1)}(i,j) = \frac{f^{(t)}(i,j) + \lambda \sum_{k,l} f^{(t)}(k,l) r(i,j,k,l)}{1 + \lambda \sum_{k,l} r(i,j,k,l)}$$
$$= f^{(t)}(i,j) + \frac{\lambda}{1 + \lambda \sum_{k,l} r(i,j,k,l)} \sum_{k,l} r(i,j,k,l) \left[f^{(t)}(k,l) - f^{(t)}(i,j)\right]$$

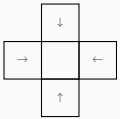
where (k,l) are the four neighbors of (i,j).

Anisotropic diffusion (cont.)

$$f^{(t+1)}(i,j) = f^{(t)}(i,j) + \frac{\lambda}{1 + \lambda \sum_{k,l} r(i,j,k,l)} \sum_{k,l} r(i,j,k,l) \left[f^{(t)}(k,l) - f^{(t)}(i,j) \right]$$

The above equation is the same as the discrete *anisotropic diffusion* equation, proposed by Perona and Malik (Perona and Malik, 1990). It acts as an "update rule": at each iteration, $f^{(t)}(i,j)$ is updated

- based on the difference w.r.t. the nearest neighbors ("diffusion")
- according to a range kernel ("anisotropic")



Anisotropic diffusion is an edge preserving filtering:

original image

Gaussian filtered

anisotropic diffusion filtered

Morphology

Non-linear filters are frequently used to process *binary images*. Binary images are obtained by a *thresholding* operation:

$$\theta(f, t) = \begin{cases} 1 & \text{if } f \geq t \\ 0 & \text{otherwise} \end{cases}$$

Morphological operations consist of two steps:

1. convolve the image with a binary structuring element (for instance a 3×3 box filter)
2. select a binary output based on the thresholded result of the convolution

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

a box-shaped structuring element

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

a “disk”-shaped structuring element

Morphology (cont.)

Let $c = f \otimes s$ be the count of the number of 1s inside each structuring element s as it is scanned over the image, and S be the number of 1s in the structuring element. The standard morphology operations are

- dilation: dilate$(f, s) = \theta(c, 1)$
- erosion: erosion$(f, s) = \theta(c, S)$
- majority: maj$(f, s) = \theta(c, S/2)$
- opening: open$(f, s) =$ dilate(erode$(f, s), s)$
- closing: close$(f, s) =$ erode(dilate$(f, s), s)$

original

dilation

erosion

majority

opening

closing

Distance transforms

The *distance transform* $D(i,j)$ of a binary image $b(i,j)$ is defined as

$$D(i,j) = \min_{k,l:b(k,l)=0} d(i-k, j-l)$$

where $d(i,j)$ is some *distance metric* between pixel offsets. Thus, the distance transform is *the distance to the nearest pixel whose value is 0*.

Commonly used metrics are:

City-block distance: $d(i,j) = |i| + |j|$

Euclidean distance: $d(i,j) = \sqrt{i^2 + j^2}$

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

original

0	0	0	0	1	0	0
0	0	1	1	1	0	0
0	1	2	2	2	1	0
0	1	2	2	1	1	0
0	1	2	1	0	0	0
0	0	1	0	0	0	0
0	0	0	0	0	0	0

city-block

0.00	0.00	0.00	0.00	1.00	0.00	0.00
0.00	0.00	1.00	1.00	1.00	0.00	0.00
0.00	1.00	1.41	2.00	1.41	1.00	0.00
0.00	1.00	2.00	1.41	1.00	1.00	0.00
0.00	1.00	1.41	1.00	0.00	0.00	0.00
0.00	0.00	1.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00

Euclidean

Connected components

Connected components are defined as regions of adjacent pixels having the same value. Finding connected components amounts to performing a sequence of neighborhood operations (basically, scanning the image horizontally downwards and then upwards).

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

original, binary image

0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	2	2	2	0	0
0	1	1	1	1	1	0	0	2	2	2	2	0	0
0	1	1	1	1	1	0	0	2	2	2	2	0	0
0	1	1	1	0	0	0	2	2	2	2	0	0	0
0	0	0	0	0	0	0	0	0	2	2	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3	3	3	0	0
0	0	0	0	0	0	0	0	0	3	3	3	3	0
0	0	0	0	0	0	0	0	0	3	3	3	3	0
0	0	0	0	0	0	0	0	0	0	3	3	3	0
0	0	0	0	0	0	0	0	0	0	0	0	3	0

labeled connected components

Connected components (cont.)

For each region some statistics may be useful, for instance:

- area (number of pixels)
- perimeter (number of boundary pixels)
- centroid:

$$(\bar{x}, \bar{y}) = (\frac{\sum x^{(i)}}{N}, \frac{\sum y^{(i)}}{N})$$

- the 2×2 matrix of second moments:

$$I = \begin{bmatrix} I_{xx} & I_{xy} \\ I_{yx} & I_{yy} \end{bmatrix} = \sum_{i=1}^{N} \begin{bmatrix} x^{(i)} - \bar{x} \\ y^{(i)} - \bar{y} \end{bmatrix} [x^{(i)} - \bar{x} \quad y^{(i)} - \bar{y}]$$

  from which the major and minor axis orientation and lengths can be computed using singular value decomposition.

- the *Hu moment invariants* (Hu, 1962).

Hu moment invariants

By defining the family of *central moments* (that are invariant to translation)

$$\mu_{pq} = \sum_i (x^{(i)} - \bar{x})^p (y^{(i)} - \bar{y})^q, \qquad p, q \geq 0,$$

and the scale invariants

$$\eta_{ij} = \frac{\mu_{ij}}{\mu_{00}^{\left(1+\frac{i+j}{2}\right)}}, \qquad i + j \geq 2,$$

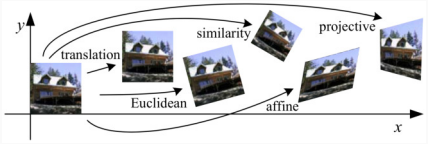
the following are the *Hu moment invariants* (Hu, 1962) and are invariant to translation, scaling and rotation and may be used a features for shape recognition:

$$\begin{aligned}
I_1 &= \eta_{20} + \eta_{02} \\
I_2 &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\
I_3 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\
I_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\
I_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + \\
&\quad (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
I_6 &= (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \\
I_7 &= (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] - \\
&\quad (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2].
\end{aligned}$$

Image warping

2D coordinate transformations

transformation	matrix	#dof	preserves	icon
translation	$\begin{bmatrix} I & t \end{bmatrix}_{2\times 3}$	2	orientation	
rigid	$\begin{bmatrix} R & t \end{bmatrix}_{2\times 3}$	3	lengths	
similarity	$\begin{bmatrix} sR & t \end{bmatrix}_{2\times 3}$	4	angles	
affine	$\begin{bmatrix} A \end{bmatrix}_{2\times 3}$	6	parallelism	
homography	$\begin{bmatrix} H \end{bmatrix}_{3\times 3}$	8	straight lines	



The transformations above:

- change the *domain* of an image;
- are global and parametric (apply a global deformation, controlled by a small number of parameters).

The radial distortion transformation is another example of global parametric transformation.

Forward warping

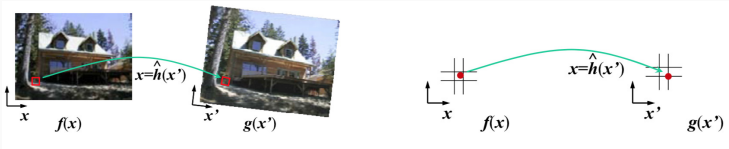
Algorithm Forward warping algorithm for transforming an image $f(x)$ into an image $g(x')$ through the transform $x' = h(x)$

procedure FORWARDWARP(f,h,out g)
For every pixel x in $f(x)$
  Compute the destination location $x' = h(x)$.
  Copy the pixel $f(x)$ to $g(x')$.
end procedure

Forward warping is the most natural approach but has two severe limitations:

- it is not clear what to do when x' has a non-integer value;
- cracks and holes appear, especially when magnifying an image.

Inverse warping

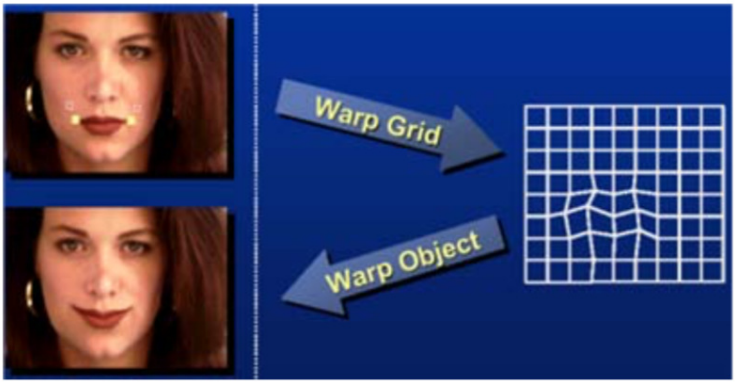


Algorithm Inverse warping algorithm for transforming an image $f(x)$ into an image $g(x')$ through the transform $x' = h(x)$

procedure INVERSEWARP(f,h,out g)
For every pixel x' in $g(x')$
  Compute the source location $x = h^{-1}(x) \doteq \hat{h}(x)$.
  Resample $f(x)$ at location x and copy to $g(x')$.
end procedure

The preferable way to perform warping. Resampling requires a carefully selected interpolation filter, because for a general (non-zoom) image transformation, the resampling rate r is not well defined (see (Szeliski, 2010), pg. 147).

Non-parametric transforms

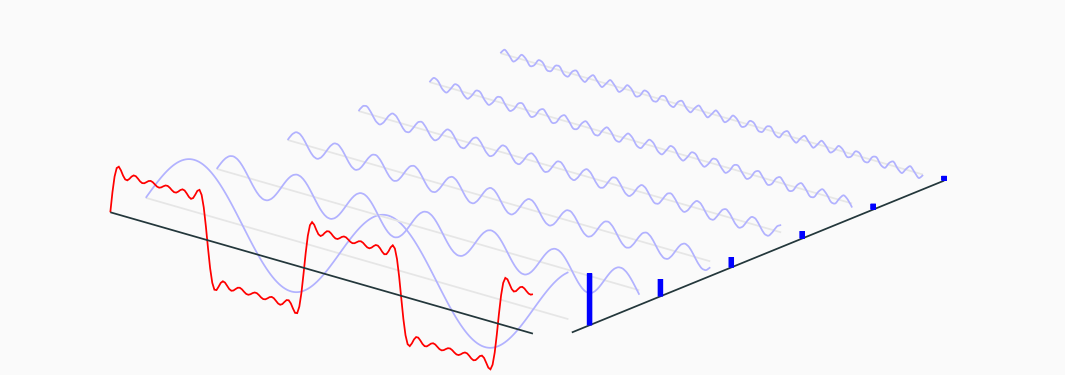


Sometimes, different parts of the image require a different amount of transformation, thus *local* transforms are needed. A popular approach is:

- define a *sparse* set of corresponding points;
- interpolate to a *dense displacement field* via *triangulation* followed by the application of an *affine* motion model within triangles.

Fourier transforms

Underlying idea



Underlying idea, due to Joseph Fourier (1768-1830):

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

The weights of the sum may provide useful information about the function itself. How the weights can be found?

Fourier transform a 1D signal

We define the Fourier transform of a signal $h(x)$ as

$$H(\omega) = \mathcal{F}\{h(x)\} = \int_{-\infty}^{\infty} h(x) e^{-j\omega x} dx \tag{10}$$

where j is the imaginary unit and ω is called the *angular frequency*.

Since $e^{-j\omega x} = \cos(\omega x) - j\sin(\omega x)$, we get

$$\begin{aligned} H(\omega) &= \int_{-\infty}^{\infty} h(x)(\cos(\omega x) - j\sin(\omega x))dx \\ &= \underbrace{\int_{-\infty}^{\infty} h(x)\cos(\omega x)dx}_{\mathrm{Re}[H(\omega)]} + j\underbrace{\int_{-\infty}^{\infty} h(x)(-\sin(\omega x))dx}_{\mathrm{Im}[H(\omega)]} \end{aligned}$$

Clearly, $H(\omega) \in \mathbb{C}$. Regarding the integral as a dot product:

- $\mathrm{Re}(H(\omega))$ is the amount of $h(x)$ along $\cos(\omega x)$
- $\mathrm{Im}(H(\omega))$ is the amount of $h(x)$ along $-\sin(\omega x)$

Fourier transform a 1D signal (cont.)

On the other hand:

$$\begin{aligned}\mathsf{Re}[H(\omega)]\cos(\omega x) + \mathsf{Im}[H(\omega)]\sin(\omega x) &= \sqrt{\mathsf{Re}^2[H(\omega)] + \mathsf{Im}^2[H(\omega)]}\sin(\omega x + \angle H(\omega)) \\ &= |H(\omega)|\sin(\omega x + \angle H(\omega))\end{aligned}$$

Thus:

- $|H(\omega)|$ is the *magnitude* of the component of $h(x)$ of angular frequency ω
- $\angle H(\omega)$ is the *phase* of the component of $h(x)$ of angular frequency ω

The discrete domain counterpart of (10) is called *Discrete Fourier Transform* (DFT) and takes the form:

$$H(k) = \mathcal{F}\{h(x)\} = \frac{1}{N}\sum_{x=0}^{N-1} h(x)e^{-j\frac{2\pi kx}{N}} \tag{11}$$

where N is the length of the signal and $k \in [-\frac{N}{2}, \frac{N}{2}]$. Values of k outside that range are not meaningful because larger values of k alias with lower frequencies.

From a computational point of view:

- DFT require $O(N^2)$ operations (multiply-adds)
- A faster algorithm, called the *Fast Fourier Transform* (FFT) exists that requires $O(N \log_2 N)$ operations

Inverse Fourier transform

The original signal $h(x)$ can be recovered from its transform $H(\omega)$ by applying the inverse Fourier transform.

In the continuous domain:

$$h(x) = \mathcal{F}^{-1}\{H(\omega)\} = \frac{1}{2\pi}\int_{-\infty}^{\infty} H(\omega)e^{j\omega x}dx \tag{12}$$

In the discrete domain:

$$h(x) = \mathcal{F}^{-1}\{H(k)\} = \sum_{N} H(k)e^{j\frac{2\pi kx}{N}} \tag{13}$$

where $\sum_{N}$ means that the sum may be performed over any interval of length N.

Properties

property	signal		transform
superposition	$f_1(x) + f_2(x)$		$F_1(\omega) + F_2(\omega)$
shift	$f(x - x_0)$		$F(\omega)e^{-j\omega x_0}$
reversal	$f(-x)$		$F^*(\omega)$
convolution	$f(x) * h(x)$		$F(\omega)H(\omega)$
correlation	$f(x) \otimes h(x)$		$F(\omega)H^*(\omega)$
multiplication	$f(x)h(x)$		$F(\omega) * H(\omega)$
differentiation	$f'(x)$		$j\omega F(\omega)$
domain scaling	$f(ax)$		$1/aF(\omega/a)$
real images	$f(x) = f^*(x)$	$\Leftrightarrow$	$F(\omega) = F(-\omega)$
Parseval's Theorem	$\sum_x f^2(x)$	$=$	$\sum_\omega F^2(\omega)$

Fourier transform pairs

Name	Signal		Transform
impulse	$\delta(x)$	$\Leftrightarrow$	1
shifted impulse	$\delta(x-u)$	$\Leftrightarrow$	$e^{-j\omega u}$
box filter	$\text{box}(x/a)$	$\Leftrightarrow$	$a\text{sinc}(a\omega)$
tent	$\text{tent}(x/a)$	$\Leftrightarrow$	$a\text{sinc}^2(a\omega)$
Gaussian	$G(x;\sigma)$	$\Leftrightarrow$	$\frac{\sqrt{2\pi}}{\sigma}G(\omega;\sigma^{-1})$
Laplacian of Gaussian	$(\frac{x^2}{\sigma^4}-\frac{1}{\sigma^2})G(x;\sigma)$	$\Leftrightarrow$	$-\frac{\sqrt{2\pi}}{\sigma}\omega^2 G(\omega;\sigma^{-1})$

$$\text{sinc}\,\omega = \begin{cases} 1 & \omega = 0 \\ \dfrac{\sin\omega}{\omega} & \text{otherwise} \end{cases}$$

Fourier transform a 2D signal

We define the Fourier transform of a signal $h(x, y)$ as

$$H(\omega_x, \omega_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(x, y) e^{-j(\omega_x x + \omega_y y)} dx dy \tag{14}$$

where j is the imaginary unit and ω_x and ω_y are horizontal and vertical angular frequencies, respectively.

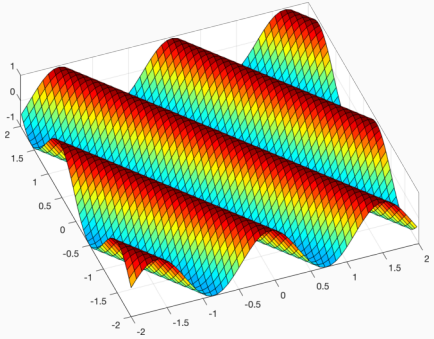
Regarding the integral as a dot product, the value $H(\omega_x, \omega_y)$ represents the amount of $h(x, y)$ along the function

$$e^{-j(\omega_x x + \omega_y y)} = \cos(\omega_x x + \omega_y y) - j \sin(\omega_x x + \omega_y y).$$

Fourier transform a 2D signal (cont.)

For fixed ω_x and ω_y, the function $\omega_x x + \omega_y y$ is constant along the lines orthogonal to the vector (ω_x, ω_y) and so are the terms $\cos(\omega_x x + \omega_y y)$ and $\sin(\omega_x x + \omega_y y)$.

In analogy to the 1D case, the "components" are now *oriented sinusoids* of angular frequency $\omega = \sqrt{\omega_x^2 + \omega_y^2}$.



Plot of the function $\cos(4x + 3y)$. The angular frequency is $\sqrt{4^2 + 3^2} = 5$ radians per unit of length.

The discrete domain counterpart of (14) takes the form:

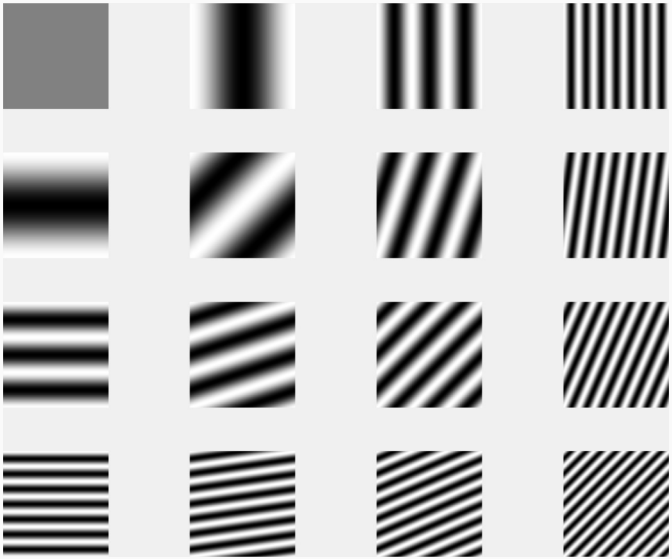
$$H(k_x, k_y) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} h(x, y) e^{-j2\pi \frac{k_x x + k_y y}{MN}} \tag{15}$$

where M and N are the width and height of the image.

`Change of basis`

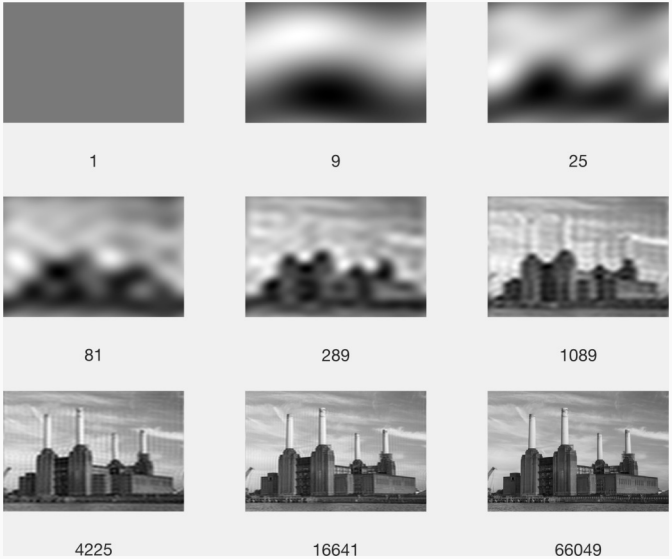
The discrete Fourier transform may be seen as a *change of basis*: instead of representing an image as a superimposition of NM properly scaled and shifted impulses (one for each pixel), we represent the same image as a weighted sum of NM oriented sinusoids.
Each components is associated to a magnitude $|H(k_x, k_y)|$ and a phase $\angle H(k_x, k_y)$.

Examples of basis functions



Some elements of a Fourier basis. The constant term, corresponding to $\omega_x = \omega_y = 0$ is on the top left corner.

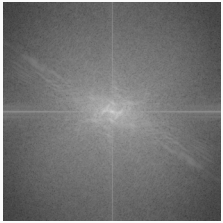
Example of reconstruction



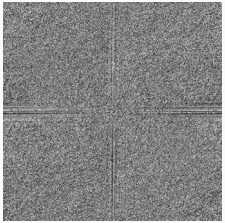
Example of truncated reconstruction using a given number of low-frequency components. The original image is 400×266.

Magnitude and phase

Zebra

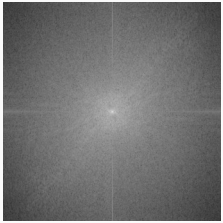


Magnitude spectrum

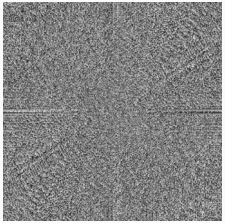


Phase spectrum

Cheetah

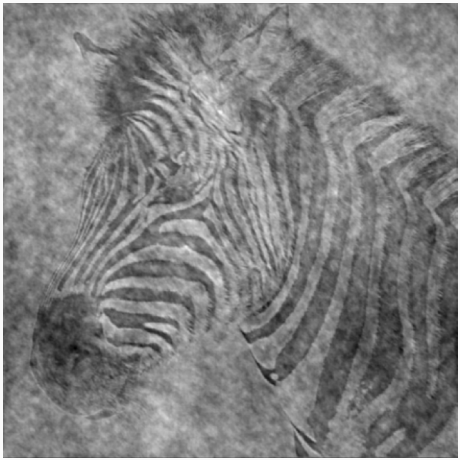


Magnitude spectrum

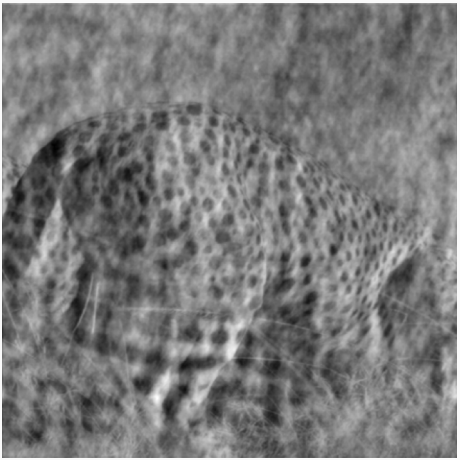


Phase spectrum

Swapping the magnitude spectra



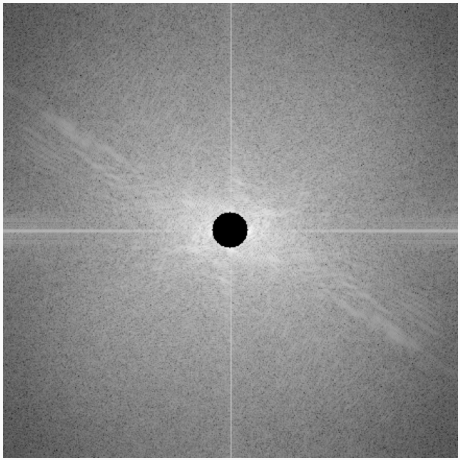
Magnitude: cheetah; phase: zebra.



Magnitude: zebra; phase: cheetah.

Phase spectrum appears to be more important for perception than the magnitude spectrum.

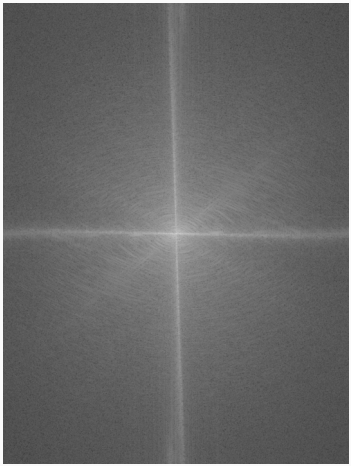
Frequency editing: box filter



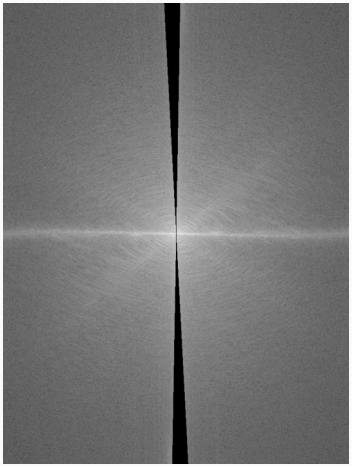
Setting to zero the magnitude of the low frequency components, results in a high pass filtering. In particular, the constant term (also called the “DC component”) is removed thus the mean value of the resulting image is zero.

Frequency editing: oriented filter

Original image (Porticato di San Luca, Bologna)



Magnitude spectrum



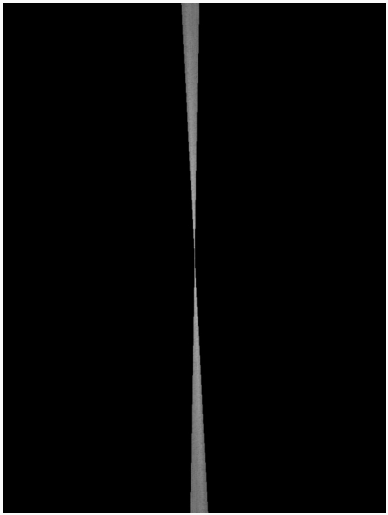
Modified magnitude spectrum

Frequency editing: oriented filter (cont.)

Original image

Horizontally filtered

Frequency editing: oriented filter (cont.)

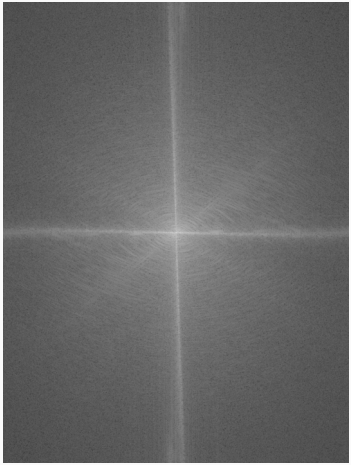


Removed magnitude spectrum

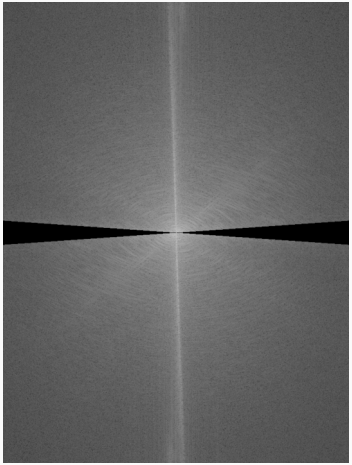
Inverse transform of the removed part

Frequency editing: oriented filter (cont.)

Original image



Magnitude spectrum



Modified magnitude spectrum

Frequency editing: oriented filter (cont.)

Original image

Vertically filtered

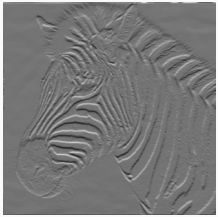
The convolution theorem

- The convolution theorem states that
$$\mathcal{F}[g * h] = \mathcal{F}[g]\mathcal{F}[h]$$
i.e. the Fourier transform of the convolution of two functions is the product of the Fourier transforms.
- Thus, *convolution in spatial domain is equivalent to multiplication in frequency domain*:
$$g * h = \mathcal{F}^{-1}\left[\mathcal{F}[g]\mathcal{F}[h]\right].$$
- Filtering in the frequency domain may result in faster computation (it depends on the matrices' size).

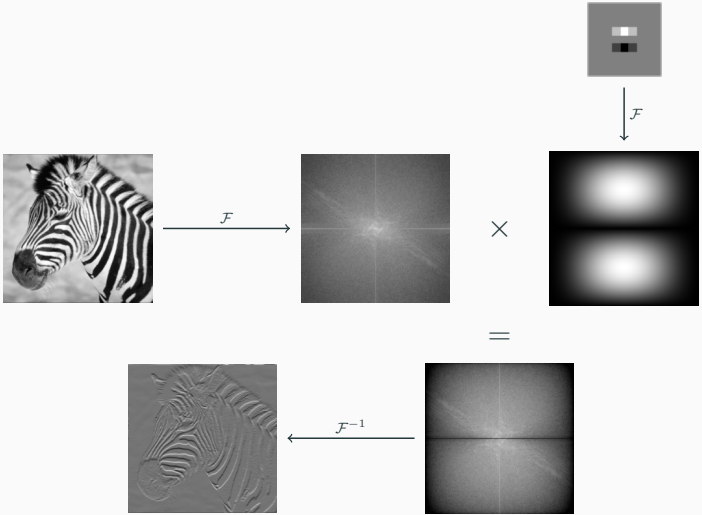
Filtering in the spatial domain

$*$

$=$



Filtering in the frequency domain



IMPORTANT: the phase is multiplied too!

Theorem (Duality of Fourier transform)

$$\mathcal{F}\{h(x)\} = H(\omega) \Longrightarrow \mathcal{F}\{H(x)\} = 2\pi h(-\omega)$$

Proof.
Since h and H are a Fourier pair, we get

$$h(x) = \mathcal{F}^{-1}\{H(\omega)\} = \frac{1}{2\pi}\int_{-\infty}^{\infty} H(\omega)e^{j\omega x}d\omega$$

Let $x' = -x$:

$$h(-x') = \frac{1}{2\pi}\int_{-\infty}^{\infty} H(\omega)e^{-j\omega x'}\,d\omega.$$

Now, interchange x' and ω:

$$2\pi h(-\omega) = \int_{-\infty}^{\infty} H(x')e^{-j\omega x'}\,dx' = \mathcal{F}\{H(x)\}$$

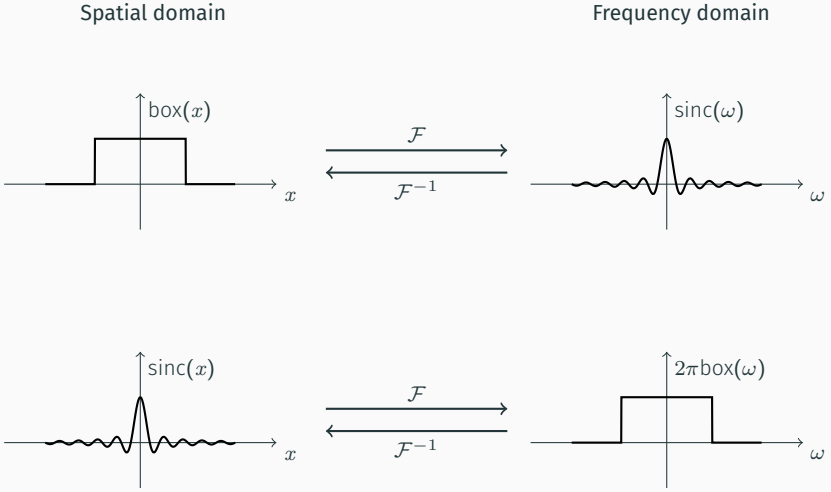
□

In particular, for even functions, i.e. $h(x) = h(-x)$, we get

$$\mathcal{F}\{h(x)\} = H(\omega) \Longrightarrow \mathcal{F}\{H(x)\} = 2\pi h(\omega).$$

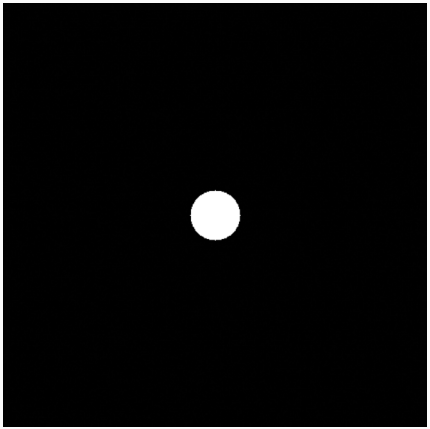
Box-sinc duality

The duality theorem implies the *box-sinc duality*.

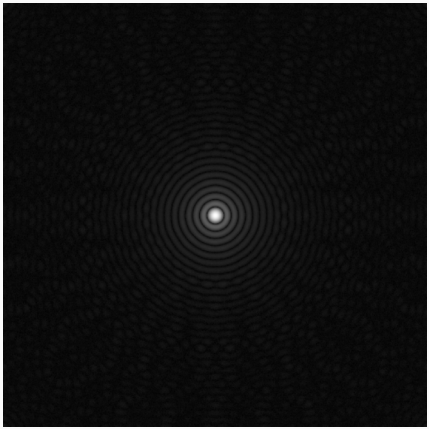


Sharp spatial filters

Sharp *spatial* filters $\Longrightarrow$ *frequency* ripples



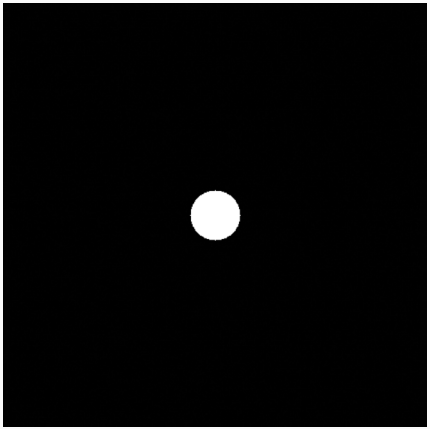
2D spatial box filter



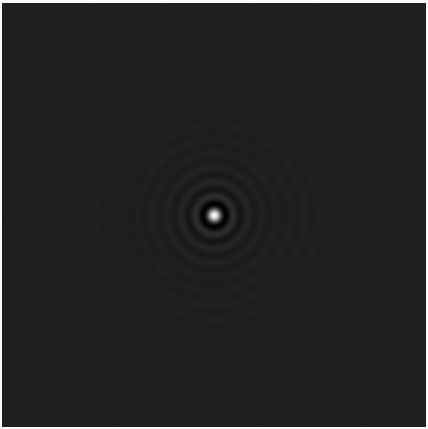
Fourier transform (magnitude)

Sharp frequency filters

Sharp *frequency* filters $\Longrightarrow$ *spatial* ripples



2D frequency box filter (magnitude)



Inverse Fourier transform

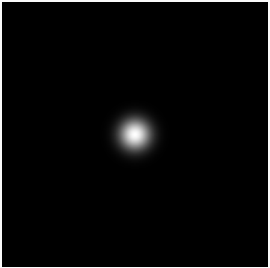
Gaussian duality

Fourier transform of a Gaussian is another Gaussian:

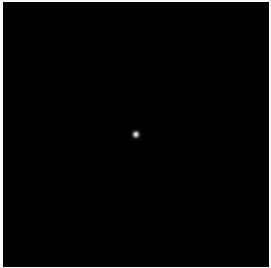
$$\mathcal{F}\{G(x;\sigma)\} = \frac{\sqrt{2\pi}}{\sigma} G(\omega;\sigma^{-1})$$

Advantages:

- Smooth degradation in frequency components
- No sharp cut-off
- No negative values
- Never zero (infinite extent)

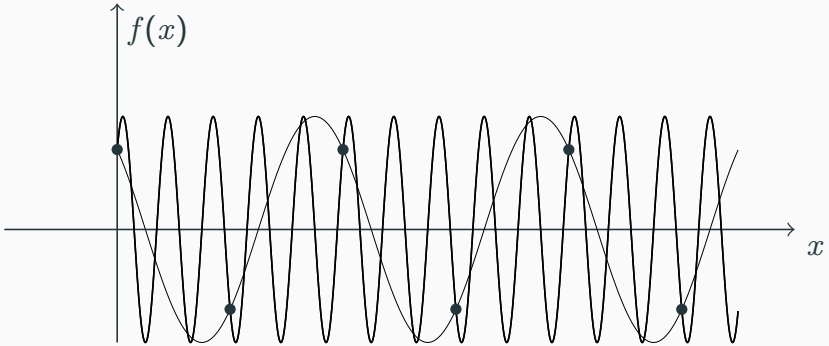


2D Gaussian filter (spatial)



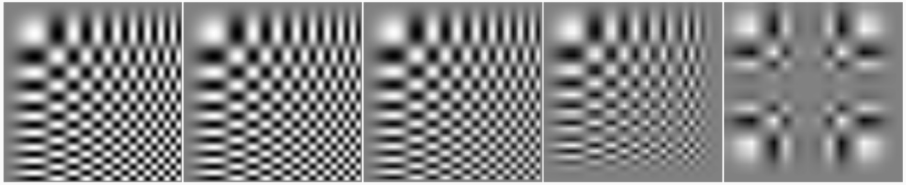
Fourier transform (magnitude)

Aliasing

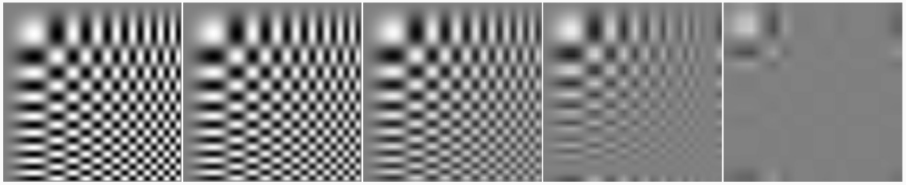


- Sampling a signal at too low sampling rate causes *aliasing*
- The result is indistinguishable from sampling of lower frequency
- Nyquist sampling theorem: Need to sample at least 2 times the maximum frequency
- What to do when using a high sampling rate is impossible? Low-pass filter the signal

Aliasing (cont.)



From left to right: original 256×256 image and resampled versions by a factor of two (128×128, 64×64, 32×32, 16×16). Artifacts arise due to aliasing.



From left to right: original 256×256 image and resampled versions by a factor of two (128×128, 64×64, 32×32, 16×16). Before resampling, each image is smoothed with a Gaussian of σ one pixel.

Multi-resolution representations

Upsampling an downsampling

Sometimes it is convenient to *change the size* of an image.

`Upsampling` (scaling up the image) may be useful for instance for:

- matching the resolution of a device (such as screen, or printer);
- matching the size required by an off-the-shelf algorithm (such as a pre-trained Convolutional Neural Network).

`Downsampling` (scaling down the image) may be useful for instance for:

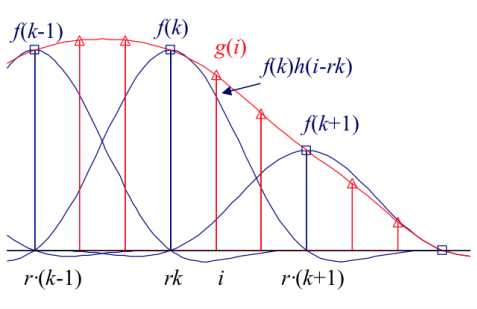
- applying *coarse-to-fine matching* (find a match in a small, heavily smoothed and resampled image, and refine the match in images of increasing size);
- searching patterns at different scales, using the same template;
- applying *fine-to-coarse feature tracking* (improving a set of features found a fine scales by rejecting the features that do not have parents at a coarser scale);
- matching the size required by an off-the-shelf algorithm (such as a pre-trained Convolutional Neural Network).

Upsampling

Upsampling requires *interpolation*, to produce an image of higher resolution:

$$g(i,j) = \sum_{k,l} f(k,l)h(i - rk, j - rl) \tag{16}$$

where f is the original image, h is the *interpolation kernel* and r is the upsampling rate.



Upsampling in 1D

Upsampling (cont.)

Original

Upsampled (impulse kernel)

Upsampled (bilinear kernel)

$$\begin{bmatrix} 0.25 & 0.5 & 0.25 \\ 0.5 & 1 & 0.5 \\ 0.25 & 0.5 & 0.25 \end{bmatrix}$$

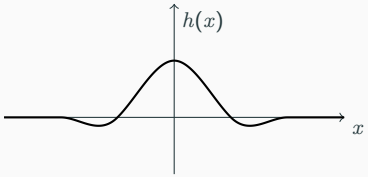
Upsampling (cont.)

Common interpolation kernels:

- bilinear kernel;
- *bicubic* kernel: the outer product of two piecewise-cubic *splines* of the form

$$h(x) = \begin{cases} 1 - (a+3)x^2 + (a+2)|x|^3 & \text{if } |x| < 1 \\ a(|x|-1)(|x|-2)^2 & \text{if } 1 \leq |x| < 2 \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

where a specifies the derivative at $x = 1$ and is often set to -1, as this best matches the shape of a sinc function.



Downsampling

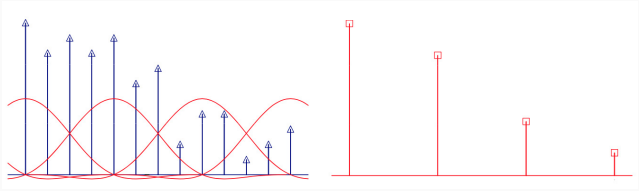
Downsampling is, conceptually, a two step procedure:

1. convolve the image with a low-pass filter (to avoid aliasing)
2. keep every rth sample

In practice the convolution can be evaluated every rth sample:

$$g(i,j) = \sum_{k,l} f(k,l)h(ri-k, rj-l). \tag{18}$$

The *smoothing kernel* h is usually a stretched and re-scaled version of an interpolation kernel.



Downsampling in 1D

Downsampling (cont.)

Original

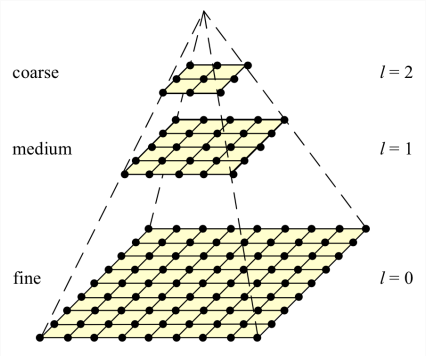
Low-pass filtered (Gaussian kernel)

Downsampled

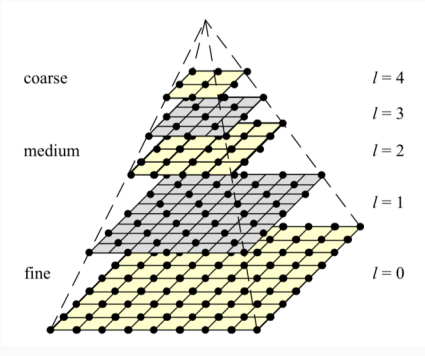
Multi-resolution representations

Image pyramids are multi-resolution representations.

A common pyramid is the *octave pyramid*, obtained by subsequent downsampling operations at constant rate $r = 2$. Another common pyramid is the *half-octave pyramid*, in which $r = \sqrt{2}$.



Octave pyramid



Half-octave pyramid

Gaussian pyramid

When the smoothing filter applied at each downsampling is Gaussian, the pyramid is called a *Gaussian pyramid*.

Six levels of a Gaussian pyramid, represented at the same size (pixels become bigger from left to right).

Six levels of a Gaussian pyramid, represented at the actual size.

References

References

Durand, F. and Dorsey, J. (2002). Fast bilateral filtering for the display of high-dynamic-range images. In *ACM transactions on graphics (TOG)*, volume 21, pages 257–266. ACM.

Freeman, W. T., Adelson, E. H., et al. (1991). The design and use of steerable filters. *IEEE Transactions on Pattern analysis and machine intelligence*, 13(9):891–906.

Hu, M.-K. (1962). Visual Pattern Recognition by Moment Invariants. *Information Theory, IRE Transactions on*, 8(2):179–187.

Perona, P. and Malik, J. (1990). Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on pattern analysis and machine intelligence*, 12(7):629–639.

Szeliski, R. (2010). *Computer vision: algorithms and applications*. Springer Science & Business Media.

Tomasi, C. and Manduchi, R. (1998). Bilateral filtering for gray and color images. In *Computer Vision, 1998. Sixth International Conference on*, pages 839–846. IEEE.

554SM –Fall 2018

Lecture 3
Image Processing

END