Máster Universitario en
Nuevas Tecnologías en Informática

Asignatura “Visión Artificial”

Procesamiento de imagen
(Técnicas más relevantes en aplicaciones de visión)

Facultad de Informática
Universidad de Murcia
Curso 2018/19

Imágenes como funciones

- Una imagen es una función de R^2 a R^M:

$$f: [a,b] \times [c,d] \rightarrow [0,255]$$

Dominio: *X* x *Y* Rango: *luminancia*

  - Imágenes discretas → muestreadas en un *grid* regular.
  - Está definida sobre un **dominio** rectangular, y el **rango** es finito.
  - P.e., *f(x,y)* da la *luminancia* (*M*=1) en la posición *(x,y)*.
  - Para una imagen a color, *M*=3:

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

Filtrado de imágenes

- Objetivos del filtrado:
  - Formar una nueva imagen cuyos píxeles se obtienen a partir de una cierta combinación de los píxeles de la(s) imagen(es) de entrada.
- Útil para:
  - Extraer información útil para procesos de interpretación/*matching* posteriores:
    - ***Features*** (bordes, esquinas, *blobs*, etc.)
  - Modificar o resaltar ciertas propiedades:
    - Superresolución, realce (*enhancement*), eliminación de ruido (*denoising*)...

Procesamiento de imagen: ejemplos

Original

Incremento de contraste

Cambio en *Hue*

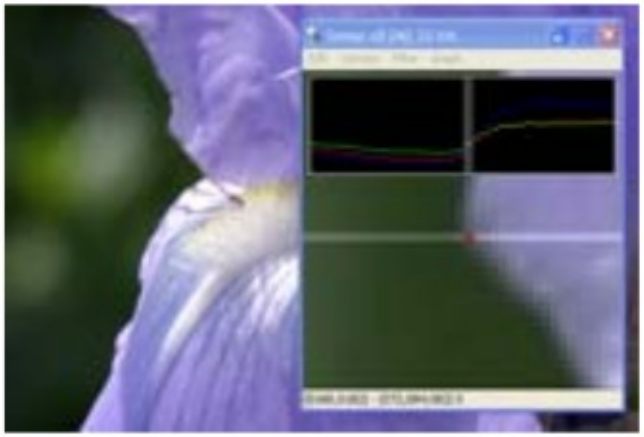
Color *quantization*

Suavizado *(Blurring)*

Rotación

Visualización de los datos de imagen

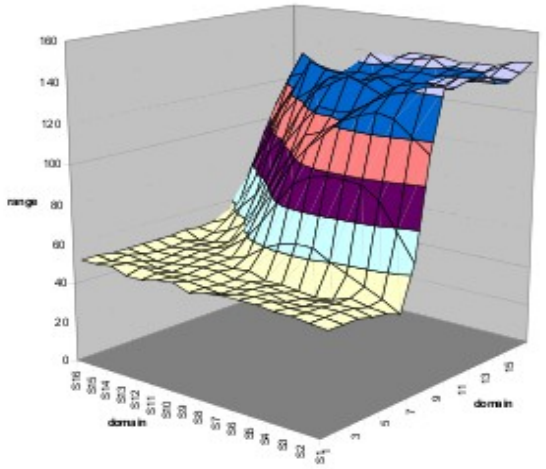
Imagen original



Scanline(s)

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

Array de números por plano
(aquí, sólo luminancia)



Superficie(s) 3D

Grupo 1: Transformaciones de píxel

Operadores “píxel a píxel”, toman una o más imágenes de entrada y generan una de salida:

$$g(\boldsymbol{x}) = h(f(\boldsymbol{x})) \text{ or } g(\boldsymbol{x}) = h(f_0(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x}))$$

- Dominio discreto X x Y:

$$g(i,j) = h(f(i,j)).$$

- Ejemplo lineal (*gain*+*bias*):

$$g(\boldsymbol{x}) = af(\boldsymbol{x}) + b.$$

- Ejemplo *blending* (mezcla de dos imágenes):

$$g(\boldsymbol{x}) = (1-\alpha)f_0(\boldsymbol{x}) + \alpha f_1(\boldsymbol{x})$$

Transformaciones de color

RGB

Red

Green

Blue

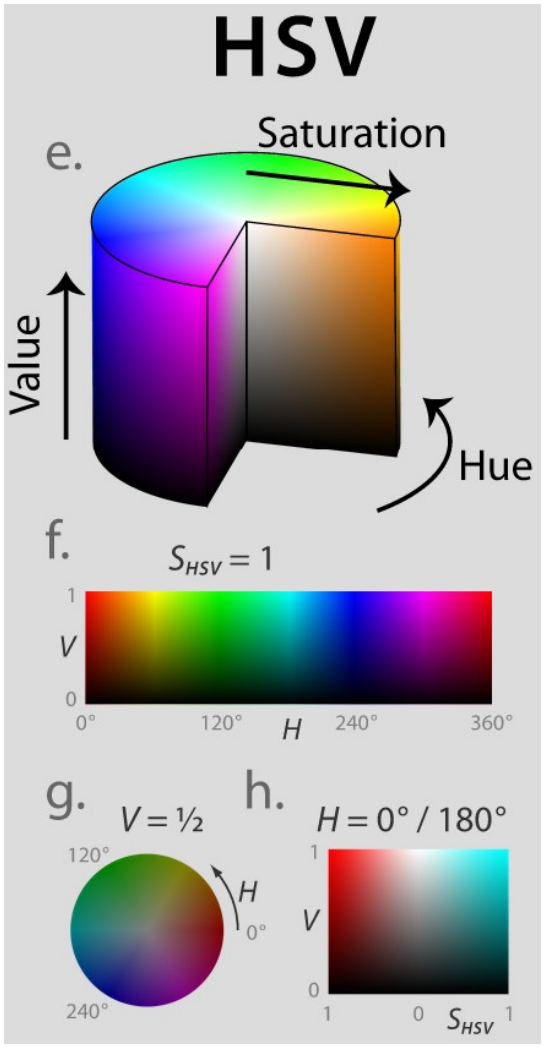
Luminancia

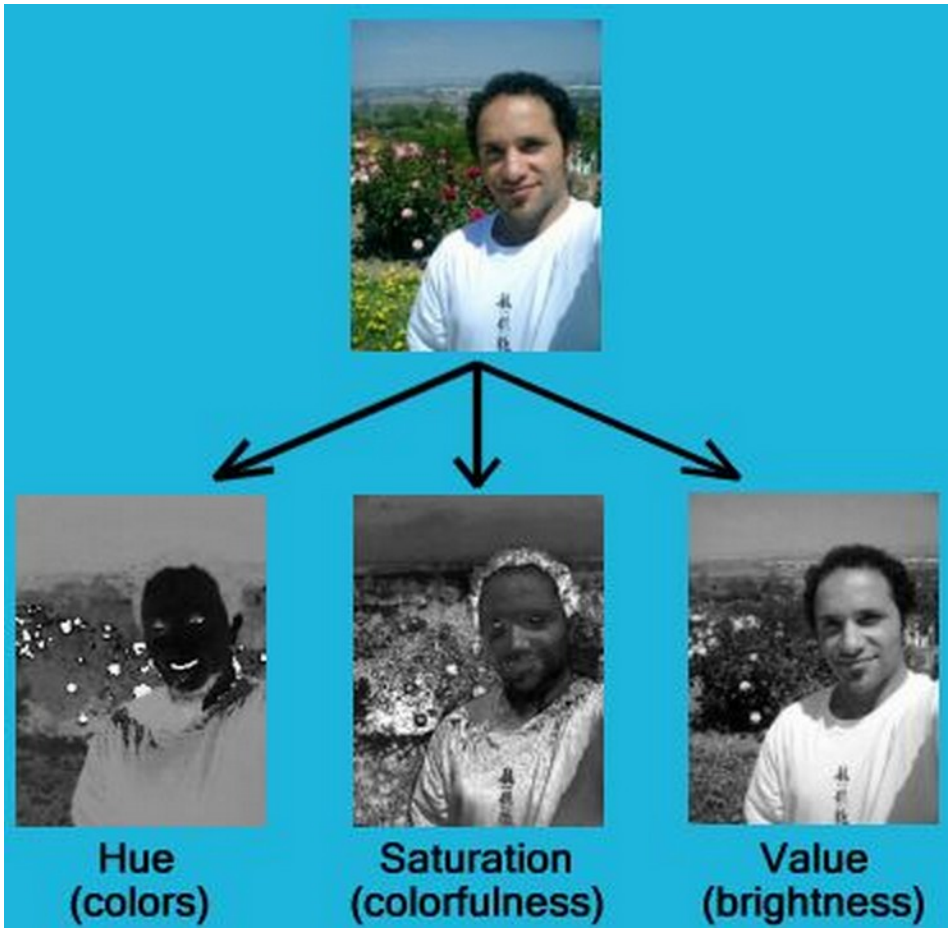
Crominancia 1

Crominancia 2

- RGB, YUV 420 (Y nivel de gris; planos U y V de crominancia, a mitad de resolución que plano Y)

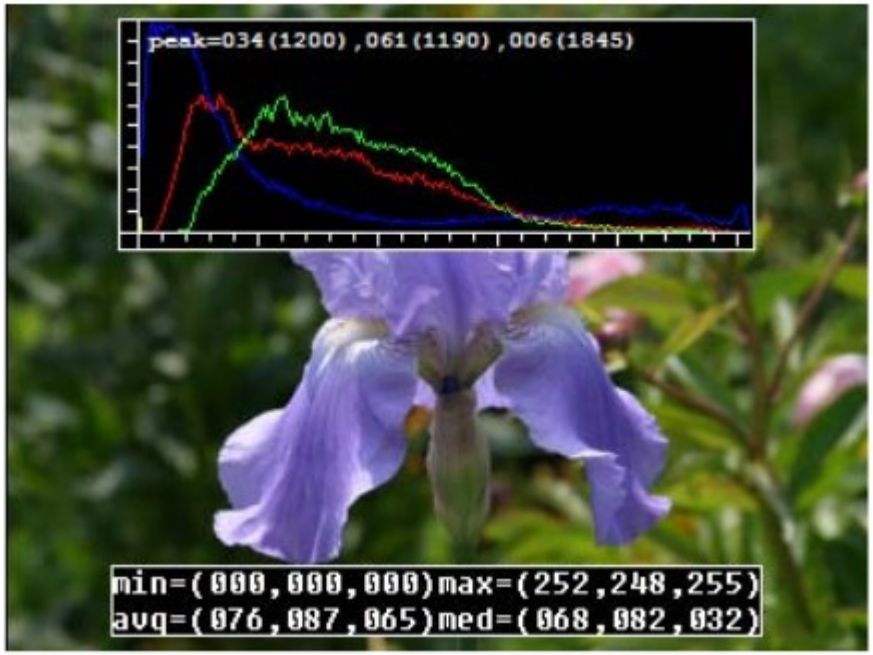
Transformaciones de color



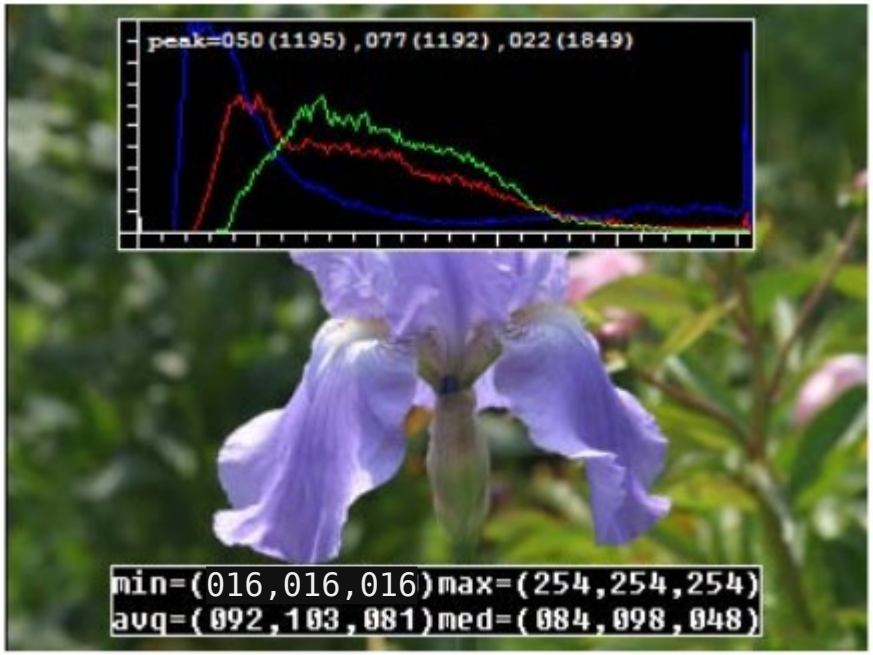


- HSV (Hue, Saturation, Value) → Separa “color puro”, grado de saturación e intensidad de luz

Incremento de brillo



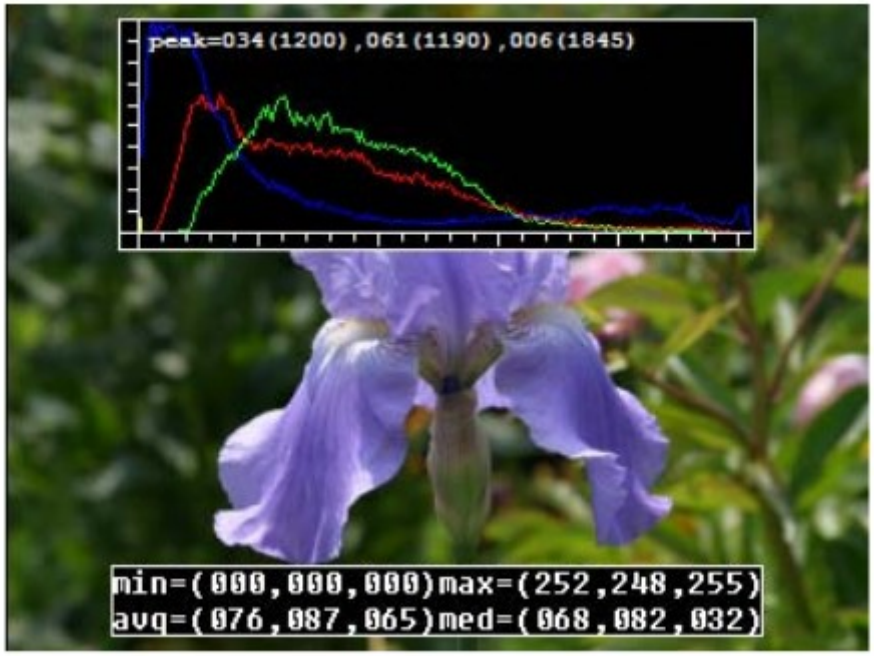
Original



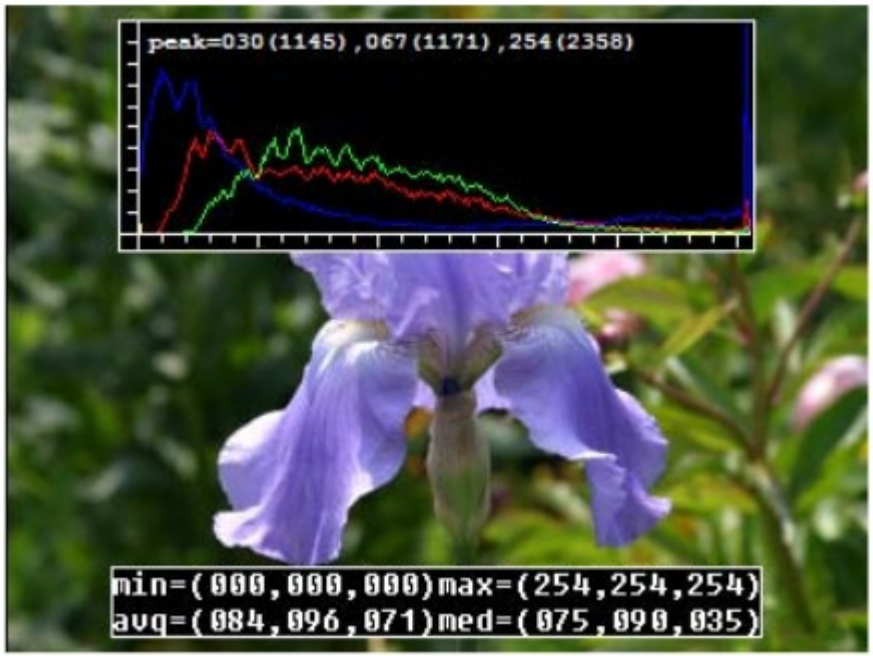
Brillo incrementado

- Observar el efecto del incremento de brillo en el histograma (*bias, +16*)

Incremento de contraste



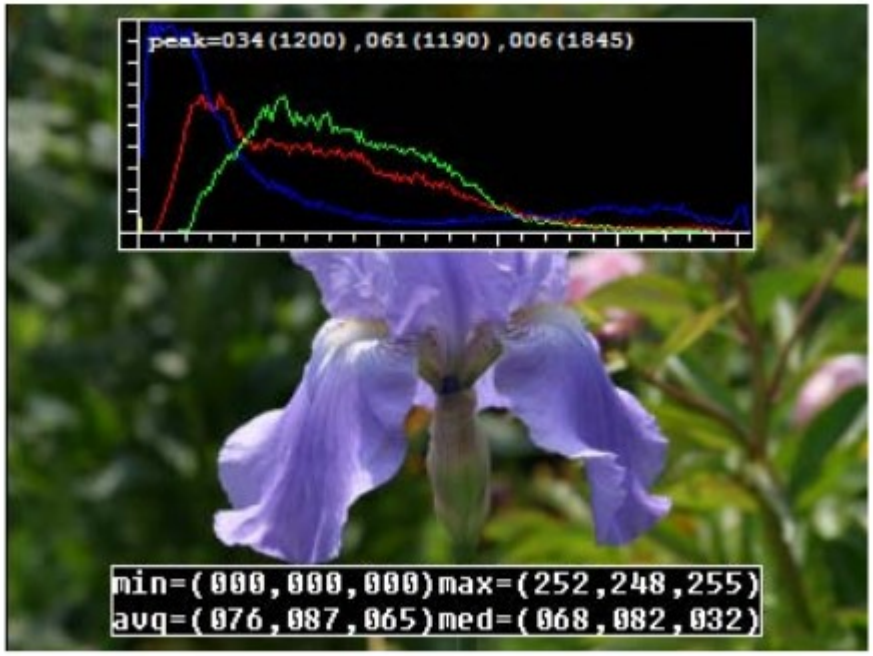
Original



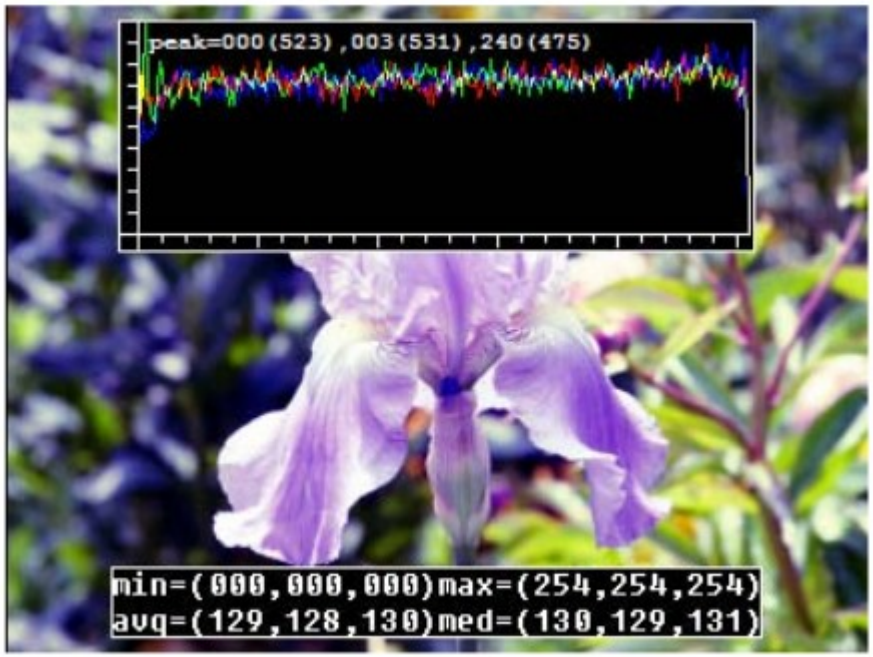
Contraste incrementado

- Observar el efecto del incremento de contraste en el histograma (*gain, x 1.1*)

Ecualización de histograma

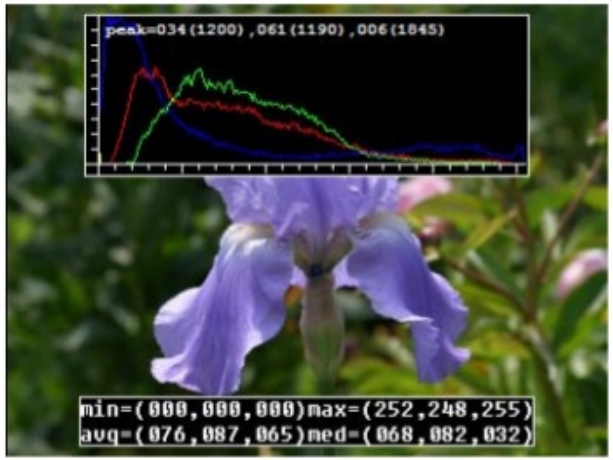


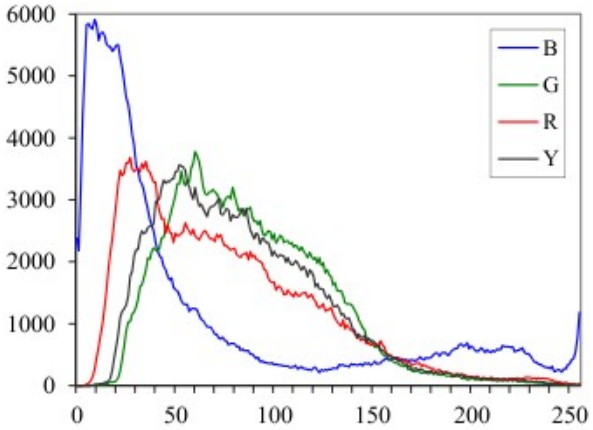
Original



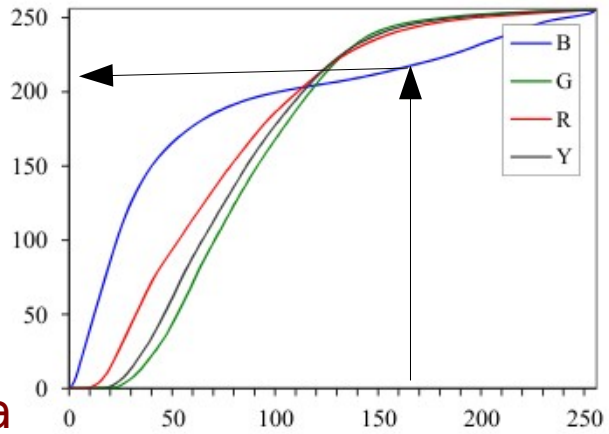
Equalización de histograma

Ecualización de histograma

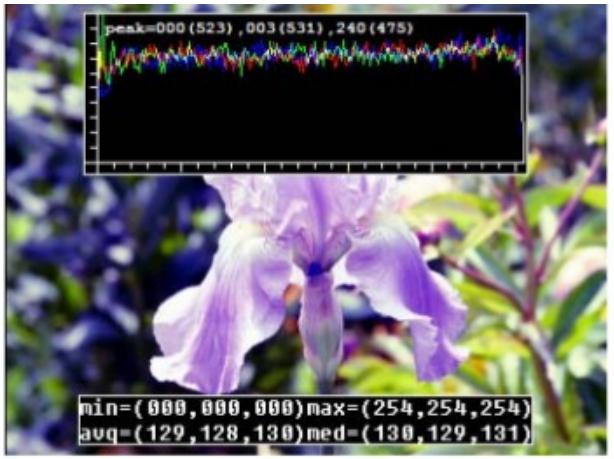




output



input



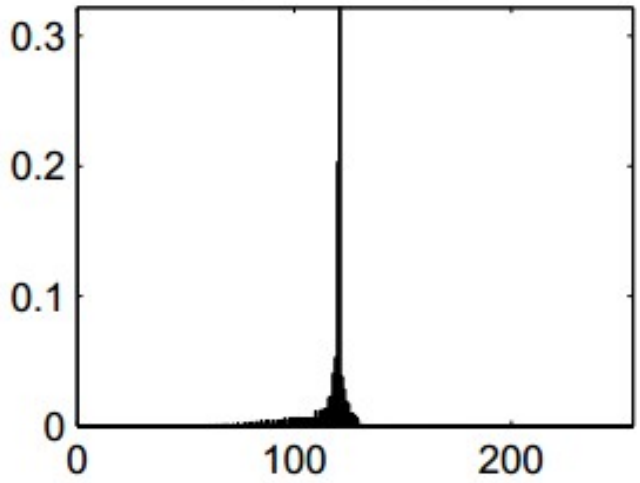
Usado como función de Transferencia (c(I)∈ [0,1]) x 255

Histograma acumulado $$c(I) = \frac{1}{N}\sum_{i=0}^{I} h(i) = c(I-1) + \frac{1}{N}h(I)$$

Ecualización de histograma

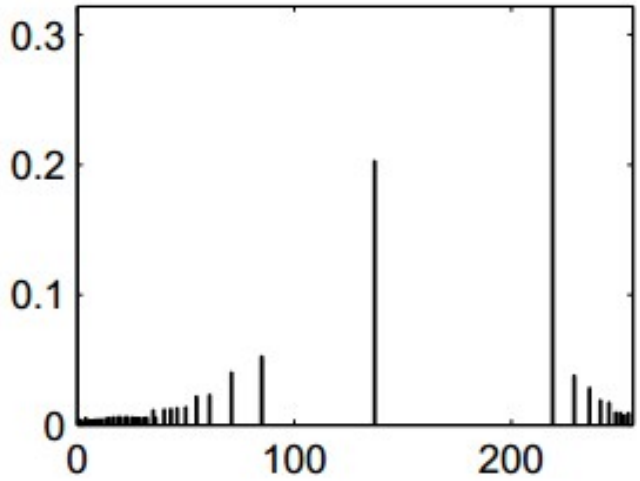
original image

original histogram



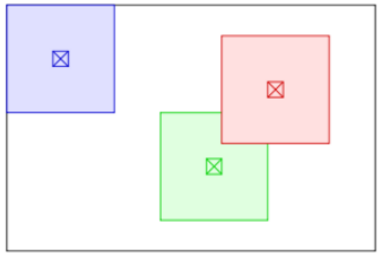
transformed image

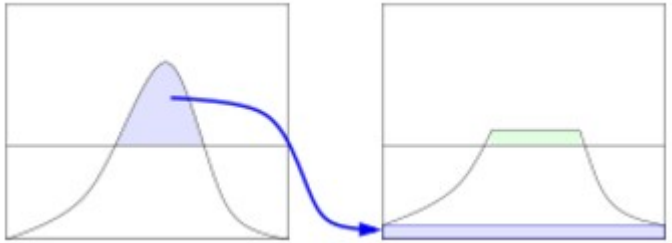
transformed histogram

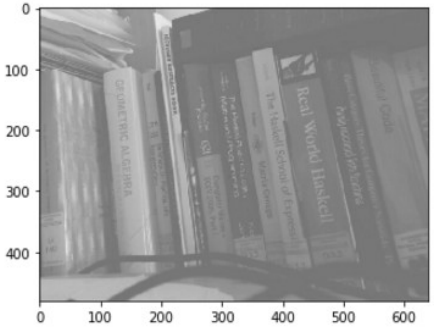


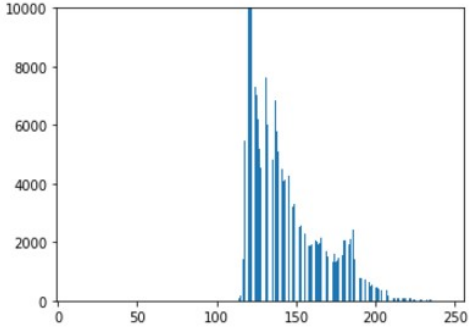
Ecualización adaptativa

Variante: (Contrast Limited) Adaptive Histogram Equalization (**CLAHE**) →
Histogramas locales, cortando picos y repartiendo uniformemente

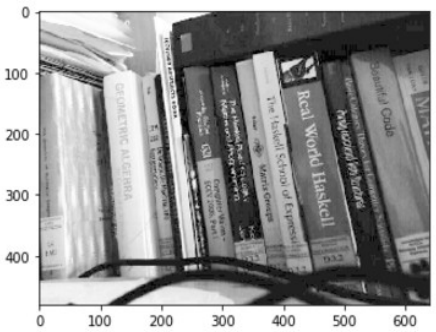


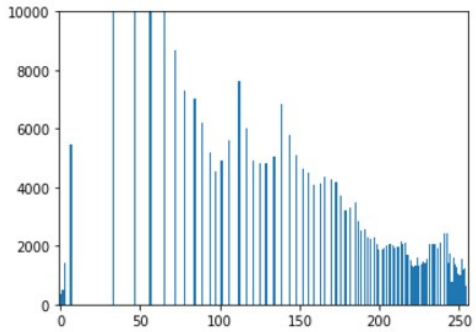




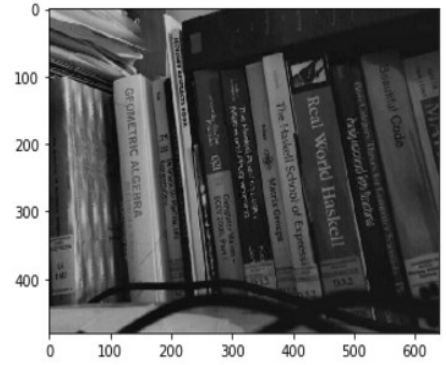


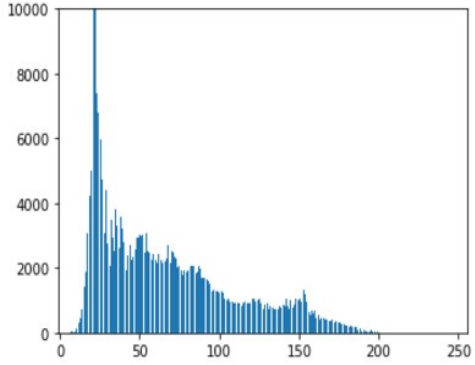
Original





Ecualización normal





Ecualización adaptativa

Umbralizado

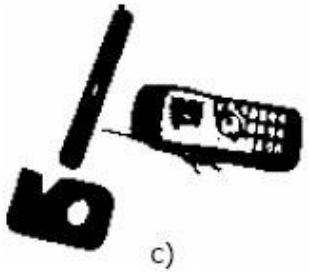
- Segmentación (habitualmente *binarización*) de imagen basada en umbralizado simple:

$$g[n,m] = \begin{cases} 1, & f[n,m] > 150 \\ 0, & \text{otherwise.} \end{cases}$$

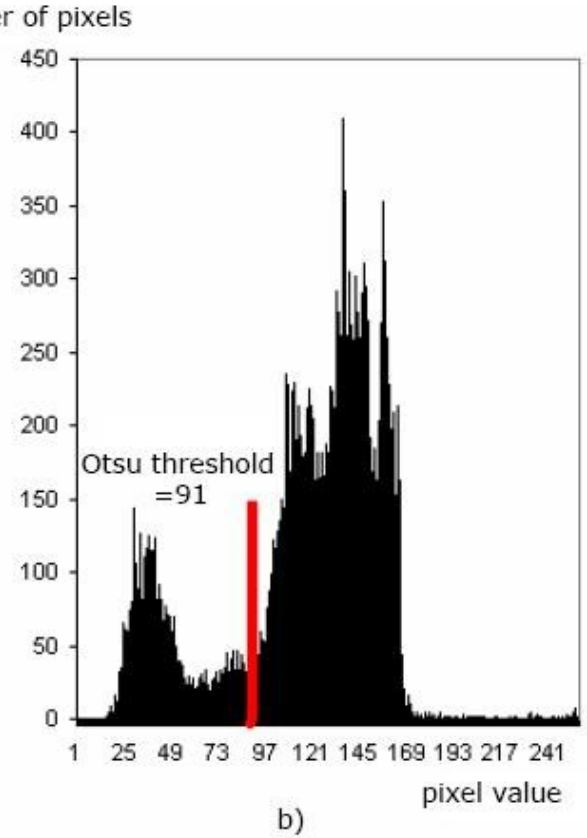
Umbralizado

- Umbralizado de Otsu (“invariante” a iluminación):
  - Búsqueda exhaustiva del umbral que minimiza la varianza intraclase (suma ponderada de varianzas de ambas clases):

a)



c)



b)

Varianza intraclase

t = umbral

$$\sigma_w^2(t) = \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_2^2(t)$$

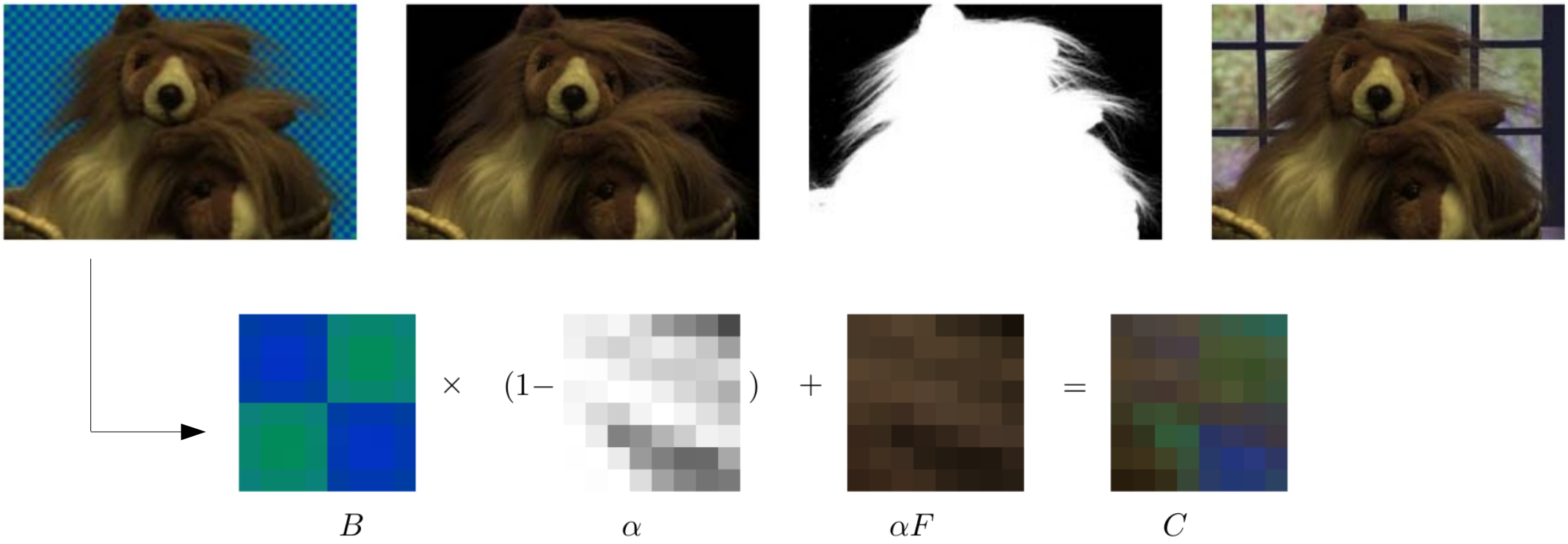
Varianza clase i=1,2

Prob. clase i=1,2

Matting y *compositing*

- Corte de objetos *foreground* para ponerlos en otro *background*:

Con fondo (B) 1 | *Matted image* (F) | Máscara | Con otro fondo (B) 2



Operación de *blending*: $C = (1 - \alpha)B + \alpha F$

Grupo 2: Operaciones en base a vecinos

- El valor de cada píxel de salida se obtiene a partir de su “vecindad” en la imagen de entrada:

Original

Suavizado (*blurred*)

Realzado

Suavizado con preservación de borde

Grupo 2: Operaciones en base a vecinos

- Más ejemplos:

Original (binarizada)

Dilatación

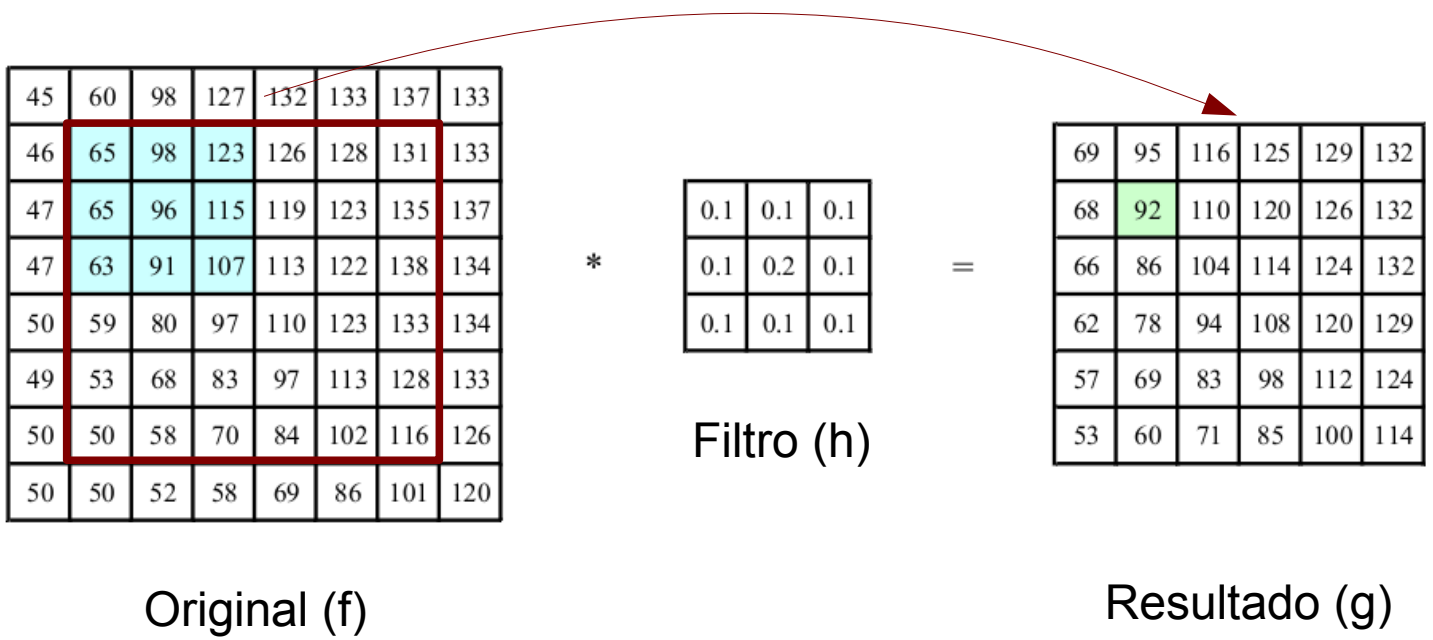
Transformada de distancia

Componentes conexas

Filtros lineales

- Cada píxel se obtiene como combinación lineal de sus vecinos (convolución):

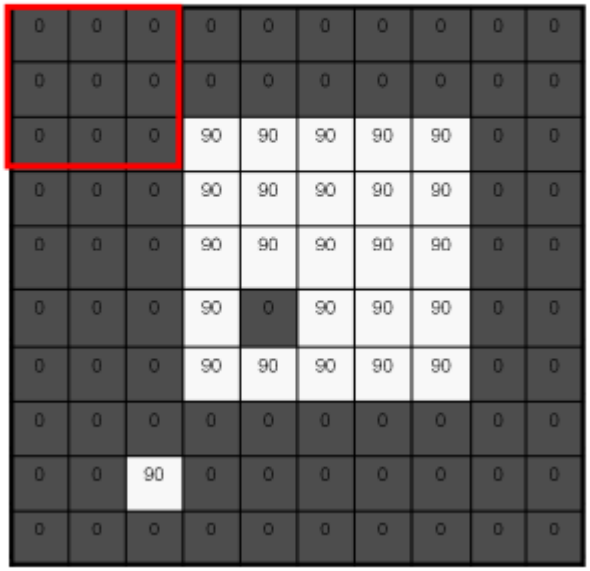
$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l) \longrightarrow g = f \otimes h$$



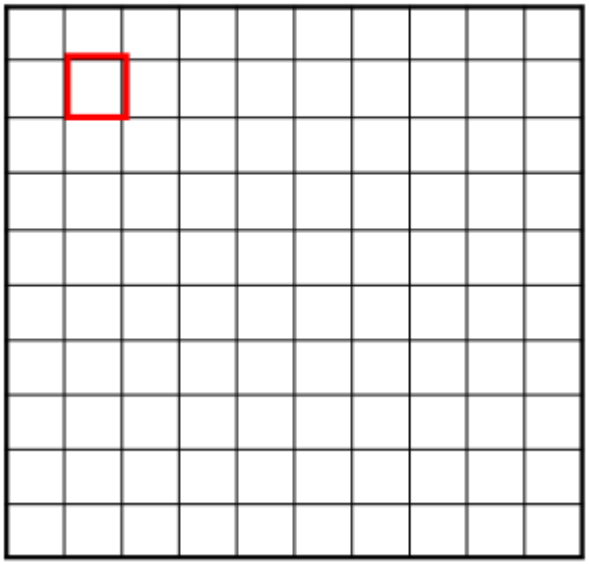
Filtros lineales

- Operación (ejemplo filtro de media; a menudo llamado *FilterBox*):

Original



Resultado

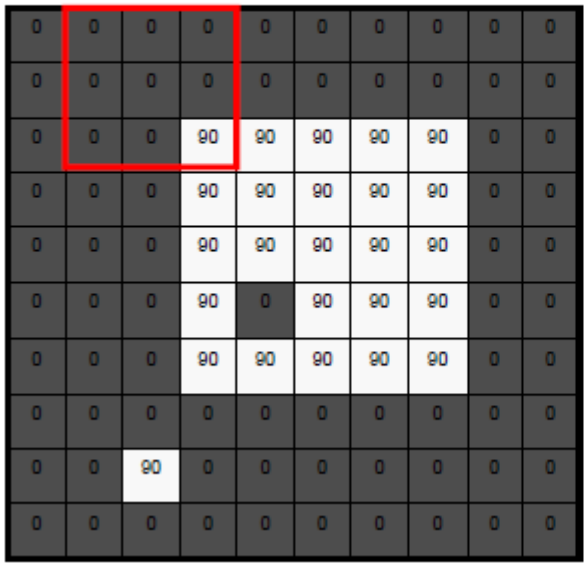


Máscara de convolución aquí: simplemente ventana 3x3 con valores = 1/9

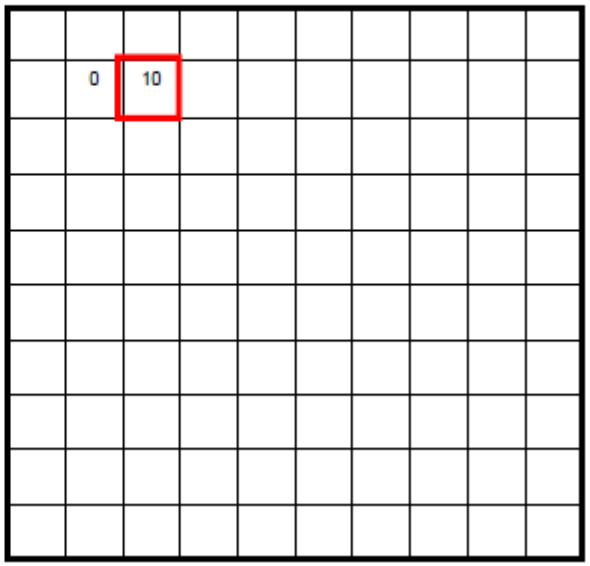
Filtros lineales

- Operación:

Original



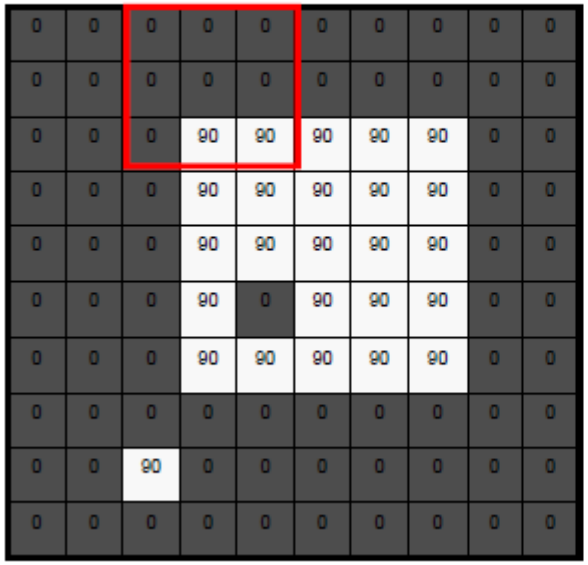
Resultado



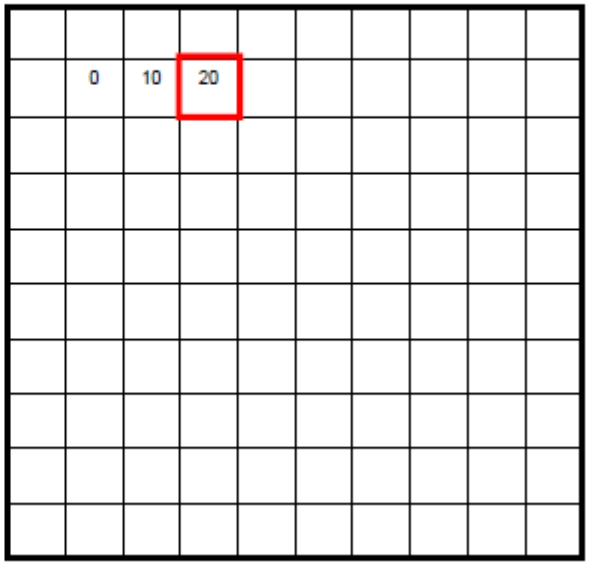
Filtros lineales

- Operación:

Original



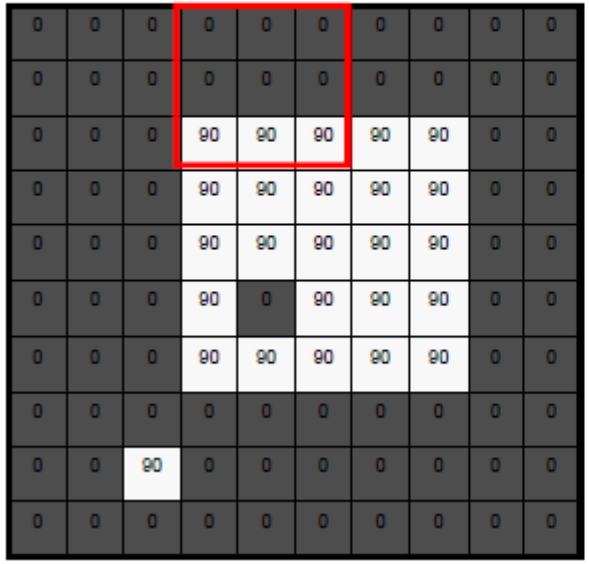
Resultado



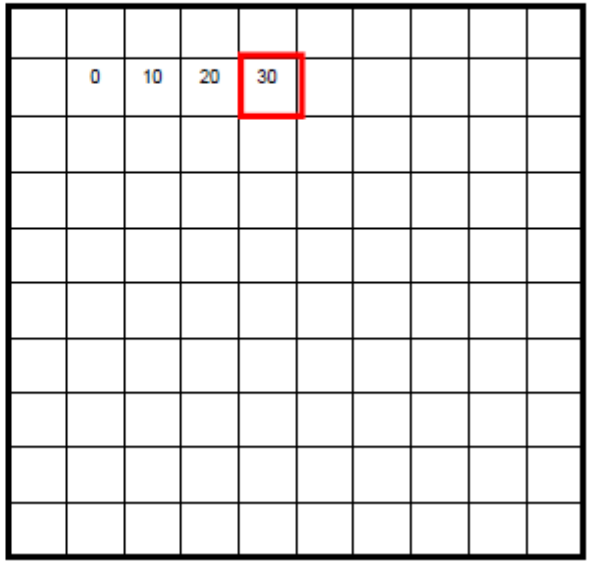
Filtros lineales

- Operación:

Original



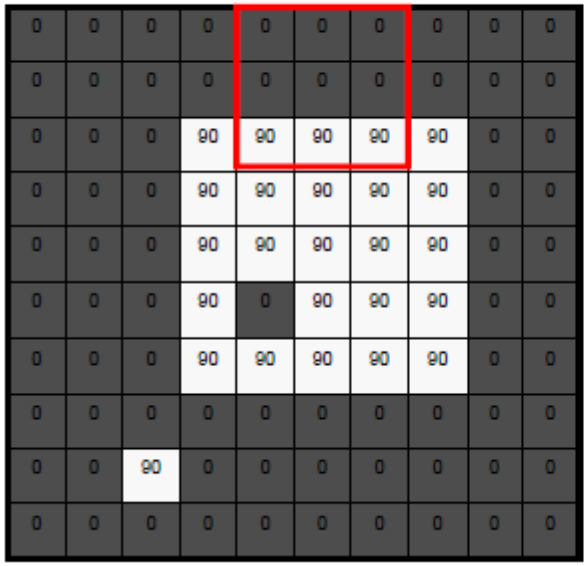
Resultado



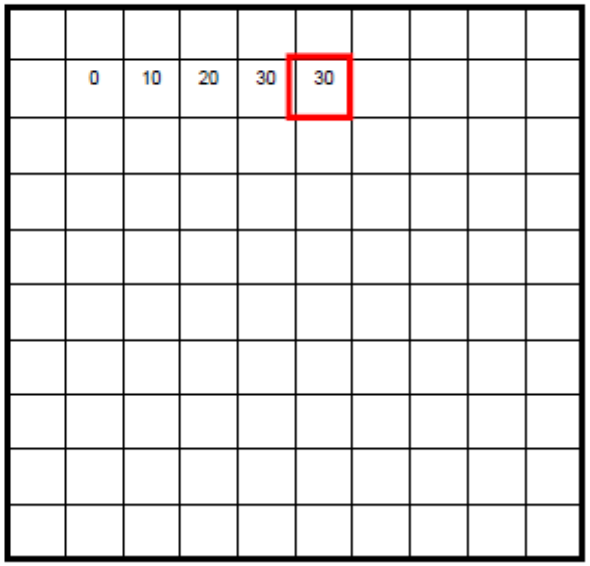
Filtros lineales

- Operación:

Original



Resultado



Filtros lineales

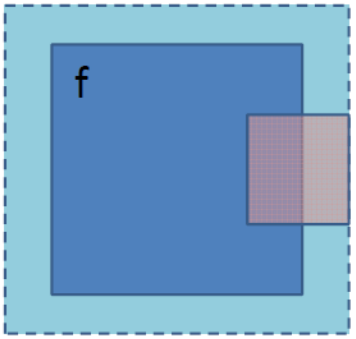
- Resultado final:

Original

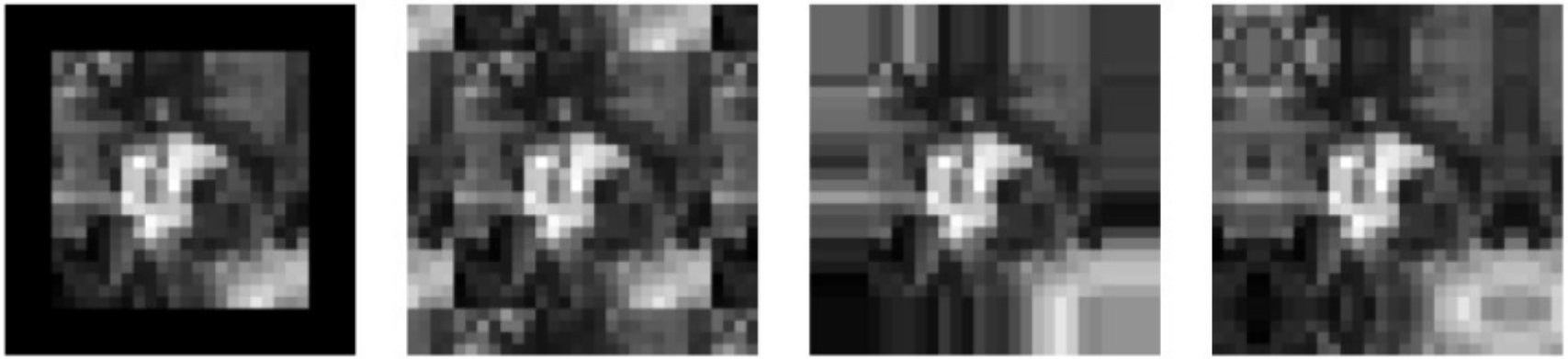
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Resultado

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

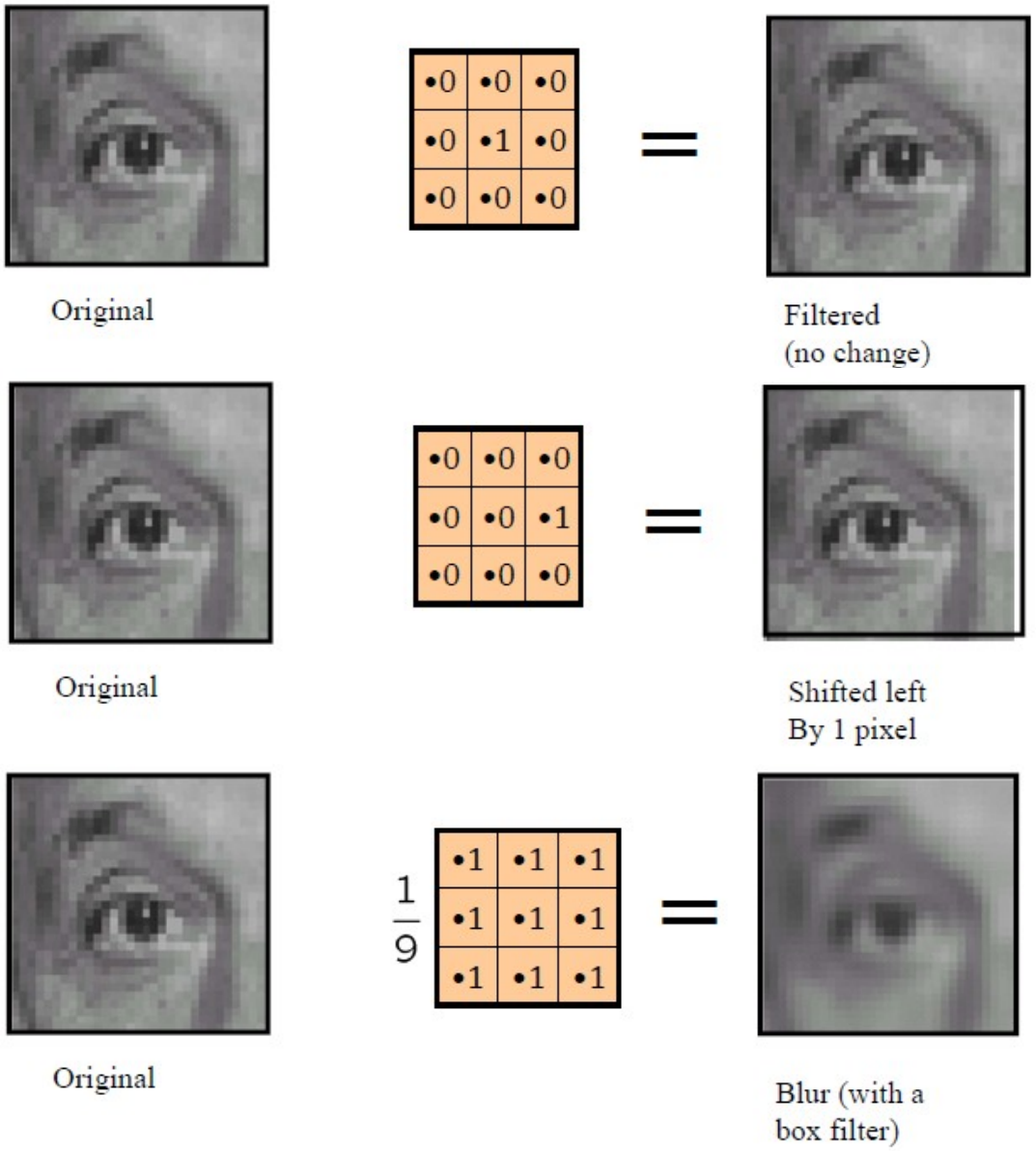


¡Ojo con los bordes!



zero wrap clamp mirror

Filtros lineales



Filtros lineales

- Ejemplo FilterBox
  - Reemplaza el valor de cada píxel con la media de su vecindad 3x3 (8-vecinos).
  - Efecto suavizado (elimina detalles de alta frecuencia).

$$g[n,m] = \frac{1}{9} \sum_{k=n-1}^{n+1} \sum_{l=m-1}^{m+1} f[k,l]$$

$$= \frac{1}{9} \sum_{k=-1}^{1} \sum_{l=-1}^{1} f[n-k, m-l]$$

h

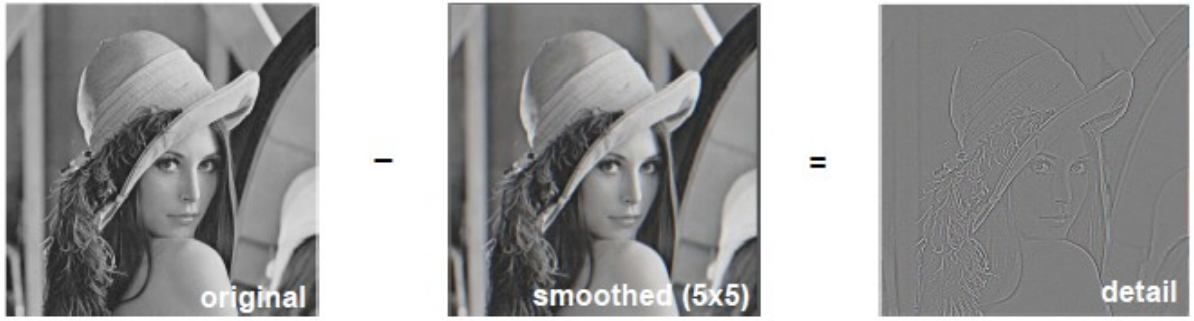
$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$$(f * h)[m,n] = \frac{1}{9} \sum_{k,l} f[k,l]\, h[m-k, n-l]$$

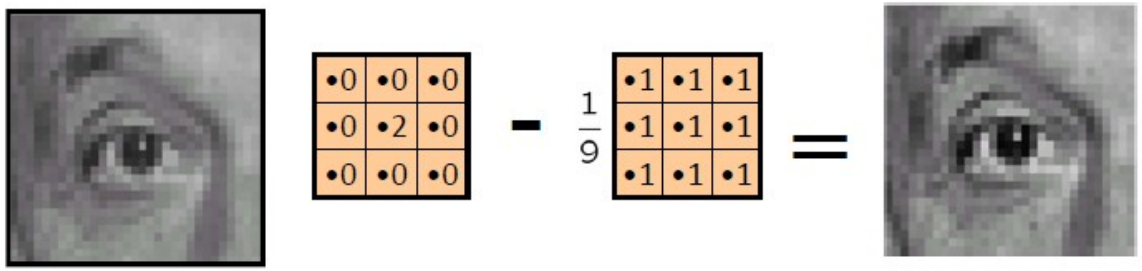
Filtros lineales

- ¿Qué se lleva el suavizado? → el detalle



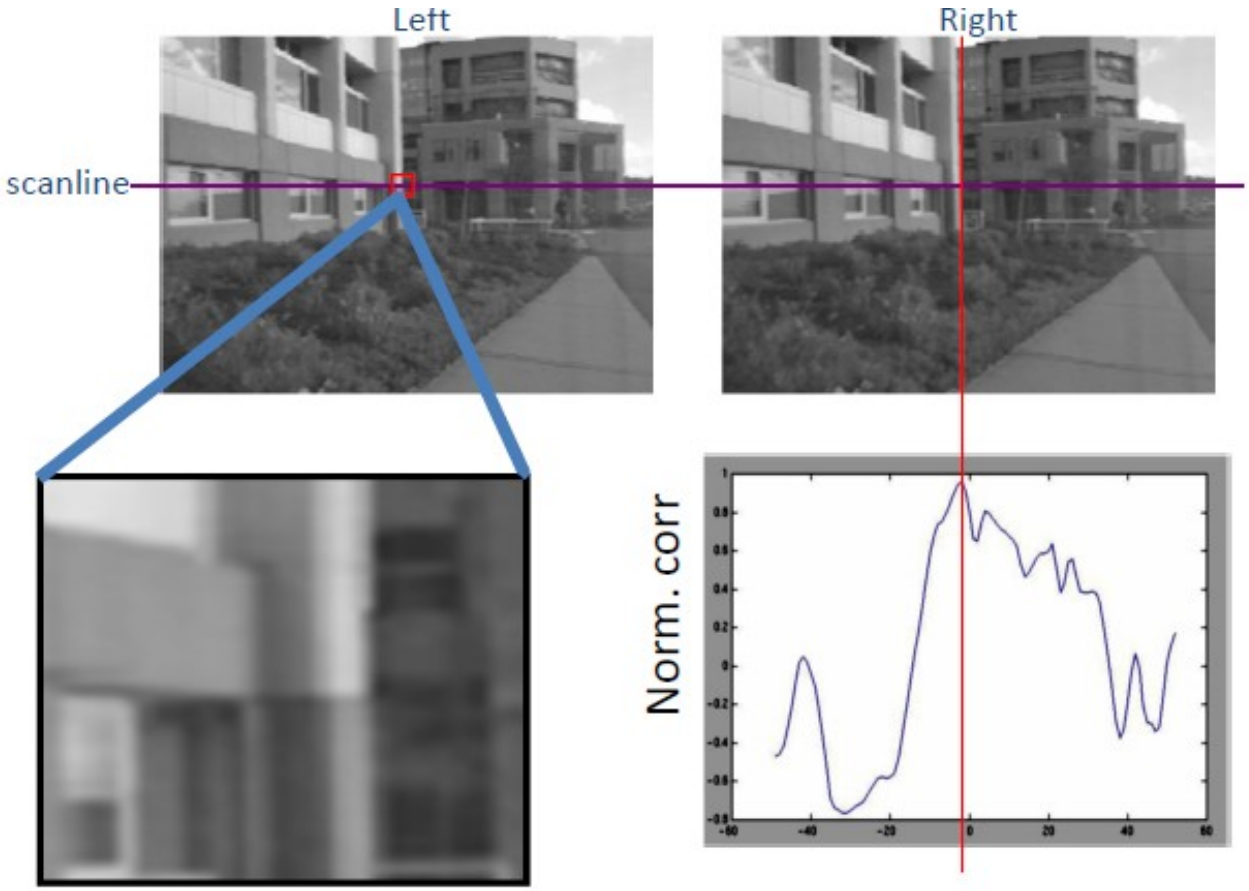
- Si lo volvemos a añadir

- Filtro *sharpening* (realce):



Filtros lineales

- Correlación cruzada (*cross-correlation*):
  - Medida de similaridad entre dos señales cuando la una es desplazada sobre la otra (alcanza un máximo cuando hay máxima similaridad)



Ejemplo: búsqueda de correspondencias en un par estéreo calibrado

$$\sum \widetilde{f}[n+k, m+l] * \widetilde{t}[k, l]$$

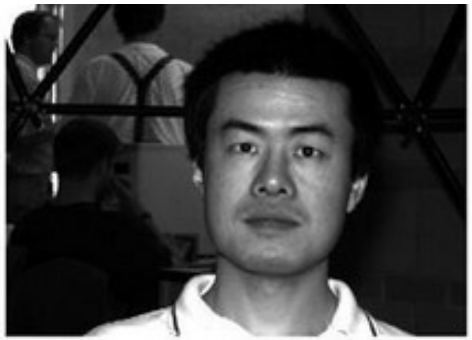
Se hace normalmente normalizando los parches de la imagen *f* y la plantilla *t* (restando la media y dividiendo por la desviación típica):

$$\frac{1}{n}\sum_{x,y}\frac{(f(x,y)-\overline{f})(t(x,y)-\overline{t})}{\sigma_f \sigma_t}$$

Template matching

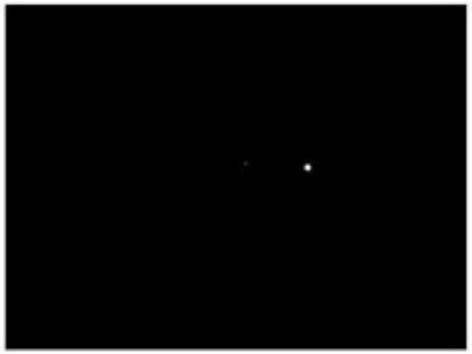
- Misma idea que lo anterior, para buscar zonas de imagen que se parezcan a un modelo dado:
  - Por supuesto, no es invariante a rotaciones, cambios de escala, etc. (método “ingenuo”).

1. Imagen de entrada



2. Plantilla (*template*)

3. NCC (Normalized cross-correlation)



4. Máximo NCC (posición de salida)

Propiedades de los filtros lineales

- Principio de superposición:

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

- Invarianza al desplazamiento:

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

- Conmutatividad:

$$f \circ h = h \circ f$$

- Asociatividad:

$$f \circ (h_1 \circ h_2) = (f \circ h_1) \circ h_2$$

Filtrado separable

- Aquel que puede reducirse a una pasada por filas y otra por columnas:

$\frac{1}{K^2}$

1	1	$\cdots$	1
1	1	$\cdots$	1
$\vdots$	$\vdots$	1	$\vdots$
1	1	$\cdots$	1

$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

$\frac{1}{256}$

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

$\frac{1}{8}$

−1	0	1
−2	0	2
−1	0	1

$\frac{1}{4}$

1	−2	1
−2	4	−2
1	−2	1

$\frac{1}{K}$

1	1	$\cdots$	1

$\frac{1}{4}$

1	2	1

$\frac{1}{16}$

1	4	6	4	1

1/2

-1	0	1

$*$

1/4

1
2
1

$\frac{1}{2}$

1	−2	1





Filtrado separable

- ¿Cómo saber si el *kernel* de un filtro es separable?
- La matriz del filtro debe poder ponerse en la forma de un *outer product* de dos vectores:

$$\boldsymbol{K} = \boldsymbol{v}\boldsymbol{h}^T$$

- Se hace la descomposición SVD, y ha de salir un sólo valor singular (σ_1) distinto de cero:

$$\boldsymbol{K} = \sum_i \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T$$

- Es decir, todas las filas son una combinación lineal de la primera.

Más ejemplos de filtrado lineal

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

Realce (*sharpening*)

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

Filtro gaussiano (blurring)

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

Operador laplaciano

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

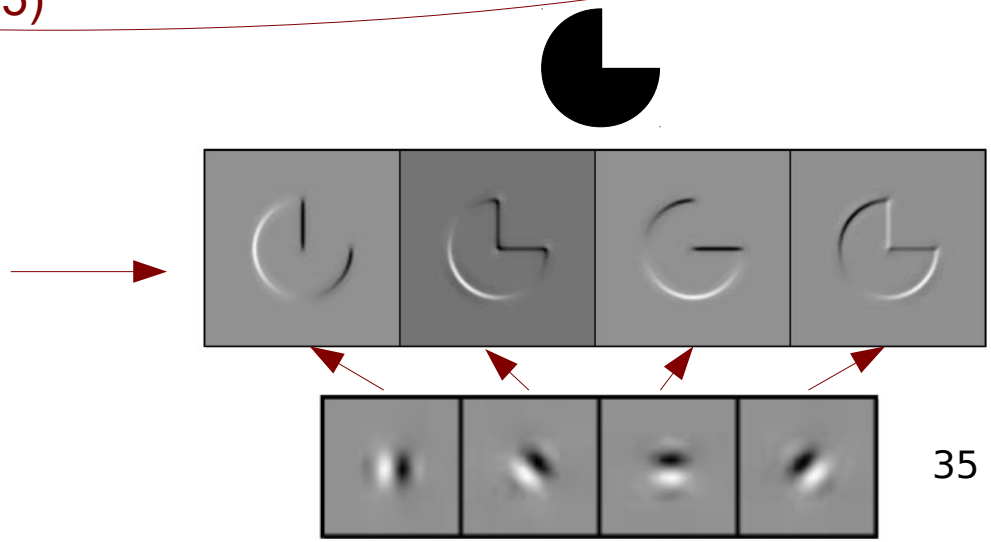
Laplaciana de la gaussiana

Importante para SIFT (Tema 5)

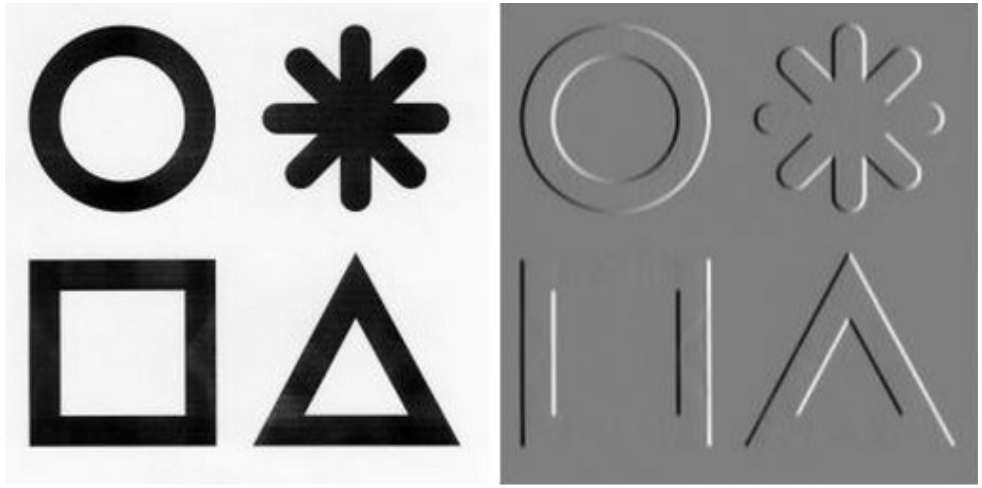
$$\hat{\boldsymbol{u}} \cdot \nabla(G * f) = \nabla_{\hat{\boldsymbol{u}}}(G * f) = (\nabla_{\hat{\boldsymbol{u}}} G) * f$$

$$G_{\hat{\boldsymbol{u}}} = uG_x + vG_y = u\frac{\partial G}{\partial x} + v\frac{\partial G}{\partial y}$$

Filtros direccionables (ej. Sobel Operator)

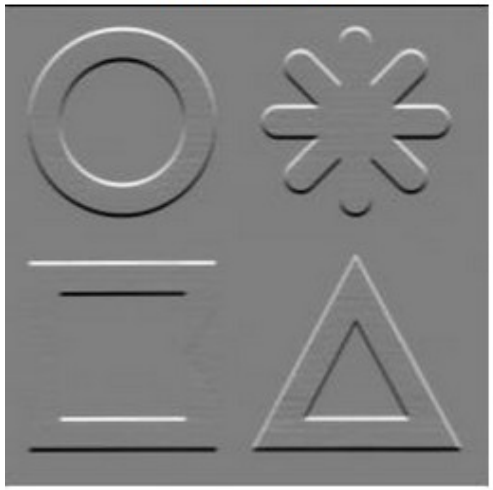


Ejemplos

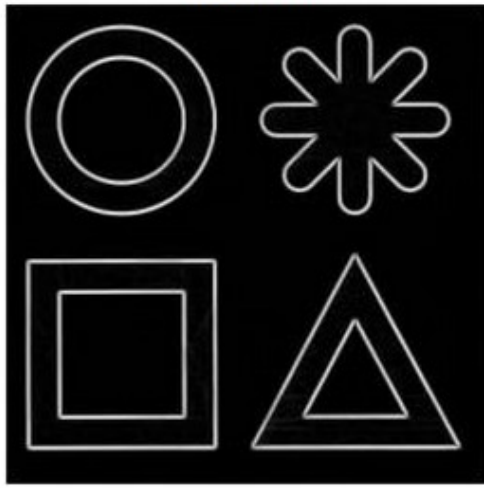


original image

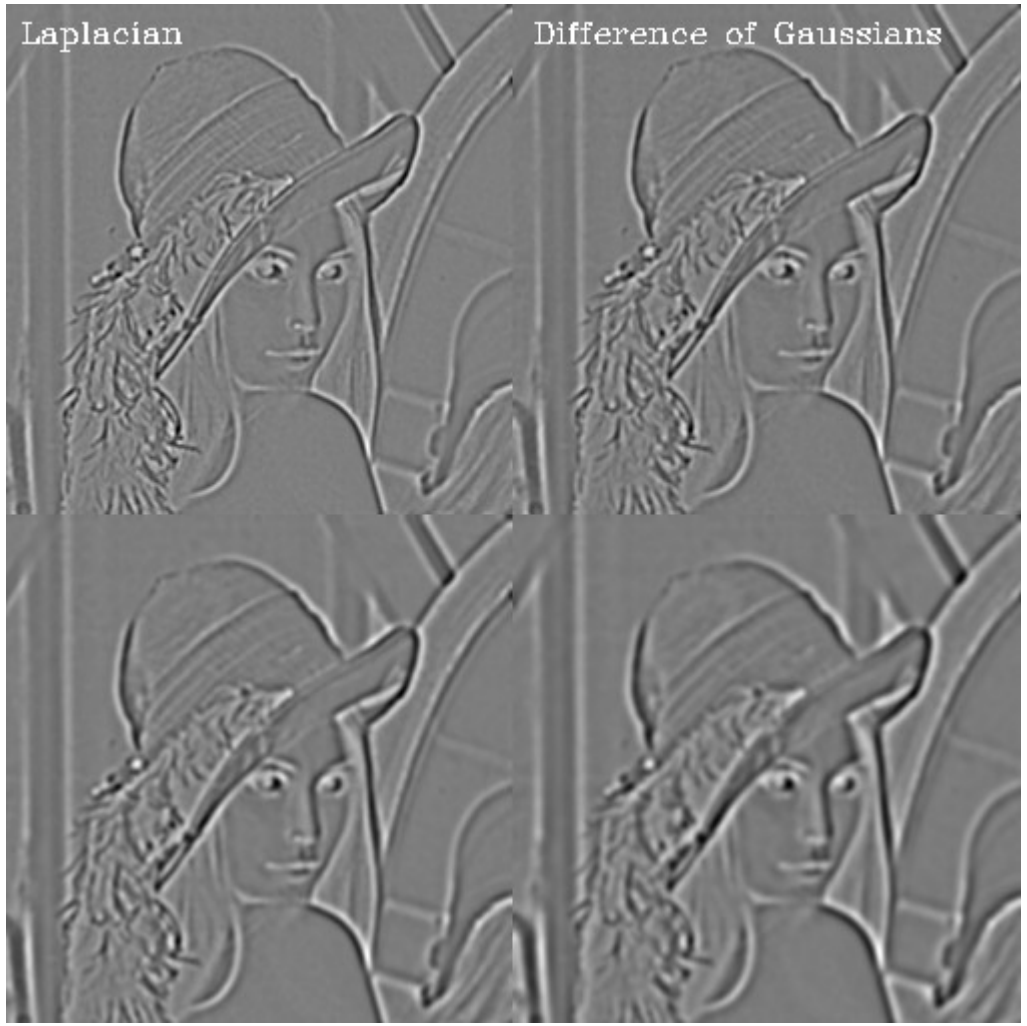
x-gradient



y-gradient



gradient magnitude



Ejemplos numéricos de lo anterior

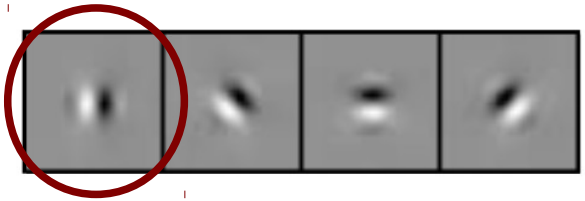
-1	-1	-1
-1	9	-1
-1	-1	-1

Realce (*sharpening*)

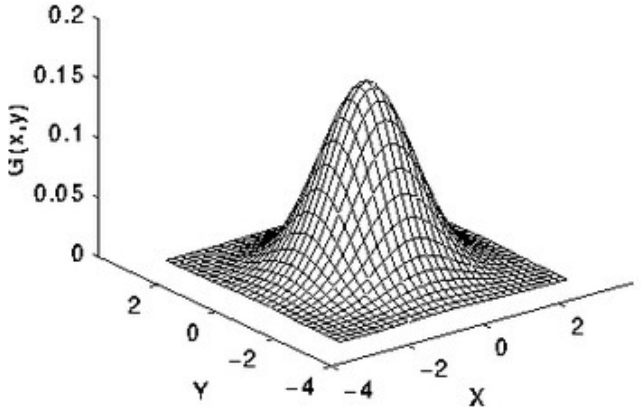
-1	-1	-1
-1	8	-1
-1	-1	-1

Operador laplaciano

$$S_x = \begin{bmatrix} 1 & 2 & 0 & -2 & -1 \\ 4 & 8 & 0 & -8 & -4 \\ 6 & 12 & 0 & -12 & -6 \\ 4 & 8 & 0 & -8 & -4 \\ 1 & 2 & 0 & -2 & -1 \end{bmatrix}$$

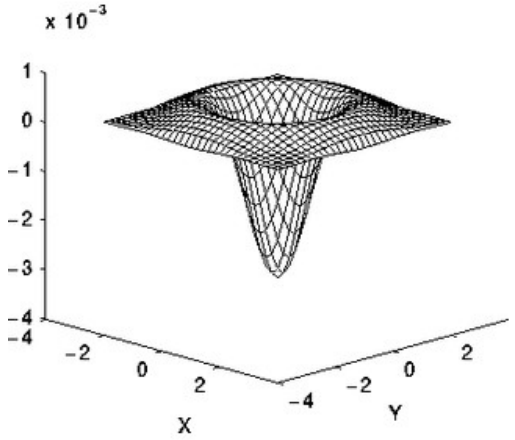


Sobel Operator (en X)



2	4	5	4	2
4	9	12	9	4
5	12	15	12	5
4	9	12	9	4
2	4	5	4	2

Filtro gaussiano (blurring)

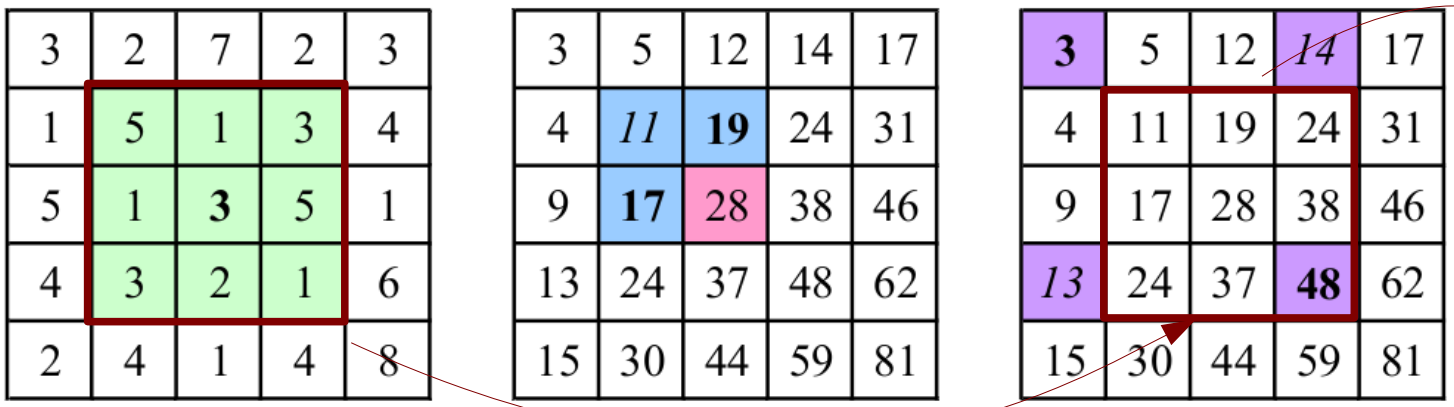


$$\begin{pmatrix} 0 & 0 & 3 & 2 & 2 & 2 & 3 & 0 & 0 \\ 0 & 2 & 3 & 5 & 5 & 5 & 3 & 2 & 0 \\ 3 & 3 & 5 & 3 & 0 & 3 & 5 & 3 & 3 \\ 2 & 5 & 3 & -12 & -23 & -12 & 3 & 5 & 2 \\ 2 & 5 & 0 & -23 & -40 & -23 & 0 & 5 & 2 \\ 2 & 5 & 3 & -12 & -23 & -12 & 3 & 5 & 2 \\ 3 & 3 & 5 & 3 & 0 & 3 & 5 & 3 & 3 \\ 0 & 2 & 3 & 5 & 5 & 5 & 3 & 2 & 0 \\ 0 & 0 & 3 & 2 & 2 & 2 & 3 & 0 & 0 \end{pmatrix}$$

Laplaciana de la gaussiana

Imagen integral

- Ejemplo de filtrado recursivo:
  - También llamados “de respuesta infinita al impulso” (IIR), frente a los de respuesta finita (FIR).



S=24 s=28 S=24

Suma de toda la ventana:
48-14-13+3 = 24
(=5+1+3+...+2+1)

Sólo 3 operaciones ¡para cualquier tamaño de ventana!

Computación recursiva de la imagen integral (cada píxel resulta de sumar todos los píxeles que le quedan “arriba y a la izquierda”

$$s(i,j) = \sum_{k=0}^{i} \sum_{l=0}^{j} f(k,l)$$

$$s(i,j) = s(i-1,j) + s(i,j-1) - s(i-1,j-1) + f(i,j)$$

Filtros no lineales

- La salida depende de una combinación no lineal de los vecinos.
- P.e., los usados para suavizado "*edge-preserving*":

Original

Filtrado Gaussiano
(lineal)

Filtrado de mediana
(no lineal)

Muy utilizado (valor mediano,
en lugar de medio como FilterBox)
de los valores de la ventana

Filtrados α-*mean* y bilateral

- Otras posibilidades para suavizado "*edge-preserving*":

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

Mediana

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

α-mean

Se descartan los valores menores y mayores (prop. α) y se hace la media de los restantes

	2	1	0	1	2
2	0.1	0.3	0.4	0.3	0.1
1	0.3	0.6	0.8	0.6	0.3
0	0.4	0.8	1.0	0.8	0.4
1	0.3	0.6	0.8	0.6	0.3
2	0.1	0.3	0.4	0.3	0.1

Filtro de dominio

0.0	0.0	0.0	0.0	0.2
0.0	0.0	0.0	0.4	0.8
0.0	0.0	1.0	0.8	0.4
0.0	0.2	0.8	0.8	1.0
0.2	0.4	1.0	0.8	0.4

Filtro de rango

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

$$g(i,j) = \frac{\sum_{k,l} f(k,l)w(i,j,k,l)}{\sum_{k,l} w(i,j,k,l)} \qquad r(i,j,k,l) = \exp\left(-\frac{\|f(i,j)-f(k,l)\|^2}{2\sigma_r^2}\right)$$

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

Filtrados α-*mean* y bilateral

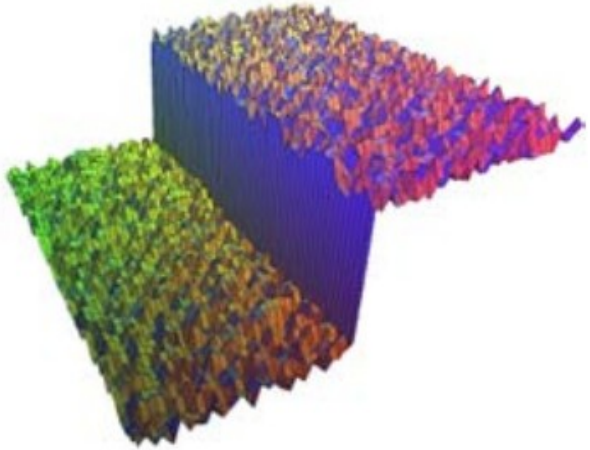
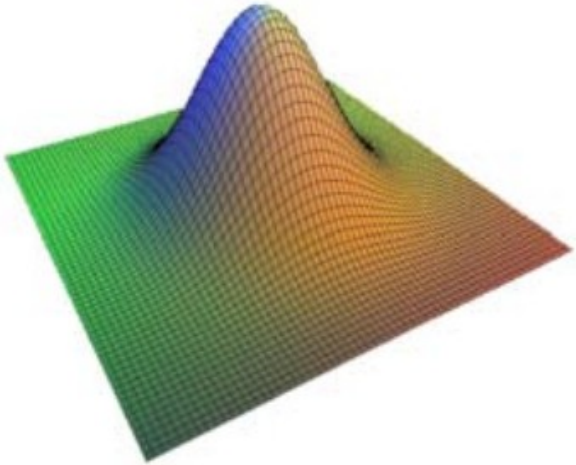
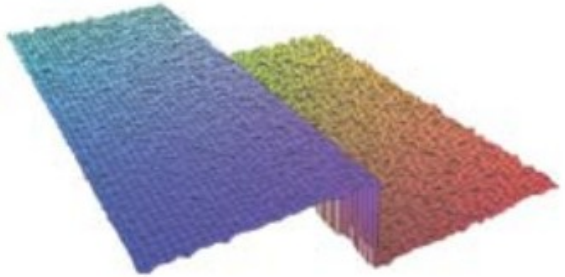


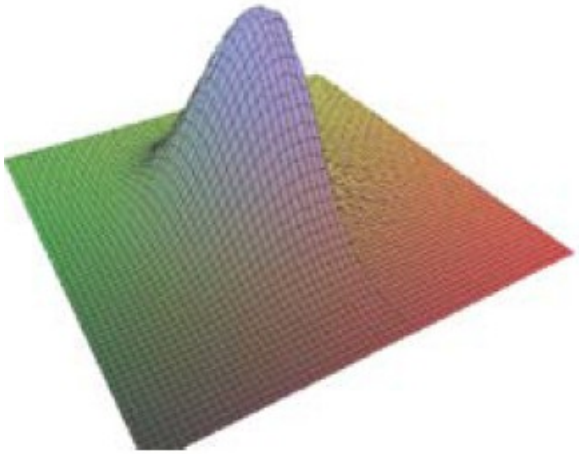
Imagen original



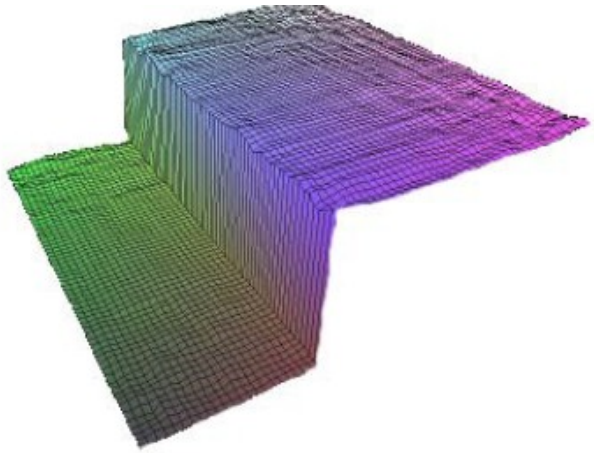
Filtro de dominio



Filtro de rango
(similaridad al pixel central)



Filtro bilateral resultante
(combinación dominio/rango)



Resultado

Ejemplo de filtrado bilateral

Original Resultado

Sigue eliminando ruido tipo “*salt and pepper*”, y respetando los bordes, pero el resultado es más “natural” que en el filtro de mediana

Operaciones morfológicas

(Consideramos aquí blanco=0, negro=1)

Original | Dilatación | Erosión | Apertura | Cierre

Podría haberse eliminado el punto con un elemento estructural más grande

Se suelen aplicar tras *binarizar* la imagen, y la salida es también binaria:

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

Cuenta de píxeles activos dentro de la máscara en la imagen de entrada

Umbral

$$\begin{aligned} \text{dilate}(f,s) &= \theta(c,1) \\ \text{erode}(f,s) &= \theta(c,S) \\ \text{open}(f,s) &= \text{dilate}(\text{erode}(f,s),s) \\ \text{close}(f,s) &= \text{erode}(\text{dilate}(f,s),s) \end{aligned}$$

c = cuenta de unos dentro del elemento estructural *s* (normalmente ventana 3x3), de tamaño S

Transformada de distancia

- Cada píxel de salida almacenará la distancia al píxel activo (aquí *b(k,l)=0*) de la entrada (imagen binaria *b*) más cercano:

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

$$d_1(k,l) = |k| + |l|$$

Distancia de Manhattan

$$d_2(k,l) = \sqrt{k^2 + l^2}$$

Distancia euclídea

  - Ejemplo de cómputo con Manhattan (euclídea más difícil):

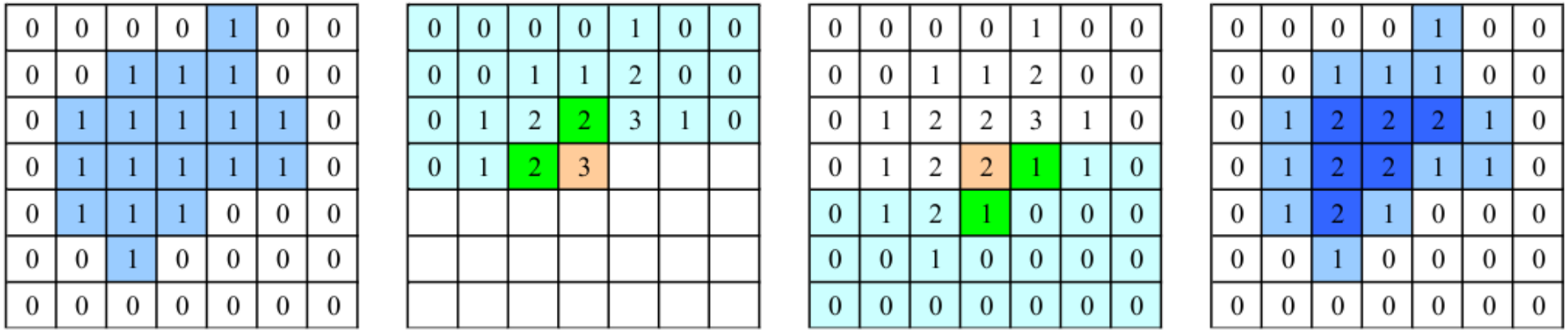


Imagen binaria original (aquí “activo” = 0)

Mitad primera pasada (hacia adelante)

Mitad segunda pasada (hacia atrás)

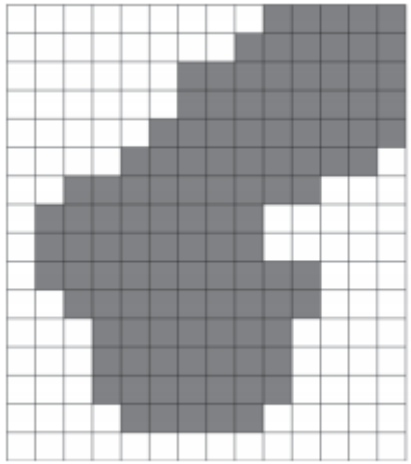
Resultado final

Cada pixel no cero se reemplaza por mínimo de 1 más la distancia de vecino N o W

Idem, pero mínimo de valor actual, 1+dist(S),1+dist(W)

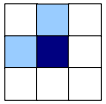
Transformada de distancia

- Ejemplo completo:

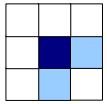


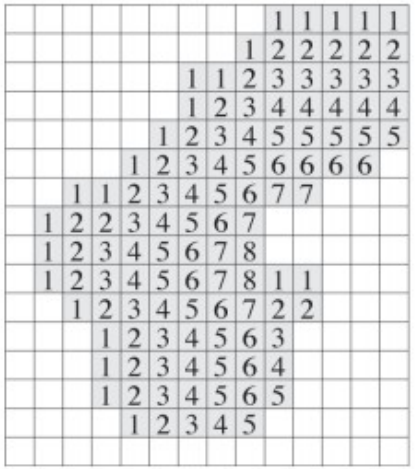
Distancia Manhattan 4 conectada

Primera:

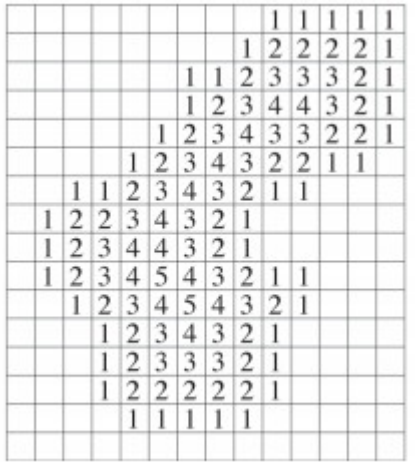


Segunda:



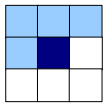


Tras la primera pasada

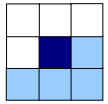


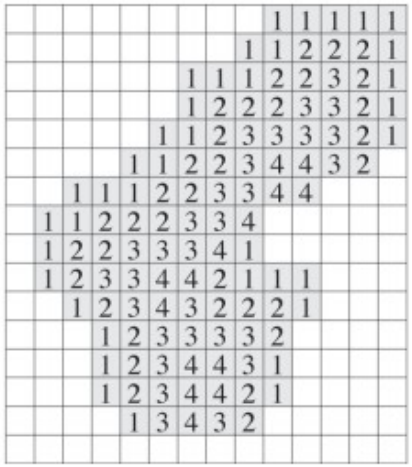
Distancia Manhattan 8 conectada

Primera:

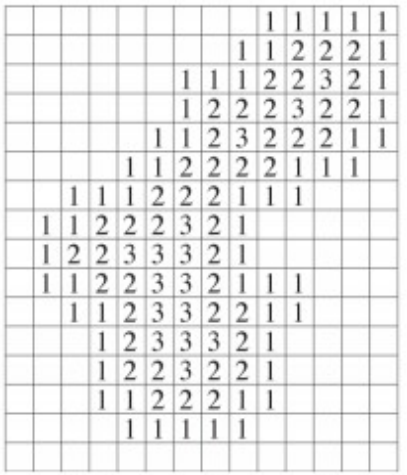


Segunda:





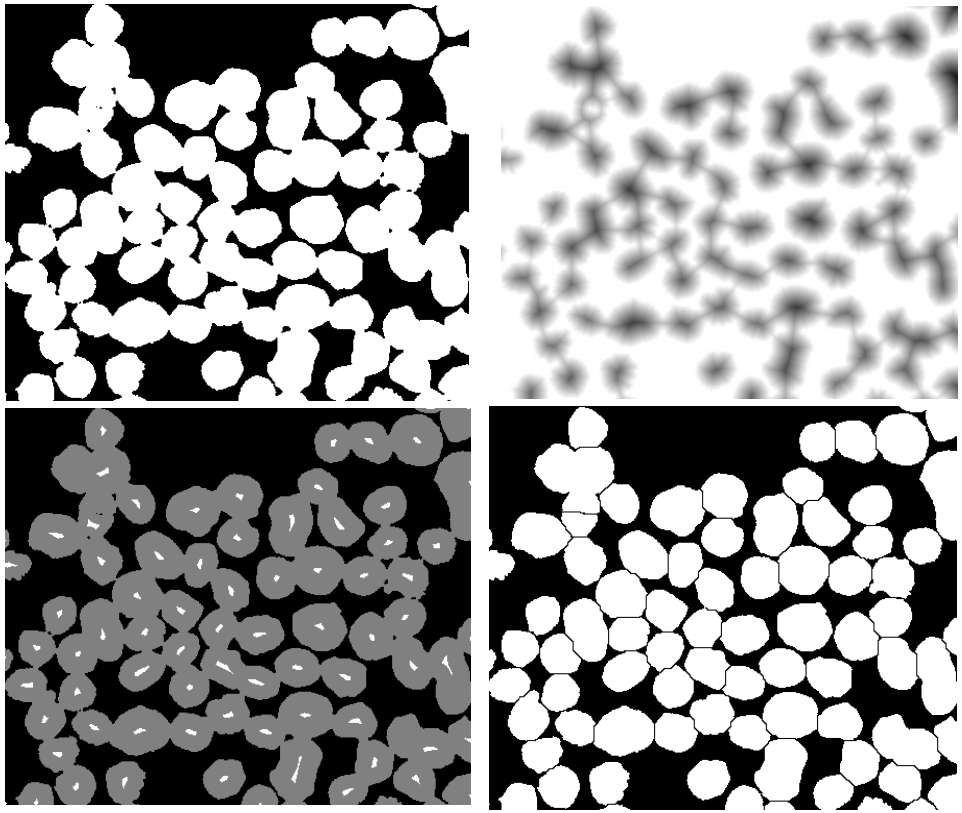
Tras la segunda pasada



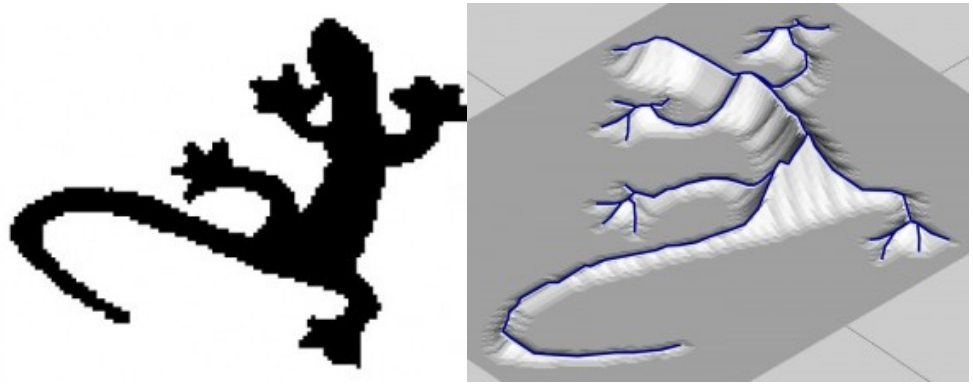
Tras la segunda pasada (final)

Transformada de distancia

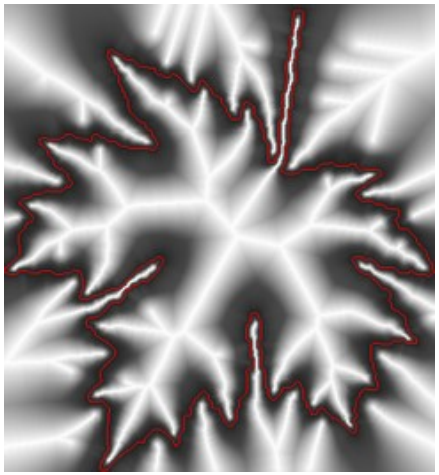
- Ejemplos de aplicación:



Conteo / separación de células
(máximos locales de la DT +
usar etiquetas de píxeles “fuente”)



“Esqueletización”
(Medial axis)



Distancia entre
contornos

Componentes conexas

- Cada píxel de salida etiquetado con número (color) distinto para cada componente 4- u 8-conectada.
- Ejemplo 4-conectada:

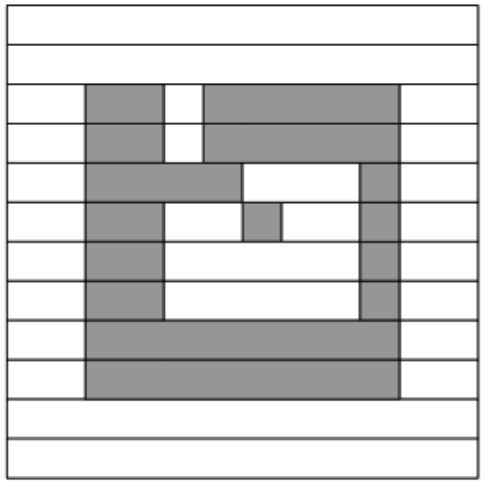
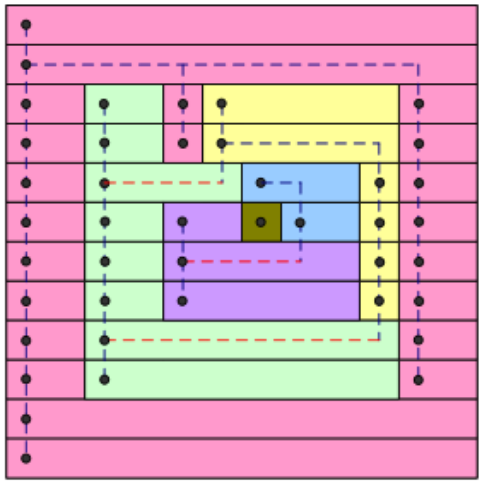
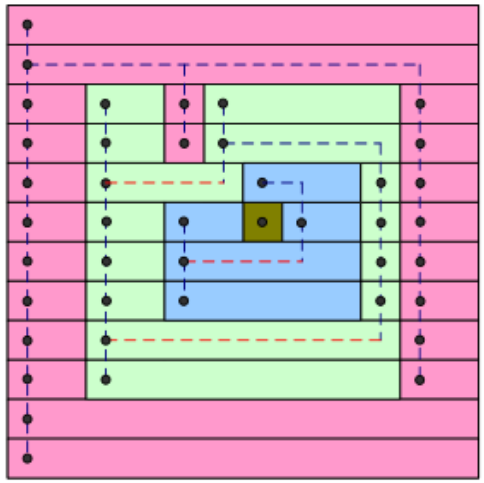


Imagen binaria original



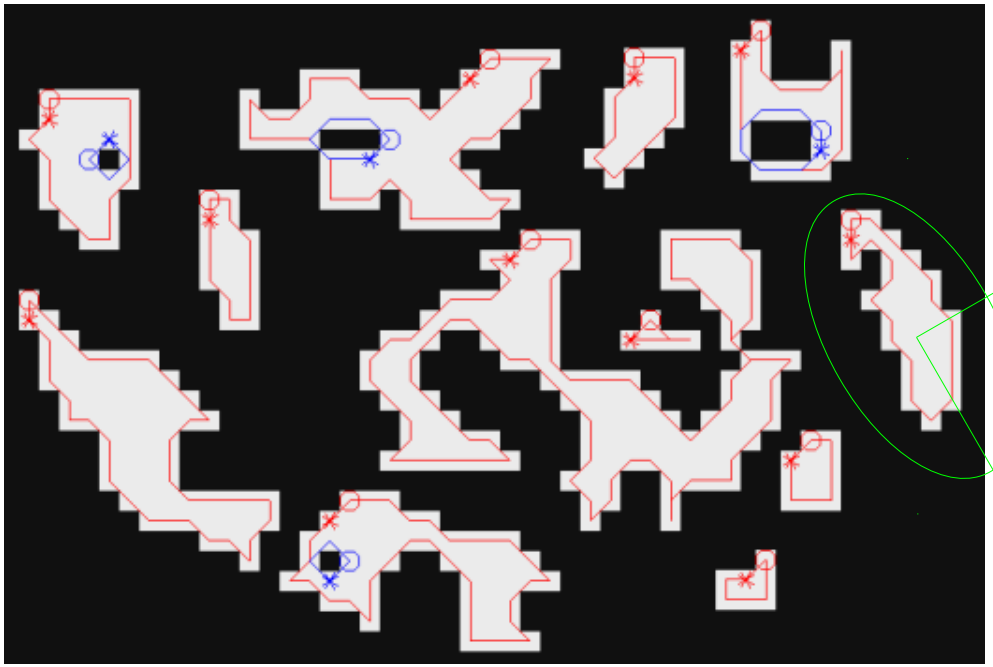
Primera pasada (mirando siempre arriba y a la izquierda) (*conexiones negras*)



Pasada final uniendo componentes no unidos en recorrido anterior (*conexiones rojas*)

Componentes conexas

- Es habitual calcular después para cada componente su contorno, perímetro, centroide, …
- … incluso segundos momentos estadísticos y, a partir de ellos, las direcciones de los ejes principales (vectores principales, *eigensystem*):

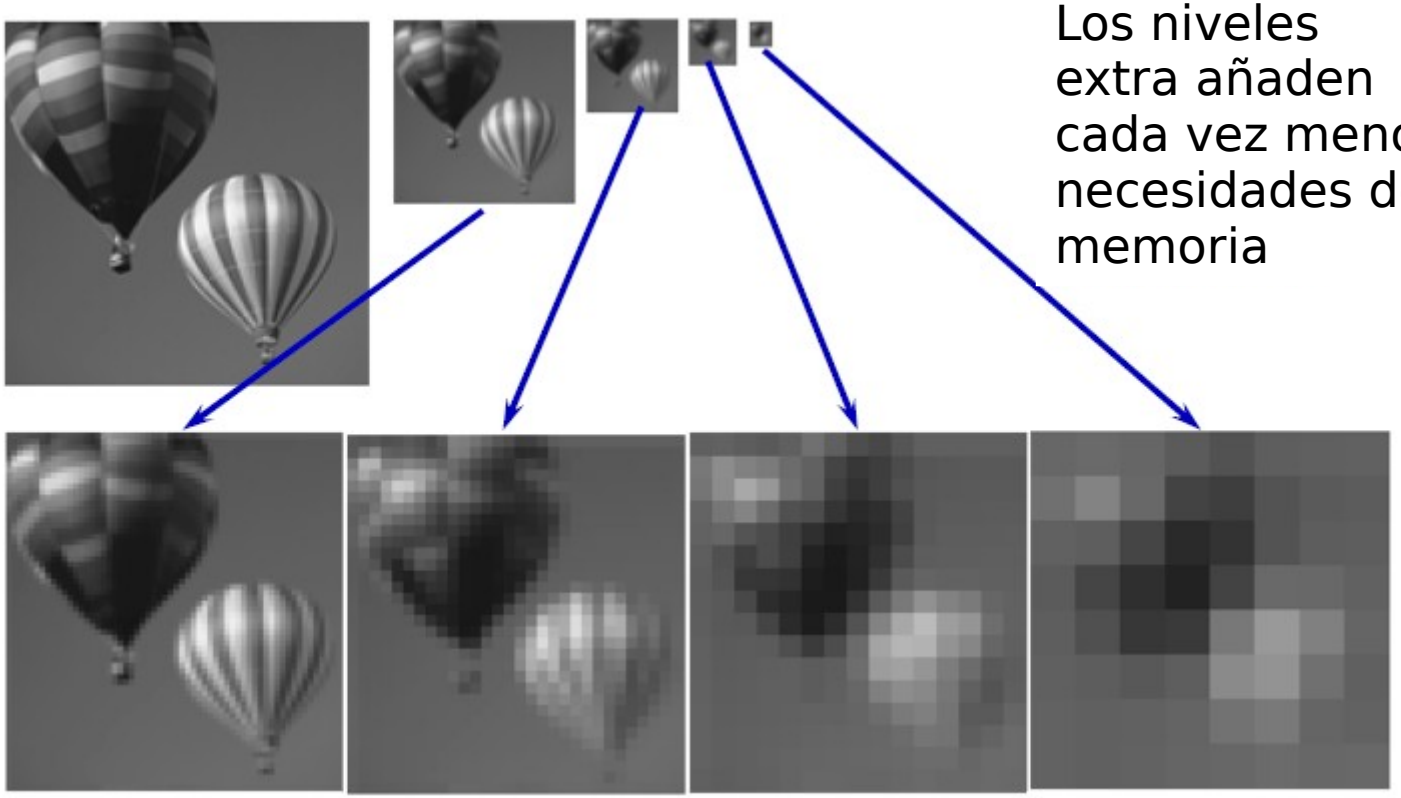


$$\boldsymbol{M} = \sum_{(x,y)\in\mathcal{R}} \begin{bmatrix} x-\overline{x} \\ y-\overline{y} \end{bmatrix} \begin{bmatrix} x-\overline{x} & y-\overline{y} \end{bmatrix}$$

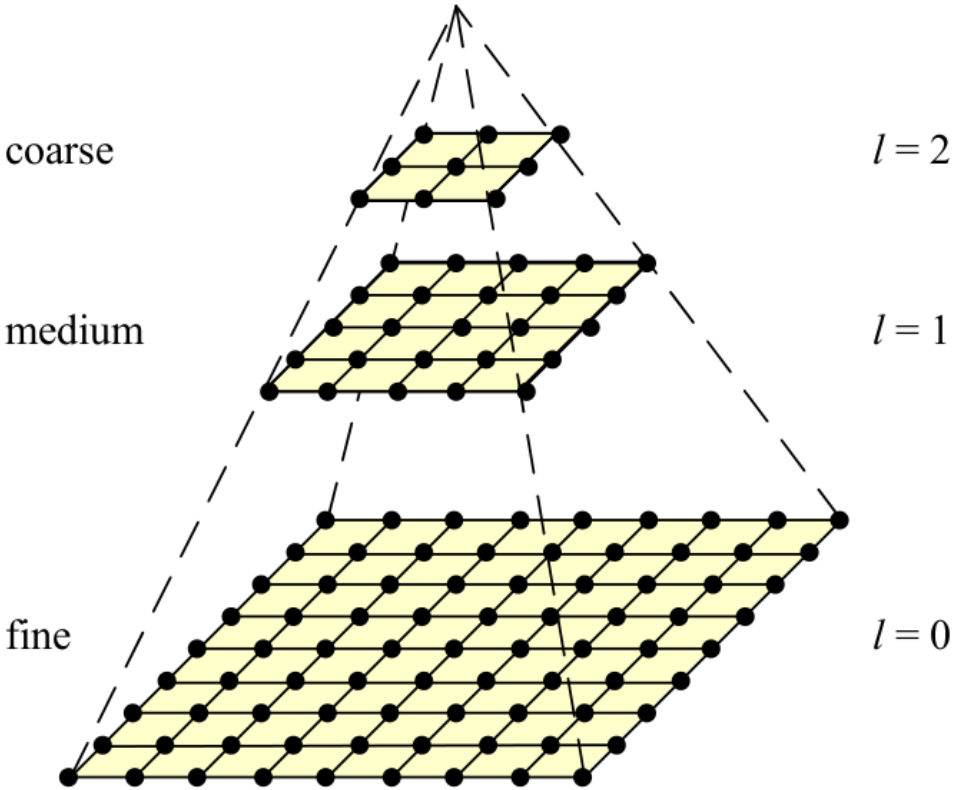
$$M = V D V^T$$

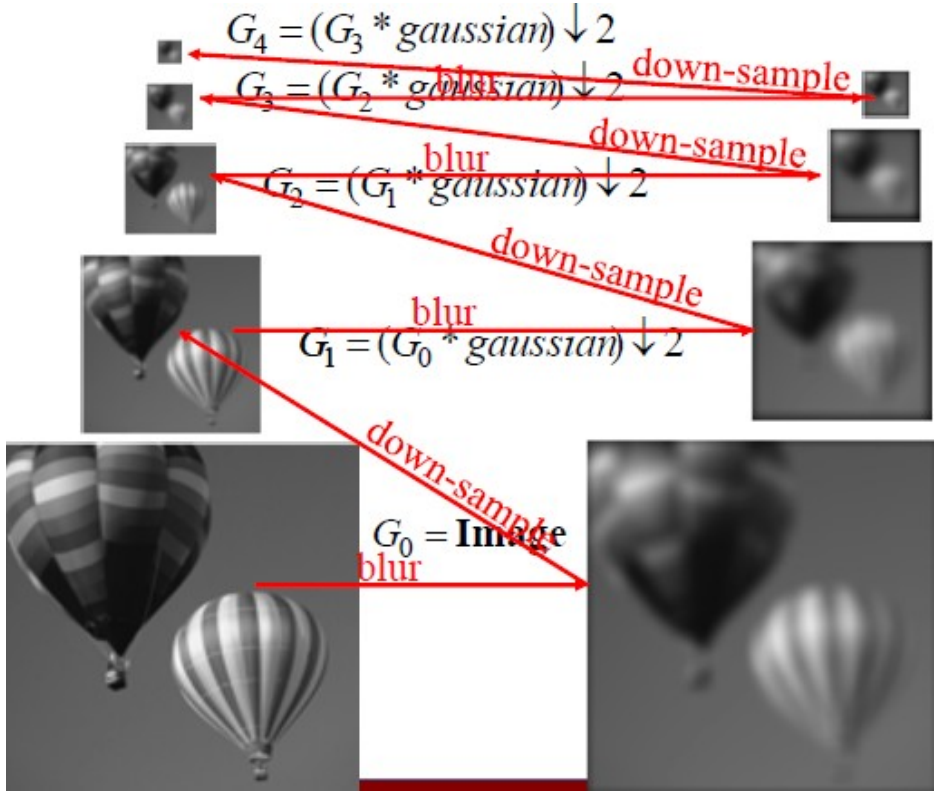
Pirámides de imagen

- Reducción iterativa del tamaño de la imagen.
  - P.e. para realizar búsquedas multiescala.
- Suavizar antes de submuestrear → evita *aliasing*.



Pirámides de imagen

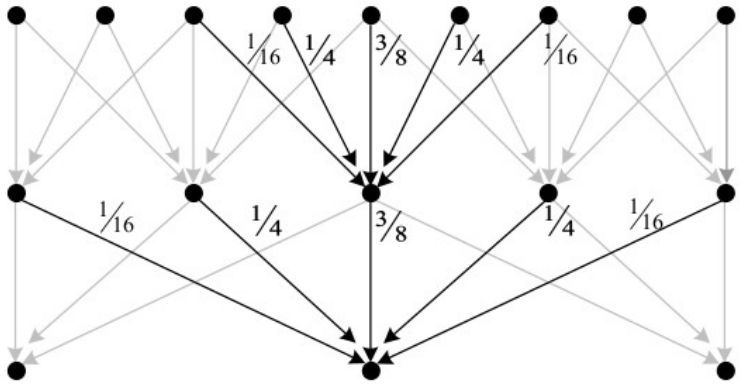




$$\frac{1}{16}\begin{array}{|c|c|c|c|c|}\hline 1 & 4 & 6 & 4 & 1 \\ \hline\end{array}$$

Típico filtro
five-tap

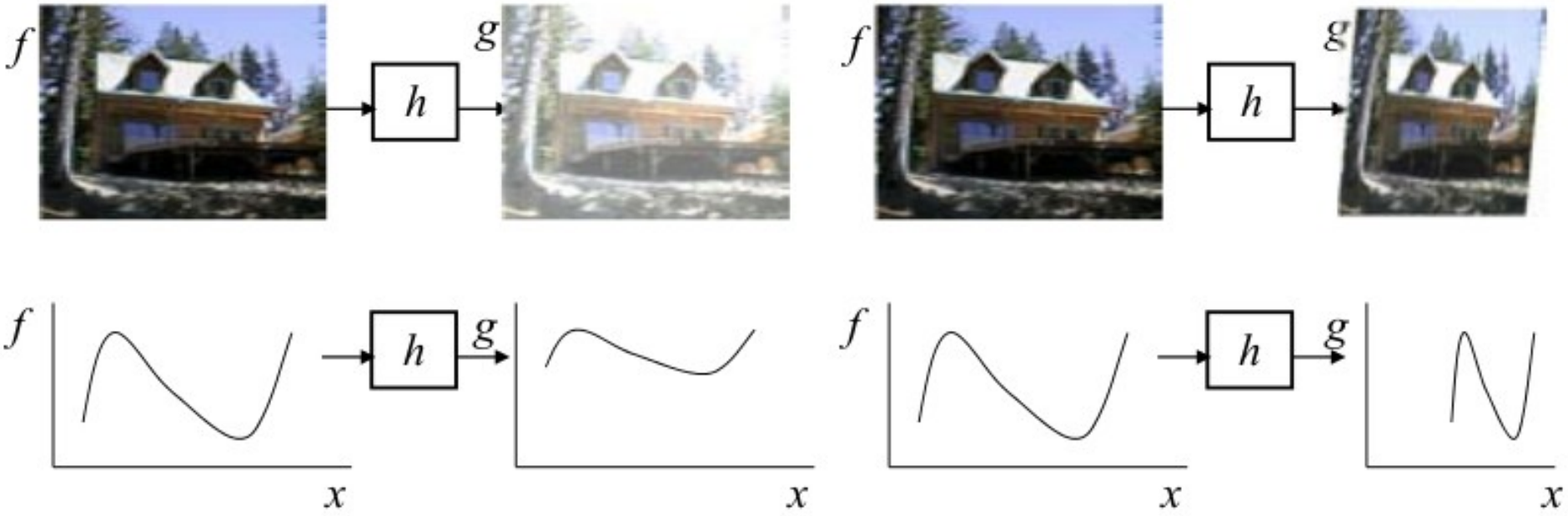
Importante para SIFT (Tema 5)



Grupo 3: Transformaciones geométricas

- Se realiza una modificación del **dominio** de la imagen, en lugar del **rango** (*warping*):

$$g(\boldsymbol{x}) = f(\boldsymbol{h}(\boldsymbol{x})) \quad \text{en lugar de} \quad g(\boldsymbol{x}) = h(f(\boldsymbol{x}))$$



Transformación de RANGO

Transformación de DOMINIO

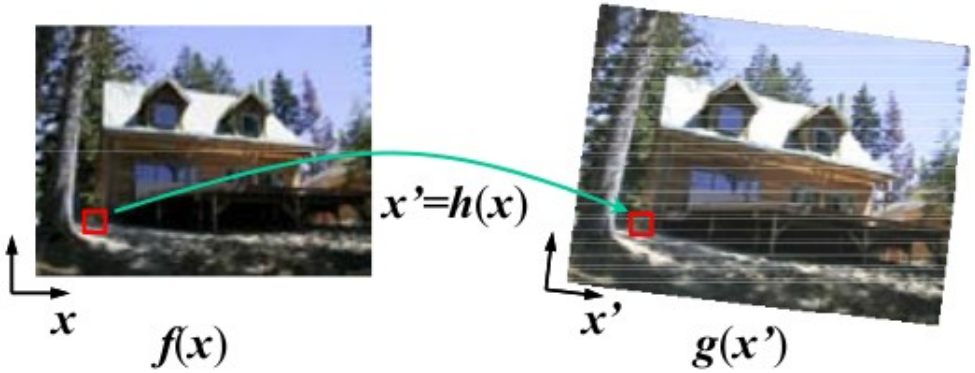
Transformaciones geométricas

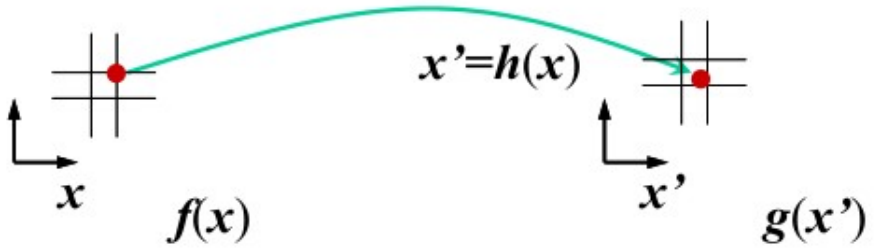
- Procedimiento de *warping*:

procedure *forwardWarp*$(f, \boldsymbol{h}, \textbf{out}\ g)$:

For every pixel $\boldsymbol{x}$ in $f(\boldsymbol{x})$

1. Compute the destination location $\boldsymbol{x}' = \boldsymbol{h}(\boldsymbol{x})$.
2. Copy the pixel $f(\boldsymbol{x})$ to $g(\boldsymbol{x}')$.





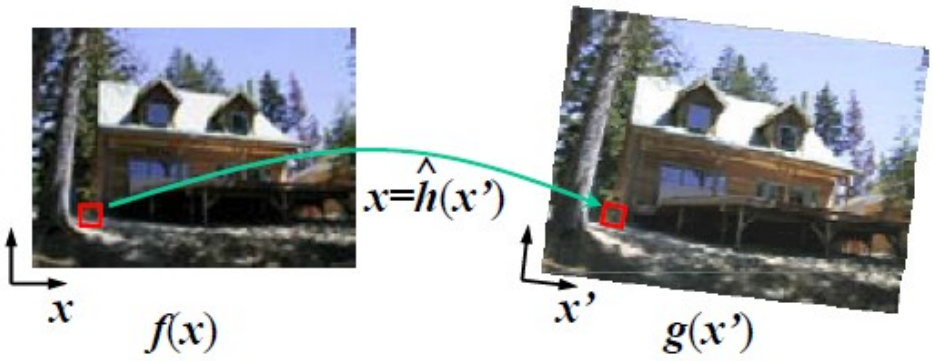
Transformaciones geométricas

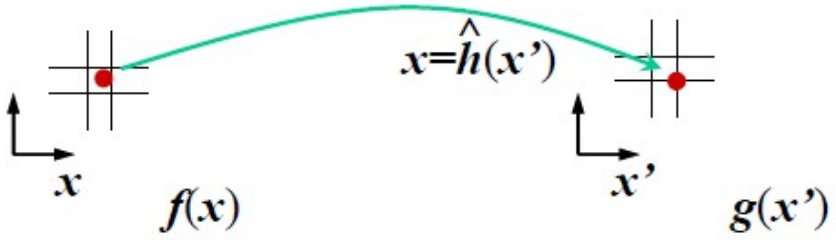
- Procedimiento de *warping* *inverso*:

procedure *inverseWarp*$(f, \boldsymbol{h}, \textbf{out}\ g)$:

For every pixel $\boldsymbol{x}'$ in $g(\boldsymbol{x}')$

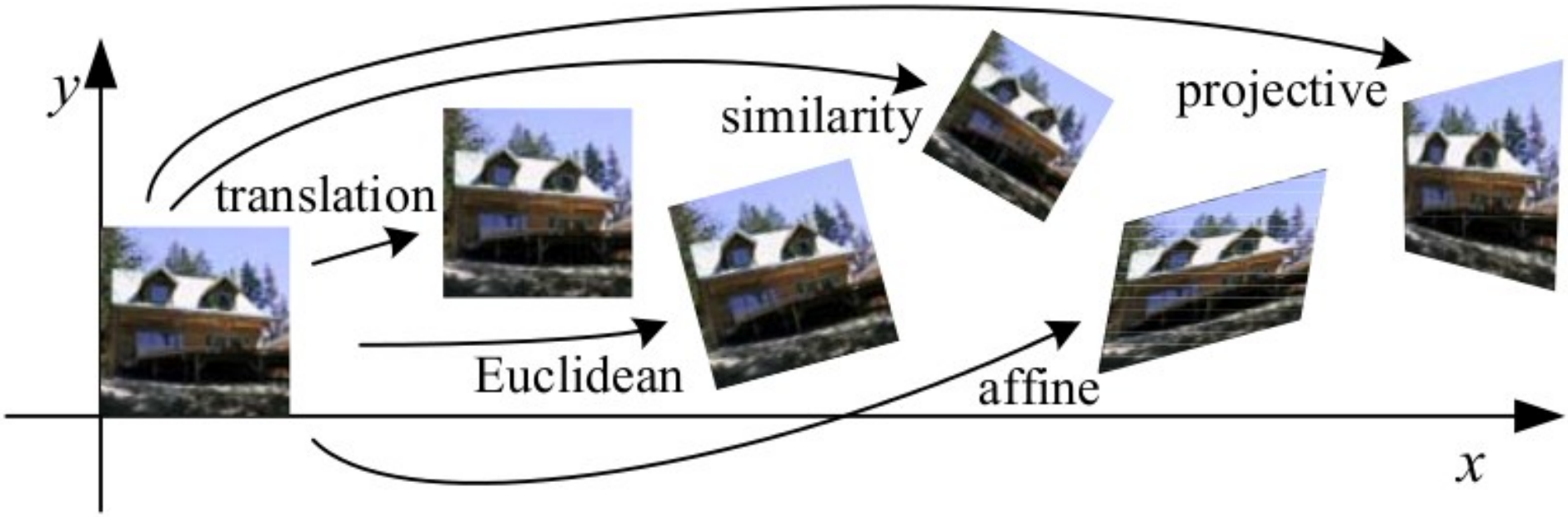
1. Compute the source location $\boldsymbol{x} = \hat{\boldsymbol{h}}(\boldsymbol{x}')$
2. Resample $f(\boldsymbol{x})$ at location $\boldsymbol{x}$ and copy to $g(\boldsymbol{x}')$





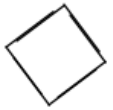
Transformaciones geométricas

- Repertorio básico de transformaciones geométricas (*warpings*) 2D:



Transformaciones geométricas

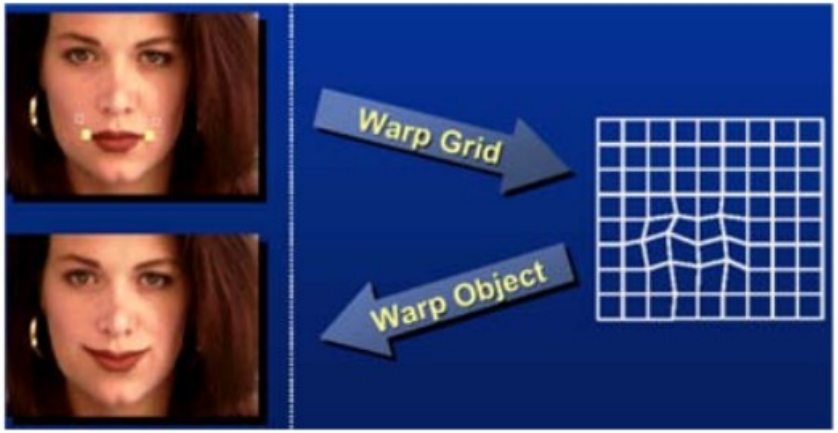
- Jerarquía de transformaciones 2D (funcionan con coordenadas homogéneas, esto es, $(x,y,1)^T$.
  - La coordenada adicional generada por la homografía *H* divide a las otras dos.

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\left[\begin{array}{c\|c} \boldsymbol{I} & \boldsymbol{t} \end{array}\right]_{2\times 3}$	2	orientation	
rigid (Euclidean)	$\left[\begin{array}{c\|c} \boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{2\times 3}$	3	lengths	
similarity	$\left[\begin{array}{c\|c} s\boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{2\times 3}$	4	angles	
affine	$\left[\begin{array}{c} \boldsymbol{A} \end{array}\right]_{2\times 3}$	6	parallelism	
projective	$\left[\begin{array}{c} \tilde{\boldsymbol{H}} \end{array}\right]_{3\times 3}$	8	straight lines	

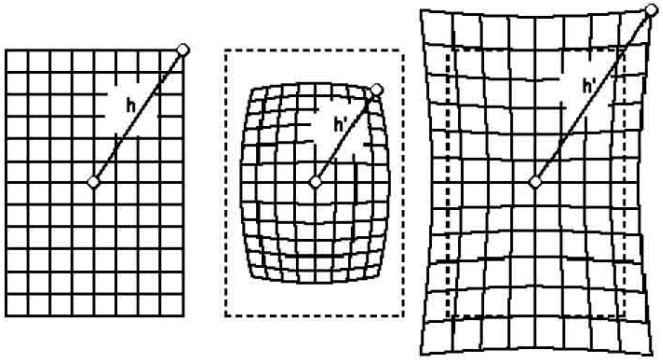
A menudo se extienden las matrices 2x3 con una tercera fila *(0,0,1)* para no tratar casos especiales.

Otras transformaciones geométricas

- Transformaciones basadas en mallas (*meshes*):

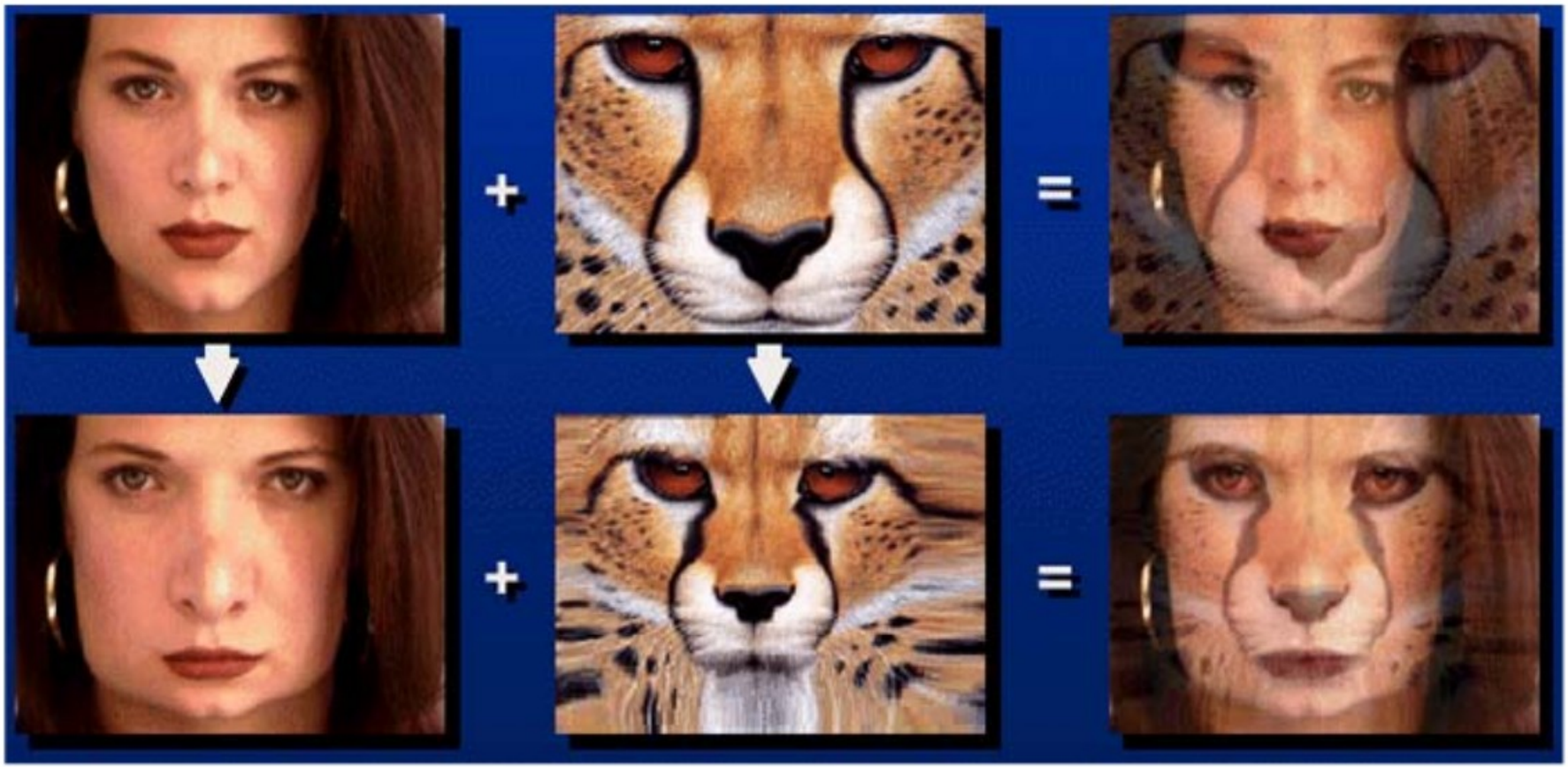


- Ejemplo: corrección de la **distorsión radial**:



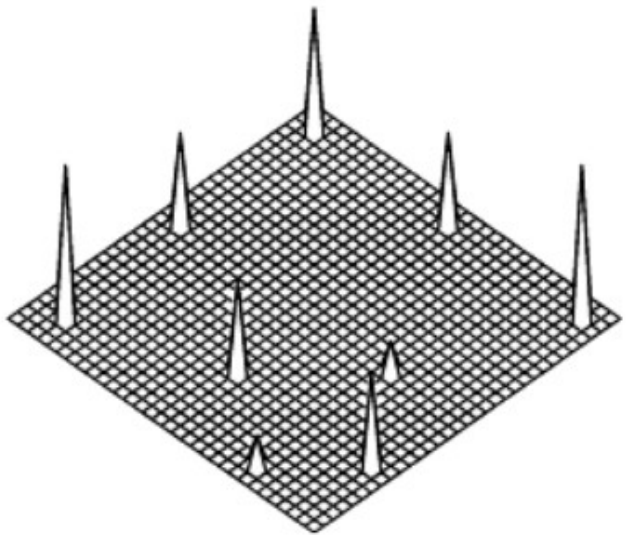
Otras transformaciones geométricas

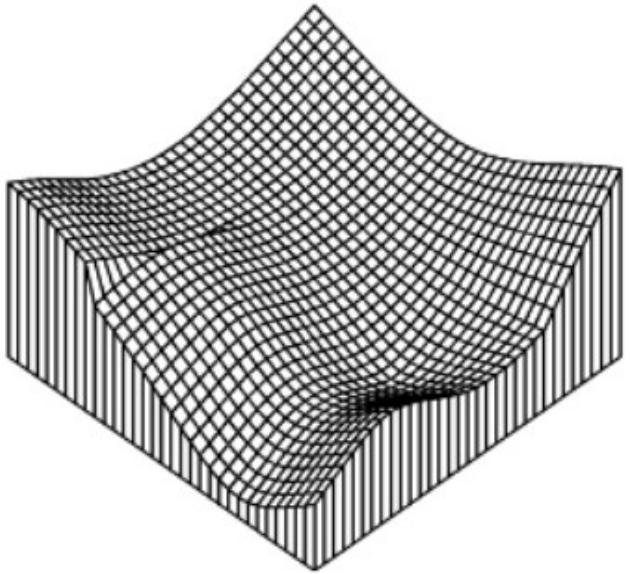
- *Warping* combinado con *blending* → *morphing* más suave:



Grupo 4: Optimización global sobre la imagen

- Inferir la imagen de salida optimizando algún criterio sobre la de entrada, dadas unas restricciones.
  - Ejemplo: interpolación “suave”, respetando ciertos valores prefijados:





Grupo 4: Optimización global sobre la imagen

- Ejemplo de interpolación:
  - Reconstrucción de una imagen a partir de sus bordes.
  - Proceso de difusión (ecuación del calor):
    - A cada píxel de salida se asigna la media de sus vecinos, y se itera.
  - En realidad lo anterior minimiza el laplaciano de la imagen (módulo del vector de segunda derivada en cada píxel)

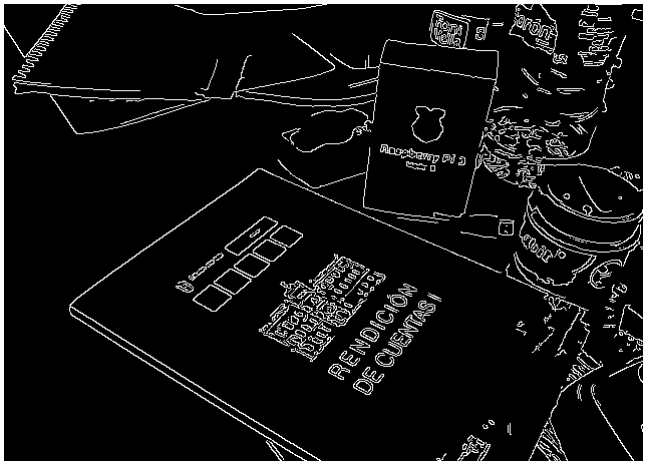
Original

Bordes (Canny)

Información relevante (intensidad)

Reconstrucción (final)

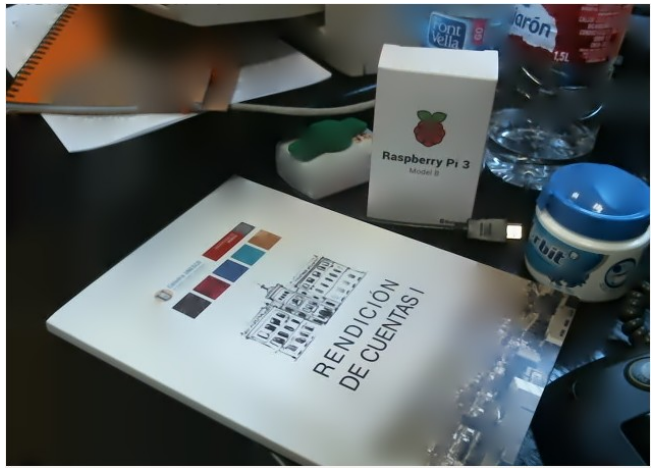
Grupo 4: Optimización global sobre la imagen



1. Canny

2. Dilatación + Muestreo color

3. Rellenado color
con transformada de distancia



4. Reconstrucción final
tras difusión

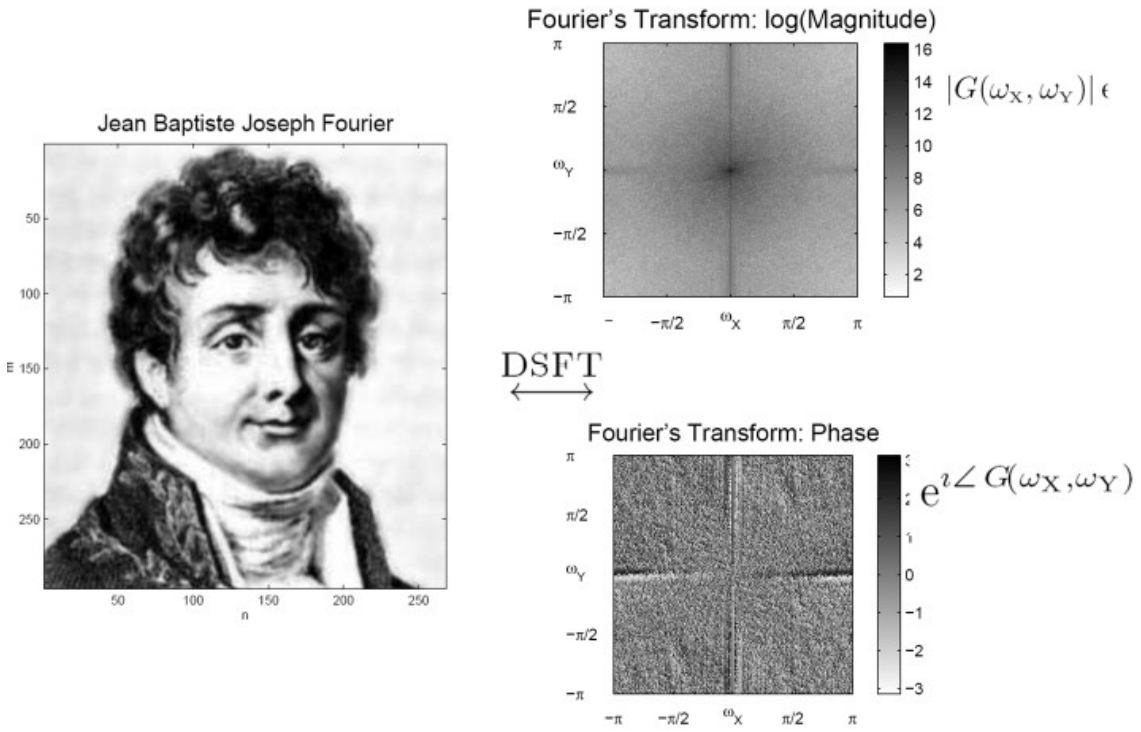
Otras técnicas globales

Denoising and inpainting

Mezcla de imágenes
(algoritmos de corte de grafos, *Graph cuts*)

Otras operaciones sobre imágenes

- *Wavelets*, transformadas de frecuencia...
  - FFT (*Fast Fourier Transform*)
  - DCT (*Discrete Cosine Transform*)
- Más usadas en compresión de imagen (JPG) o vídeo (MPEG, H264, etc.) que en visión.



Procesamiento de imagen en OpenCV

- Prácticamente todas las operaciones estudiadas están disponibles en la OpenCV:
  - Una gran mayoría, a través de wrappers de las Librerías Intel IPP (*Integrated Performance Primitives*)
    - Operaciones vectorizadas (extensiones SIMD).
- También hay implementaciones alternativas disponibles para compilar como extensiones a la OpenCV:
  - CUDA, OpenCL, Intel TBB (Threading Building Blocks), etc.

Referencias

- *"Computer Vision: Algorithms and Applications"*, R. Szeliski (2010), Springer (http://szeliski.org/Book/).
- Algunos notebooks de Python+OpenCV:
  - Primitivas gráficas, RGB ↔ gris, superficie imagen: ↵
  - Espacios de color: ↵; segmentación por color (probabilística): ↵
  - Histogramas, ecualización: ↵; Chroma (máscaras): ↵
  - Cuantización de color: ↵
  - Transformaciones de dominio, distorsión radial (con numpy): ↵
- Tutorial Python+OpenCV (secciones relevantes):
  - Image Processing in OpenCV: ↵
    - Colorspaces, geometric transformation of images, image thresholding, smoothing images, morphological transformations, image gradients, image pyramids, contours, histograms, image transforms (Fourier, cosine), template matching, grab cut.