PROCESAMIENTO DE IMAGENES

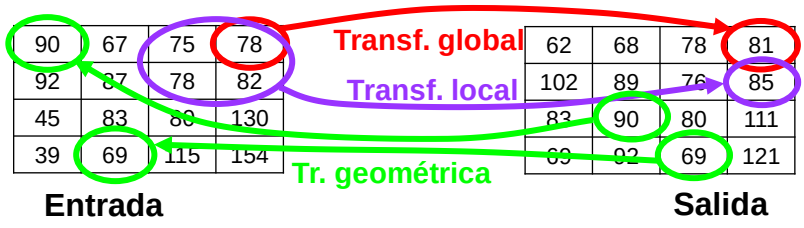
Programa de teoría

1. Adquisición y representación de imágenes.
2. Procesamiento global de imágenes.
3. Filtros y transformaciones locales.
4. **Transformaciones geométricas.**
5. Espacios de color y el dominio frecuencial.
6. Análisis de imágenes.

Tema 4. Transformaciones geométricas.

4.1. Interpolación y transformaciones básicas.
4.2. Transformaciones afines.
4.3. Transformaciones bilineal y perspectiva.
4.4. Transformaciones de mapeo.
A.4. Transf. geométricas OpenCV.

4.1. Interpolación y transf. básicas.

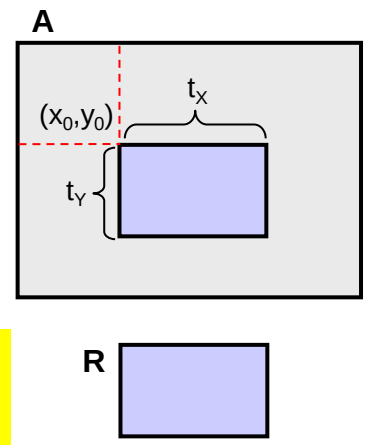


$R(x,y) := A(f_1(x,y), f_2(x,y))$

- **Transformación geométrica:** el valor de un píxel depende de otro píxel (o varios) cuya posición es calculada a través de un par de funciones $\mathbf{f_1}$ y $\mathbf{f_2}$.
- El tamaño de la imagen de salida puede ser distinto del tamaño de la imagen de entrada.

4.1. Interpolación y transf. básicas.

- **Ejemplo. Desplazamiento y recorte (*trim*):** dada una imagen **A**, seleccionar un trozo rectangular, desde el punto (x_0, y_0) con tamaño (t_X, t_Y).
- **R**: imagen de $(0..t_X\text{-}1, 0..t_Y\text{-}1)$
  $R(x,y) := A(x+x_0, y+y_0), \forall (x,y) \in R$
- ¿Qué pasa si $A(x+x_0, y+y_0)$ está fuera de rango?
  - Asignar alguna constante a R.
  - No modificar lo que hubiera antes en R.



Normalmente esta transformación no aparece de forma explícita, sino implícitamente, al trabajar con ROI

4.1. Interpolación y transf. básicas.

- **Reflejos y rotaciones exactas** (s. horario)

  Sea **A** la imagen de entrada, de (0..m_X, 0..m_Y)

  La esquina superior izquierda es (0, 0)

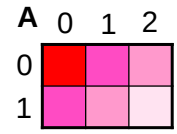
  – **Espejo horizontal**: R(x,y):= A(m_X-x, y)
  R: (0..m_X, 0..m_Y)

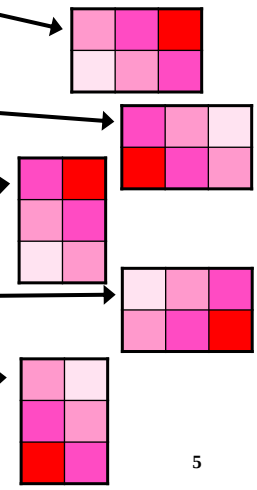
  – **Espejo vertical**: R(x,y):= A(x, m_Y-y)
  R: (0..m_X, 0..m_Y)

  – **Rotar 90º**: R(x,y):= A(y, m_Y-x)
  R: (0..m_Y, 0..m_X)

  – **Rotar 180º**: R(x,y):= A(m_X-x, m_Y-y)
  R: (0..m_X, 0..m_Y)

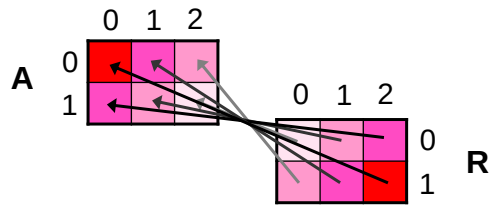
  – **Rotar 270º**: R(x,y):= A(m_x-y, x)
  R: (0..m_Y, 0..m_X)





4.1. Interpolación y transf. básicas.

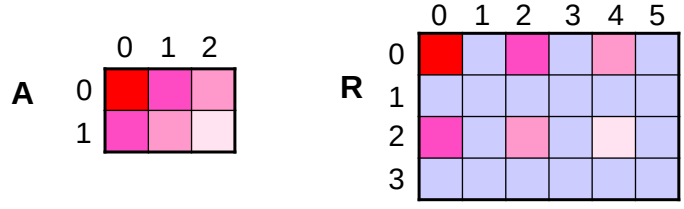
- En general la transformación tendrá la forma:
  R(x, y):= A(f_1(x,y), f_2(x,y))
- Siendo f_1 y f_2 dos funciones cualesquiera del tipo:
  f_1, f_2: N x N → R
  – **f_1**: posición en X del original para el píxel resultante (x,y)
  – **f_2**: posición en Y del original para el píxel resultante (x,y)
- **Ejemplo.** En la rotación de 180º
  f_1(x,y):= m_X-x f_2(x,y):= m_Y-y



4.1. Interpolación y transf. básicas.

- ... → **R** ¿Qué ocurre si el resultado es un número no entero?
- Por ejemplo, se puede conseguir un **aumento de 2x** con una transformación del tipo:

  R(x,y):= A(x/2, y/2), con R: $(0..2m_X+1, 0..2m_Y+1)$

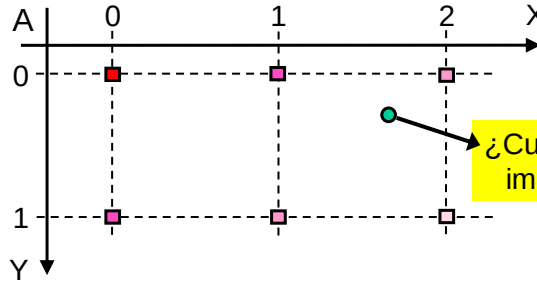


- R(0, 0):= A(0/2, 0/2) = A(0, 0) **OK**
- R(1, 0):= A(1/2, 0/2) = A(0.5, 0)
- R(1, 1):= A(1/2, 1/2) = A(0.5, 0.5)

Índices no definidos en el array. ¿Qué hacer ahí?

4.1. Interpolación y transf. básicas.

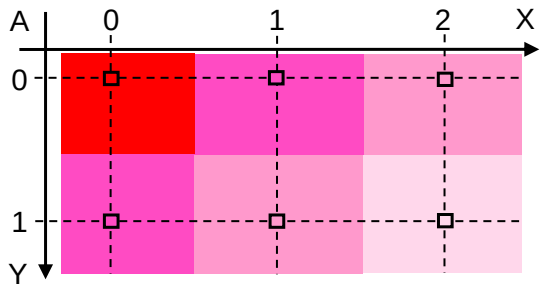
- **Problema:** las imágenes son **señales discretas**, pero la transformación geométrica las trata como si fueran **continuas** (definidas en todo el plano).



- **Solución:** aplicar una **interpolación**.
- **Tipos de interpolación:** vecino más próximo, bilineal, bicúbica, supermuestreo.

4.1. Interpolación y transf. básicas.

- **Interpolación: Vecino más próximo**
  Cualquier punto del espacio toma el valor del píxel más cercano.



- **Implementación:**

$$\left.\begin{array}{l} f_1(x,y) \rightarrow \lfloor f_1(x,y) + 0{,}5 \rfloor \\ f_2(x,y) \rightarrow \lfloor f_2(x,y) + 0{,}5 \rfloor \end{array}\right\} R(x,y) = A(\lfloor f_1(x,y) + 0{,}5 \rfloor, \lfloor f_2(x,y) + 0{,}5 \rfloor)$$

4.1. Interpolación y transf. básicas.

- **Ejemplo.** Zoom de 10x con vecino más próximo.
  $R(x,y) := A(\lfloor x/10 \rfloor, \lfloor y/10 \rfloor)$

Imagen original 25x26

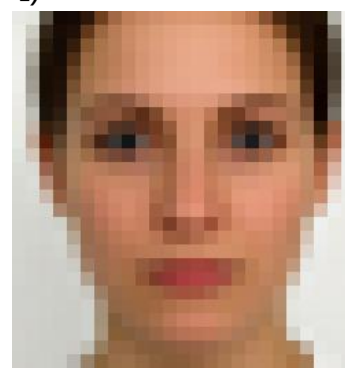
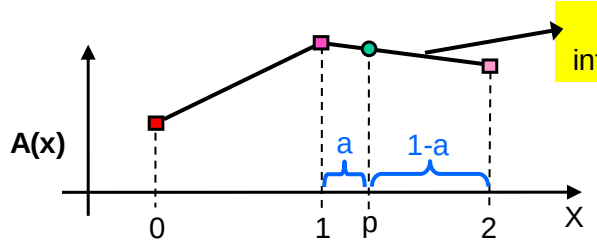


Imagen ampliada 250x260

- **Ventajas:**
  - Es muy sencilla y rápida de calcular.
- **Inconvenientes:**
  - El efecto de cuadriculado es evidente, y da lugar imágenes de poca calidad.

4.1. Interpolación y transf. básicas.

- **Interpolación bilineal**
- En una dimensión, una **interpolación lineal** significa trazar una línea recta entre cada par de puntos consecutivos.



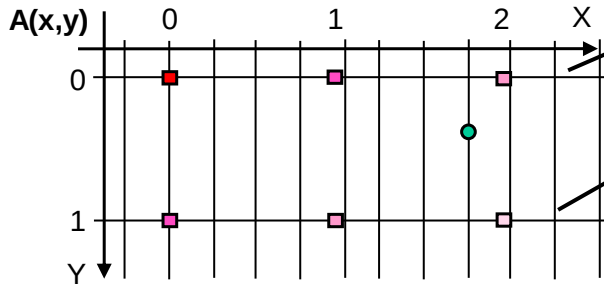
Valores interpolados

Media ponderada, de los valores de **i** y **d** según la distancia **a**

- **Cálculo de la interpolación lineal.** Sea **p** el punto que queremos interpolar.
- Sup. que **p** se encuentra entre **i** y **d**, es decir: **i**= $\lfloor p \rfloor$, **d**= i+1
- El valor interpolado en **p** será: A'(p):= (1-a)A(i) + aA(d) siendo **a**= p-i

4.1. Interpolación y transf. básicas.

- En dos dimensiones, la **interpolación bilineal** consiste en aplicar dos interpolaciones lineales:
  1. Interpolar la función **horizontalmente**, en las filas existentes.
  2. Interpolar la función **verticalmente** en todo el espacio (usando la anterior).



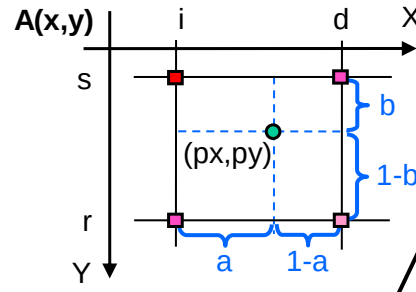
1º interpolar horizontalmt.

2º interpolar verticalmt.

- ¿Cómo calcular el valor interpolado de un punto (px, py), A'(px,py)?

4.1. Interpolación y transf. básicas.

- Sea **p**= (**px**, **py**), con **i**= ⌊px⌋, **d**= i+1, **s**= ⌊py⌋, **r**= s+1
  con **a**= px-i, **b**= py-s



- **Cálculo de la interpolación bilineal:**
  - A'(px, s) = (1-a)A(i, s) + aA(d, s)
  - A'(px, r) = (1-a)A(i, r) + aA(d, r)
  - A'(px, py) = (1-b)A'(px, s) + bA'(px, r)

  → **A'(px, py) = (1-a)(1-b)A(i, s) + a(1-b)A(d, s) + (1-a)bA(i, r) + abA(d, r)**

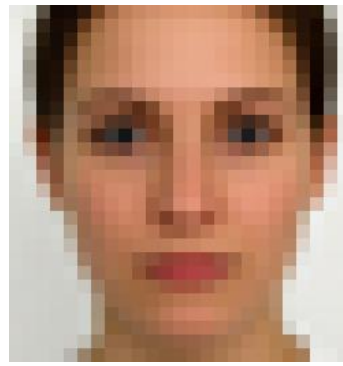
Media ponderada de los 4 píxeles circundantes

Esto recuerda a una convolución, ¿no?

4.1. Interpolación y transf. básicas.

- **Ejemplo.** Zoom de 10x con interpolación bilineal.
  R(x,y):= A'(x/10, y/10)

Imagen original 25x26



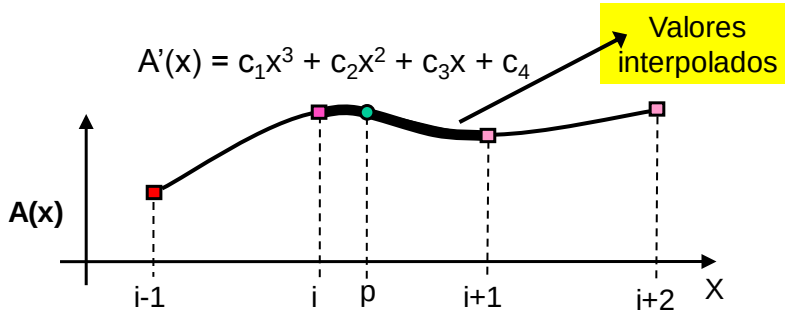
Vecino más próximo

Interpolación bilineal

- **Indicación:** un zoom entero de K con interpolación bilineal es parecido (= a veces) a un zoom de K con vecino más próximo, seguido de un filtro de media de KxK.

4.1. Interpolación y transf. básicas.

- La interpolación bilineal mejora la de vecino más próximo, pero produce un efecto de "zonas rectangulares".
- **Interpolación bicúbica:** basada en dos interpolaciones cúbicas.
- En una dimensión, la **interpolación cúbica** consiste en trazar una cúbica entre los 4 puntos más próximos (2 a la izquierda y 2 a la derecha).



4.1. Interpolación y transf. básicas.

- **Cálculo de la interpolación cúbica.**
  - Sea **p** el punto que queremos interpolar. **i**= ⌊p⌋
  - Obtener las 4 ecuaciones:
    A'(i-1)=A(i-1); A'(i)=A(i); A'(i+1)=A(i+1); A'(i+2)=A(i+2)
  - 4 ecuaciones, 4 incógnitas → despejar y obtener c_1, c_2, c_3, c_4
  - Aplicar las constantes, obteniendo A'(p)

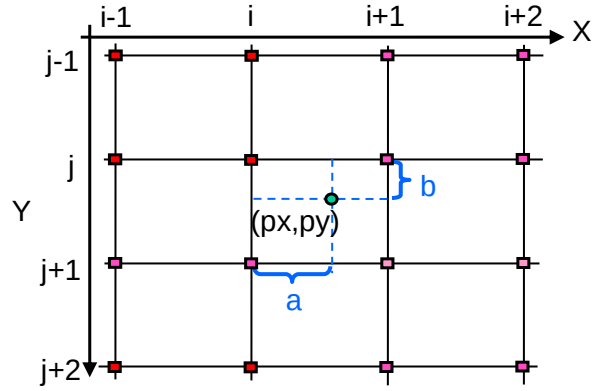
- **Interpolación bicúbica.** Igual que la bilineal, se basa en dos interpolaciones cúbicas:
  1. Interpolación cúbica **horizontal**, en las filas existentes (usando 4 puntos).
  2. Interpolación cúbica **vertical** en todo el espacio usando 4 puntos (usando la anterior interpolación).

  → En la interpolación bicúbica de un punto (px, py) intervienen los 16 puntos circundantes.

4.1. Interpolación y transf. básicas.

- **Cálculo de la interpolación bicúbica.**

- Igual que con la bilineal, el valor del punto se puede calcular como una **media ponderada** de los 4x4 píxeles circundantes.



$$A'(px,py) = \sum_{n=-1..2}\sum_{m=-1..2} A(i+n,j+m)\cdot P(n-a)\cdot P(b-m)$$

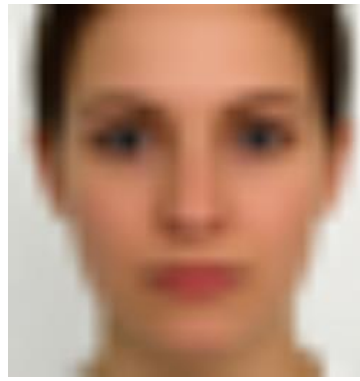
Siendo: $P(k) = 1/6(C(k+2)^3-4C(k+1)^3+6C(k)^3-4C(k-1)^3)$

$C(k) = \max(0, k)$

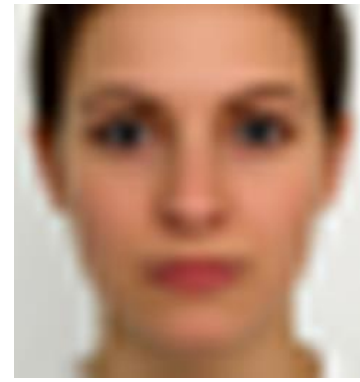
4.1. Interpolación y transf. básicas.

- **Ejemplo.** Zoom de 10x con interpolación bicúbica.

Imagen original 25x26



Aumento de 10x con interpolación bilineal

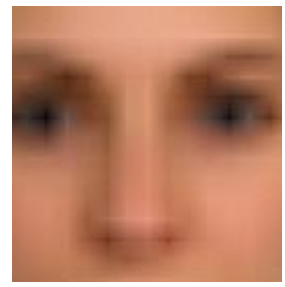


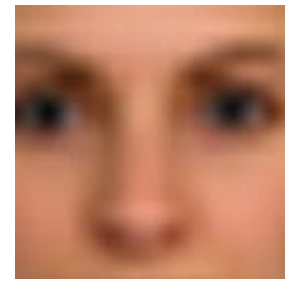
Aumento de 10x con interpolación bicúbica

4.1. Interpolación y transf. básicas.

- **Comparación.** Detalle del zoom de 10x, con vecino más próximo, interpolación bilineal y bicúbica. Se ha aplicado un perfilado en las 3, para destacar el efecto del zoom.


Vecino más próximo


Interpolación bilineal


Interpolación bicúbica

- En todos los casos se nota la falta de detalle (obviamente), pero en la bilineal son más evidentes los **artificios horizontales** y **verticales** que en la bicúbica.

4.1. Interpolación y transf. básicas.

- La interpolación también es importante en las **rotaciones** no exactas (que veremos más adelante) y, en general, en cualquier transformación geométrica.

Vecino más próximo | Interpolación bilineal | Interpolación bicúbica

- La interpolación bicúbica siempre suele producir el **mejor resultado**, aunque es algo **más costosa**.
- ¿Merece la pena el aumento de tiempo respecto a la bilineal?

4.1. Interpolación y transf. básicas.

- En las operaciones de **reducción** también se aplican interpolaciones, pero...
- **Reducción de k:** R(x, y):= A(k·x, k·y),
- **Ejemplo.** Reducción de 5x.

Observar estas estructuras extrañas. Esto tb. es **aliasing**

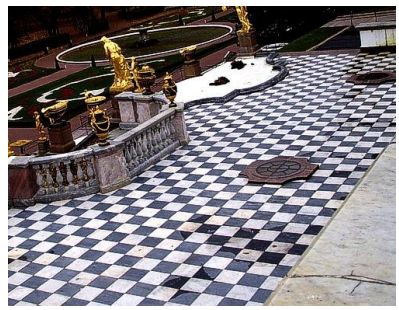


Imagen original 500x386

Reducción de 5x con vecino más próximo

4.1. Interpolación y transf. básicas.

- El problema no se soluciona con interpolación bilineal o bicúbica.

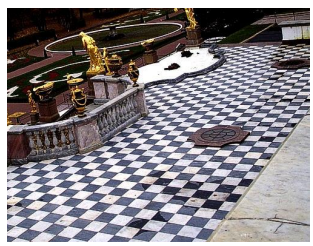


Imagen original 500x386

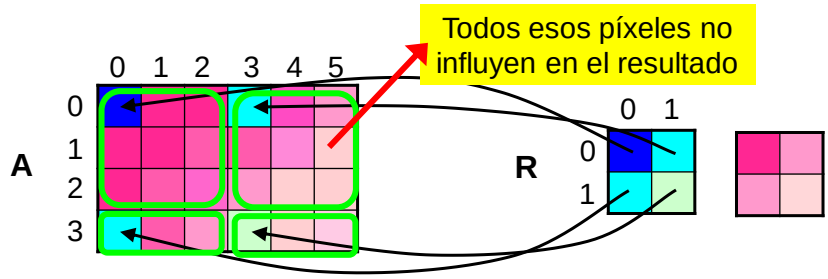
Reducción de 5x con interpolación bilineal

Reducción de 5x con interpolac. bicúbica

- El problema se debe a que los detalles son más pequeños que la **resolución** de salida. Pero, además, los métodos de interpolación no mejoran la situación: cada píxel de salida es un muestreo ordenado de uno de entrada.

4.1. Interpolación y transf. básicas.

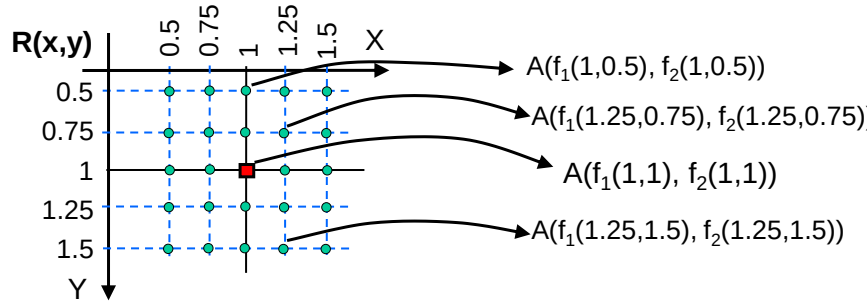
- **Ejemplo.** Reducción de 3x. R(x,y) = A(3x, 3y)



- **Solución:** cada píxel de salida debería ser la media de los 3x3 píxeles de entrada correspondientes.
- **Interpolación por supermuestreo (*super sampling*)**
  Idea: considerar el píxel como un "volumen" con cierto área. Aplicar varias veces la transformación y tomar la media.

4.1. Interpolación y transf. básicas.

- **Supermuestreo uniforme**, en rejilla cuadrada.



R(x,y):= media {$A(f_1(x-0.5,y-0.5), f_2(x-0.5,y-0.5))$, ...
$A(f_1(x-0.5,y+0.5), f_2(x-0.5,y+0.5))$, ..., $A(f_1(x,y-0.5), f_2(x,y-0.5))$, ..., $A(f_1(x,y+0.5), f_2(x,y+0.5))$, ..., $A(f_1(x+0.5,y-0.5), f_2(x+0.5,y-0.5))$, ..., $A(f_1(x+0.5,y+0.5), f_2(x+0.5,y+0.5))$}

4.1. Interpolación y transf. básicas.

- **Ejemplo.** Reducción de 5x, con supermuestreo.

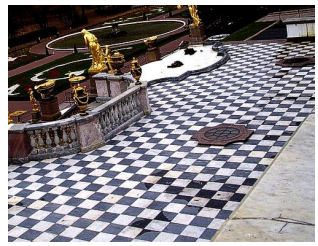


Imagen original 500x386

Reducción de 5x con interpolación bilineal

Reducción de 5x con supermuestreo

- **Resultado:** el supermuestreo logra un resultado de mucha más calidad. Evita el problema del aliasing.
- Sin embargo, el supermuestreo es mucho más costoso, requiere más cálculos.
- Cuanto mayor reducción, mayor es el efecto del aliasing.

4.1. Interpolación y transf. básicas.

- Una **alternativa** al supermuestreo es aplicar primero un filtro de suavizado (por ejemplo, de media) y después un simple vecino más próximo.

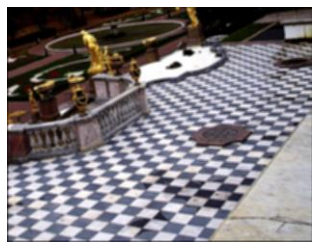


Imagen suavizada con media de 5x5

Reducción de 5x con vecino más próximo, de la suavizada

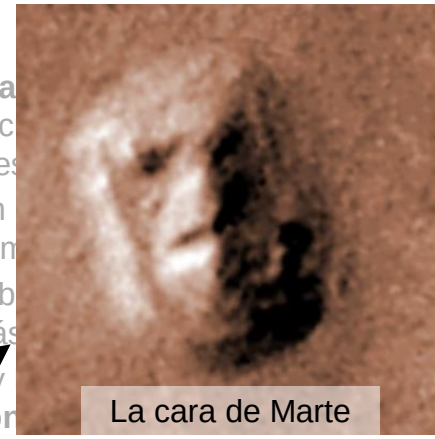
Reducción de 5x con supermuestreo, de la original

- Pero esto sólo es aplicable en las transformaciones que impliquen una reducción de resolución.

4.1. Interpolación y transf. básicas.

Conclusiones

- **Transformación geométrica** depende de uno de entrada c acuerdo a un par de funciones
- Como las posiciones pueden aplicar interpolaciones: vec. m
- En **zoom**, funciona mejor la b supermuestreo. Pero son más
- Las interpolaciones bilineal y avanzadas) dan la **sensación** la imagen, pero cuidado...
- Cualquier detalle aparente de resolución inferior a un píxel es una mera **alucinación**.



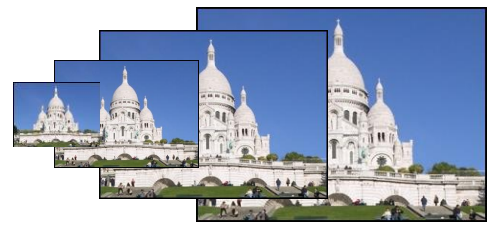
La cara de Marte

4.2. Transformaciones afines.

- Las **transformaciones afines** son cualquiera de los cuatro tipos siguientes, o combinaciones de las mismas.

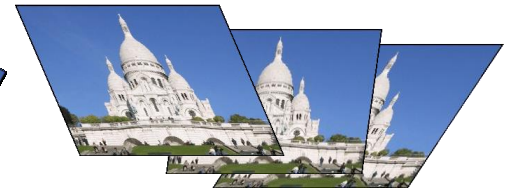
Traslación

Escala



Rotación

Inclinación



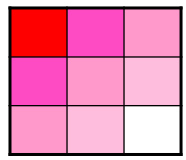
4.2. Transformaciones afines.

- **Transformación de traslación (desplazamiento):**

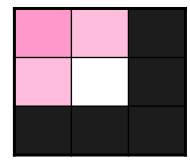
  $R(x,y) := A(x+d_x, y+d_y)$

- **Ejemplos.**

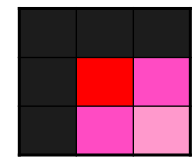
A



Traslación (1, 1)
$R_1(x,y) := A(x+1,y+1)$

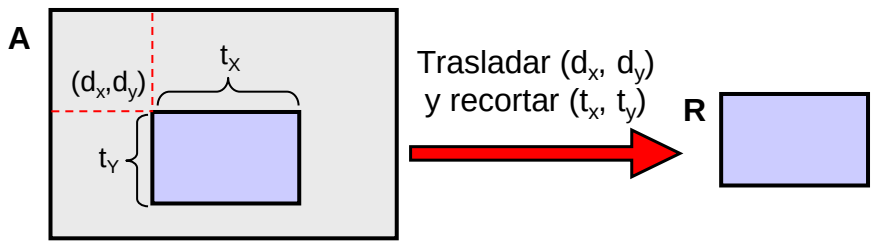


Traslación (-1, -1)
$R_2(x,y) := A(x-1,y-1)$



- Si las imágenes A y R tienen el mismo tamaño, algunos píxeles **caen fuera** de la imagen A. ¿Qué valor toman?
- Más que una interpolación sería una **extrapolación**.
- Se puede usar un valor **constante** predefinido. O **no modificar** el contenido previo.

4.2. Transformaciones afines.



- **Aplicaciones:**
  - **Seleccionar** y **recortar** una región rectangular.
  - Aunque, como ya hemos visto, no suele aparecer de forma explícita, sino al manejar ROI.
  - También suele aparecer en combinación con las otras operaciones, para centrar la imagen resultante.
  - **Ejemplo.** Recordar la operación de rotación de 180º:

    $R(x, y) := A(m_X-x, m_Y-y)$

4.2. Transformaciones afines.

- **Transformación de escala:**

  $R(x,y) := A(e_x \cdot x, e_y \cdot y)$

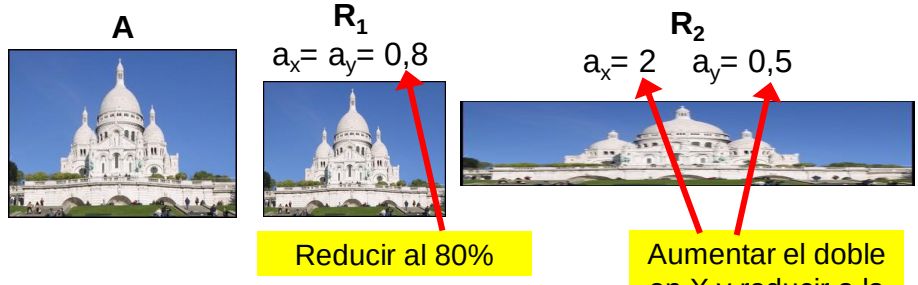
  - **e_x** = escala en el eje X
  - **e_y** = escala en el eje Y
- Normalmente será igual en ambos ejes ($e_x = e_y$), aunque puede ser distinta.
- Pero es más intuitivo el concepto de **aumento** o **zoom**:

  $R(x,y) := A(x/a_x, y/a_y)$

  - **a_x** = aumento en el eje X = $1/e_x$
  - **a_y** = aumento en el eje Y = $1/e_y$
- Si la imagen **A** es de tamaño m_X x m_Y, la imagen resultante **R** será de $m_X \cdot a_x$ x $m_Y \cdot a_y$

4.2. Transformaciones afines.

- **a_x, a_y** mayor que 1 → **Aumento** o **zoom** de la imagen → Aplicar **interpolación bilineal** o **bicúbica**.
- **a_x, a_y** menor que 1 → **Reducción** (*decimate*) de la imagen → Aplicar **supermuestreo** o **suavizado** previo.
- **Ejemplos.** Transformaciones de escala.

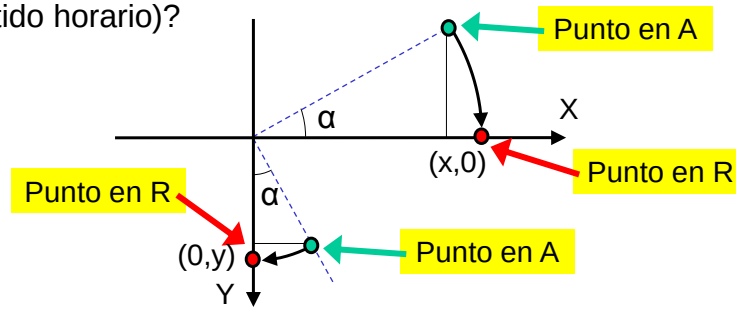


- El aumento sirve para hacer **zoom digital**, pero recordar sus limitaciones.

4.2. Transformaciones afines.

Transformación de rotación:

- Hemos visto las **rotaciones exactas** (90º, 180º, 270º), pero ¿cómo realizar una rotación de un **ángulo cualquiera α** (en sentido horario)?



- El punto (x, 0) en R corresponde en A a (x·cos α, -x·sen α)
- El punto (0, y) en R " en A a (y·sen α, y·cos α)
- (x, y) en R → (x·cos α + y·sen α, -x·sen α + y·cos α) en A

4.2. Transformaciones afines.

- **Rotación** de una imagen A en un **ángulo α**:

  R(x,y):= A(x·cos α + y·sen α, -x·sen α + y·cos α)

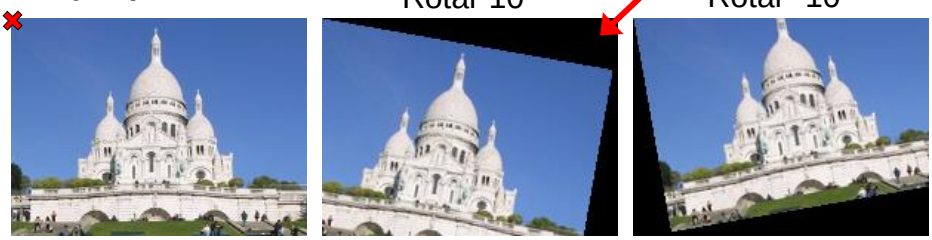
- La rotación se suele expresar **matricialmente**:

$$R(x,y) := A\left(\begin{bmatrix} \cos\alpha & \text{sen}\,\alpha \\ -\text{sen}\,\alpha & \cos\alpha \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}\right)$$

Lo que cae fuera de la imagen no se modifica

- **Ejemplos.**

Rotar 10º | Rotar -10º



- **Ojo**: estas rotaciones son respecto al punto (0,0). ¿Cómo hacerlas respecto a un centro arbitrario (c_x,c_y)?

4.2. Transformaciones afines.

- **Rotación** de A en un **ángulo α**, respecto a **un centro** (c_x,c_y)
- **Idea**: es equivalente a una rotación respecto a (0,0), seguida de un desplazamiento (d_x, d_y) adecuado. ¿Cuánto?
- El centro no se debe modificar: $R(c_x,c_y):= A(c_x,c_y)$

  $R(c_x,c_y):= A(c_x \cdot \cos\alpha + c_y \cdot \text{sen}\,\alpha + d_x,\ -c_x \cdot \text{sen}\,\alpha + c_y \cdot \cos\alpha + d_y)$
- **Solución**:

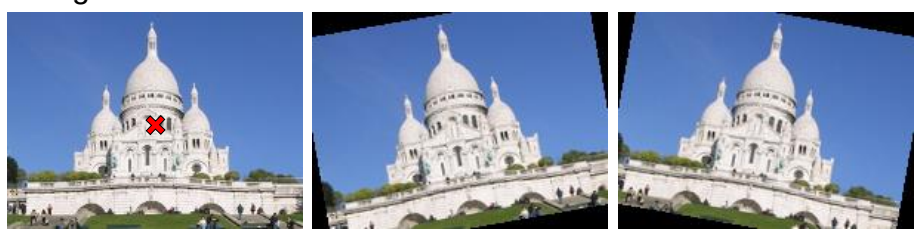
  $d_x = c_x - c_x \cdot \cos\alpha - c_y \cdot \text{sen}\,\alpha$; $d_y = c_y + c_x \cdot \text{sen}\,\alpha - c_y \cdot \cos\alpha$
- Matricialmente, la **rotación α con centro (c_x,c_y)** sería:

$$R(x,y):= A\left(\begin{array}{|c|c|c|}\hline \cos\alpha & \text{sen}\,\alpha & c_x - c_x\cdot\cos\alpha - c_y\cdot\text{sen}\,\alpha \\ \hline -\text{sen}\,\alpha & \cos\alpha & c_y + c_x\cdot\text{sen}\,\alpha - c_y\cdot\cos\alpha \\ \hline \end{array} \cdot \begin{array}{|c|}\hline x \\ \hline y \\ \hline 1 \\ \hline \end{array}\right)$$

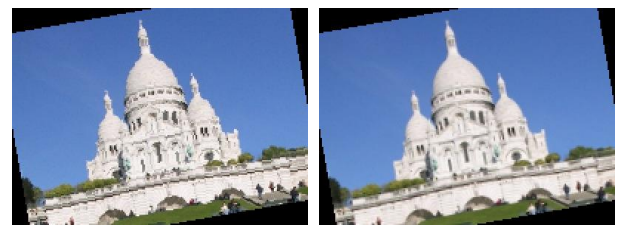
4.2. Transformaciones afines.

- **Ejemplos**. Rotaciones respecto a un centro cualquiera.

Imagen de entrada | Rotar -10º | Rotar 10º



- Se ha utilizado **interpolación bicúbica**. Recordar la importancia de la interpolación.



Vecino más próximo | Interpolación bilineal

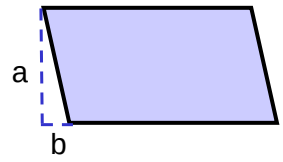
4.2. Transformaciones afines.

Transformación de inclinación (*shear*):

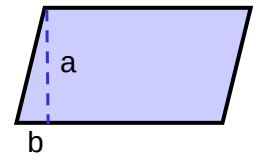
- La inclinación transforma una región rectangular en un rombo. Sirve para "simular" una perspectiva.
- **Posibilidades**: inclinación en X, en Y o en ambos ejes.

Imagen de entrada

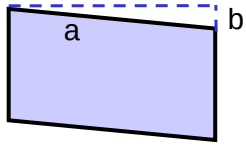
Inclinación en X de b/a



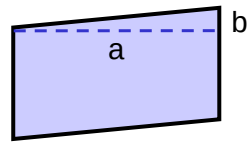
Inclinación en X de -b/a



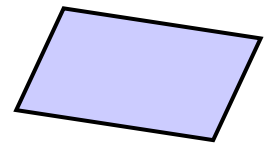
Inclinación en Y de b/a



Inclinación en Y de -b/a



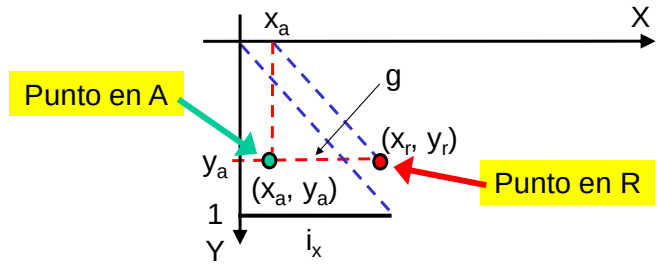
Inclinación en X e Y



- El valor de inclinación es la tangente del ángulo.

4.2. Transformaciones afines.

- ¿Cómo obtener la transformación de inclinación?
- **Ejemplo**. Inclinación en X de cantidad i_x.



- $(x_a, y_a) = (x_r\text{-}g, y_r)$. Pero, ¿cuánto vale **g**?
- Por **analogía** de **triángulos**: $g/y_r = i_x/1 \rightarrow g = i_x \cdot y_r$
- **Inclinación en X en cantidad i_x:**

  $R(x, y) := A(x - i_x \cdot y, y)$

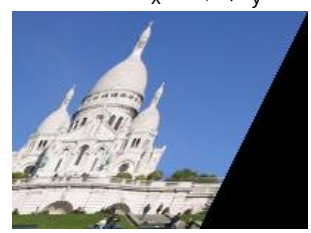
4.2. Transformaciones afines.

- De manera similar... **inclinación en Y en cantidad i_y:**
  $R(x, y) := A(x, y - i_y \cdot x)$
- **Inclinación X en i_x, Y en i_y:**
  $R(x, y) := A(x - i_x \cdot y, y - i_y \cdot x)$
- **Matricialmente**:

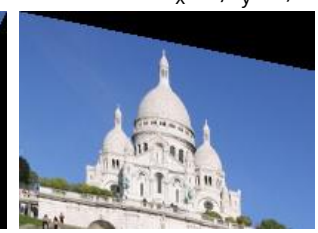
$$R(x,y) := A\left(\begin{bmatrix} 1 & -i_x \\ -i_y & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}\right)$$

- **Ejemplos.**

Inclinación i_x=-0,4; i_y=0



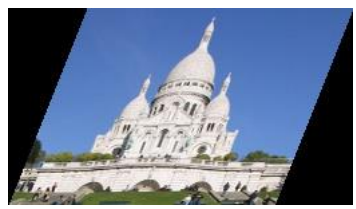
Inclinación i_x=0; i_y=0,2



4.2. Transformaciones afines.

- Parte de la imagen se sale.
- **Solución**: igual que antes, aplicar un **desplazamiento** para **centrar** el resultado. ¿De cuánto?
- O bien, **ampliar** el tamaño de la imagen resultado para que quepa toda la imagen.
- ¿Cuánto? → En X: $abs(i_x) \cdot m_Y$; en Y: $abs(i_y) \cdot m_X$
- **Ejemplos.**

Inclinación i_x=-0,4



Inclinación i_y=0,2

Inclinación i_x=0,1;i_y=0,3

4.2. Transformaciones afines.

- ¿Qué tienen en común todas las transformaciones afines?
- Todas ellas se pueden expresar de **forma matricial**:

$$R(x,y) := A\left(\begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}\right)$$

Traslación

$$\begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \end{bmatrix}$$

Escala

$$\begin{bmatrix} e_x & 0 & 0 \\ 0 & e_y & 0 \end{bmatrix}$$

Rotación

$$\begin{bmatrix} \cos\alpha & \text{sen}\,\alpha & 0 \\ -\text{sen}\,\alpha & \cos\alpha & 0 \end{bmatrix}$$

Inclinación

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

- Es más, cualquier **combinación de transformaciones afines** es una transformación afín, y se puede expresar de forma matricial.

4.2. Transformaciones afines.

- Podemos definir una **transformación afín genérica**:

$$R(x,y) := A\left(\begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}\right)$$

- La matriz $((c_{11},c_{12},c_{13}),(c_{21},c_{22},c_{23}))$ son los **parámetros de la transformación**. Puede implicar escalas, traslaciones, rotaciones, etc.
- ¿Cómo calcular la tr. afín equivalente a dos tr. afines?
- Usar el **producto matricial**, añadiendo una fila con (0, 0, 1).

$$\begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} d_{11} & d_{12} & d_{13} \\ d_{21} & d_{22} & d_{23} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

4.2. Transformaciones afines.

- **Ejemplo**. Sobre una imagen aplicamos 1º) traslación (a, b), 2º) escala (c, d), 3º) rotación e, y 4º) traslación (f, g).
- Podemos calcular la matriz de transformación **equivalente**:

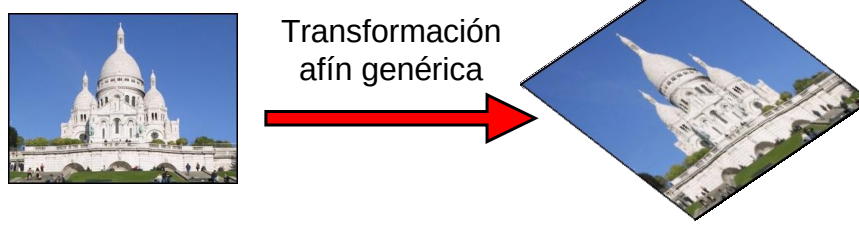
1	0	f	x	cos e	sen e	0	x	c	0	0	x	1	0	a
0	1	g		-sen e	cos e	0		0	d	0		0	1	b
0	0	1		0	0	1		0	0	1		0	0	1

=	c·cos e	d·sen e	a·c·cos e+f+b·d·sen e
	-c·sen e	d·cos e	b·d·cos e+g-a·c·sen e
	0	0	1

- En lugar de aplicar 4 transformaciones sobre toda la imagen, basta con aplicar la transformación equivalente.

4.2. Transformaciones afines.

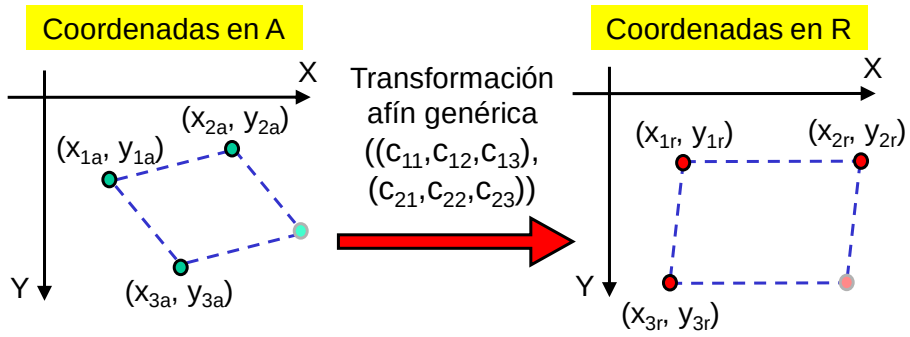
- **Uso**. Una transformación afín permite "mapear" una región rectangular cualquiera en un rombo cualquiera. O, en general, cualquier rombo en otro rombo (se supone que un rectángulo es también un rombo).



- ¿Cómo calcular los parámetros de esta transformación?
- **Problema**: dado un rombo en la imagen original y otro rombo en la imagen de destino, calcular la transformación afín que realiza ese mapeo.

4.2. Transformaciones afines.

- **Nota**: un rombo queda completamente definido por 3 puntos.



- 6 incógnitas (c_{ij}) → Necesitamos **6 ecuaciones** para resolverlas. ¿Cuáles?
- Cada punto produce dos ecuaciones:

$$\begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix} \cdot \begin{bmatrix} x_{ia} \\ y_{ia} \\ 1 \end{bmatrix} = \begin{bmatrix} x_{ir} \\ y_{ir} \end{bmatrix}$$

4.2. Transformaciones afines.

- **Ecuaciones a resolver:**

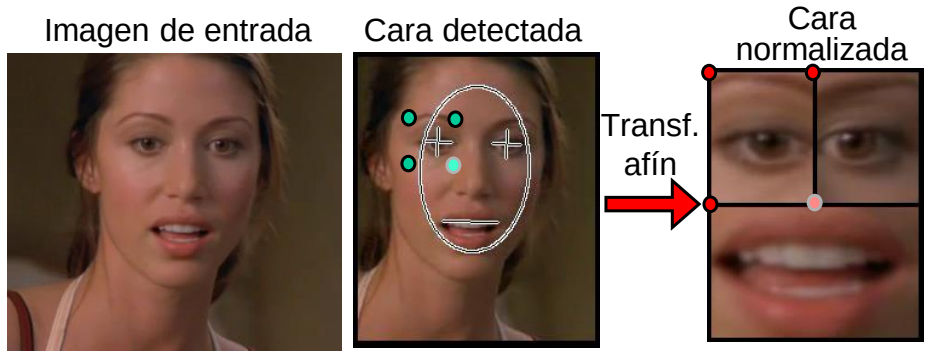
$$c_{11}x_{1a} + c_{12}y_{1a} + c_{13} = x_{1r} \qquad c_{21}x_{1a} + c_{22}y_{1a} + c_{23} = y_{1r}$$
$$c_{11}x_{2a} + c_{12}y_{2a} + c_{13} = x_{2r} \qquad c_{21}x_{2a} + c_{22}y_{2a} + c_{23} = y_{2r}$$
$$c_{11}x_{3a} + c_{12}y_{3a} + c_{13} = x_{2r} \qquad c_{21}x_{3a} + c_{22}y_{3a} + c_{23} = y_{3r}$$

- 6 ecuaciones y 6 incógnitas → Se **resuelven las incógnitas** (método de Cramer) y despachados.
- Existirá solución si los 3 puntos de A no están en la misma recta, y los 3 de R tampoco.
- Una vez resueltas las incógnitas, aplicamos la transformación afín buscada:

$$R(x,y) := A\left(\begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \right)$$

4.2. Transformaciones afines.

- **Ejemplo 1.** La aplicación más inmediata y típica de las transformaciones afines es extraer y redimensionar un área de interés, dándole una forma predefinida de antemano. Esto es lo que se llama **normalización**.
- Por ejemplo, **detectar una cara** humana, seleccionar los ojos y la boca y mapearlos a un rectángulo predefinido.



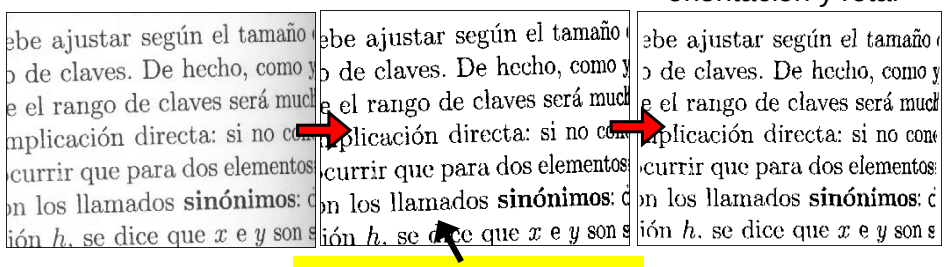
4.2. Transformaciones afines.

- Este proceso de normalización se puede aplicar sobre **vídeo**, para conseguir una **estabilización** de los objetos de interés.

4.2. Transformaciones afines.

- **Ejemplo 2.** La **normalización** es fundamental en muchas aplicaciones de reconocimiento de objetos, como los OCR (*Optical Character Recognition*).
- Se aplica: umbralización, segmentación, normalización y comparación.

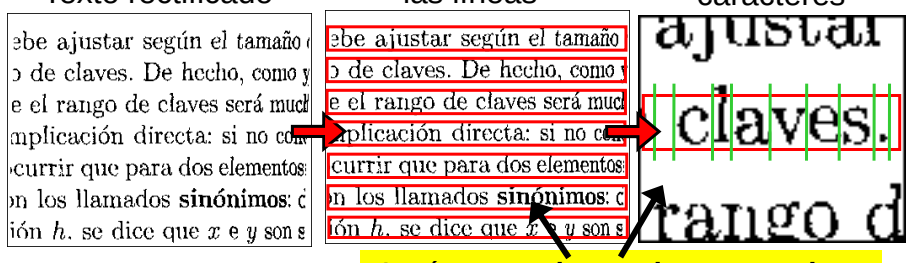
Imagen de entrada → **Umbralizar** → **Rectificar**: detectar orientación y rotar



Sería conveniente una técnica **adaptativa**

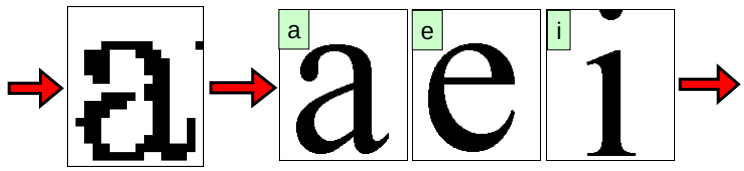
4.2. Transformaciones afines.

Texto rectificado → **Detectar** y separar las líneas → **Segmentar** los caracteres



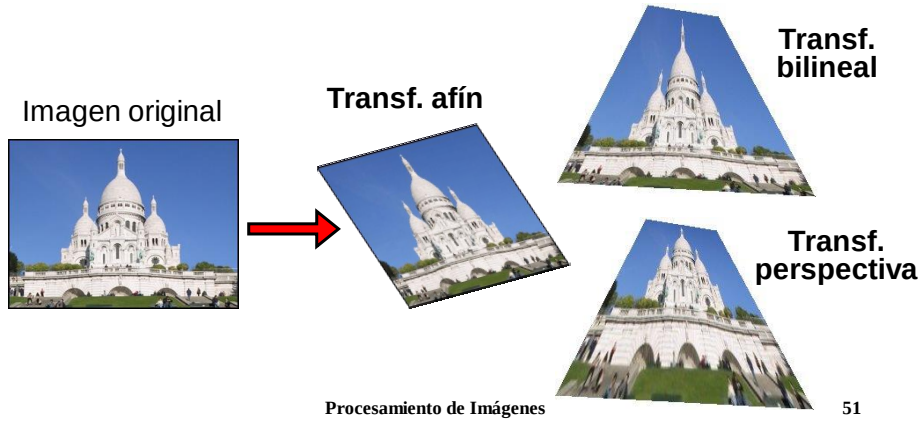
Aquí se usan **integrales proyectivas**

Normalizar cada carácter a un tamaño estándar → **Comparar** con un conjunto de patrones (p.ej. diferencia) → Tomar el **máximo**



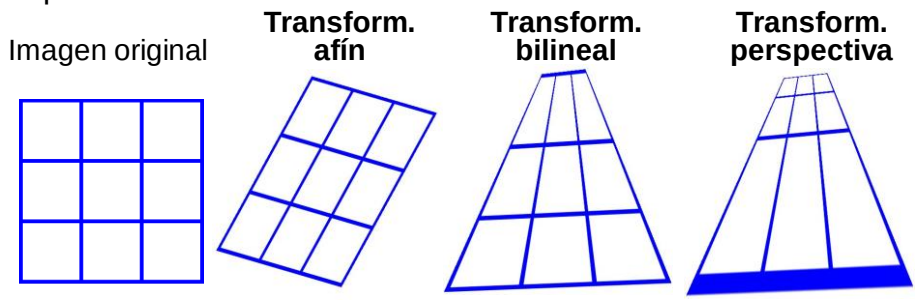
4.3. Transformación bilineal y perspectiva.

- Las transformaciones **bilineal** y **perspectiva** se pueden ver como **generalizaciones** de las afines:
  - Transformación **afín**: cualquier rombo se mapea en un rombo.
  - Transf. **bilineal** y **perspectiva**: cualquier cuadrilátero se transforma en otro cuadrilátero (ambos convexos).



4.3. Transformación bilineal y perspectiva.

- Las transformaciones afines conservan el **paralelismo** de las líneas; bilineales y perspectivas no.
- La transf. perspectiva es la **proyección perspectiva** de un plano, colocado en un espacio 3D.
- La transf. bilineal se suele usar como una **variante rápida** de la transf. perspectiva, aunque no es exactamente igual. Es decir, se debería aplicar perspectiva, pero se usa bilineal por eficiencia.



4.3. Transformación bilineal y perspectiva.

- **Transformaciones afines:**

$$R(x,y) := A\left(\begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}\right)$$

- **Transformaciones bilineales:**

$$R(x,y) := A\left(\begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ x \cdot y \\ 1 \end{bmatrix}\right)$$

Observar el nuevo factor que aparece

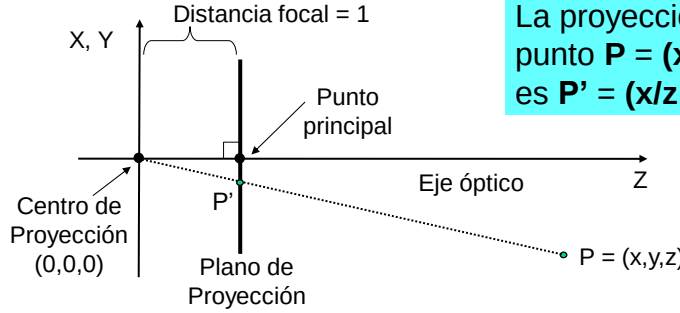
Pero, ¿cuántos grados de libertad hay aquí?

- **Transformaciones perspectivas:**

$$R(x,y) := A(x'/z',\ y'/z') \qquad \text{con:} \quad \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

4.3. Transformación bilineal y perspectiva.

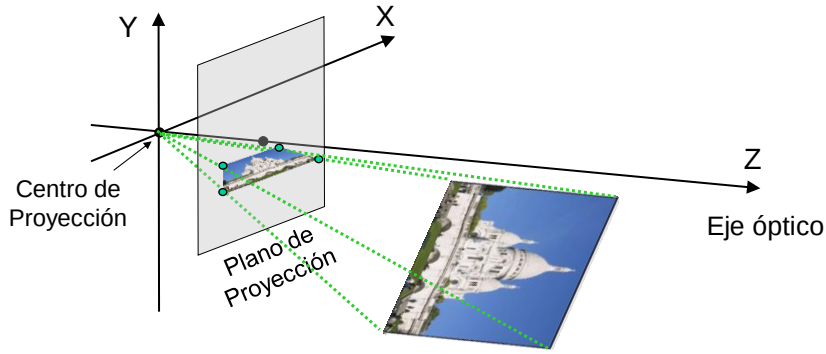
- Recordar la **proyección perspectiva** en el proceso de formación de imágenes.



La proyección del punto **P** = **(x, y, z)** es **P'** = **(x/z, y/z)**

- La idea de la **transf. perspectiva** es: dado un plano (la imagen de entrada) colocarlo en una posición cualquiera del espacio 3D y después proyectarlo sobre el plano de imagen Z=1.

4.3. Transformación bilineal y perspectiva.



1) Colocar la imagen plana en el espacio 3D

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2) Proyección perspectiva de la imagen en el espacio

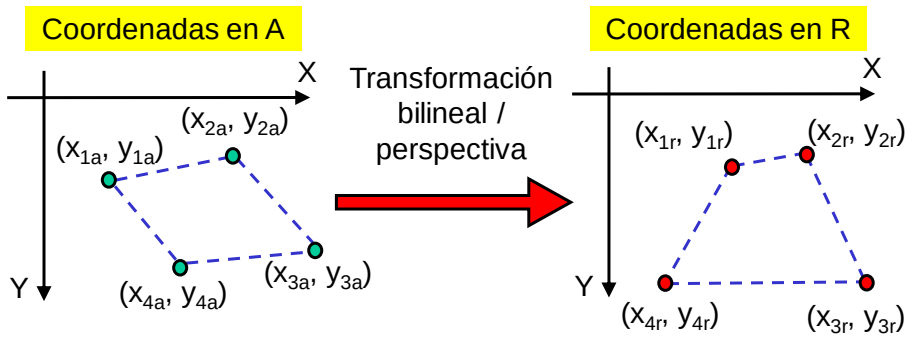
$$R(x,y) := A(x'/z', y'/z')$$

4.3. Transformación bilineal y perspectiva.

- La transf. bilineal es una **simulación** de la perspectiva. También mapea un rectángulo en un cuadrilátero.
- Pero el resultado no es exactamente una perspectiva. La diferencia es mayor cuanto mayor efecto de perspectiva.

4.3. Transformación bilineal y perspectiva.

- **Problema**: dados 4 puntos en la imagen original y otros 4 en la imagen de destino, calcular las transformaciones bilineal y perspectiva que producen ese mapeo.
- **Solución**: plantear los sistemas de ecuaciones correspondientes y resolver las incógnitas.



4.3. Transformación bilineal y perspectiva.

- **Transformación bilineal:** 8 incógnitas. Cada par de puntos equivalentes produce 2 ecuaciones → con 4 puntos es necesario y suficiente.

$$R(x,y) := A\left(\begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ x \cdot y \\ 1 \end{bmatrix}\right)$$

- **Ecuaciones a resolver:**

$$c_{11}x_{1a}+c_{12}y_{1a}+c_{13}x_{1a}y_{1a}+c_{14}=x_{1r};\quad c_{21}x_{1a}+c_{22}y_{1a}+c_{23}x_{1a}y_{1a}+c_{24}= y_{1r}$$

$$\cdots\cdots\cdots \qquad \cdots\cdots\cdots$$

$$c_{11}x_{4a}+c_{12}y_{4a}+c_{13}x_{4a}y_{4a}+c_{14}=x_{4r};\quad c_{21}x_{4a}+c_{22}y_{4a}+c_{23}x_{4a}y_{4a}+c_{24}= y_{4r}$$

- Para que haya solución, los cuadriláteros deben ser **convexos** y no deben haber tres puntos en la misma recta.

4.3. Transformación bilineal y perspectiva.

- **Transformación perspectiva:** 9 incógnitas, 4 puntos... ¿?

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$R(x,y) := A(x'/z', y'/z')$

- Cada par de puntos (x_{ia},y_{ia}), (x_{ir},y_{ir}) produce dos ecuaciones:
  $(c_{11}x_{ia}+c_{12}y_{ia}+c_{13})/(c_{31}x_{ia}+c_{32}y_{ia}+c_{33}) = x_{ir}$
  $(c_{21}x_{ia}+c_{22}y_{ia}+c_{23})/(c_{31}x_{ia}+c_{32}y_{ia}+c_{33}) = y_{ir}$
- Sistema **homogéneo** e **indeterminado** (8 ec., 9 inc.).
- Observar que aparece un factor de escala. Si multiplicamos todas las constantes por k el sistema no cambia.
- Se puede resolverlo "fijando" la incógnita $c_{33}=1$. Nos quedamos con 8 incógnitas, resolvemos y listos.

4.3. Transformación bilineal y perspectiva.

Indicaciones:

- Las transformaciones bilineales y perspectivas "**contienen**" a las afines:
  - **Transf. bilineal**: será equivalente a una afín si $c_{13}=c_{23}=0$
  - **Transf. perspectiva**: equivalente a una afín si $c_{31}=c_{32}=0$
- Los tres tipos de transformaciones son **invertibles**: dada una transf. se puede definir la transf. inversa, de manera que se obtenga la imagen original (o casi).

4.3. Transformación bilineal y perspectiva.

Transformaciones inversas:

- ¿Cómo obtener las transformaciones inversas?
- Si una **transf. perspectiva** está definida por una matriz A (3x3), la **transf. perspectiva inversa** usará la matriz A^{-1}.
- En el caso de la **transf. afín**, se puede extender la matriz de (3x2) a una de (3x3) y obtener la inversa.

c_{11}	c_{12}	c_{13}
c_{21}	c_{22}	c_{23}

→

c_{11}	c_{12}	c_{13}
c_{21}	c_{22}	c_{23}
0	0	1

→ Calcular la inversa de la matriz

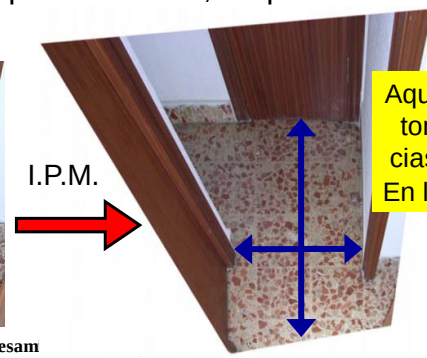
- **Ejercicio**. Calcular la **inversa** de una **transf. bilineal** dada por una matriz de coeficientes, de (4x2).

4.3. Transformación bilineal y perspectiva.

- **Ejemplo 1**. La "invertibilidad" de la proyección perspectiva puede ser útil en **navegación de robots**.
- **Idea:** dada una imagen tomada con una cámara desde el robot, obtener una **vista superior**. De esta forma, el robot puede conocer las zonas por las que se puede mover.
- **Inverse Perspective Mapping**: transformación inversa a la perspectiva producida por la cámara, respecto a cierto plano.

Imagen de entrada

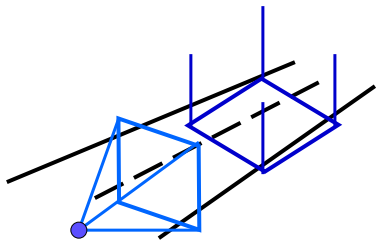
I.P.M.



Aquí se pueden tomar distancias y ángulos. En la original no

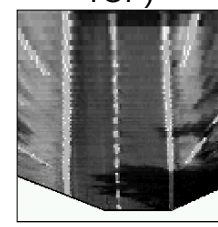
4.3. Transformación bilineal y perspectiva.

- La técnica de I.P.M. se ha aplicado en **conducción automática de vehículos**.



Imag. capturada desde el coche

I.P.M. de la imag. capturada (vista TOP)



http://www.argo.ce.unipr.it/ARGO/

El proyecto es un poco antiguo, pero bueno...

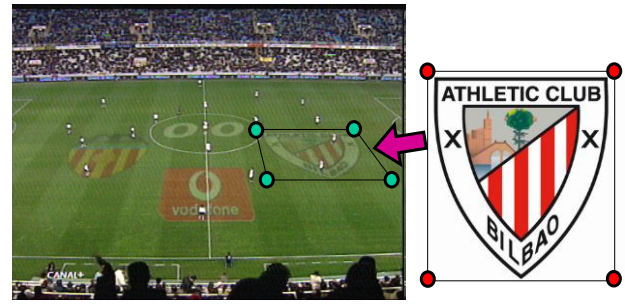
4.3. Transformación bilineal y perspectiva.

- **Ejemplo 2**. Integración de **elementos visuales artificiales** en un entorno. El objetivo es hacer que algo que no está parezca que realmente está.
- **Problema**: qué transformación perspectiva se debe aplicar para que la integración sea realista. → **Calibración**.
- **Pasos:**

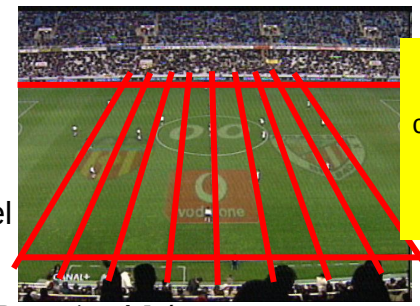
1) Detectar el suelo (color verde).
2) Transformación perspectiva del elemento.
3) Media ponderada entre el suelo y la imagen transformada.

4.3. Transformación bilineal y perspectiva.

- ¿Cómo hacer la **calibración**?
- **Manualmente**: seleccionar el cuadrilátero en la imagen donde se proyectará el elemento.



- **Automáticamente**. Más complejo. P.ej. encontrando las líneas del campo, las blancas y las del corte del césped.



Una vez **localizado** el campo, colocar los elementos en unos sitios predefinidos

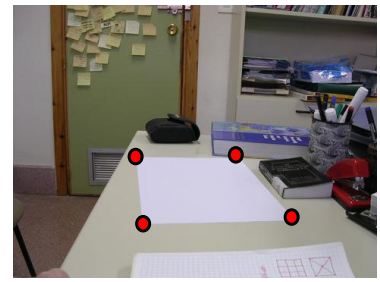
4.3. Transformación bilineal y perspectiva.

- **Ejemplo 3**. **Engañando** a la perspectiva. Usando mapeo inverso de perspectiva, se puede hacer que un dibujo (real) visto en perspectiva parezca tener otra perspectiva, y por tanto ocupe un espacio que realmente no ocupa.
- Este efecto visual sólo funciona desde un punto de vista dado. Está diseñado para ese punto.
- **Idea**: el dibujo real a poner es el I.P.M. del original, según la transf. perspectiva observada en la escena.

4.3. Transformación bilineal y perspectiva.

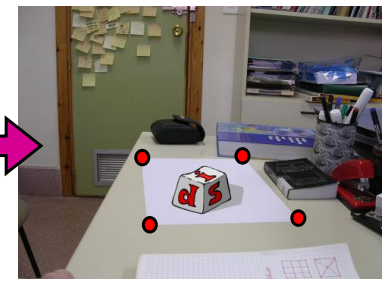
- **Pasos:**

1) Poner el panel donde irá el dibujo en la escena, y **capturar** una **imagen** desde el punto deseado.



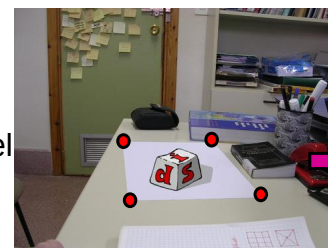
2) **Calibrar**: encontrar los 4 puntos del rectángulo del panel.

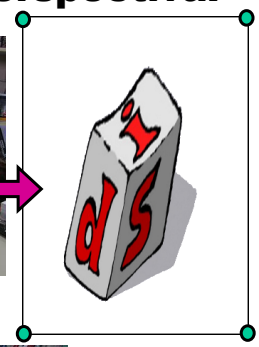
3) **Añadir** sobre la imagen tomada el dibujo que se quiere poner (con la escala y traslación adecuadas).



4.3. Transformación bilineal y perspectiva.

4) Sobre la imagen del paso 3, aplicar una transformación **perspectiva**, desde el cuadrilátero del paso 2, hasta el rectángulo dado por el tamaño del panel.





5) **Imprimir** el panel y colocarlo en el mismo sitio.

6) Y... Voilà!!

4.3. Transformación bilineal y perspectiva.

- **Ejemplo 4**. Transformaciones de **malla**. Es una transformación libre y que "da mucho juego". Sobre la imagen original definimos una malla de **puntos de control**. Estos puntos se pueden mover, y la imagen resultande debe modificarse coherentemente con los puntos de control.

Imagen original

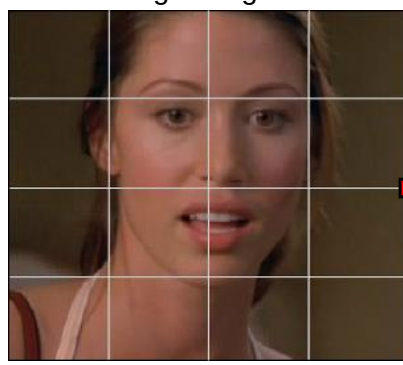
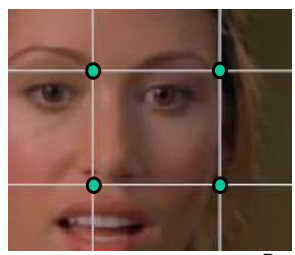


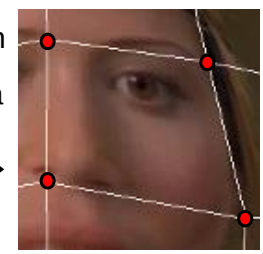
Imagen transformada

4.3. Transformación bilineal y perspectiva.

- ¿Cómo conseguir la transformación de malla?
- La **idea** es muy sencilla:
  - La malla define una **serie de rectángulos** de posiciones conocidas.
  - Después de **mover los puntos** de la malla, conocemos el cuadrilátero al que se debe mapear cada rectángulo original.
  - En este caso, suele ser más conveniente aplicar una **transformación bilineal**.
  - Se **repite** el proceso para todos los rectángulos originales.



Transformación bilineal específica para cada trozo



4.3. Transformación bilineal y perspectiva.

Conclusiones:

- Las transformaciones afines, bilineales y perspectivas son esenciales en generación, procesamiento, análisis de imágenes, y en visión artificial.
  - **Transformaciones afines**: mapean un rombo en otro rombo. 3 puntos en la imagen origen y 3 en el destino.
  - **Transf. bilineales y perspectivas**: mapean un cuadrilátero convexo en otro cuadrilátero convexo. 4 puntos en el origen y 4 en el destino.
- Las transformaciones afines conservan el **paralelismo** de las rectas.
- Hay que conocer el **significado** de cada transformación para saber cuál conviene aplicar.
- ¿Cómo se podría extender la idea a vídeo, considerando la escala temporal?

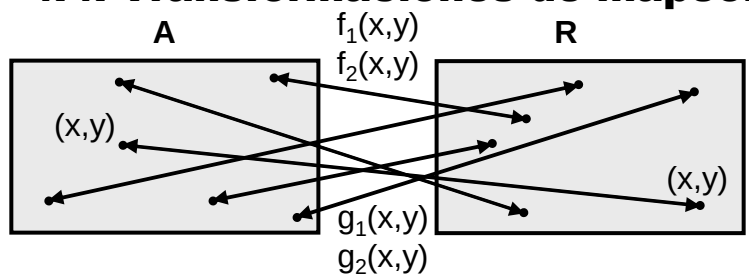
4.4. Transformaciones de mapeo.

- Recordemos que una **transformación geométrica** es cualquier operación del tipo:

  $R(x, y) := A(f_1(x,y), f_2(x,y))$
- Siendo f_1 y f_2 dos funciones cualesquiera:

  $f_1, f_2: N \times N \rightarrow R$
  - **f_1**: posición en X del original para el píxel resultante (x,y)
  - **f_2**: posición en Y del original para el píxel resultante (x,y)
- Las transformaciones vistas hasta ahora tienen formas particulares y son continuas.
- Un **mapeo** (*mapping*) es cualquier transformación arbitraria, definida por un par de funciones f_1 y f_2, continuas o escalonadas.

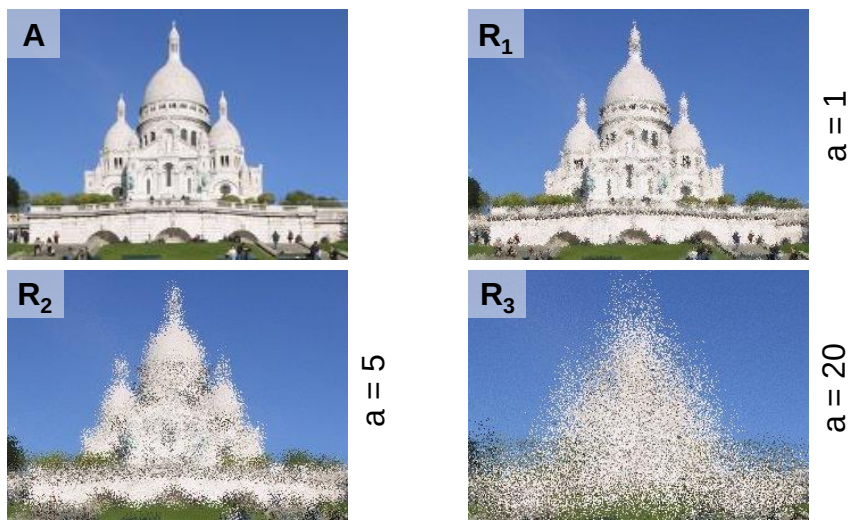
4.4. Transformaciones de mapeo.



- **Mapeo inverso**: el mapeo puede venir dado al revés: $R(g_1(x,y), g_2(x,y)) := A(x,y)$
- **Significado**: el píxel (x,y) en la imagen original se mueve a la posición $(g_1(x,y), g_2(x,y))$.
- Normalmente trabajaremos con mapeo directo.
- Existen infinitos mapeos. Cualquier par de funciones locas, (f_1, f_2), es posible, pero ¿cuáles son plausibles?

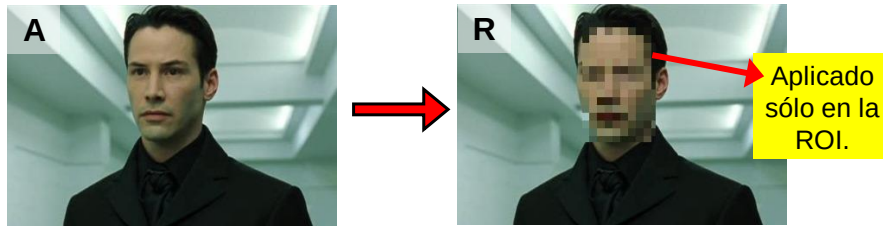
4.4. Transformaciones de mapeo.

- **Ejemplo 1**. **Difuminado aleatorio**, de radio **a**:
  $f_1(x,y) := x+random(2a+1)-a$ $f_2(x,y) := y+random(2a+1)-a$



4.4. Transformaciones de mapeo.

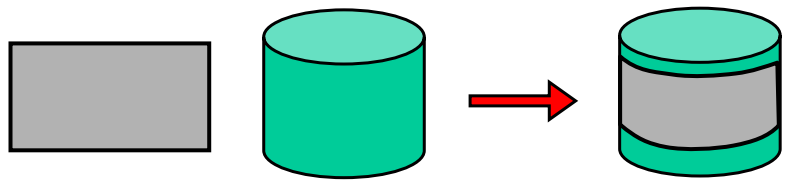
- **Ejemplo 2**. **Pixelado**: $f_1(x,y):=\lfloor x/8 \rfloor *8$; $f_2(x,y):=\lfloor y/8 \rfloor *8$



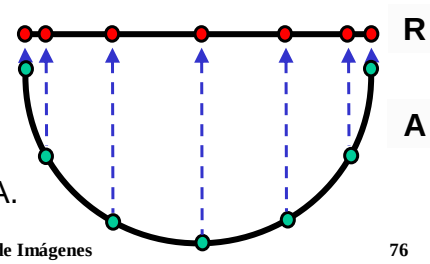
- **Ejemplo 3**. Efecto de **cristal** a cuadros:
  $f_1(x,y):= x–x \bmod 30+y \bmod 30$; $f_2(x,y):= y–y \bmod 30+x \bmod 30$

4.4. Transformaciones de mapeo.

- Los mapeos pueden servir para **simular** las deformaciones producidas por **fenómenos físicos naturales**.
- Por ejemplo, ¿cómo se deforma una imagen pegada a un cilindro (como una etiqueta de una botella)?



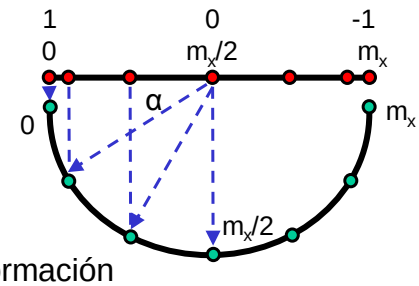
- La coordenada Y no se modifica: $f_2(x, y):= y$
- ¿Qué pasa con la X?
- La X de R es el coseno del ángulo correspondiente en A.



4.4. Transformaciones de mapeo.

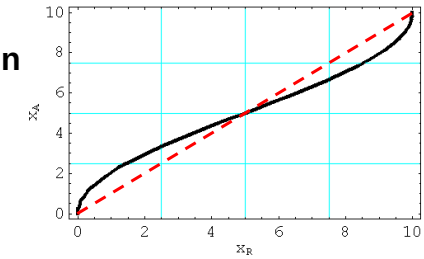
$\alpha = \arcsin(1-x_R/(m_x/2))$

$x_A = \alpha \cdot m_x/\pi$



- **Conclusión.** Transformación **cilíndrica** en X: $R(x,y) := A(\arcsin(1-x/(m_x/2)) \cdot m_x/\pi,\ y)$

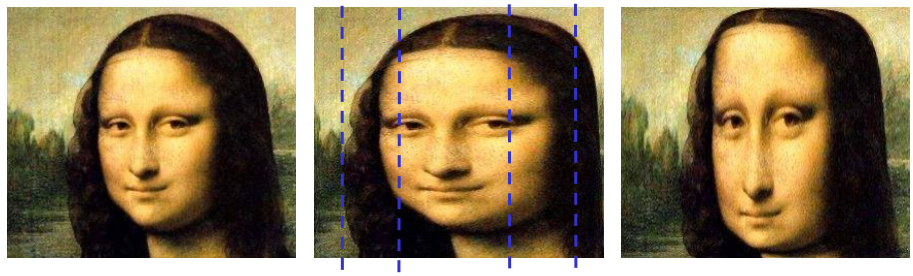
Representación de la función
$f(x) := \arcsin(1-x/(m_x/2)) \cdot m_x/\pi$
con $m_x=10$



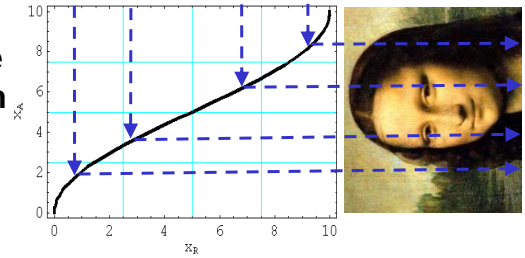
4.4. Transformaciones de mapeo.

- **Ejemplo.** Aplicación de la transformación cilíndrica.

Imagen de entrada | Tr. cilíndrica en X | Tr. cilíndrica en Y

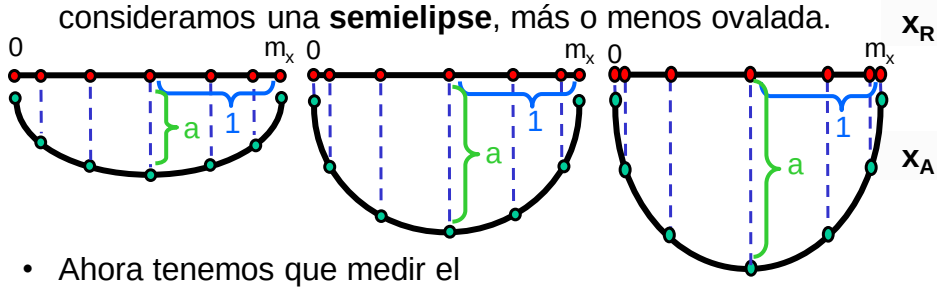


Interpretación de la transformación cilíndrica



4.4. Transformaciones de mapeo.

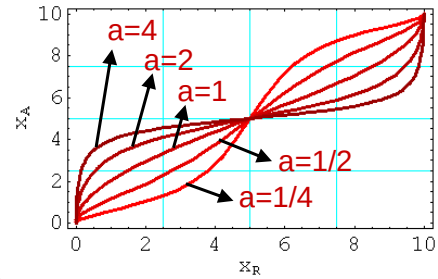
- El efecto se puede **graduar**, si en lugar de un semicírculo consideramos una **semielipse**, más o menos ovalada.



- Ahora tenemos que medir el **ángulo en una elipse**. Si tomamos $x' = 1-x_R/(m_x/2)$, ent.:

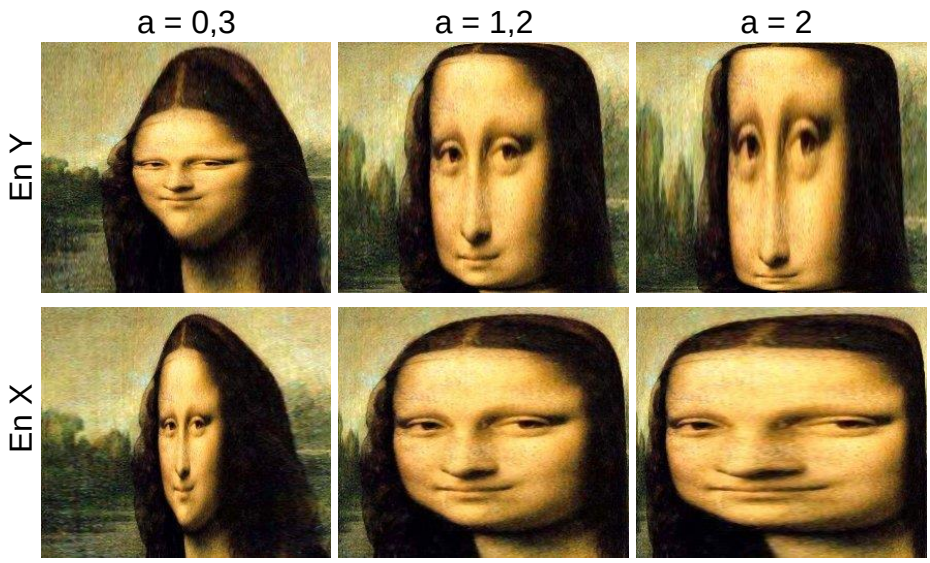
$f_1(x, y) := \text{atan}(a \cdot \text{sqrt}(1-x'^2)/x')m_x/\pi$

siendo **a** el segundo radio de la elipse (en relación al ancho de la imagen).



4.4. Transformaciones de mapeo.

- **Ejemplos**. Transformaciones elípticas.



4.4. Transformaciones de mapeo.

- De forma parecida, podemos definir otros muchos tipos de transformaciones, basados en **deformaciones** producidas por **fenómenos físicos naturales** (o no).
- **Método**: estudiar la forma (matemática) de la deformación, y obtener el par de funciones $f_1(x,y)$, $f_2(x,y)$.
- **Ejemplo**. Transformaciones geométricas genéricas.

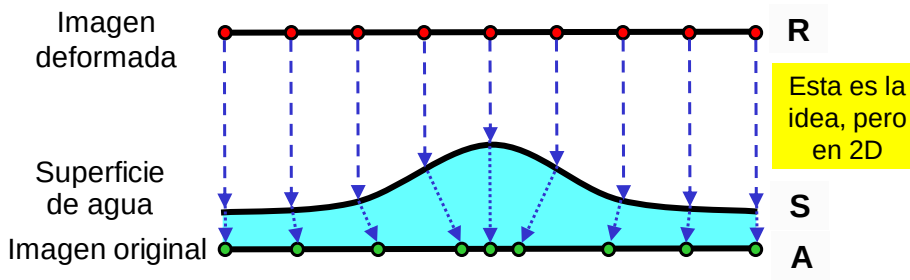
Estirar: simula un panel abombado hacia afuera

Pinchar: simula apretar la superficie del panel

Ondulación: simula una deformación por ondas de agua

4.4. Transformaciones de mapeo.

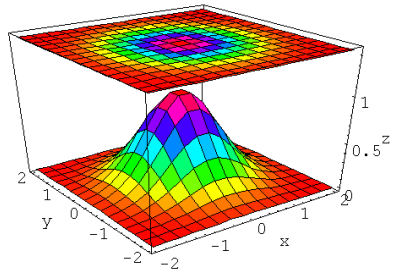
- ¿Cómo **unificar** todas estas deformaciones? Es decir, crear un **marco común** a todas ellas.
- **Idea**: supongamos que sobre una imagen podemos colocar una capa de agua. Podemos poner gotas, hacer ondas, formar olas, etc.
- La deformación de la imagen es el resultado de la **refracción** de la luz, al pasar del aire al agua.



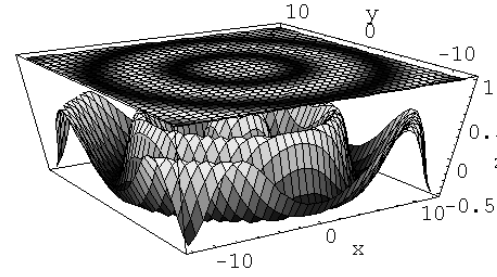
4.4. Transformaciones de mapeo.

- Ojo, la **superficie deformante**, **S**, no es ni más ni menos que una imagen, donde el valor de un píxel **S(x,y)** indica la altura del agua en ese punto.

S_1. Superficie de efecto pinchar/estirar



S_2. Superficie de efecto de ondas marinas



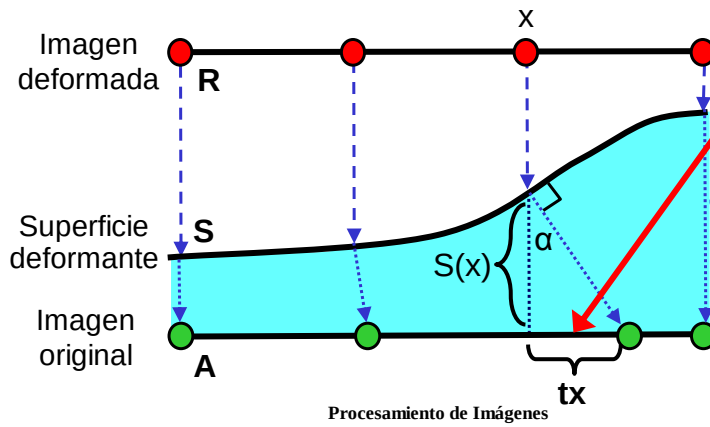
$S(x,y):= e^{-((x-cx)^2+(y-cy)^2)/\sigma^2}$

$S(x,y):= sen(f \cdot sqrt((x-cx)^2+(y-cy)^2))$

Ahora sólo hay que calcular los rayos incidentes, el ángulo de refracción de cada uno, y el sitio donde choca con el fondo...

4.4. Transformaciones de mapeo.

- **Objetivo**: dada una imagen **A**, una superficie deformante **S** (imagen de 1 solo canal) definir la transformación geométrica correspondiente: $R(x,y):= A(f_1(x,y,S), f_2(x,y,S))$
- **En una dimensión**: en el caso de mayor refracción, el rayo se desvía perpendicularmente a la superficie del agua.



¡Esto es lo que nos interesa! ¿Cuánto vale?

$tx/S(x) = \tan(\alpha)$

Pero $\tan(\alpha)$ es la pendiente de S en x → $tx = S(x) \cdot S'(x)$

4.4. Transformaciones de mapeo.

- **¡Ya está!** → $R(x) := A(x + S(x) \cdot dS(x)/dx)$
- Y **en dos dimensiones**:
  $R(x,y) := A(x + S(x,y) \cdot dS(x,y)/dx , y + S(x,y) \cdot dS(x,y)/dy)$
- Ale, ¡todos a derivar!

  $S(x,y) \cdot dS(s,y)/dx = e^{-((x-cx)^2+(y-cy)^2)/\sigma^2} \cdot d(e^{-((x-cx)^2+(y-cy)^2)/\sigma^2})/dx =$
  $= -2(x-cx)/\sigma^2 \cdot e^{-2((x-cx)^2+(y-cy)^2)/\sigma^2}$ ....

- Y así para cualquier función...

Cachis... ¿no habrá una forma más sencilla?

- Pero, ¿qué vimos en el tema anterior?
- La **derivada en X** (o en Y) de una imagen se puede calcular con un filtro de **convolución** adecuado: Sobel, Prewitt, Scharr, etc.
- Además, de esta forma podemos usar cualquier superficie deformante arbitraria.

4.4. Transformaciones de mapeo.

- **Algoritmo**. Transformación de una imagen **A** según la superficie deformante **S**.

1) Calcular Gx:= Sx ⊗ S
   siendo Sx una máscara de derivada en X
2) Calcular Gy:= Sy ⊗ S
   siendo Sy una máscara de derivada en Y
3) Calcular MapaX(x,y):= x + a·S(x,y)·Gx(x,y)
4) Calcular MapaY(x,y):= y + a·S(x,y)·Gy(x,y)
5) Obtener la imagen resultante:
   R(x,y):= A(MapaX(x,y), MapaY(x,y))

Sx

-1	0	1
-2	0	2
-1	0	1

Sy

-1	-2	-1
0	0	0
1	2	1

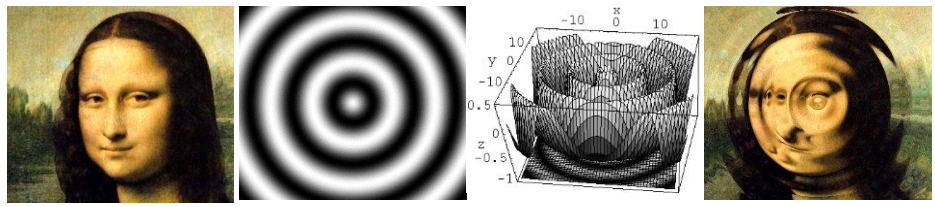
Notas: todas las imágenes (A, S, Gx, Gy, MapaX, MapaY, R) son del mismo tamaño.

El parámetro **a** indica el grado de la transformación. Cuanto mayor, más pronunciada.

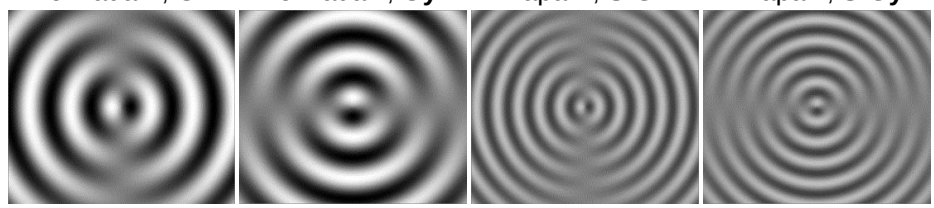
4.4. Transformaciones de mapeo.

- **Ejemplo**. Aplicación de la deformación de ondas.

Im. de entrada, **A** | Sup. deformante, **S** | Superficie en 3D | Im. de salida, **R**



Derivada X, **Gx** | Derivada Y, **Gy** | Mapa X, **S·Gx** | Mapa Y, **S·Gy**



Ojo: -1=Negro, +1=Blanco

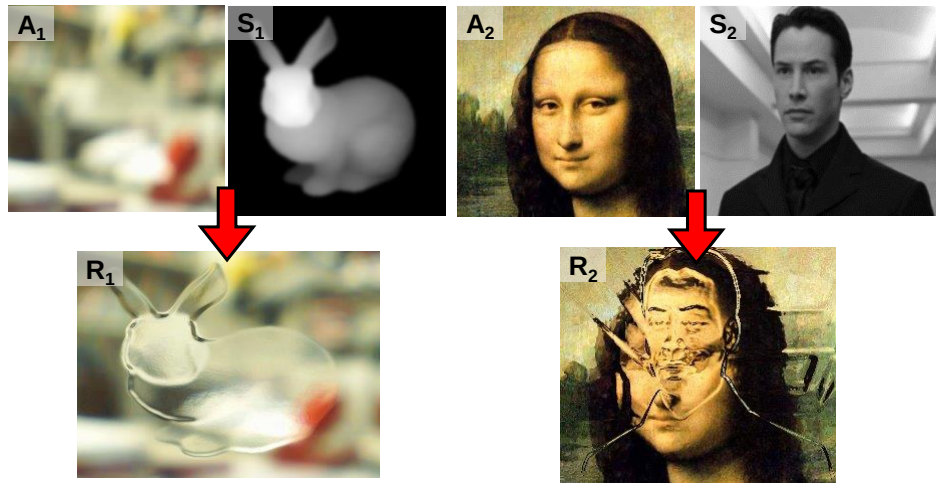
4.4. Transformaciones de mapeo.

- **Transformación de ondas**:
  $S(x,y) := \text{sen}(f \cdot \text{sqrt}((x-cx)^2+(y-cy)^2)+p)$
  - (**cx**, **cy**): centro de las ondas
  - **f**: frecuencia; **p**: fase
- **Transformación apretar/pinchar**:
  $S(x,y) := e^{-((x-cx)^2+(y-cy)^2)/\sigma^2}$
  - (**cx**, **cy**): centro de la deformación
  - **σ**: anchura de la zona deformada
  - **a**: fuerza de la deformación; **a**<0 → pinchar, **a**>0 → estirar

Variación de frecuencia | Variación de fase | Variación de fuerza

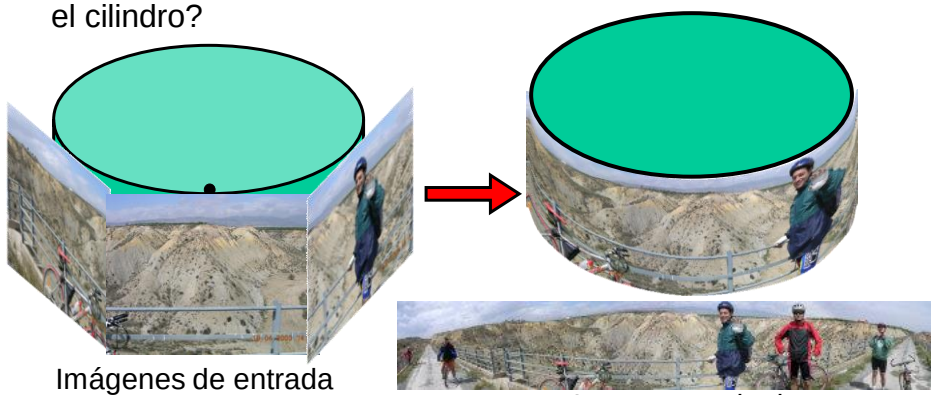
4.4. Transformaciones de mapeo.

- Lo interesante de esta transformación es que se puede usar **cualquier imagen** como superficie deformante.



4.4. Transformaciones de mapeo.

- Las transformaciones geométricas son esenciales en las **composiciones panorámicas**.
- **Idea**: obtener la imagen que debería estar pegada a un cilindro que envuelve todo el campo visual del sujeto.
- **Cuestión**: ¿cómo se proyectan las imágenes individuales en el cilindro?

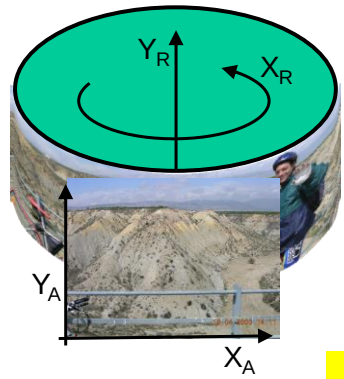


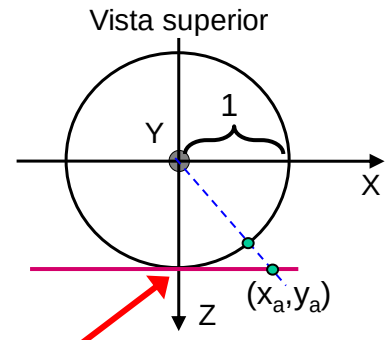
Imágenes de entrada

Imagen resultado

4.4. Transformaciones de mapeo.

- **Idea:** la X en la panorámica es el ángulo en el cilindro.
- La proyección de un punto (x_a,y_a) de la imagen viene dada por la intersección de la recta que pasa por $(x_a,y_a,1)$ y (0,0,0), y el cilindro con radio 1, a lo largo del eje Y.





¿Dónde toca la imagen con el cilindro? Centro: (x_c,y_c)

4.4. Transformaciones de mapeo.

- Si todas las fotos se toman desde el centro, sin mover la cámara (sólo girarla en Y), el punto (x_c,y_c) será el centro de la imagen $(m_X/2, m_Y/2)$.
- Si hay giro arriba o abajo, sí que se modifica y_c.
- Y también puede haber giro a lo largo de Z.
- Otro parámetro es cuántos grados corresponden al ancho de una foto, es decir cuánto es el **campo visual**. Lo podemos medir en el **número de píxeles** que representan 45º, **fp**.
- Con estos parámetros, la transformación será:

  $x_R := \arctan((x_A-x_c)/fp)$

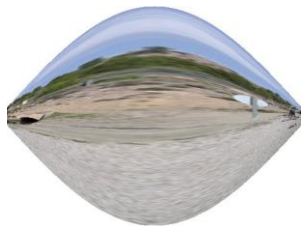
  $y_R := (y_A-y_c)/\mathrm{sqrt}((x_A-x_c)^2 + (y_A-y_c)^2 + fp))$
- x_R estará entre -90º y 90º, y_R entre $-m_Y/2$ y $m_Y/2$.

Ojo, esta **fp** no es ni más ni menos que la **distancia focal**

Podrá variar según el valor del zoom

4.4. Transformaciones de mapeo.

- Pero, cuidado, lo que necesitamos son las funciones $f_1(x,y)$ y $f_2(x,y)$ (de $R(x,y):= A(f_1(x,y), f_2(x,y))$ que vienen dadas por las **inversas**:
  $f_1(x,y):= x_c + fp \cdot \tan x$
  $f_2(x,y):= y_c + y \cdot \text{sqrt}(1+\tan^2 x)$
- En definitiva, tenemos una transformación con 3 parámetros:
  - El centro de la imagen (**cx**, **cy**).
  - La distancia focal, en píxeles, **fp**.
- **Ejemplo**. Variación de **df**, con centro ($m_x/2$, $m_y/2$), 480x360.



df = 20 pix.

df = 120 pix.

df = 530 pix.

4.4. Transformaciones de mapeo.

- La distancia focal **no cambia**, si no cambiamos el zoom entre una foto y otra. Se puede calibrar una vez y usarla en todas las fotos de la composición.

} Calibración previa

- El centro en X se puede dejar en la mitad de la imagen, si no cambiamos el eje de rotación.

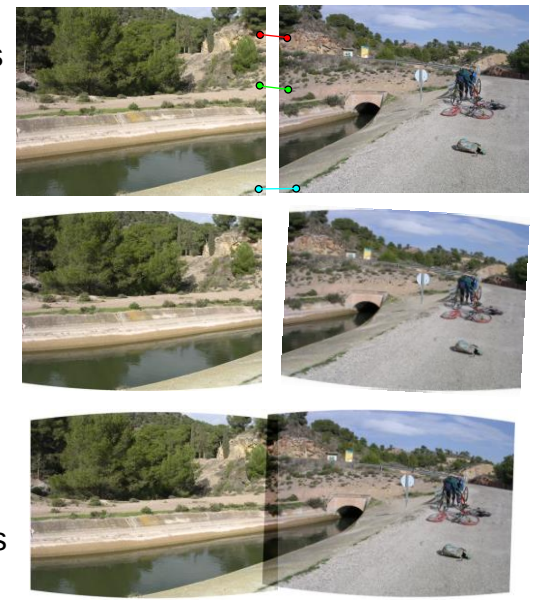
- El centro en Y, $\mathbf{c_y}$, puede cambiar.
- El giro en el eje Z también. Se puede corregir con una rotación afín a priori.
- Por último, se debe encontrar el desplazamiento en X para ajustar las imágenes.

} Calibración para cada grupo de imágenes

4.4. Transformaciones de mapeo.

- **Proceso**:

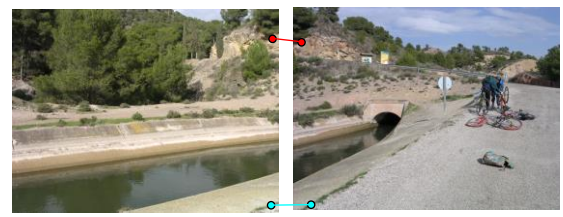
1) Buscar puntos análogos (de forma manual o automática) entre cada par de imágenes consecutivas.
2) Calcular los parámetros de la transformación, usando los puntos definidos.
3) Transformar las imágenes individualmente.
4) Componer las imágenes resultantes.



4.4. Transformaciones de mapeo.

Versión simplificada (aunque inexacta):

- Normalmente la distancia focal será grande y se puede sustituir la transf. anterior por una simple transf. afín.
- **Parámetros de la transf. afín**: desplazamiento (**dx**,**dy**), escala **s** (igual en ambos ejes) y rotación **r**. No hay inclinac.
- Con **dos puntos análogos** basta para resolver los 4 parámetros.



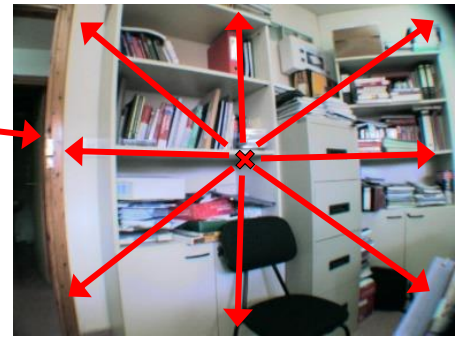
Es mucho más sencillo, pero peor. Observar la **línea quebrada** del canal

4.4. Transformaciones de mapeo.

Corrección de la distorsión radial

- La distorsión radial es una deformación introducida por las lentes de las cámaras, que da lugar a un *curvado* de las zonas periféricas de las imágenes.

Observar la curvatura de la puerta



- La distorsión radial se modela como un desplazamiento radial, según la distancia, **r**, al centro de la imagen de la forma:

$$p \cdot (1 + k_1 r^2 + k_2 r^4)$$

4.4. Transformaciones de mapeo.

- **Corrección de la distorsión radial**:

  $R(x,y) := A((x-cx)(1+k_1 r^2+k_2 r^4)+cx\ ,\ (y-cy)(1+k_1 r^2+k_2 r^4)+cy)$

  con $r^2 = (x-cx)^2 + (y-cy)^2$
- Para poder aplicarla, tenemos que encontrar valores adecuados de **k_1**, **k_2**, **cx** y **cy**. → **Calibración**.
- **Posibilidades**: hacer pruebas, o bien obtener 4 ecuaciones.

Imagen de entrada (384x288)

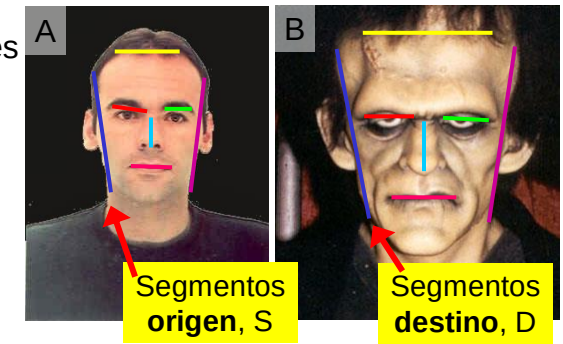
Imagen resultante con $k_1 = -2{,}9 \cdot 10^{-7}$, $k_2 = -1 \cdot 10^{-13}$

4.4. Transformaciones de mapeo.

- Otro tipo de transformaciones geométricas interesantes son las de **morphing**, muy usadas en efectos especiales.
- El **morphing** es un efecto de **transición suave** y **progresiva** entre dos (o más) imágenes.
- Está basado en algunas **ideas** ya estudiadas:
  - Una **transformación geométrica**, definida por un conjunto de puntos de origen, S, y otro de destino, D. El proceso es similar a la transformación de malla, pero en lugar de rectángulos se usan segmentos equivalentes.
  - La transf. geométrica está **graduada**, entre las posiciones de origen y de destino. 0%: posiciones de origen, 100%: posiciones de destino, 50%: término medio entre ambos.
  - El **color de un píxel** es una **media ponderada**, entre la imagen origen y destino, según el grado de la transformación.

4.4. Transformaciones de mapeo.

- **Morphing entre dos imágenes basado en líneas**.
- **Paso 1**. Establecer segmentos equivalentes entre dos imágenes.



- $S= (s_1, s_2, ..., s_n)$
- $D= (d_1, d_2, ..., d_n)$

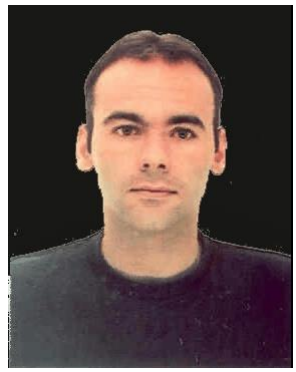
- **Paso 2**. **Repetir** el proceso para **g** desde 0 hasta 1. En el paso **g**, los segmentos intermedios son:

$$i_k= (1-g)s_k + g \cdot d_k$$

4.4. Transformaciones de mapeo.

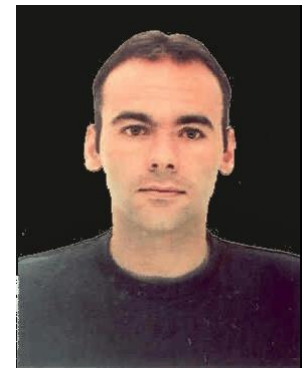
- **Morphing entre dos imágenes basado en líneas**.

2.1. Transformar **A**, moviendo los puntos de **S** a **I** → $\mathbf{R_1}$



2.2. Transformar **B**, moviendo los puntos de **D** a **I** → $\mathbf{R_2}$

2.3. Media ponderada: **R**= (1-g)R1 + g·R2



4.4. Transformaciones de mapeo.

Conclusiones:

- Las transformaciones de mapeo son el **caso general** de las transformaciones geométricas.
- Permiten modelar los efectos producidos por **fenómenos físicos naturales**.
  - Para simularlos (efecto de ondas, pinchar, estirar...).
  - Para corregirlos (distorsión radial, aberraciones en las lentes...).
- Y también otras cosas no naturales: **efectos especiales**.
- La transformación de mapeo está definida por un par de funciones $\mathbf{f_1}$, $\mathbf{f_2}$, (RxR → R) o bien un par de imágenes $\mathbf{mapa_x}$, $\mathbf{mapa_y}$.

4. Transformaciones geométricas.

Conclusiones:

- Existen muchos **tipos de transformaciones geométricas**, desde las más simples a las más complejas:
  - Afines, bilineales, perspectivas, basadas en superficies deformantes, morphing, mapeo arbitrario, etc.
- Pero todas ellas tienen el **mismo formato**:

  $R(x, y) := A(f_1(x,y), f_2(x,y))$
- Y todas requieren usar **interpolación**.
- La cuestión clave: ¿cómo están definidas las funciones f_1 y f_2 para el efecto que necesitamos?
- Ojo, en algún caso podemos tener lo contrario:

  $R(g_1(x,y), g_2(x,y)) := A(x,y)$
- En ese caso, habrá que obtener las funciones **inversas**.

Anexo A.4. Transformaciones geométricas en OpenCV.

- Transformaciones afines predefinidas
- Transformaciones afines genéricas
- Transformaciones perspectivas
- Transformaciones de mapeo arbitrario
- Ejemplos

A.4. Transformaciones geométricas OpenCV.

- Podemos clasificar las operaciones de transformación geométrica en los siguientes **tipos**:
  - 1) Transformaciones afines predefinidas.
  - 2) Transformaciones afines genéricas.
  - 3) Transformaciones perspectivas.
  - 4) Transformaciones de mapeo arbitrario.
- Un parámetro de las operaciones es el **tipo de interpolación** a aplicar. Se definen las constantes:

Interpolación	En OpenCV
Vecino más próximo	INTER_NEAREST
Bilineal	INTER_LINEAR
Bicúbica	INTER_CUBIC
Supermuestreo	INTER_AREA

A.4. Transformaciones geométricas OpenCV.

- **Comportamiento** de las operaciones de transformación geométrica en OpenCV:
  - Las operaciones permiten usar ROI. No se pueden usar máscaras (mask).
  - Se permite el modo in-place, es decir, la salida se almacena en la misma imagen de entrada.
  - En cuanto a la **extrapolación** de píxeles que caen fuera de la imagen de entrada (**borderMode**), se permiten distintos modos. Los más adecuados son: no modificar la imagen de entrada (**BORDER_TRANSPARENT**), o poner un valor constante (**BORDER_CONSTANT**) indicado en **borderValue**.
  - La principal carencia de OpenCV son las transformaciones bilineales. No se pueden hacer... hay que programárselas uno mismo con **remap**.

A.4. Transformaciones geométricas OpenCV.

- Transformaciones **afines predefinidas**:
  - resize, flip, transpose, getRotationMatrix2D
- Transformaciones **afines genéricas**:
  - warpAffine, getAffineTransform
- Transformaciones **perspectivas**:
  - warpPerspective, getPerspectiveTransform
- Transformaciones de **mapeo arbitrario**:
  - remap

A.4. Transformaciones geométricas OpenCV.

- **Redimensionar una imagen**:

  void **cv::resize** (Mat src, Mat dst, Size dsize,
  [double fx=0, double fy=0, int interp=INTER_LINEAR]);
  - Ojo, en esta función hay que poner cv::resize para que no se confunda con una función QMainWindows::resize.
  - Redimensionar la imagen **src** almacenando el resultado en **dst**. Puede ser un aumento o una reducción.
  - Dos modos posibles de dar el tamaño: indicar el tamaño de destino (con **dsize**); o indicar el factor de escala (**fx**, **fy**). En el primer caso fx,fy deben ser 0, y en el segundo dsize será 0.
  - El método de interpolación se indica en **interp**.

- **Ejemplo**. Zoom 2x de una imagen img (modo in-place):

  resize(img, img, Size(), 2, 2);

A.4. Transformaciones geométricas OpenCV.

- **Espejo de una imagen**:
  void **flip** (Mat src, Mat dst, int flipCode);
  - Calcula distintos espejos de la imagen **src** según el **flipCode**.
  - Valores de **flipCode**:
    1 → espejo horizontal; 0 → espejo vertical; -1 → ambos.
  - Permite modo in-place.
- **Transpuesta de una imagen**:
  void **transpose** (Mat src, Mat dst);
  - Calcula la transpuesta de imagen **src** en la imagen **dst**.
- Rotaciones exactas de **img**, en la matriz **rota**:
  - **Rotar 90º**: transpose(img, rota); flip(rota, rota, 1);
  - **Rotar 180º**: flip(img, rota, -1);
  - **Rotar 270º**: transpose(img, rota); flip(rota, rota, 0);

A.4. Transformaciones geométricas OpenCV.

- **Rotar una imagen**. No existe una función única para rotar una imagen, sino que hay que:
  - Calcular una matriz de rotación (**getRotationMatrix2D**).
  - Aplicar dicha transformación afín (**warpAffine**).
- Las matrices de transformación afín son **matrices Mat de tamaño 2x3** y tipo **CV_64FC1** (o **CV_32FC1**).
- **Ejemplo**. Rotar y escalar una imagen indicando ángulo y escala:

```
void Rotar (Mat imagen, Mat &salida, double angulo, double escala= 1)
{
   double w= imagen.size().width, h= imagen.size().height;
   double sa= sin(angulo*M_PI/180), ca= cos(angulo*M_PI/180);
   double cx= -w/2*ca-h/2*sa, cy= w/2*sa-h/2*ca;
   sa= fabs(sa); ca= fabs(ca);
   Size tam((w*ca+h*sa)*escala, (h*ca+w*sa)*escala);
   Mat M= getRotationMatrix2D(Point2f(0,0), angulo, escala);
   M.at<double>(0,2)= tam.width/2+cx*escala;
   M.at<double>(1,2)= tam.height/2+cy*escala;
   warpAffine(imagen, salida, M, tam);
}
```

A.4. Transformaciones geométricas OpenCV.

Transformaciones afines genéricas

- **Aplicar una transformación afín arbitraria**:

  void **warpAffine** (Mat src, Mat dst, Mat c, Size dsize,
  [int flags, int borderMode, const Scalar& borderValue]);

  – Aplicar una transformación afín genérica, dada por la fórmula:

$$dst(x,y) := src\left(\begin{bmatrix} c[0][0] & c[0][1] & c[0][2] \\ c[1][0] & c[1][1] & c[1][2] \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}\right)$$

  – La matriz **c**, de tipo CV_64FC1 o CV_32FC1, indica los coeficientes de la transformación.

  – La matriz **c** se puede rellenar usando **at**, o con el operador **<<**

  – La operación no puede funcionar en modo in-place → Eso dice la documentación, pero en realidad sí que lo admite.

A.4. Transformaciones geométricas OpenCV.

  – **flags** indica el tipo de interpolación (bilineal por defecto) y además... Si **flags** = WARP_INVERSE_MAP, la transformación es inversa, es decir, se aplica:

$$dst\left(\begin{bmatrix} c[0][0] & c[0][1] & c[0][2] \\ c[1][0] & c[1][1] & c[1][2] \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}\right) := src(x,y)$$

  – **borderMode** y **borderValue** indican el modo de rellenar las zonas exteriores (por defecto, rellenar a negro).

- **Ejemplo**. Inclinar (*shear*) la imagen **img** en X en **angulo** grados y desplazar en X para que se quede centrada.

```
double inc= tan(ui->doubleSpinBox_3->value()*M_PI/180.0);
Mat c= (Mat_<double>(2, 3) << 1,  inc,  -inc*img.size().height/2.0,
                              0,  1,    0 );
warpAffine(img, img, c, img.size());
```

A.4. Transformaciones geométricas OpenCV.

- **Calcular los coeficientes de una transformación afín**:

  Mat **getAffineTransform** (Mat src, Mat dst);

  - **src** es una matriz que almacena 3 puntos en la imagen origen. Debe ser de 1 fila y 3 columnas de tipo CV_32FC2 (para cada píxel, el primer canal es la X y el segundo canal es la Y).
  - **dst** es un array de 3 puntos, en el destino. Igual que antes.
  - **Significado**: calcular la transformación afín necesaria para mapear los puntos **src** en los puntos **dst**, almacenando el resultado en la matriz devuelta (que es de 2 filas y 3 columnas). Esta matriz es la que se debe usar en **warpPerspective**.
- Esta operación resuelve el sistema de ecuaciones de las páginas 45-46, para mapear un rombo dado en otro rombo.
- También se podrían resolver de forma explícita usando la función **solve** para resolver sistemas de ecuaciones en general.

A.4. Transformaciones geométricas OpenCV.

- **Ejemplo**. Aplicar una transformación afín a una imagen **img**, suponiendo que tenemos 3 puntos en **img** y los 3 puntos correspondientes en **destino**.

```
Mat pt1(1, 3, CV_32FC2);     // Puntos en el origen
pt1.at<Vec2f>(0,0)= Vec2f(0, 0);      // Rellenar los tres puntos
pt1.at<Vec2f>(0,1)= Vec2f(100, 0);
pt1.at<Vec2f>(0,2)= Vec2f(100, 100);
Mat pt2(1, 3, CV_32FC2);     // Puntos en el destino
pt2.at<Vec2f>(0,0)= Vec2f(10, 20);   // Rellenar los tres puntos
pt2.at<Vec2f>(0,1)= Vec2f(80, 40);
pt2.at<Vec2f>(0,2)= Vec2f(20, 70);
Mat c= getAffineTransform(pt1, pt2);
warpAffine(img, img, c, img.size());
```

A.4. Transformaciones geométricas OpenCV.

Transformaciones perspectivas

- Para las transformaciones **perspectivas**, las operaciones disponibles en OpenCV son similares a las afines.
  - **Aplicar una transformación**: warpAffine → warpPerspective
  - **Calcular los coeficientes**: getAffineTransform → getPerspectiveTransform
- En este caso la matriz que define la transformación es un **Mat** de 3x3 de tipo **CV_64FC1** o **CV_32FC1**.
- OpenCV no tiene las transformaciones bilineales.

A.4. Transformaciones geométricas OpenCV.

- **Aplicar una transformación perspectiva**:

  void **warpPerspective** (Mat src, Mat dst, Mat c, Size dsize,
  [int flags, int borderMode, const Scalar& borderValue]);

  - Aplica una transf. perspectiva de la imagen **src** en **dst**. La matriz de coeficientes **c** es de 3x3.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} c[0][0] & c[0][1] & c[0][2] \\ c[1][0] & c[1][1] & c[1][2] \\ c[2][0] & c[2][1] & c[2][2] \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

  - **flags** indica el tipo de interpolación (por defecto bilineal). Y si vale WARP_INVERSE_MAP, se usa la inversa de la matriz **c**.
  - Si **flags** vale BORDER_TRANSPARENT significa que lo que caiga fuera en dst, no se modifique.

A.4. Transformaciones geométricas OpenCV.

- **Calcular los coeficientes de una transformación afín**:
  Mat **getPerspectiveTransform** (Mat src, Mat dst);
  - **src** es un array de 4 puntos, en la imagen de origen. Igual que getAffineTransform, debe ser un Mat de 1 fila y 4 columnas, de CV_32FC2 (un píxel para cada punto).
  - **dst** es un array de 4 puntos, en la imagen de destino.
  - **Significado**: calcular la transformación perspectiva necesaria para mapear los puntos **src** en los puntos **dst**, devolviendo el resultado. Es la que se debe aplicar en **warpPerspective**.
- Esta operación resuelve el sistema de ecuaciones de la página 59, para mapear un cuadrilátero dado en otro.
- Los cuadriláteros deben ser conexos (no cóncavos).
- También se podrían resolver con **solve**.

A.4. Transformaciones geométricas OpenCV.

- **Algunas propiedades interesantes de la matriz c**:
  - Si ambos cuadriláteros son rombos, la transformación anterior será afín, por lo que **c**[2][0]= 0, **c**[2][1]= 0, **c**[2][2]= 1.
  - La inversa de una transformación perspectiva asociada a **c**, se obtiene usando la matriz inversa de **c**, es decir, $\mathbf{c}^{-1}$. Ver **invert** para invertir matrices.
  - El modo **WARP_INVERSE_MAP** consiste simplemente en usar la matriz $\mathbf{c}^{-1}$.
  - La aplicación sucesiva de dos transformaciones perspectivas **c1** y luego **c2**, equivale a la transformación asociada al producto matricial de **c2** por **c1**, que se calcula con: **c2*c1**

A.4. Transformaciones geométricas OpenCV.

- **Ejemplo**. Transformación perspectiva de una imagen, **img**, para producir un efecto similar al de la página 56.

```
double w= img.size().width, h= img.size().height;
Mat pt1(1, 4, CV_32FC2);      // Cuatro puntos en el origen
pt1.at<Vec2f>(0,0)= Vec2f(0, 0);
pt1.at<Vec2f>(0,1)= Vec2f(w, 0);
pt1.at<Vec2f>(0,2)= Vec2f(w, h);
pt1.at<Vec2f>(0,3)= Vec2f(0, h);
Mat pt2(1, 4, CV_32FC2);      // Cuatro puntos en el destino
pt2.at<Vec2f>(0,0)= Vec2f(w*0.3, 0);
pt2.at<Vec2f>(0,1)= Vec2f(w*0.7, 0);
pt2.at<Vec2f>(0,2)= Vec2f(w, h);
pt2.at<Vec2f>(0,3)= Vec2f(0, h);
Mat c= getPerspectiveTransform(pt1, pt2);
Mat resultado;
warpPerspective(img, resultado, c, img.size());
```


A.4. Transformaciones geométricas OpenCV.

Transformaciones de mapeo arbitrario:

- Consisten en definir el par de funciones $\mathbf{f_1}(x,y)$ y $\mathbf{f_2}(x,y)$ de la transformación geométrica genérica:

  $R(x,y) := A(f_1(x,y), f_2(x,y))$
- f_1 y f_2 indican para cada píxel de salida, cuál es el píxel correspondiente de entrada.
- Recordar que f_1 y f_2 se pueden ver, a su vez, como imágenes, que tendrán un solo canal y tipo CV_32FC1.
- Cuando el valor de las funciones no sea entero, se aplicará algún método de interpolación (según **interpol**).
- Si el valor asociado a un píxel cae fuera de la imagen origen, el contenido de la imagen no se modifica o se pone a un valor constante (según **borderMode**).
- La única función necesaria es **remap**.

A.4. Transformaciones geométricas OpenCV.

- **Aplicar un mapeo arbitrario**:

void **remap** (Mat src, Mat dst, Mat map1, Mat map2, int interp,
[int borderMode, const Scalar& borderValue]);

  - Aplica la transformación de mapeo:
    dst(x,y):= src(map1(x,y), map2(x,y))
  - Es decir, $\mathbf{f_1}$ está dada en **map1**, y $\mathbf{f_2}$ en **map2**.
  - **map1** y **map2** deben ser imágenes 1 solo canal, de profundidad float (**CV_32FC1**) y del mismo tamaño que **dst**.
  - La profundidad de dst será la misma que **src**, y su tamaño será el mismo que **map1** y **map2**.
  - Admite modo **in-place**.
  - Lo importante de esta función es cómo calcular las imágenes **map1** y **map2** para conseguir el efecto deseado.

A.4. Transformaciones geométricas OpenCV.

- **Ejemplo 1**. Transformación aleatoria de una imagen **img**, con un radio de deformación de **m**:

```
Mat mapa1(img.size(), CV_32FC1);
Mat mapa2(img.size(), CV_32FC1);
for (int y= 0; y<img.size().height; y++)
   for (int x= 0; x<img.size().width; x++) {
      mapa1.at<float>(y, x)= x+rand()%(2*m+1) - m;   // Mapa 1
      mapa2.at<float>(y, x)= y+rand()%(2*m+1) - m;   // Mapa2
   }
Mat resultado;
remap(img, resultado, mapa1, mapa2, INTER_NEAREST);
```

- **Ejemplo 2**. Transformación de acristalado. En el código anterior, sustituir las líneas comentadas por:

```
mapa1.at<float>(y, x)= x - x%m + y%m;
mapa2.at<float>(y, x)= y - y%m + x%m;
```