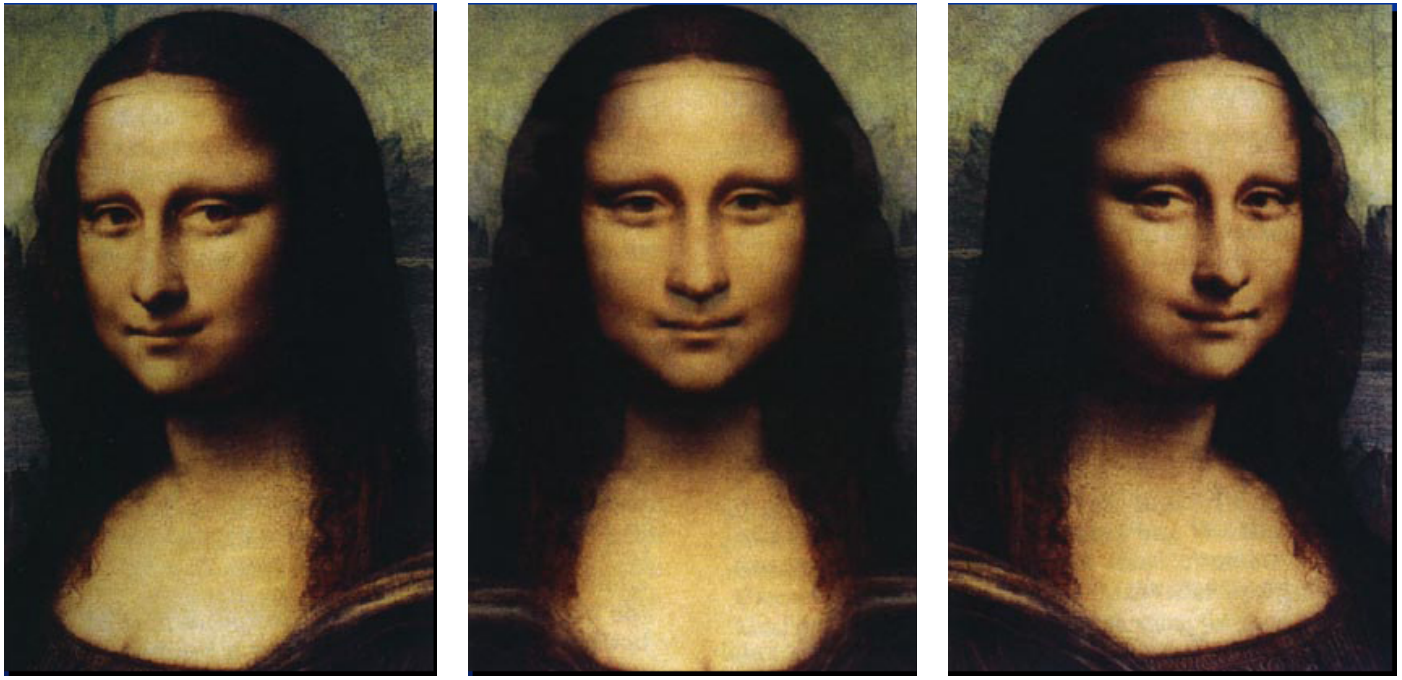
Interactive Computer Graphics: Lecture 15

Warping and Morphing

Warping and Morphing

- What is
  - warping ?
  - morphing ?



Warping and Morphing

- What is
  - warping ?
  - morphing ?

?

Warping and Morphing

- What is
  - warping ?
  - morphing ?



Warping

- The term warping refers to the geometric transformation of graphical objects (images, surfaces or volumes) from one coordinate system to another coordinate system.
- Warping does not affect the attributes of the underlying graphical objects.
- Attributes may be
  - color (RGB, HSV)
  - texture maps and coordinates
  - normals, etc.

Morphing

- The term morphing stands for metamorphosing and refers to an animation technique in which one graphical object is gradually turned into another.
- Morphing can affect both the shape and attributes of the graphical objects.

Warping and Morphing

- What is
  - warping ?
  - morphing ?



Morphing = Object Averaging

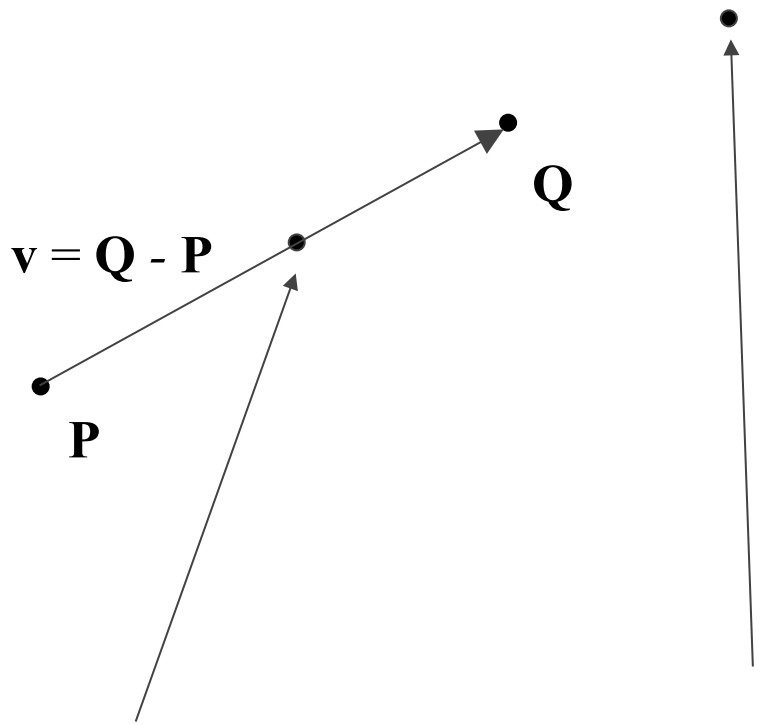
- The aim is to find "an average" between two objects
  - Not an average of two images of objects…

    …but an image of the average object!
  - How can we make a smooth transition in time?
    - Do a "weighted average" over time t

- How do we know what the average object looks like?
  - Need an algorithm to compute the average geometry and appearance

Averaging

What's the average of **P** and **Q**?



Linear Interpolation (Affine Combination):

New point $a\mathbf{P} + b\mathbf{Q}$, defined only when $a+b = 1$
So $a\mathbf{P}+b\mathbf{Q} = a\mathbf{P}+(1-a)\mathbf{Q}$

$\mathbf{P} + 0.5\mathbf{v}$
$= \mathbf{P} + 0.5(\mathbf{Q} - \mathbf{P})$
$= 0.5\mathbf{P} + 0.5\mathbf{Q}$

$\mathbf{P} + 1.5\mathbf{v}$
$= \mathbf{P} + 1.5(\mathbf{Q} - \mathbf{P})$
$= -0.5\mathbf{P} + 1.5\mathbf{Q}$
(extrapolation)

Image blending

- Determines how to combine attributes associated with geometrical primitives. Attributes may include
  - color
  - texture coordinates
  - normals
- Blending
  - cross-dissolve
  - adaptive cross-dissolve
  - alpha-channel blending
  - z-buffer blending

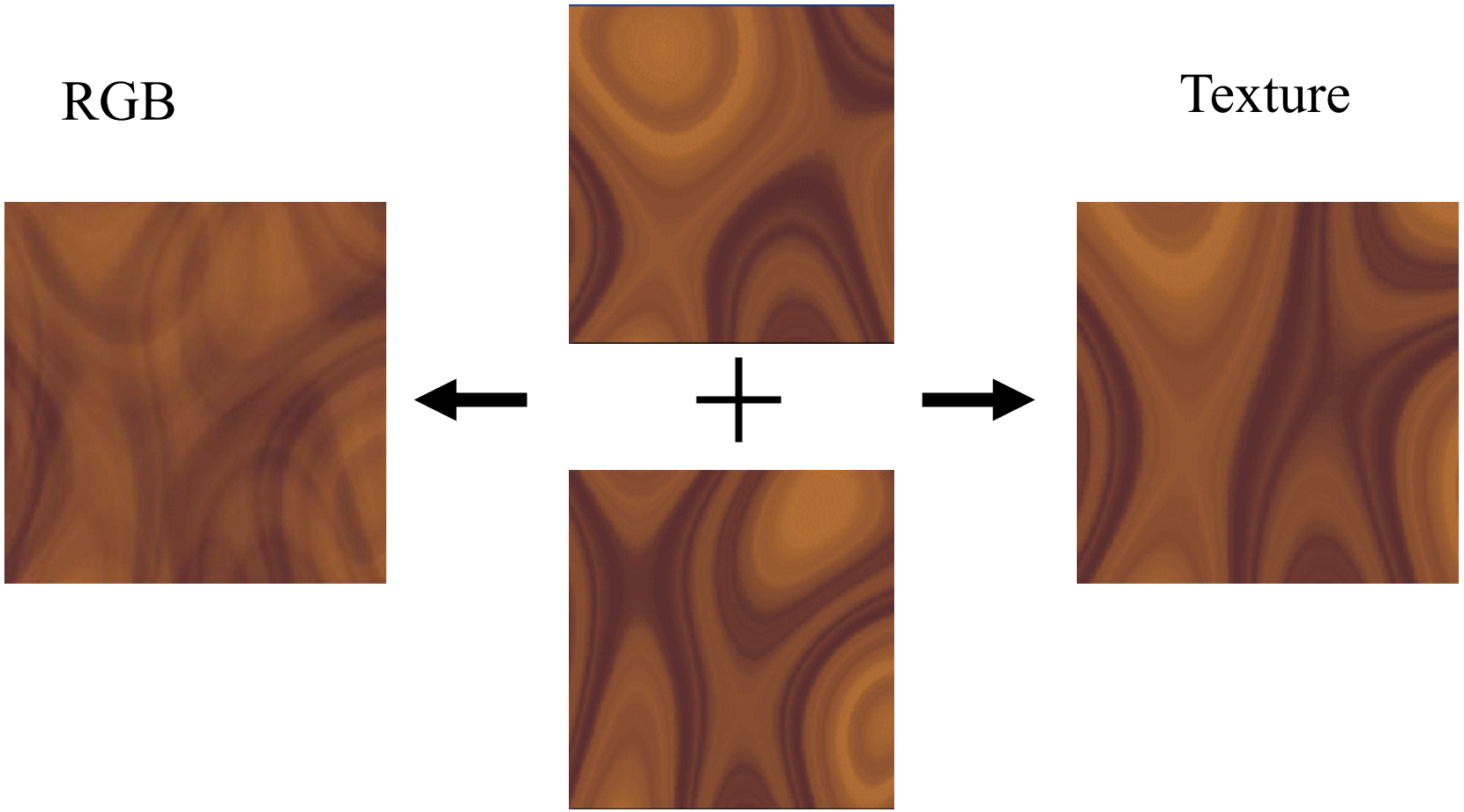
Image blending: Cross-dissolve

- Blending with cross-dissolve:

$$I = (1-t)\cdot I_A + t\cdot I_B$$

  - intensities
  - RGB space
  - HSV space
  - texture space

Image blending: Cross-dissolve



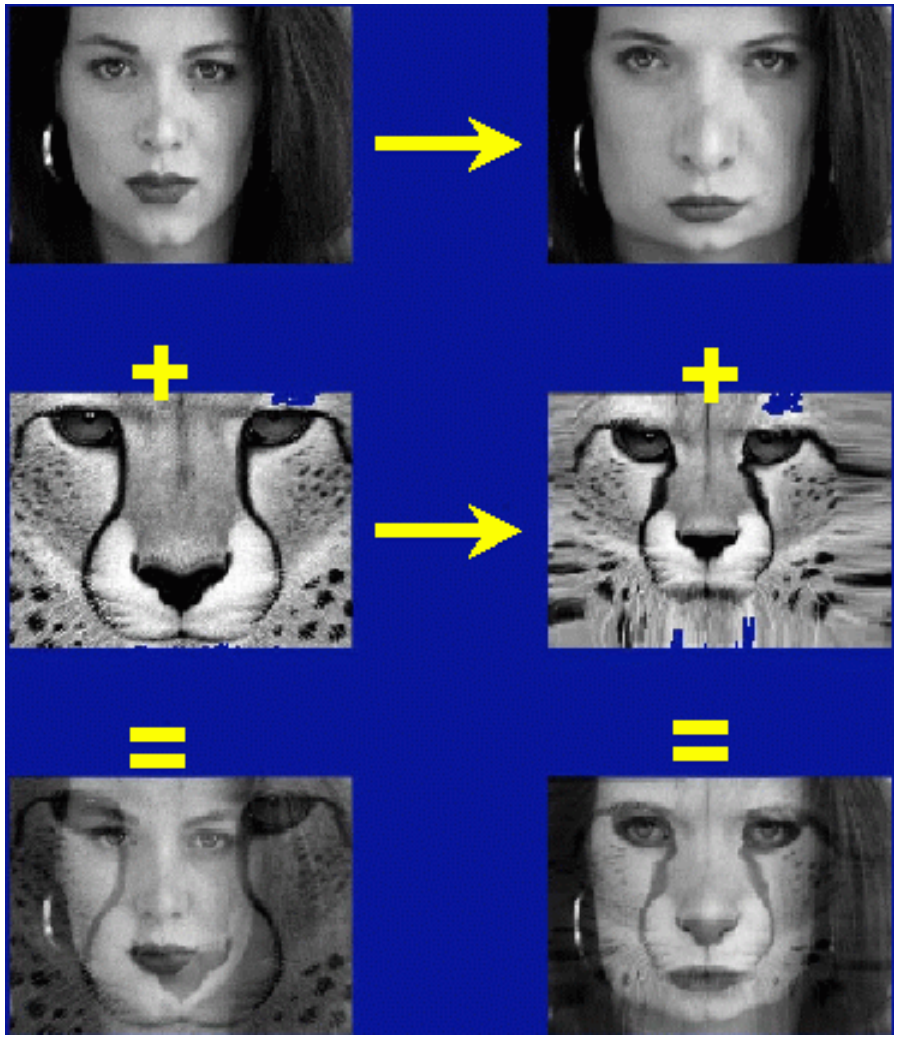
Morphing using cross-dissolve

- But what is the images are not aligned?

Morphing using warping and cross-dissolve

- Align first, then cross-dissolve

Morphing = (warping)^2 + blending



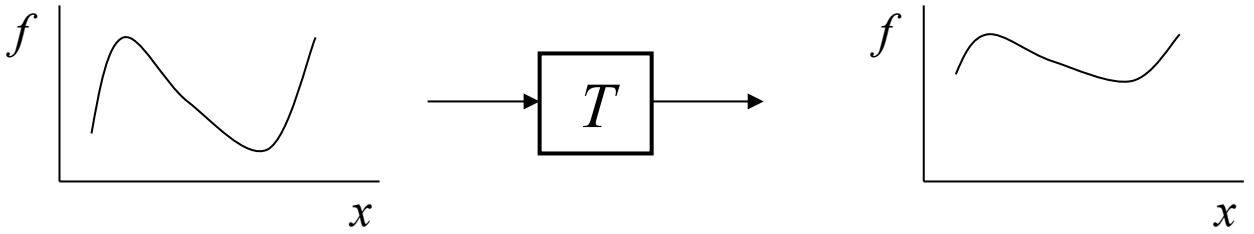
Morphing

```
GenerateAnimation(Image0, Image1)
begin
      foreach intermediate frame time t do
            Warp0 = WarpImage(Image0, t)
            Warp1 = WarpImage(Image1, t)
            foreach pixel p in FinalImage do
                  Result(p) = (1-t)Warp0 + tWarp1
            end
      end
end
```

Image warping

- Image filtering: change ***range*** of image

$$g(x) = T(f(x))$$



- Image warping: change ***domain*** of image

$$g(x) = f(T(x))$$

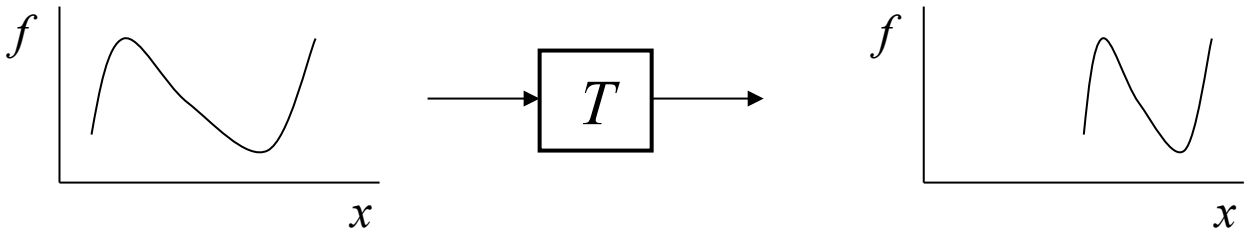
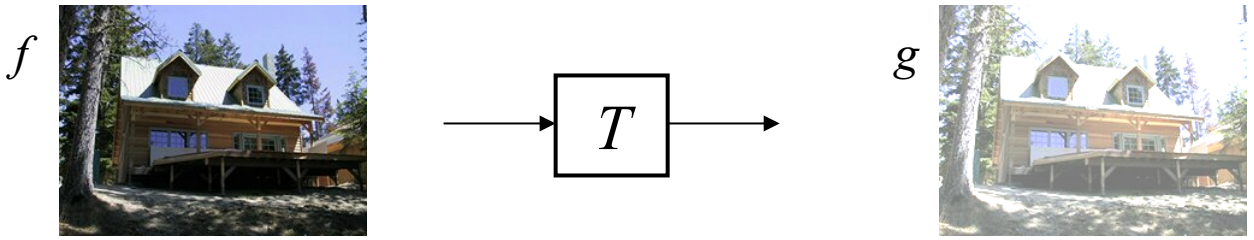


Image warping

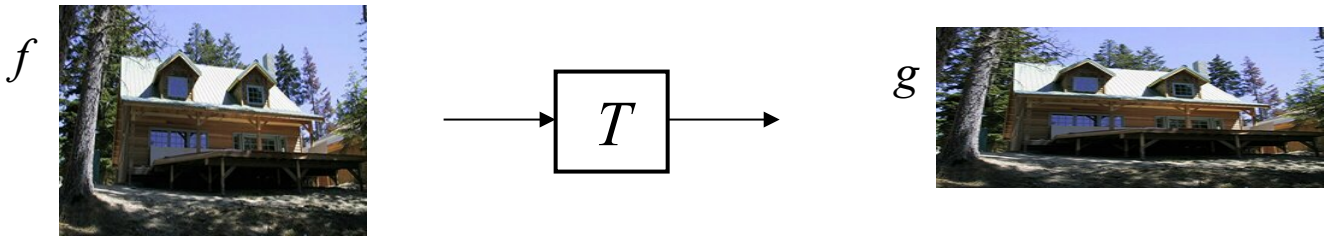
- Image filtering: change ***range*** of image

$$g(x) = T(f(x))$$



- Image warping: change ***domain*** of image

$$g(x) = f(T(x))$$



Parametric (global) warping

- Examples of parametric warps:

translation

rotation

aspect

affine

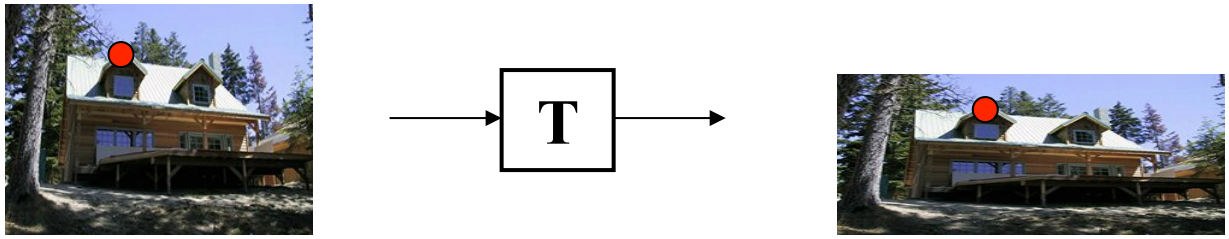
perspective

cylindrical

Parametric warps (or transformations)

- Dimensions of transformation
  - 1D: curves
  - 2D: images
  - 3D: volumes
- Types of transformations
  - rigid
  - affine
  - polynomial
    - quadratic
    - cubic
  - splines

Parametric (global) warping



$\mathbf{p} = (x,y)$ $\mathbf{p'} = (x',y')$

- Transformation T can be expressed as a mapping:

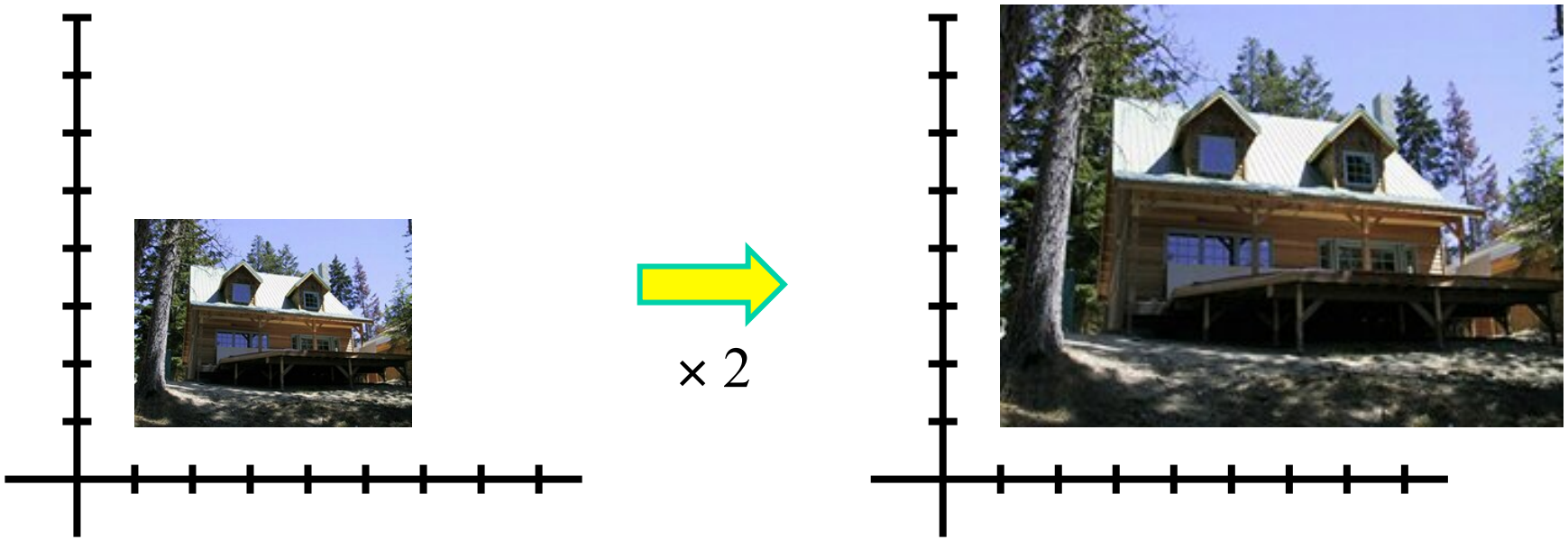
$$\mathbf{p'} = \mathbf{T(p)}$$

- Transformation T can be expressed as a matrix:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{M} \begin{bmatrix} x \\ y \end{bmatrix}$$

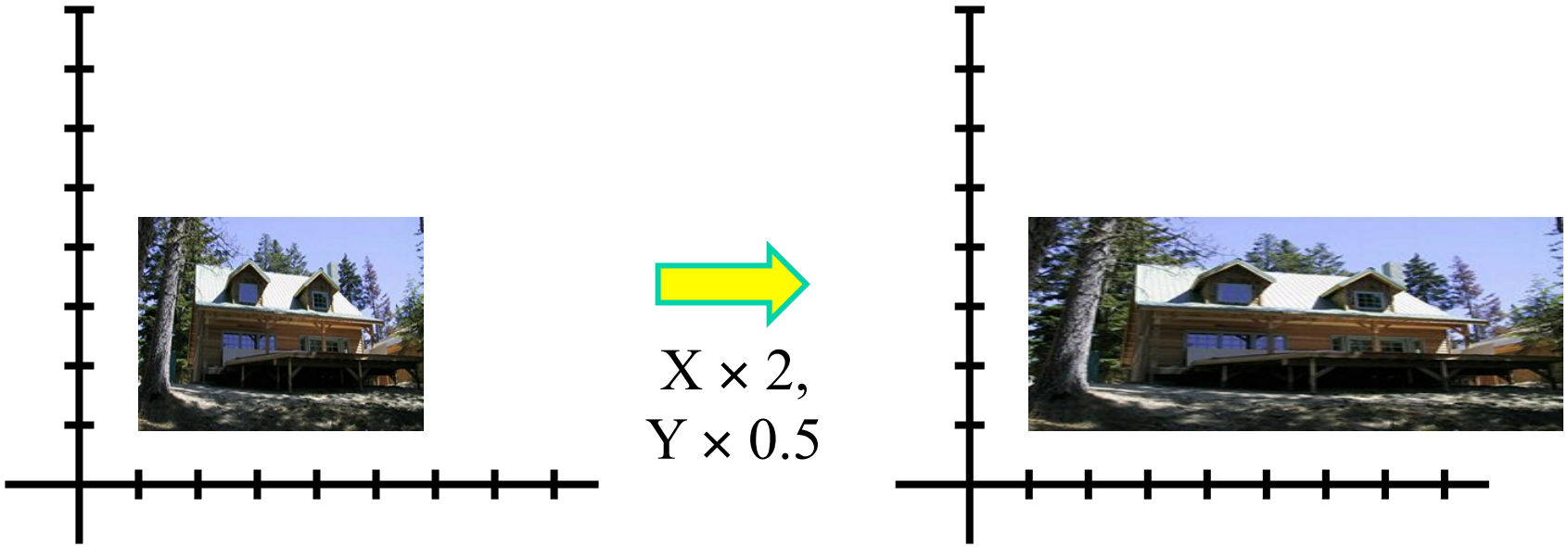
Scaling

- *Scaling* a coordinate means multiplying each of its components by a scalar
- *Uniform scaling* means this scalar is the same for all components:



Scaling

- *Non-uniform scaling*: different scalars per component:



Scaling

- Scaling operation:

$$x' = ax$$

$$y' = by$$

- Or, in matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}}_{\text{scaling matrix } \mathbf{S}} \begin{bmatrix} x \\ y \end{bmatrix}$$

- What is the inverse of $\mathbf{S}$?

2-D Rotation

- This is easy to capture in matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}}_{\mathbf{R}} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Even though $\sin(\theta)$ and $\cos(\theta)$ are nonlinear functions of θ
  - x' is a linear combination of x and y
  - y' is a linear combination of x and y
- What is the inverse transformation?
  - Rotation by $-\theta$
  - For rotation matrices, $\det(\mathbf{R}) = 1$ so $\mathbf{R}^{-1} = \mathbf{R}^{T}$

2x2 Matrices

- What types of transformations can be represented with a 2 x 2 matrix?

2D Identity?

$$x' = x$$
$$y' = y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Scaling around the origin?

$$x' = s_x x$$
$$y' = s_y y$$

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \end{bmatrix} = \begin{bmatrix} \boldsymbol{s}_x & 0 \\ 0 & \boldsymbol{s}_y \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix}$$

2x2 Matrices

- What types of transformations can be represented with a 2 x 2 matrix?

2D Rotate around the origin?

$$x' = x\cos\Theta - y\sin\Theta$$
$$y' = x\sin\Theta + y\cos\Theta$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Shear?

$$x' = x + sh_x y$$
$$y' = sh_y x + y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & sh_x \\ sh_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2x2 Matrices

- What types of transformations can be represented with a 2 x 2 matrix?

2D Mirror about Y axis?

$$x' = -x$$
$$y' = y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Mirror about origin?

$$x' = -x$$
$$y' = -y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2x2 Matrices

- What types of transformations can be represented with a 2 x 2 matrix?

2D Translation?

$$x' = x + t_x$$
$$y' = y + t_y$$

NO!

Only linear 2D transformations
can be represented with a 2 x 2 matrix

All 2D Linear Transformations

- Linear transformations are combinations of …
  - Scale,
  - Rotation,
  - Shear, and
  - Mirror

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Properties of linear transformations:
  - Origin maps to origin
  - Lines map to lines
  - Parallel lines remain parallel
  - Ratios are preserved
  - Closed under composition

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} \begin{bmatrix} i & j \\ k & l \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Homogeneous Coordinates

- Q: How can we represent translation as a matrix transformation?

$$x' = x + t_x$$
$$y' = y + t_y$$

- A: Using the translation parameters as the rightmost column:

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Basic 2D Transformations

- Basic 2D transformations as 3 x 3 matrices

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translate

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{s}_x & 0 & 0 \\ 0 & \boldsymbol{s}_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ 1 \end{bmatrix}$$

Scale

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotate

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \boldsymbol{sh}_x & 0 \\ \boldsymbol{sh}_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ 1 \end{bmatrix}$$

Shear

Image warping

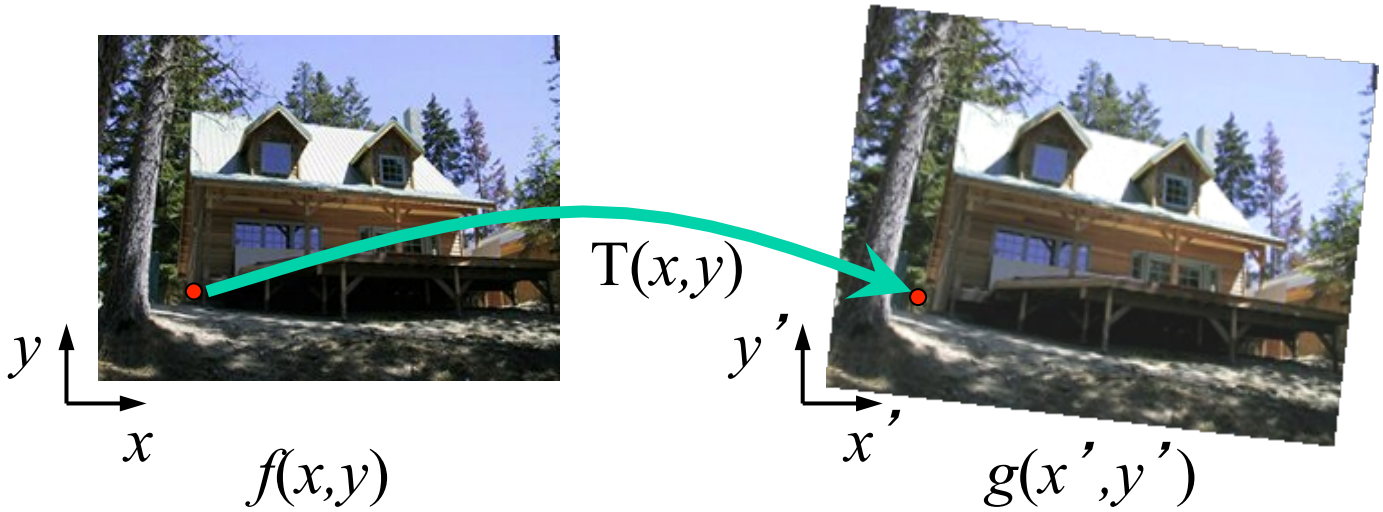
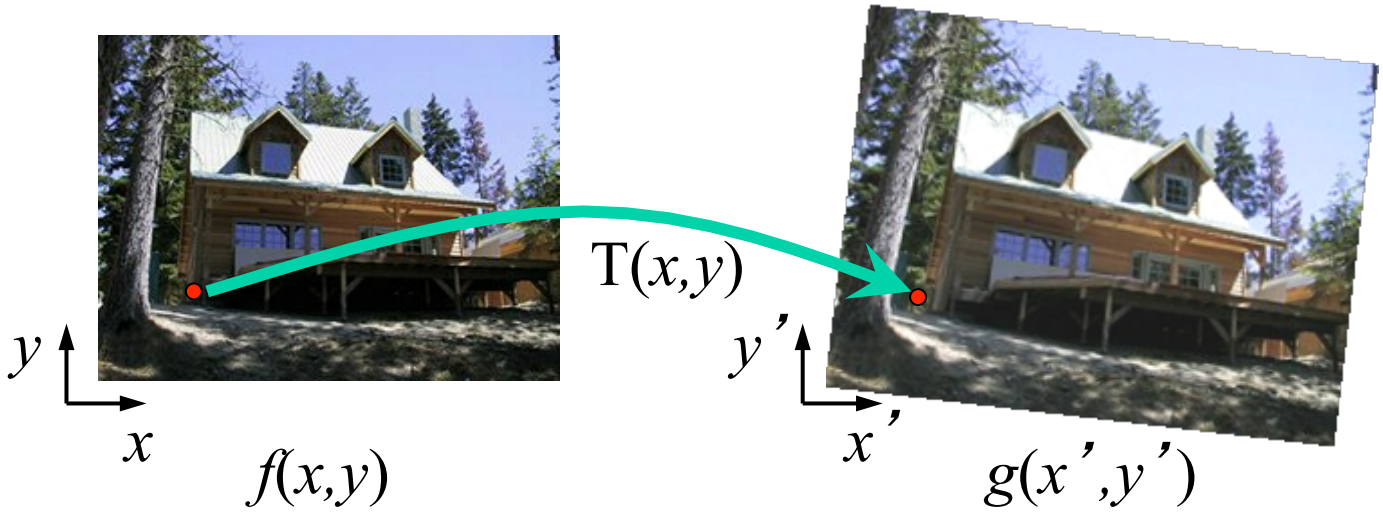
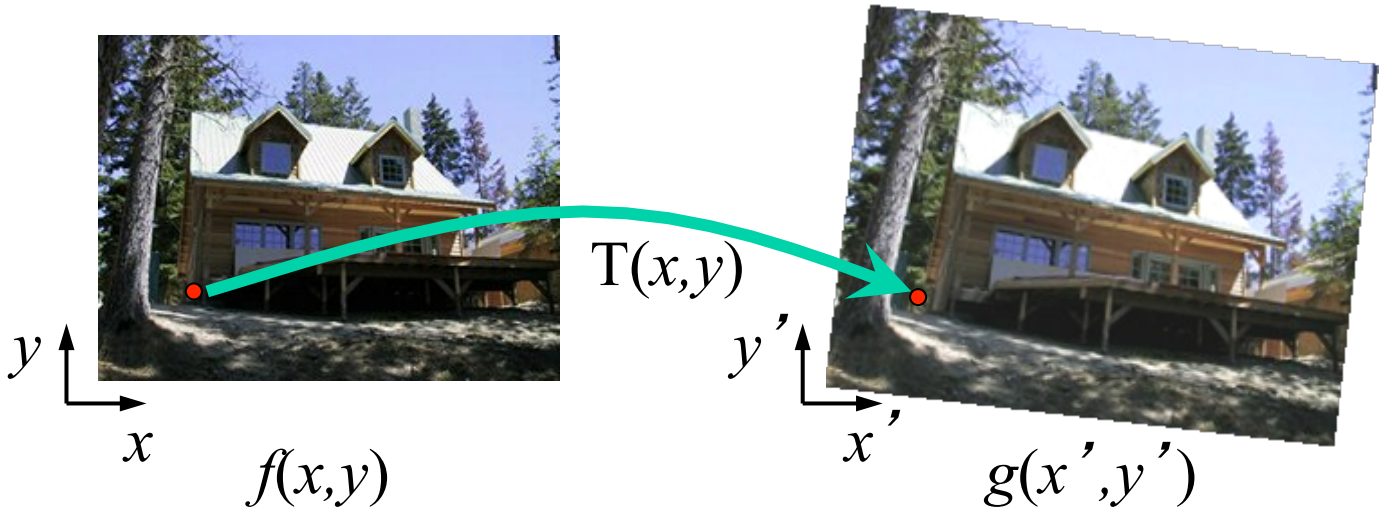


Image warping



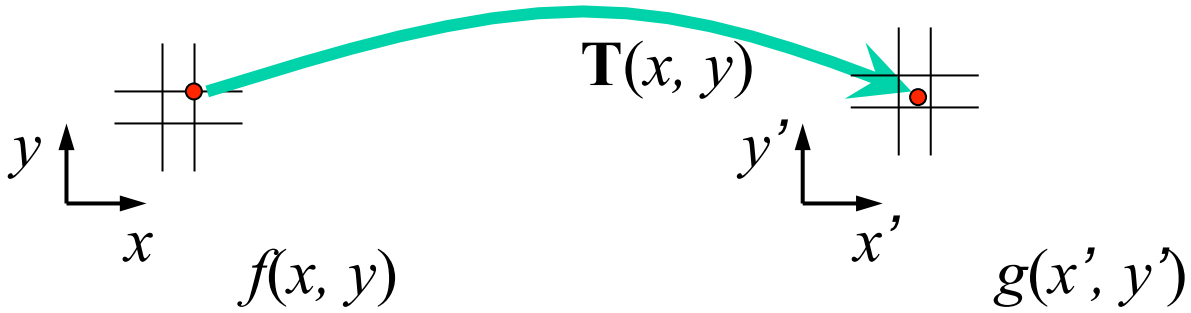
- Given a coordinate transform $(x',y') = \mathbf{T}(x, y)$ and a source image $f(x, y)$, how do we compute a transformed image $g(x', y') = f(\mathbf{T}(x, y))$?

Forward warping



- Given a coordinate transform $(x',y') = \mathbf{T}(x, y)$ and a source image $f(x, y)$, how do we compute a transformed image $g(x', y') = f(\mathbf{T}(x, y))$?
- Send each pixel $f(x,y)$ to its corresponding location $(x',y') = \mathbf{T}(x, y)$ in the second image

Forward warping

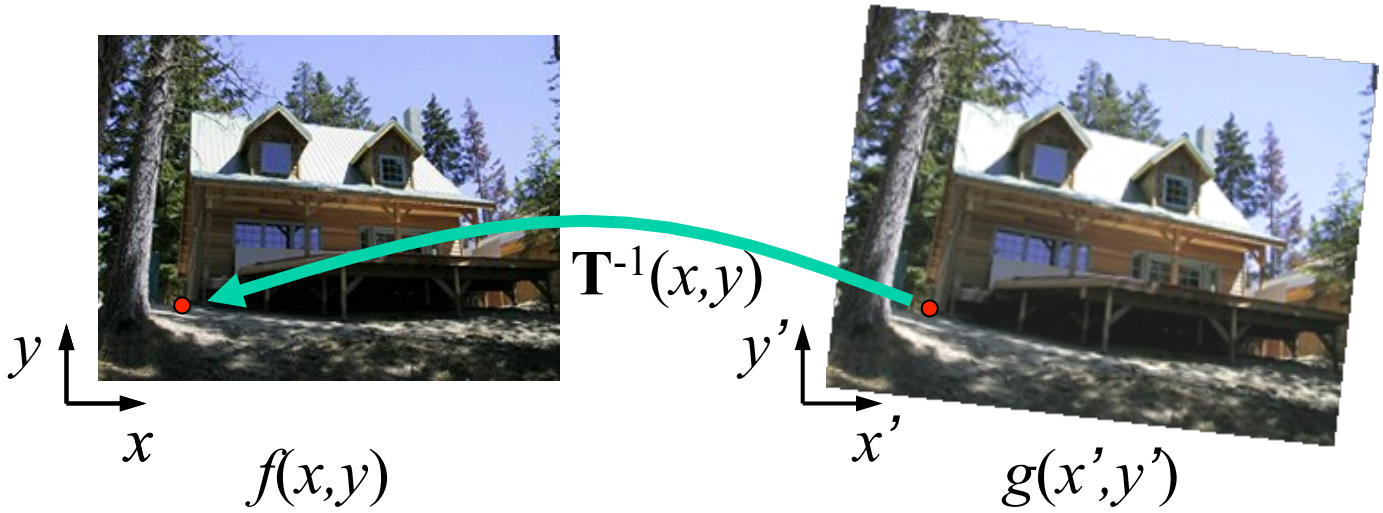


- Given a coordinate transform $(x',y') = \mathbf{T}(x, y)$ and a source image $f(x, y)$, how do we compute a transformed image $g(x', y') = f(\mathbf{T}(x, y))$?
- Send each pixel $f(x,y)$ to its corresponding location $(x',y') = \mathbf{T}(x, y)$ in the second image

  Q: what if pixel lands "between" two pixels?

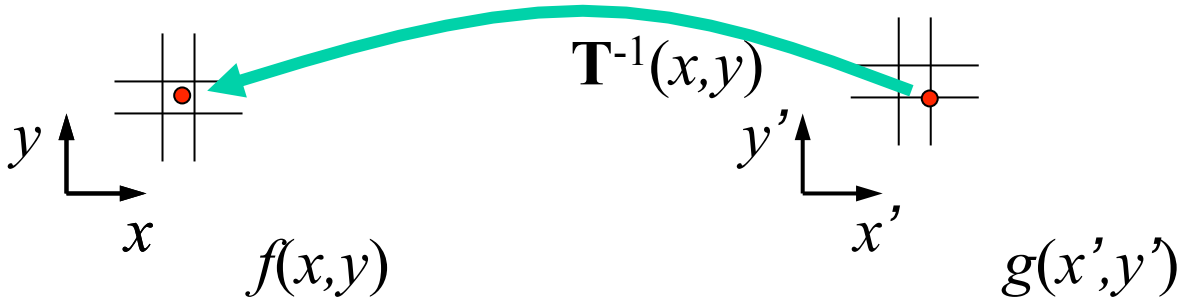
  A: distribute color among neighboring pixels (x',y')

Inverse warping



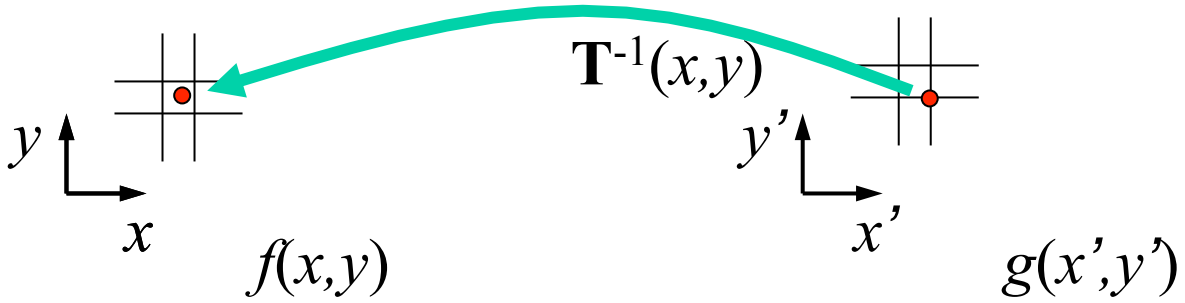
- Get each pixel $g(x', y')$ from its corresponding location $(x, y) = \mathbf{T}^{-1}(x', y')$ in the first image

Inverse warping



- Get each pixel $g(x', y')$ from its corresponding location $(x, y) = \mathbf{T}^{-1}(x', y')$ in the first image
  Q: what if pixel comes from "between" two pixels?

Inverse warping



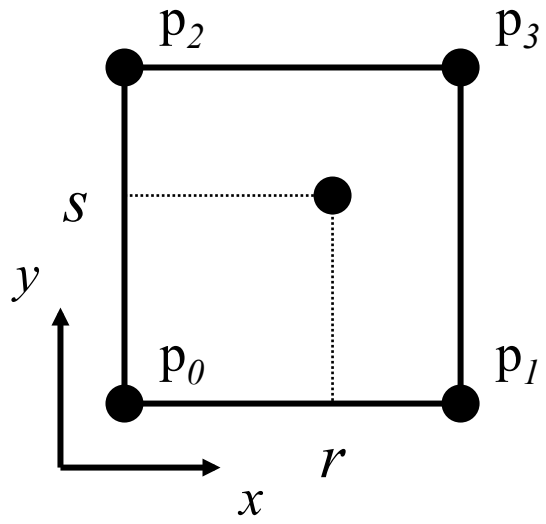
- Get each pixel $g(x', y')$ from its corresponding location $(x, y) = \mathbf{T}^{-1}(x', y')$ in the first image
  Q: what if pixel comes from "between" two pixels?
  A: Interpolate color value from neighbors
  – nearest neighbor, bilinear, Gaussian, bicubic

Interpolation

Interpolation

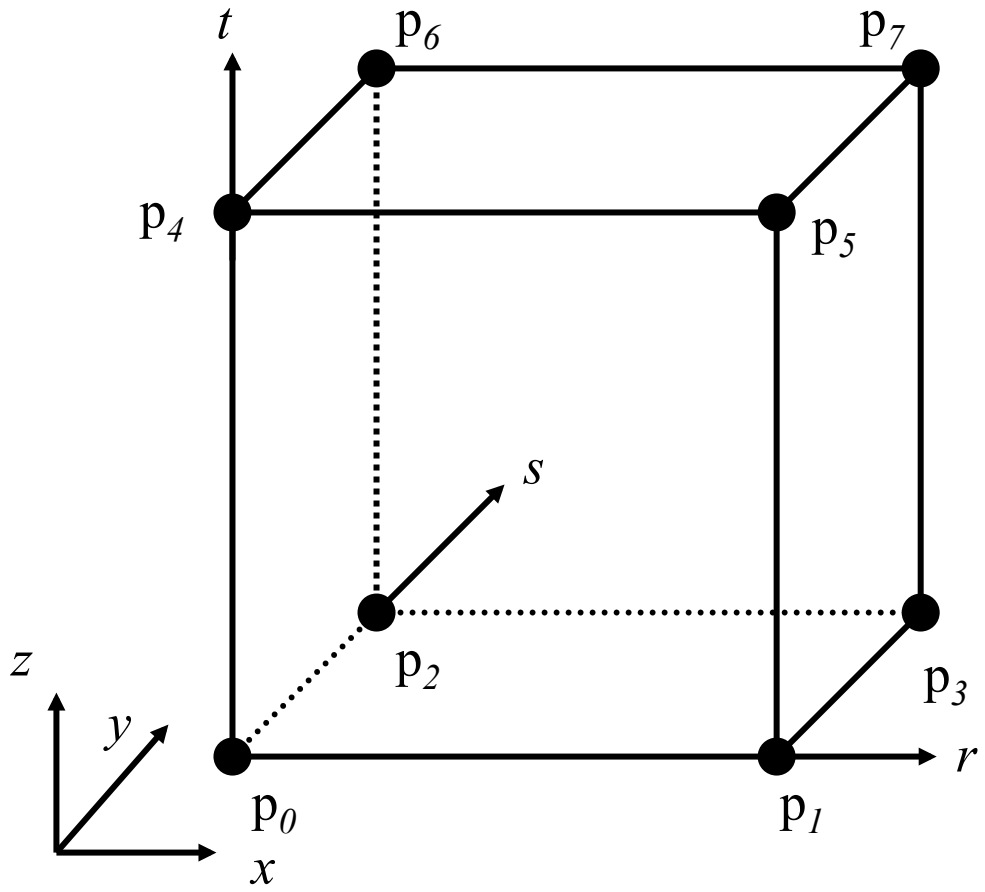
Interpolation: Linear, 2D

$$f(p) = \sum_{i=0}^{n-1} w_i f(p_i)$$



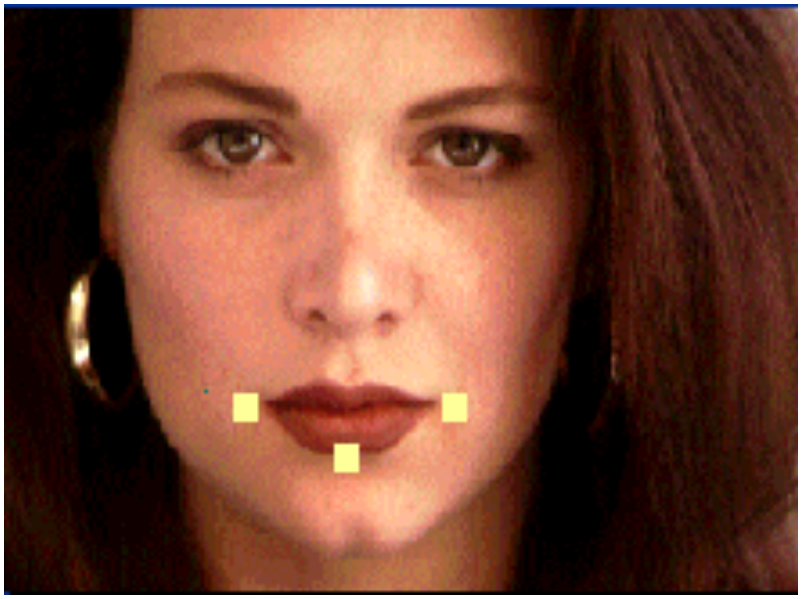
$$w_0 = (1-r)(1-s)$$
$$w_1 = r(1-s)$$
$$w_2 = (1-r)s$$
$$w_3 = rs$$

Interpolation: Linear, 3D



$$w_0 = (1-r)(1-s)(1-t)$$
$$w_1 = r(1-s)(1-t)$$
$$w_2 = (1-r)s(1-t)$$
$$w_3 = rs(1-t)$$
$$w_4 = (1-r)(1-s)t$$
$$w_5 = r(1-s)t$$
$$w_6 = (1-r)st$$
$$w_7 = rst$$

Image warping: Correspondences



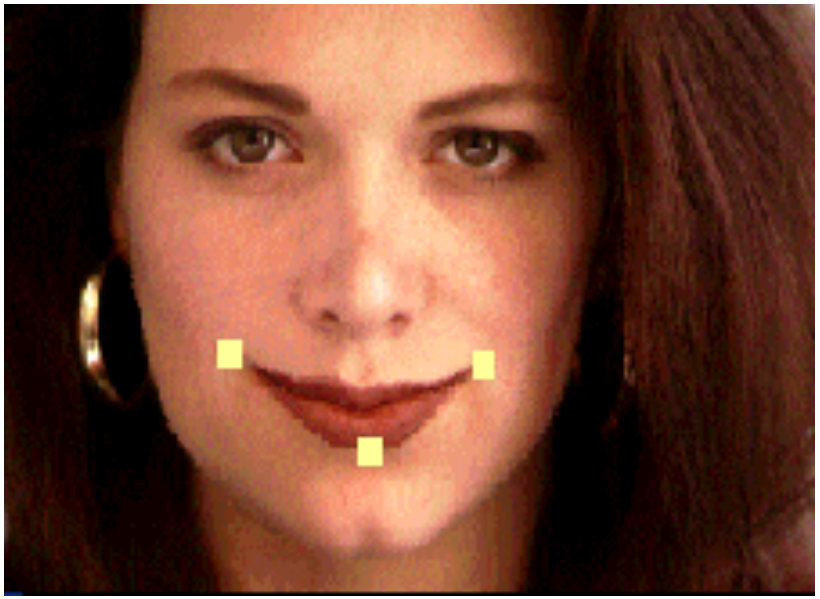


Image warping: Correspondences

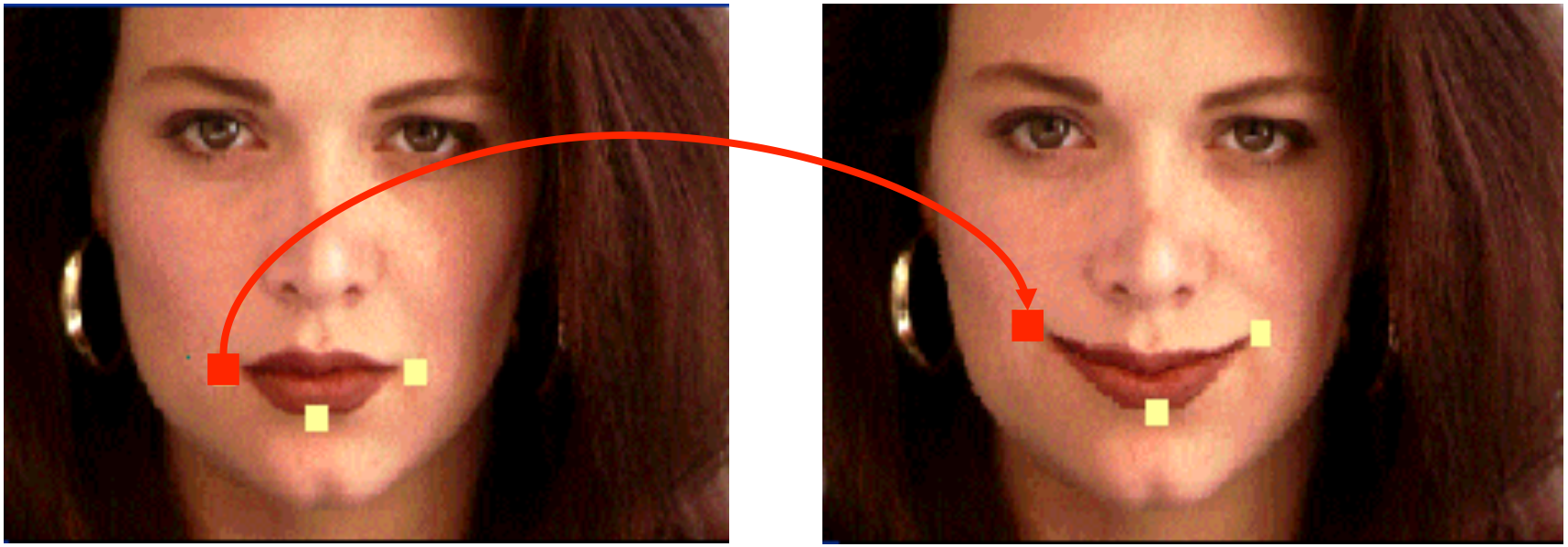
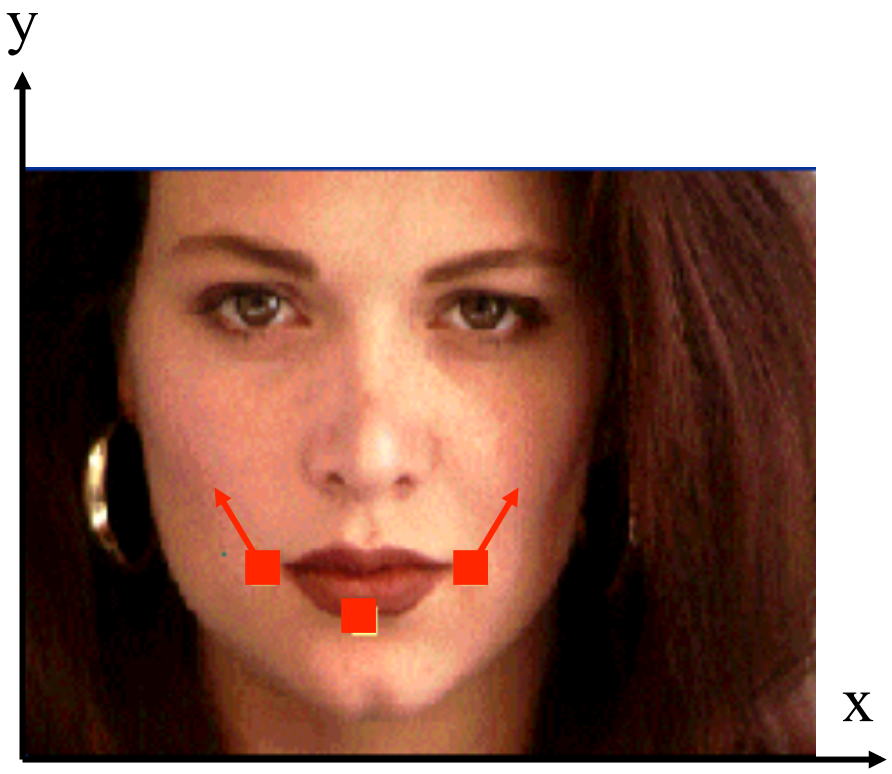
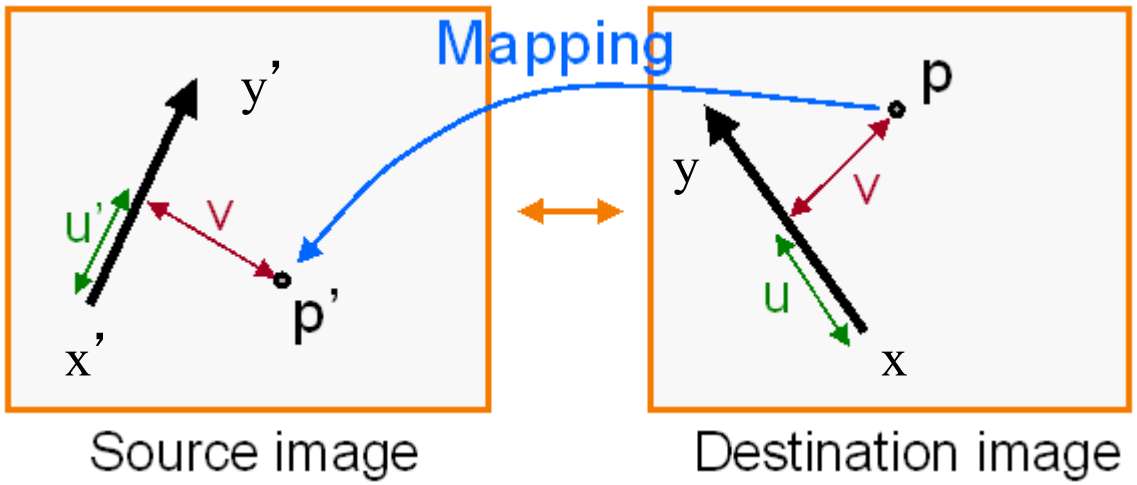


Image warping: Correspondences



Feature-Based Warping: Beier-Neeley

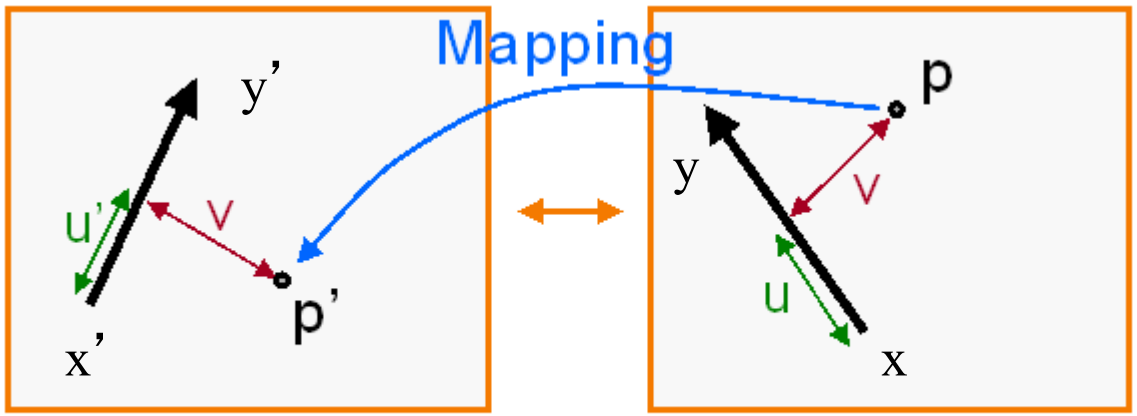
- Beier & Neeley use pairs of lines to specify warp
  - Given **p** in destination image, where is **p'** in source image?



Feature-Based Warping: Beier-Neeley

$$u = \frac{(p-x)\cdot(y-x)}{\|y-x\|^2} \qquad v = \frac{(p-x)\cdot Perpendicular(y-x)}{\|y-x\|}$$

$$p' = x + u\cdot(y'-x') + \frac{v\cdot Perpendicular(y'-x')}{\|y'-x'\|}$$

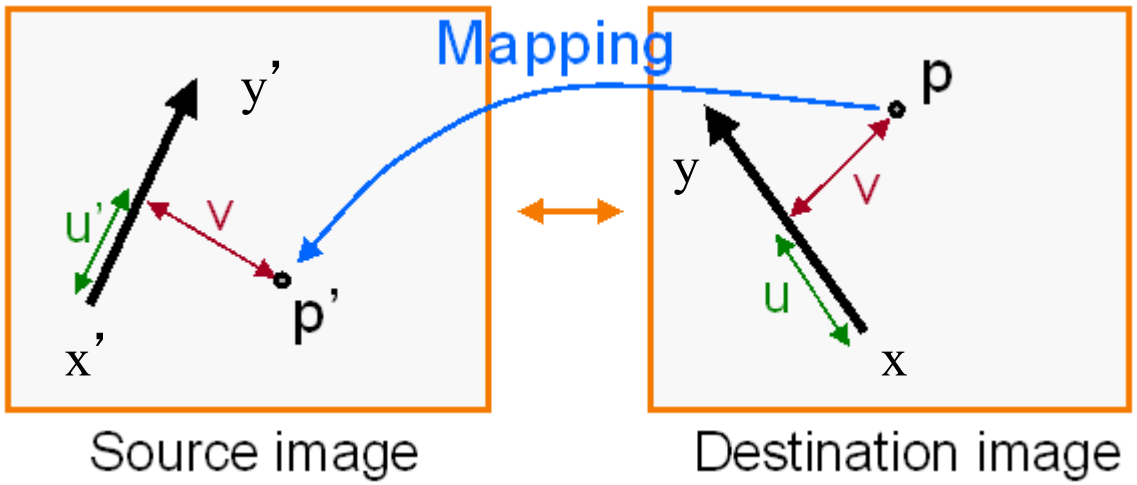


u is a fraction

v is a length (in pixels)

Feature-Based Warping: Beier-Neeley

- For each pixel p in the destination image
  - find the corresponding u,v
  - find the p’ in the source image for that u,v
  - destination(p) = source(p’)

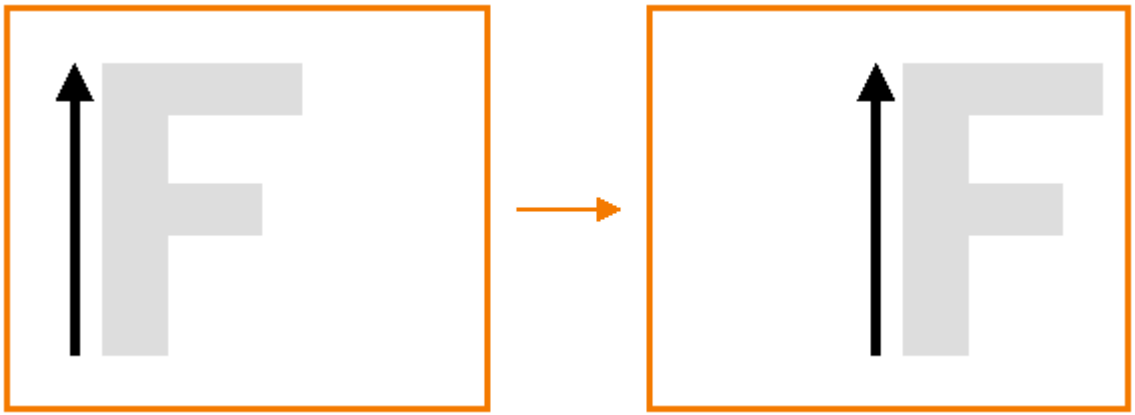


u is a fraction

v is a length (in pixels)

Warping with One Line Pair: Beier-Neeley

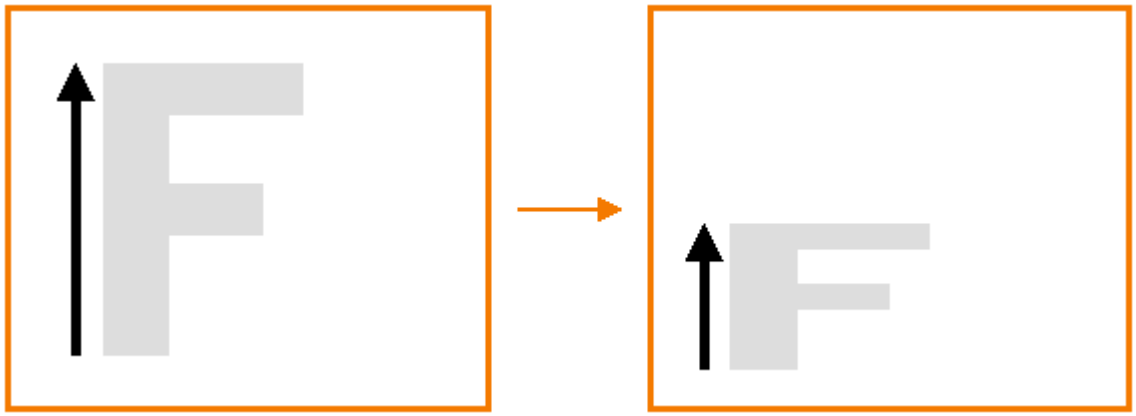
- What happens to the “F” ?



Translation !

Warping with One Line Pair (cont.): Beier-Neeley

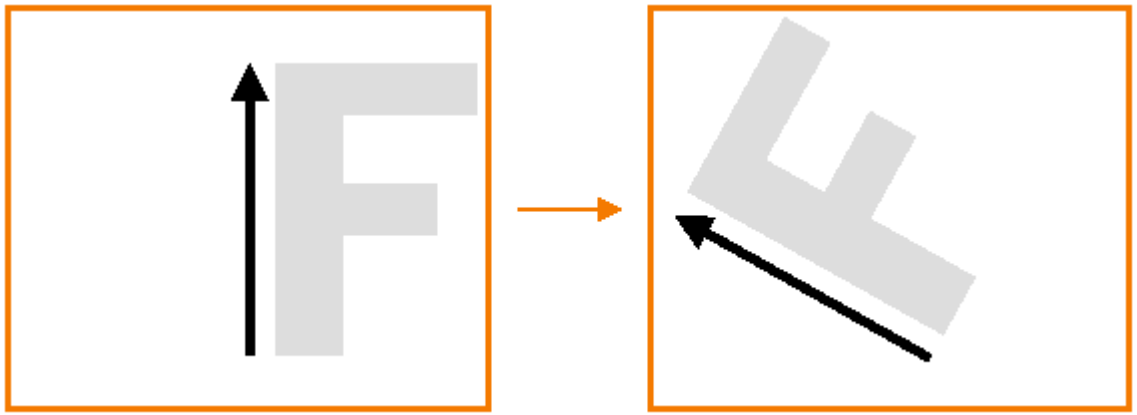
- What happens to the “F” ?



Scale !

Warping with One Line Pair (cont.): Beier-Neeley

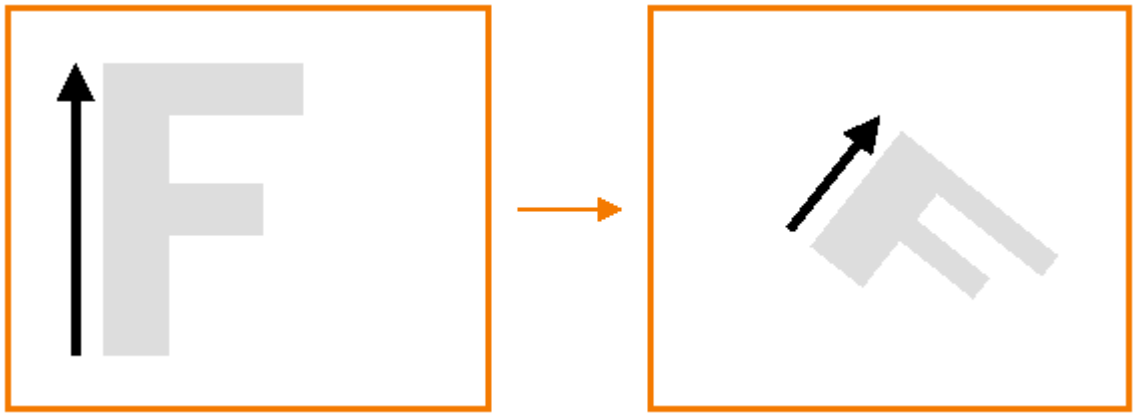
- What happens to the “F” ?



Rotation !

Warping with One Line Pair (cont.): Beier-Neeley

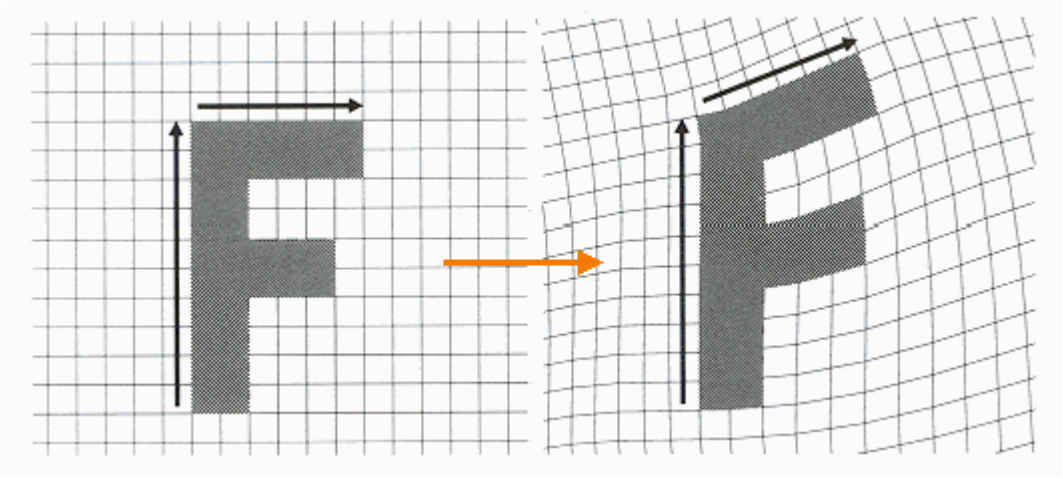
- What happens to the “F” ?



In general, similarity transformations

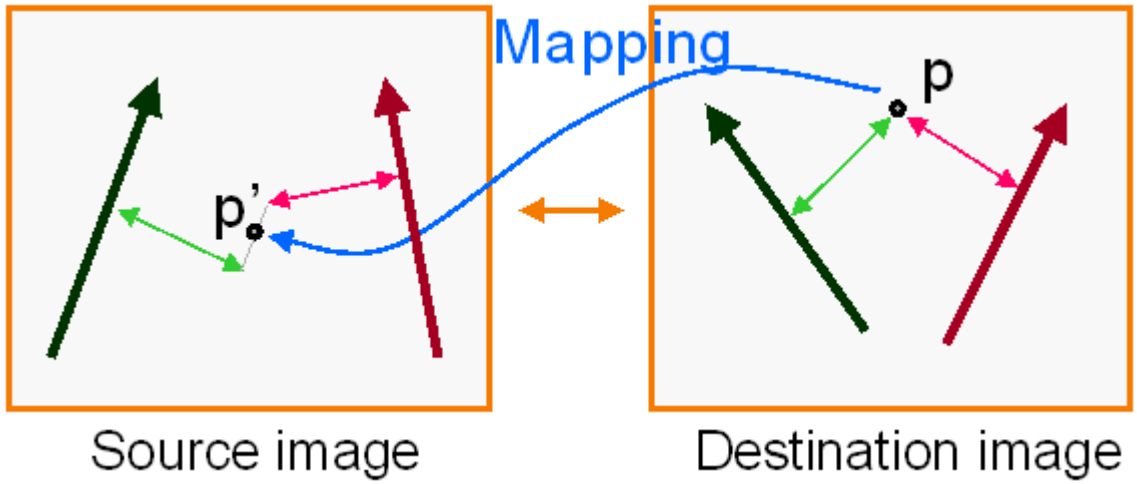
Warping with Multiple Line Pairs: Beier-Neeley

- Use weighted combination of points defined each pair of corresponding lines



Warping with Multiple Line Pairs: Beier-Neeley

- Use weighted combination of points defined by each pair corresponding lines



p' is a weighted average

Weighting Effect of Each Line Pair: Beier-Neeley

- To weight the contribution of each line pair

$$weight[i] = \left(\frac{length[i]^p}{a + dist[i]} \right)^b$$

where

- $length[i]$ is the length of $L[i]$
- $dist[i]$ is the distance from X to $L[i]$
- a, b, p are constants that control the warp

Warping Pseudocode: Beier-Neeley

```
foreach destination pixel p do
   psum = (0, 0)
   wsum = (0, 0)
   foreach line L[i] in destination do
        p'[i] = p transformed by (L[i], L'[i])
        psum = psum + p'[i] * weight[i]
        wsum += weight[i]
   end
   p' = psum / wsum
   destination(p) = source(p')
end
```