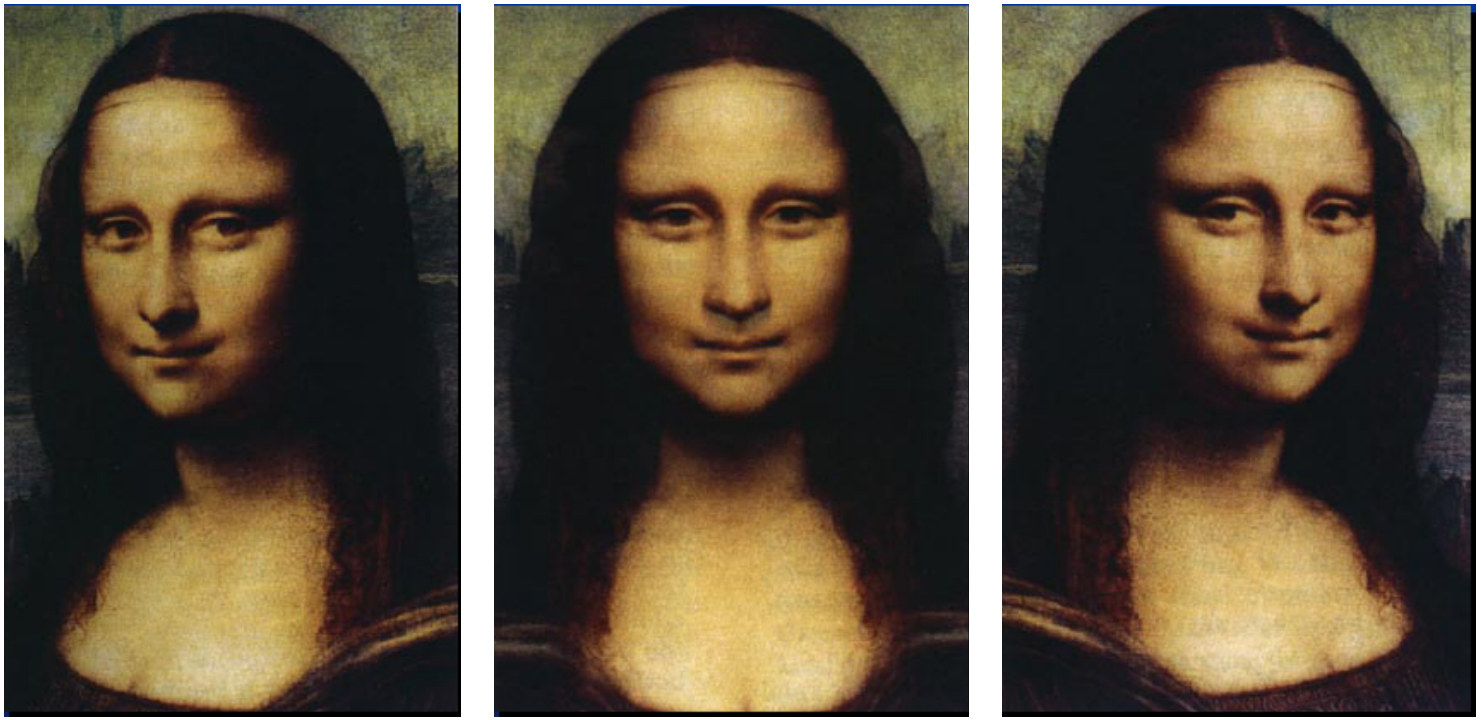
# *Interactive Computer Graphics: Lecture 14*

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*

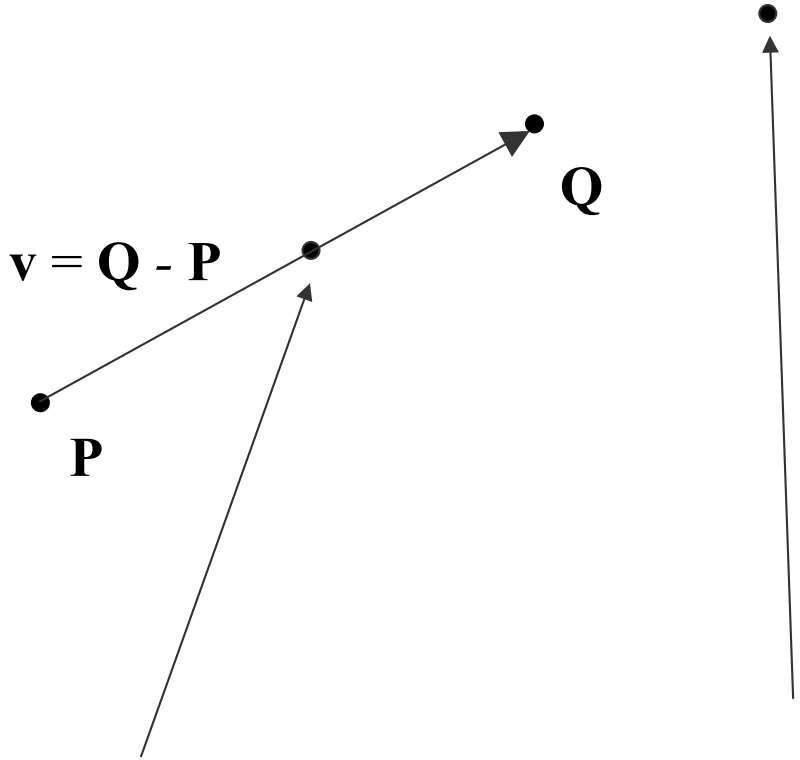
- The aim is to find "an average" between two objects
  - Not an average of two images of objects…

    …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

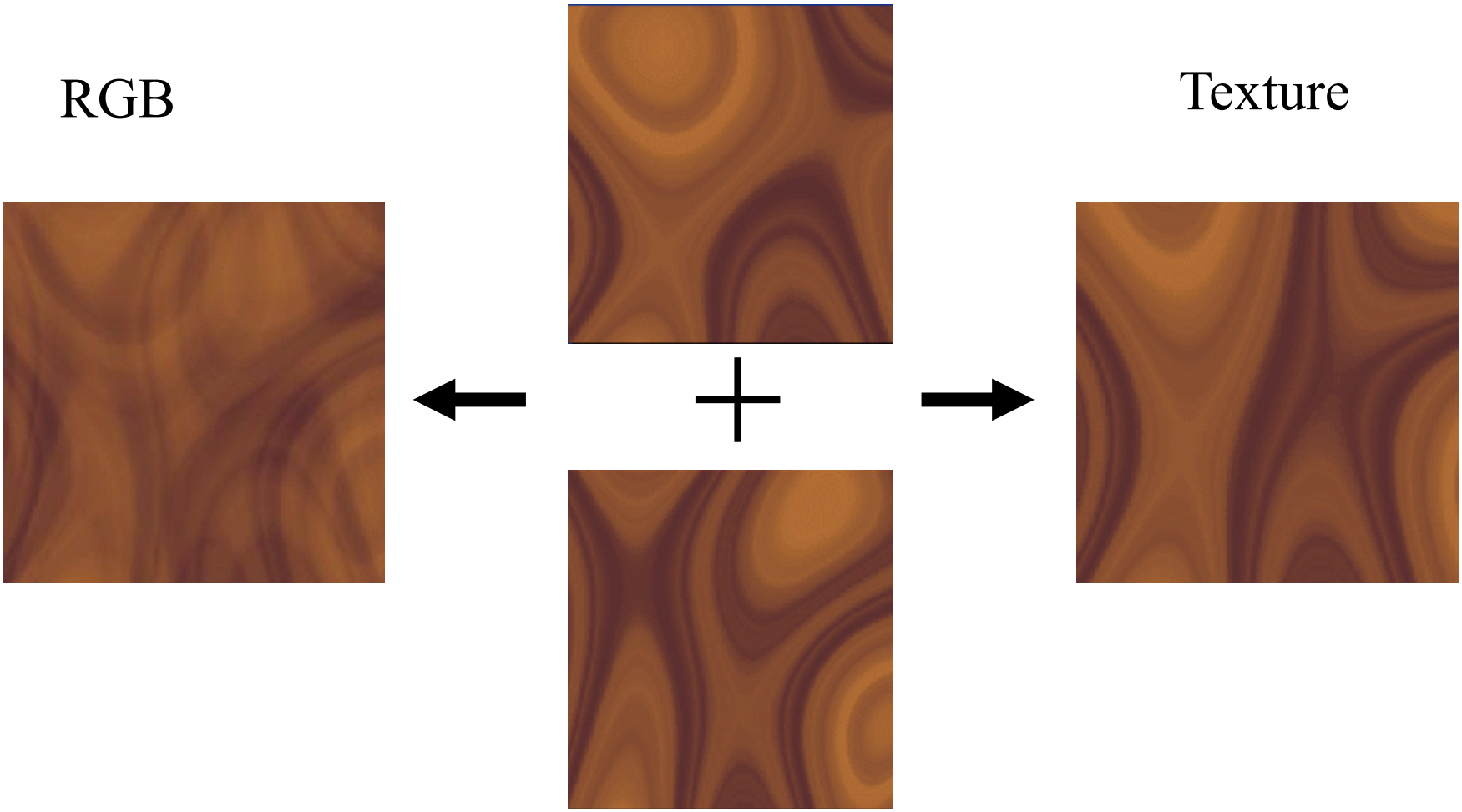
# *Image blending: Cross-dissolve*

- 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*

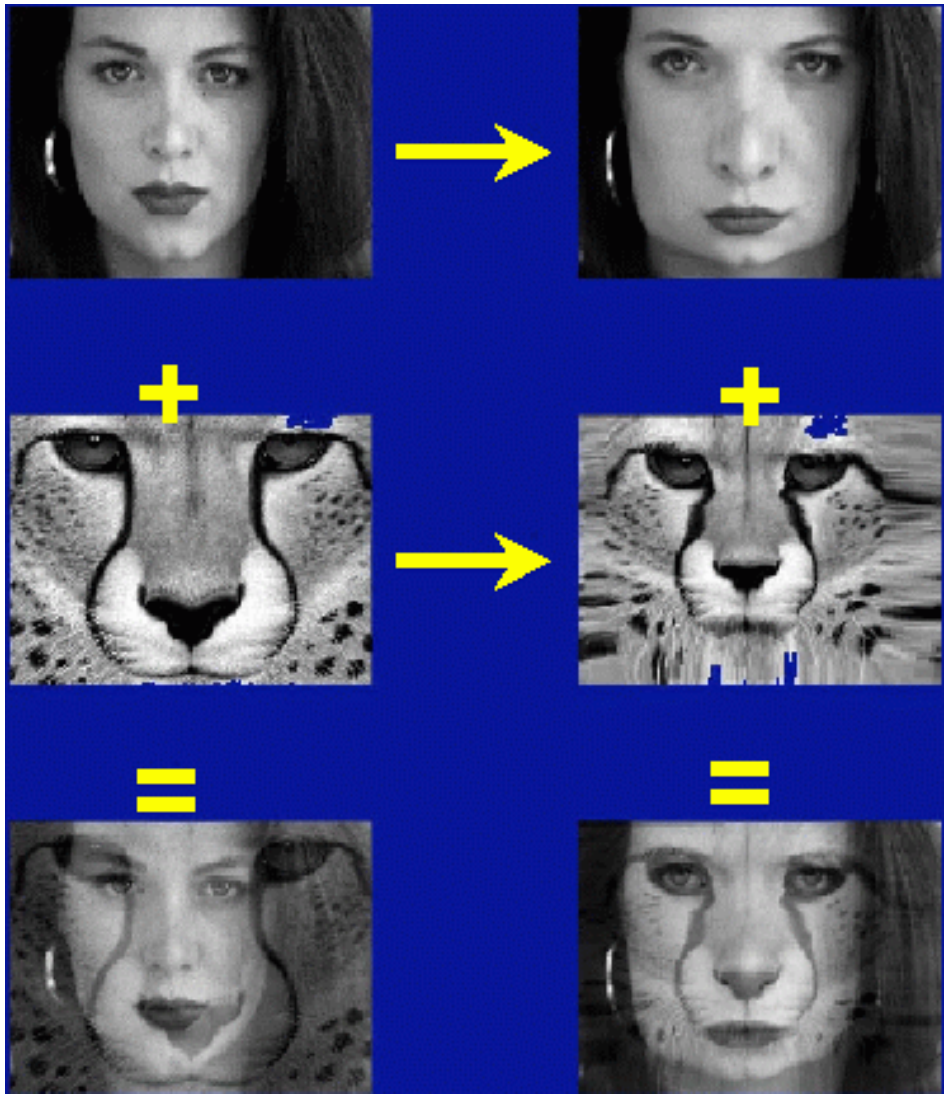



- But what is the images are not aligned?

# *Morphing using warping and cross-dissolve*


- Align first, then cross-dissolve

# *Morphing = (warping)$^2$ + blending*



# *Morphing*

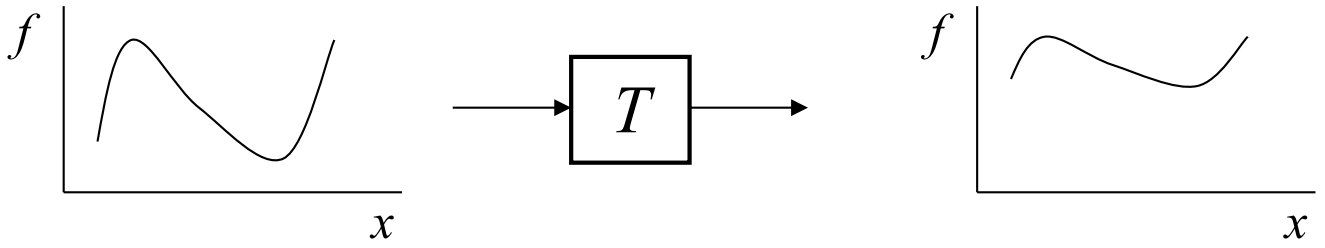
```
GenerateAnimation(Image0, Image1)
begin
        foreach intermediate frame time t do
                Warp0 = WarpImage(Image0, t)
                Warp1 = WarpImage(Image1, t)
                foreach pixel p in FinalImage do
                        Result(p) = (1-t)Warp0 + tWarp1
                end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

# *Morphing*

# *Image warping*

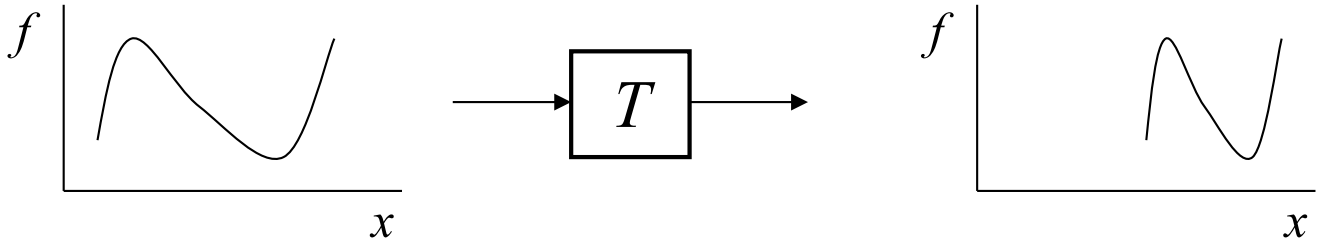
- Image filtering: change ***range*** of image

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




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

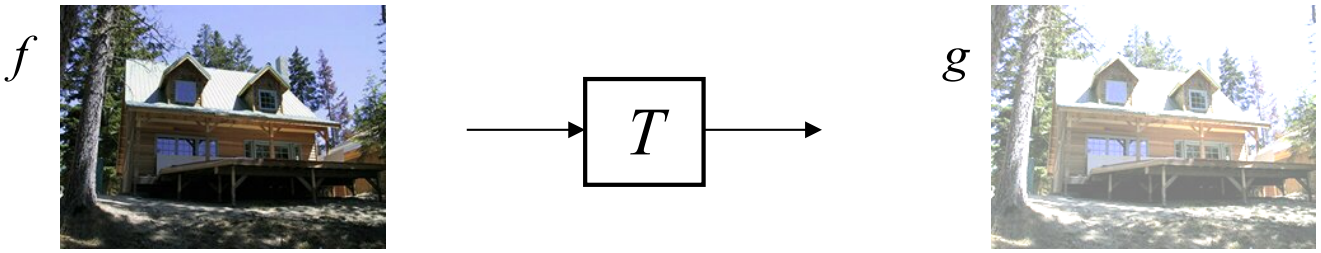
$$g(x) = f(T(x))$$



# *Image warping*

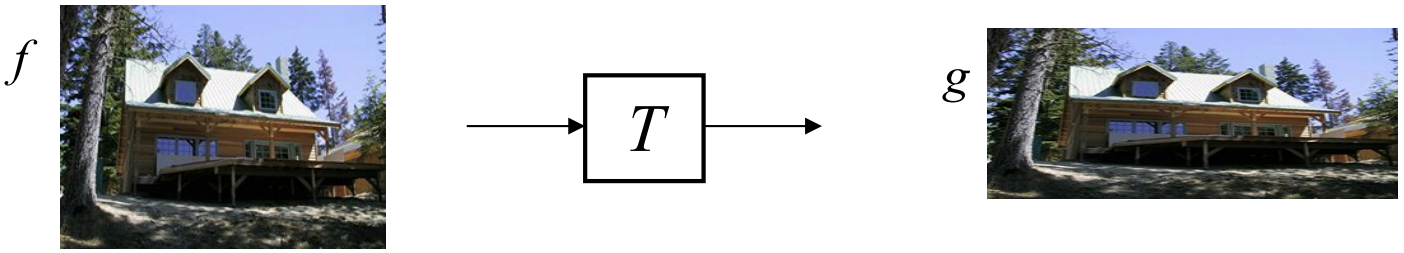
- Image filtering: change ***range*** of image

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

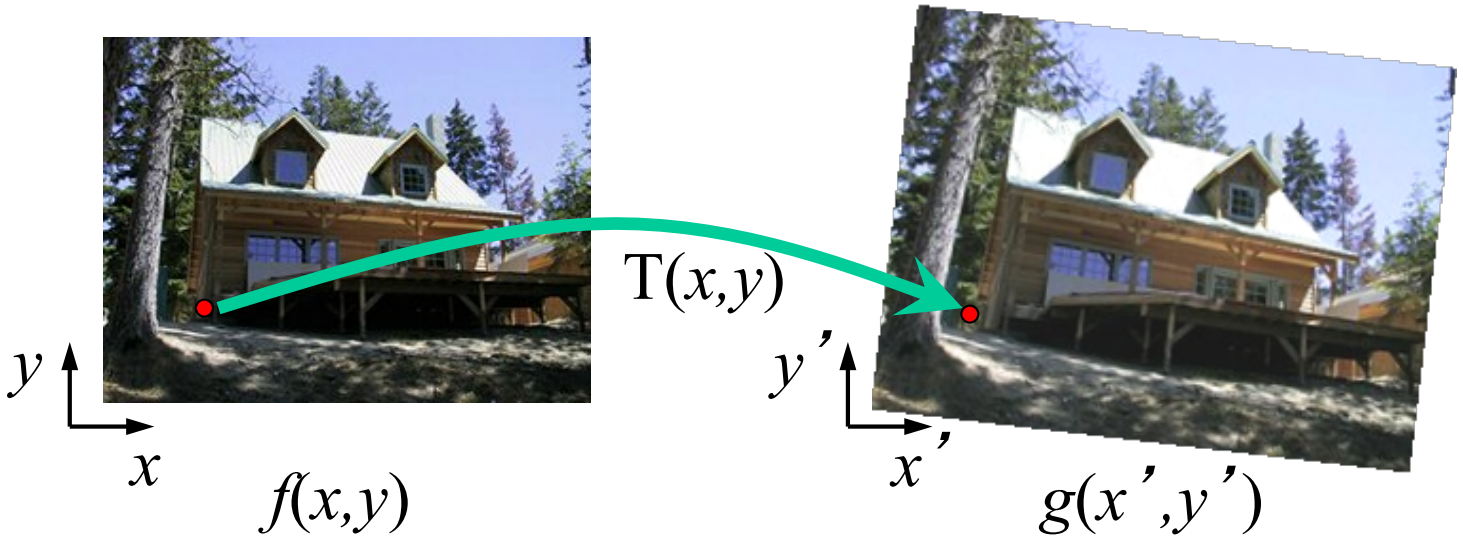



- Image warping: change ***domain*** of image

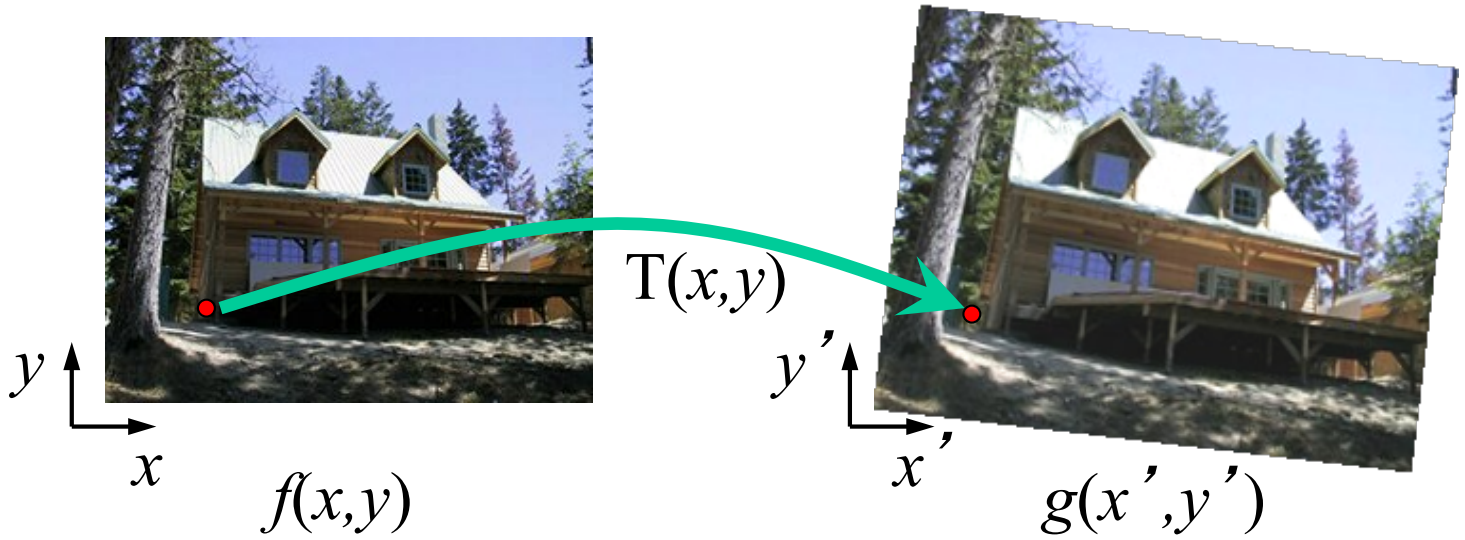
$$g(x) = f(T(x))$$



# Image warping

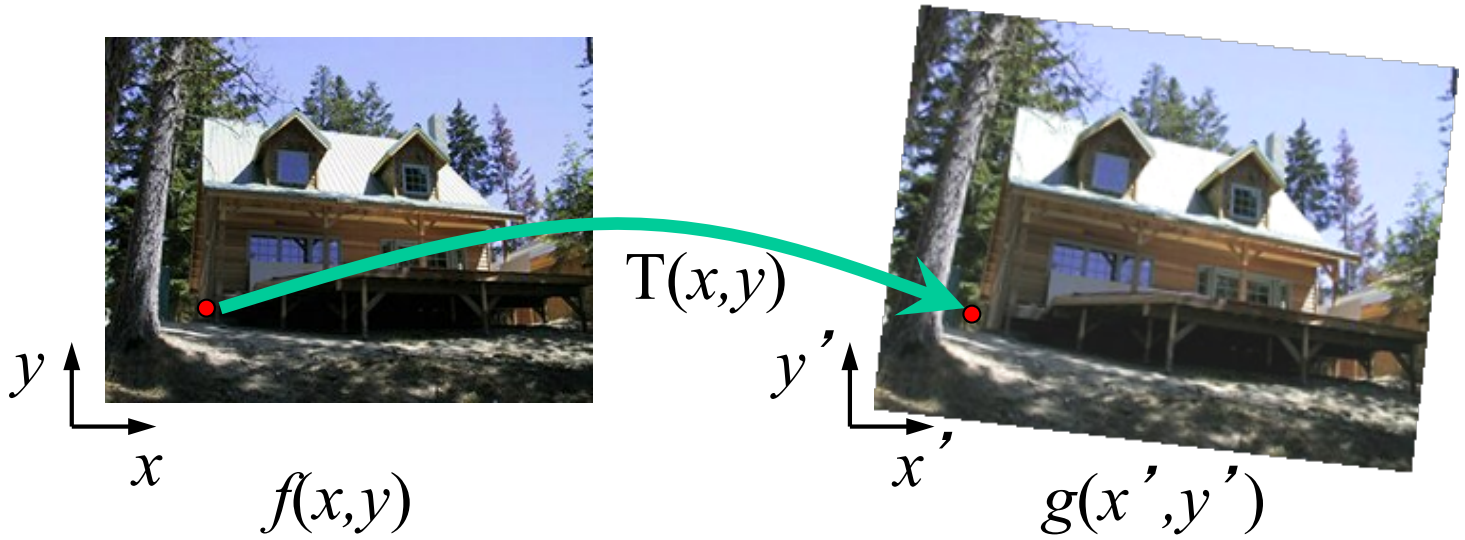


# *Image warping*



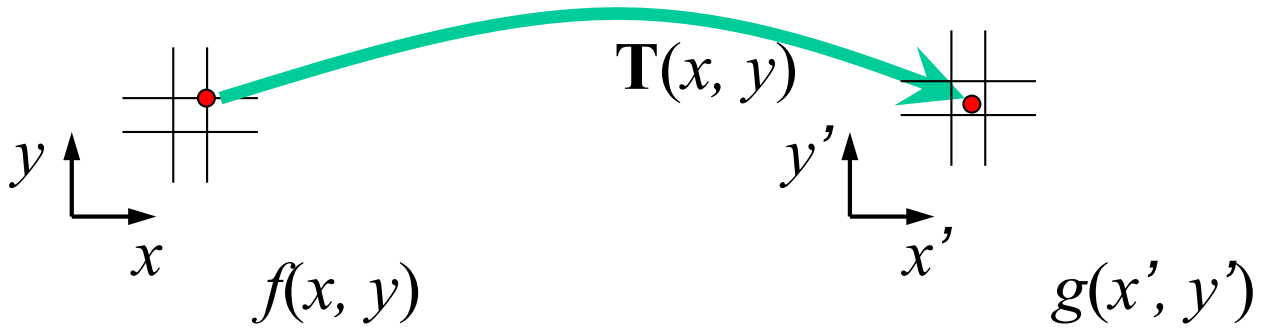

- Given a coordinate transform $(x',y') = \mathbf{T}(x, y)$ and a source image $f(x, y)$, how do we compute a transformed image $g(x', y') = f(\mathbf{T}(x, y))$?

# 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

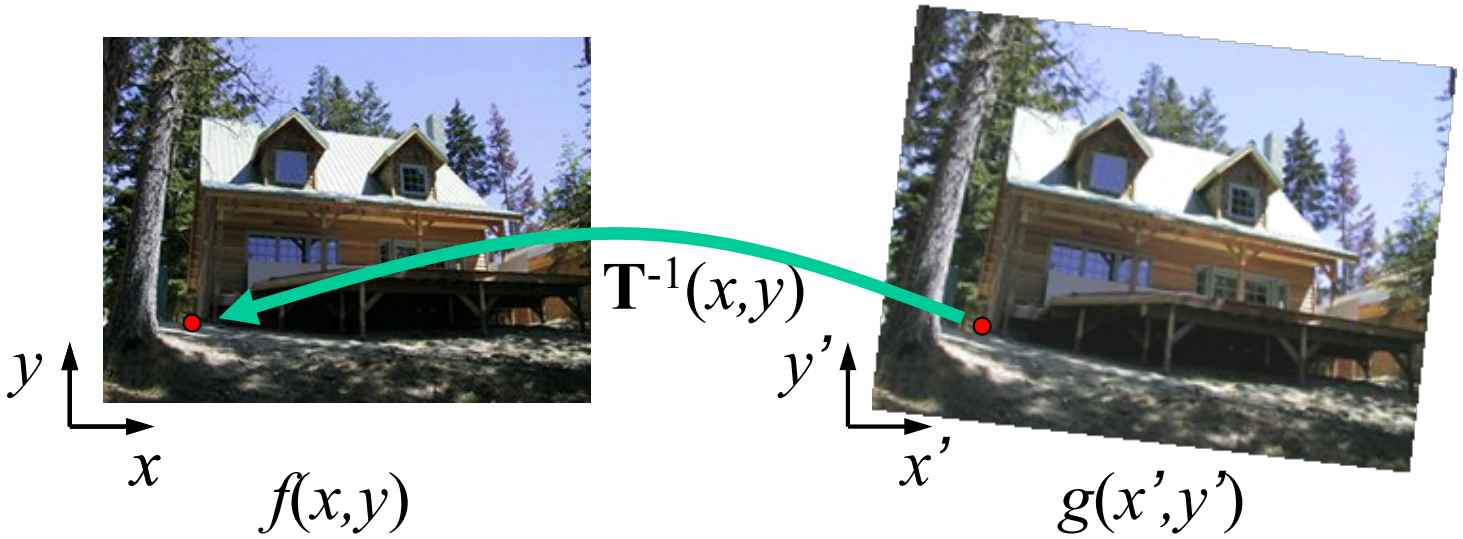



- Given a coordinate transform $(x',y') = \mathbf{T}(x, y)$ and a source image $f(x, y)$, how do we compute a transformed image $g(x', y') = f(\mathbf{T}(x, y))$?
- Send each pixel $f(x,y)$ to its corresponding location $(x',y') = \mathbf{T}(x, y)$ in the second image

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

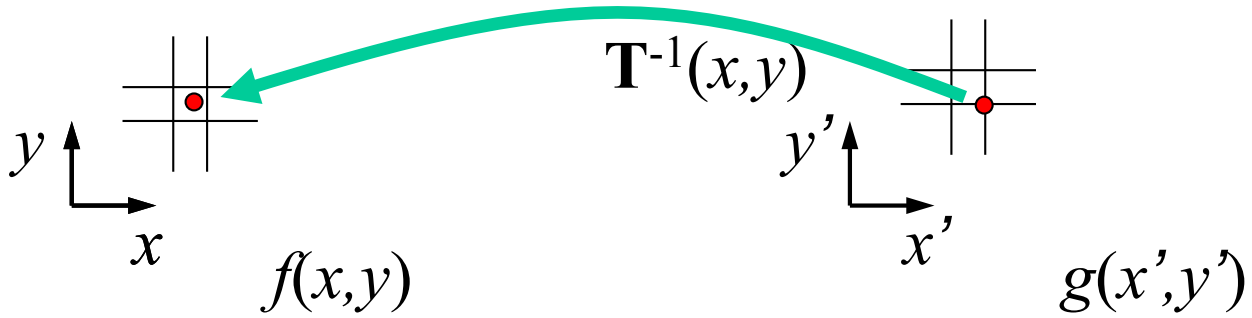
  A: distribute color among neighboring pixels $(x',y')$

# *Inverse warping*



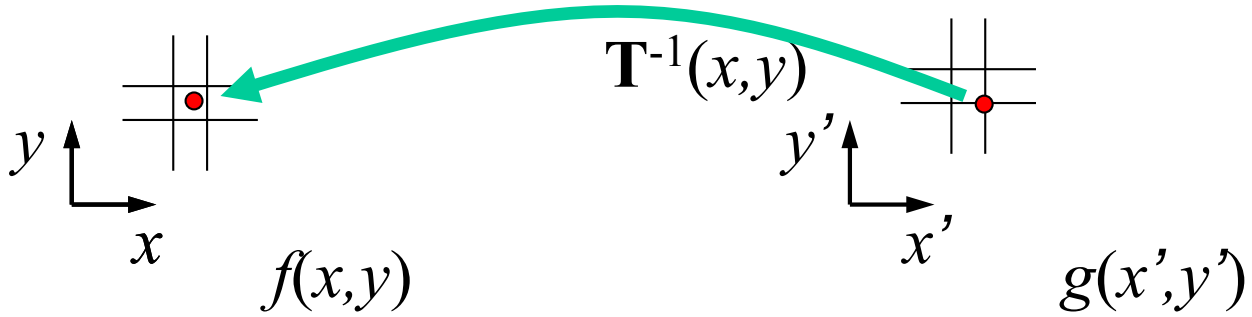

- Get each pixel $g(x', y')$ from its corresponding location $(x, y) = \mathbf{T}^{-1}(x', y')$ in the first image

# *Inverse warping*



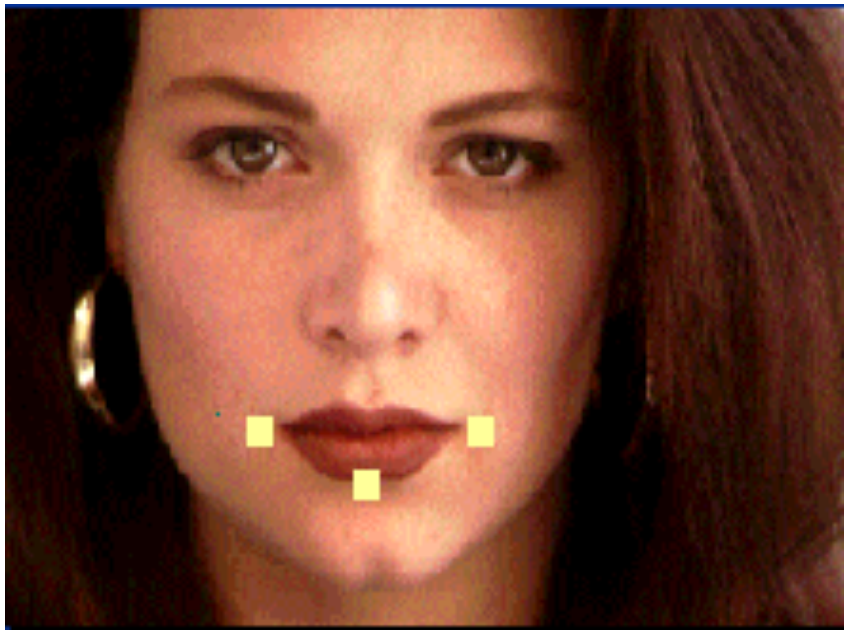

- Get each pixel $g(x', y')$ from its corresponding location $(x, y) = \mathbf{T}^{-1}(x', y')$ in the first image
  Q: what if pixel comes from "between" two pixels?

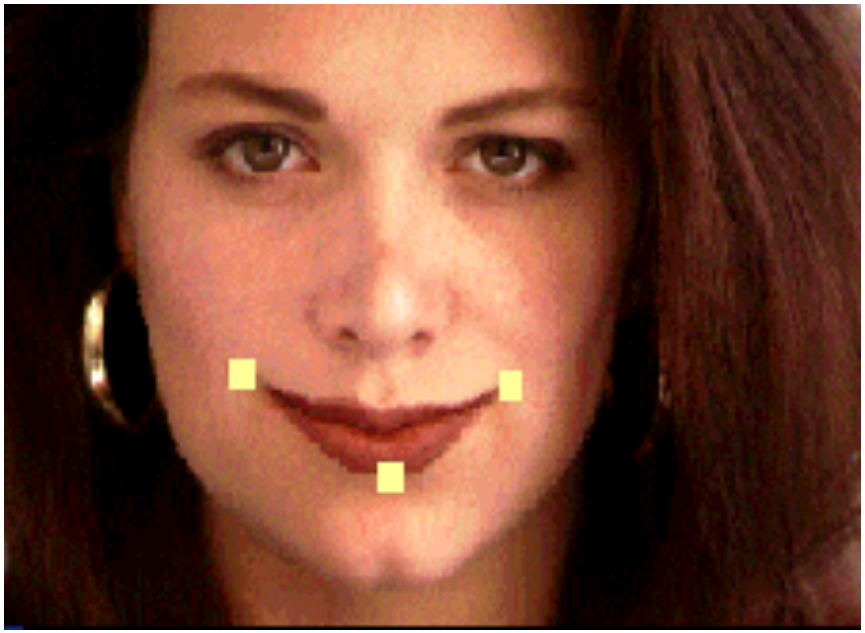
# *Inverse warping*



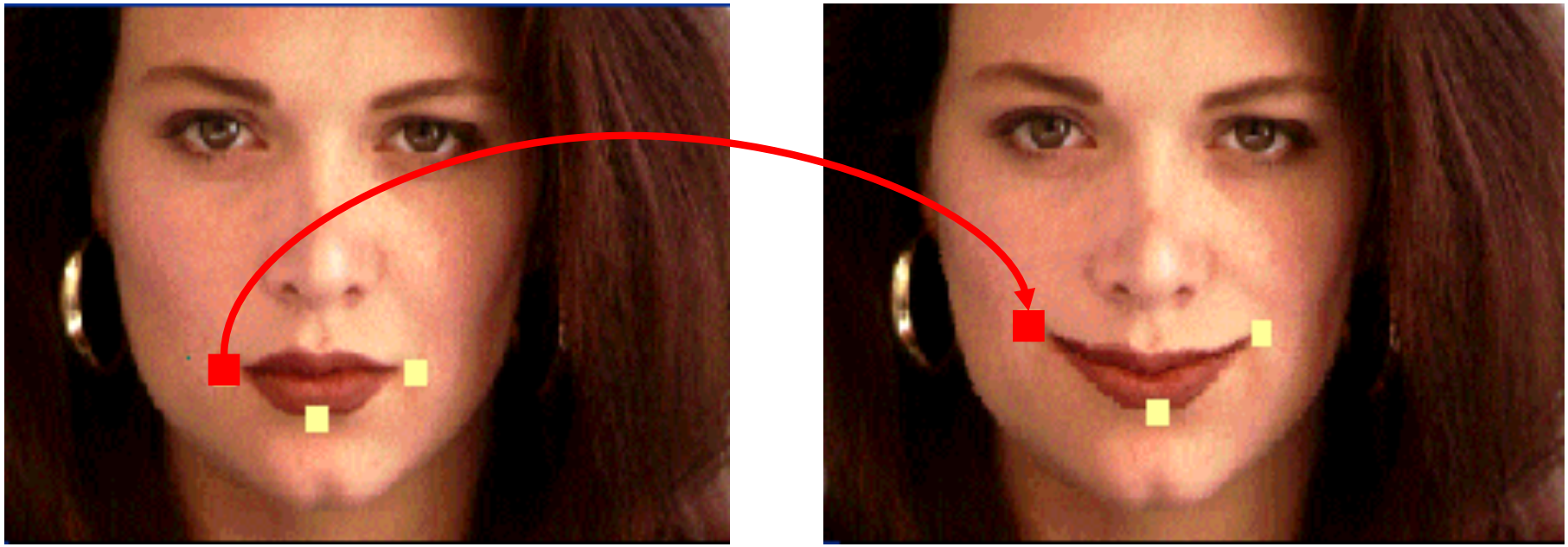

- Get each pixel $g(x', y')$ from its corresponding location $(x, y) = \mathbf{T}^{-1}(x', y')$ in the first image
  Q: what if pixel comes from "between" two pixels?
  A: Interpolate color value from neighbors
  – nearest neighbor, bilinear, Gaussian, bicubic

# *Image warping: Correspondences*

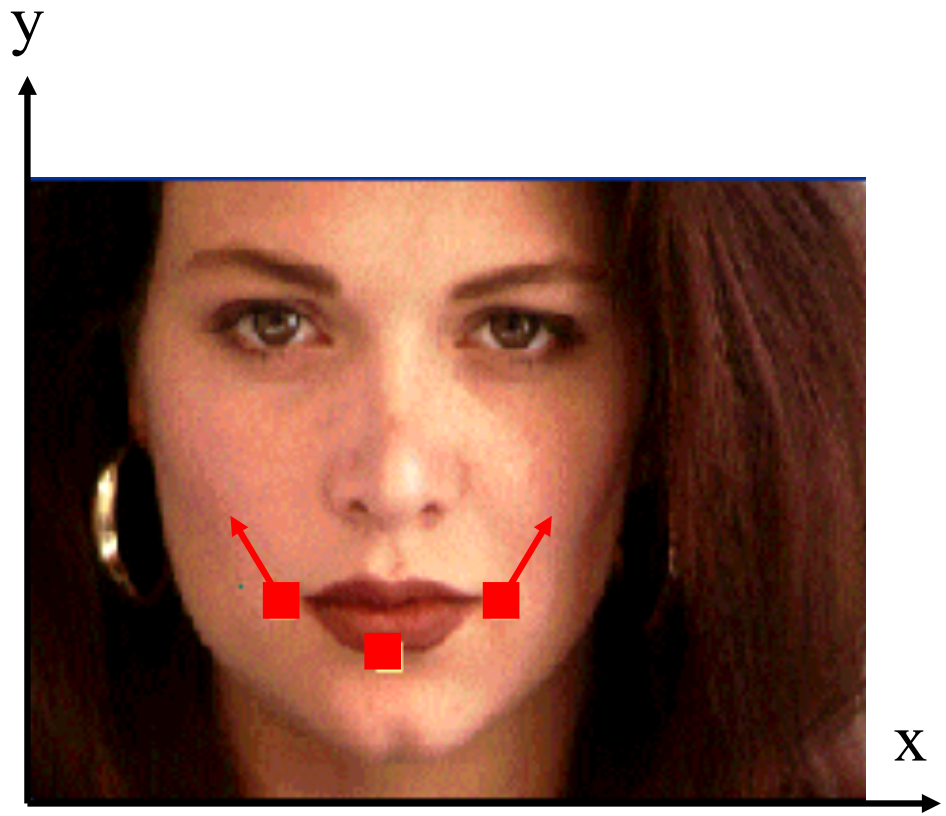




# *Image warping: Correspondences*

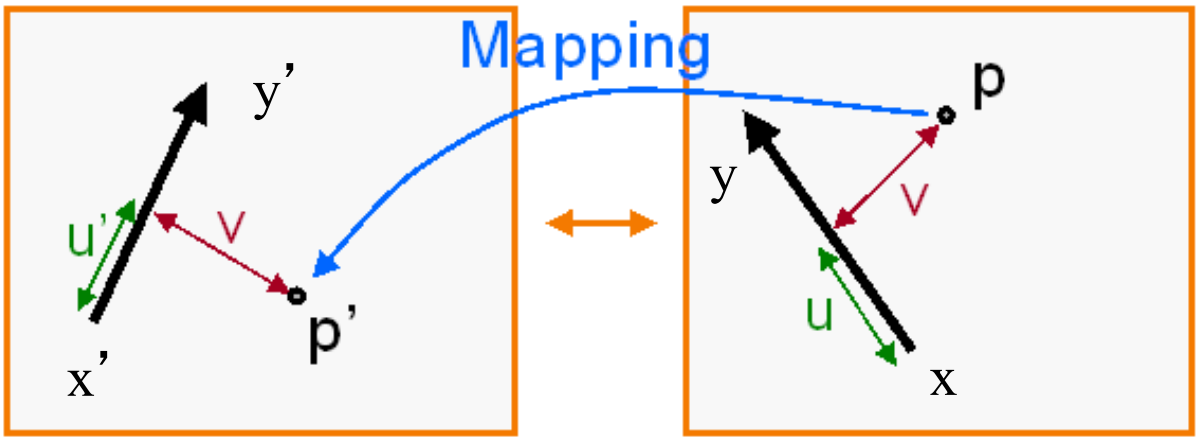


# *Image warping: Correspondences*



# *Feature-Based Warping: Beier-Neeley*

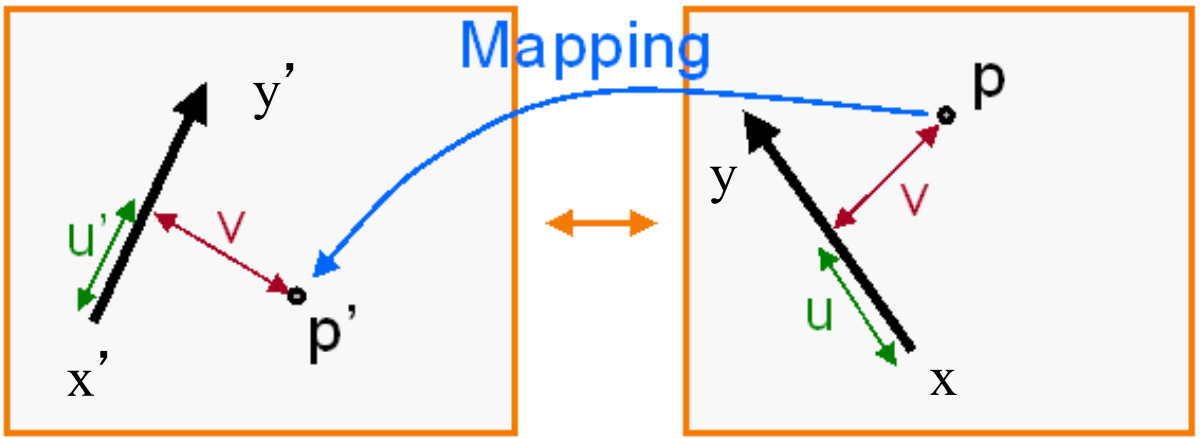
- Beier & Neeley use pairs of lines to specify warp
  - Given **p** in destination image, where is **p'** in source image?



# Feature-Based Warping: Beier-Neeley

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

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

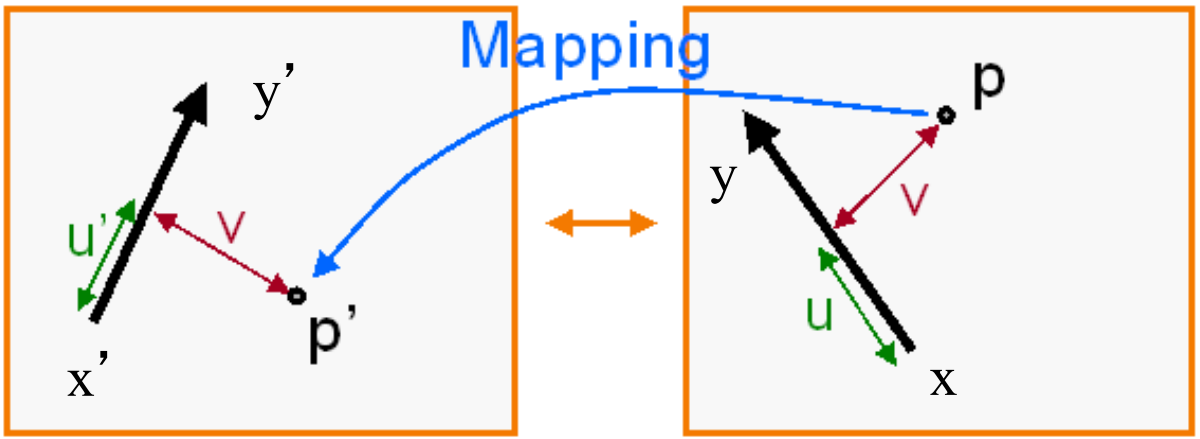



u is a fraction

v is a length (in pixels)

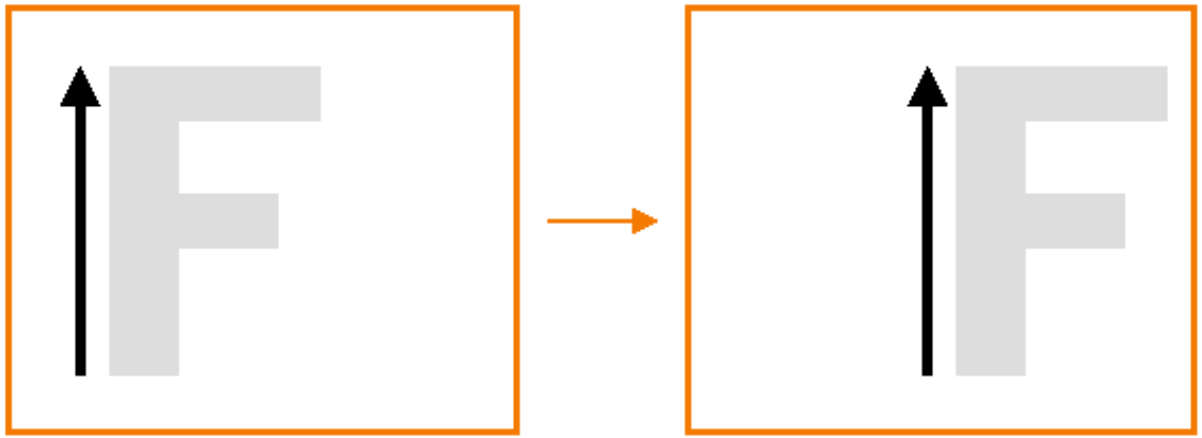
# *Feature-Based Warping: Beier-Neeley*

- 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’)



# *Warping with One Line Pair: Beier-Neeley*

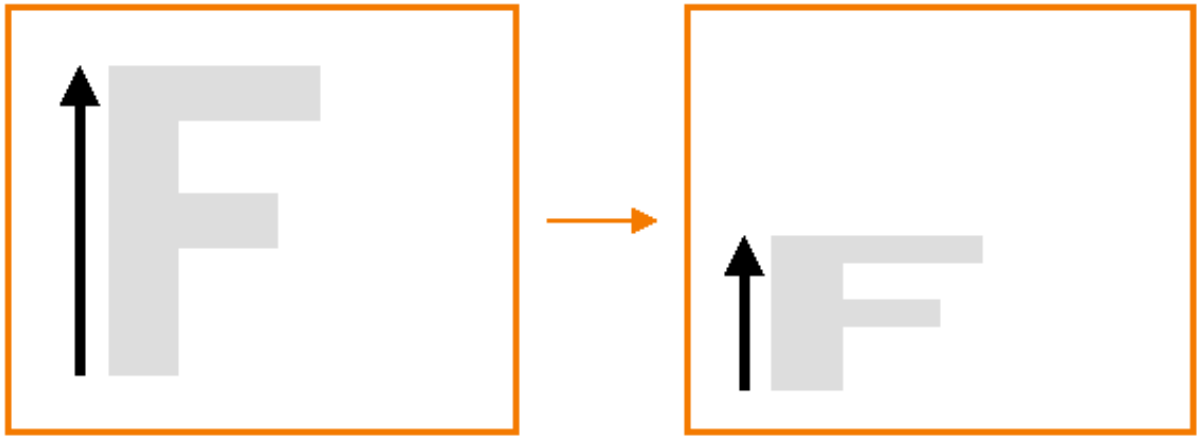
- What happens to the “F” ?



Translation !

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

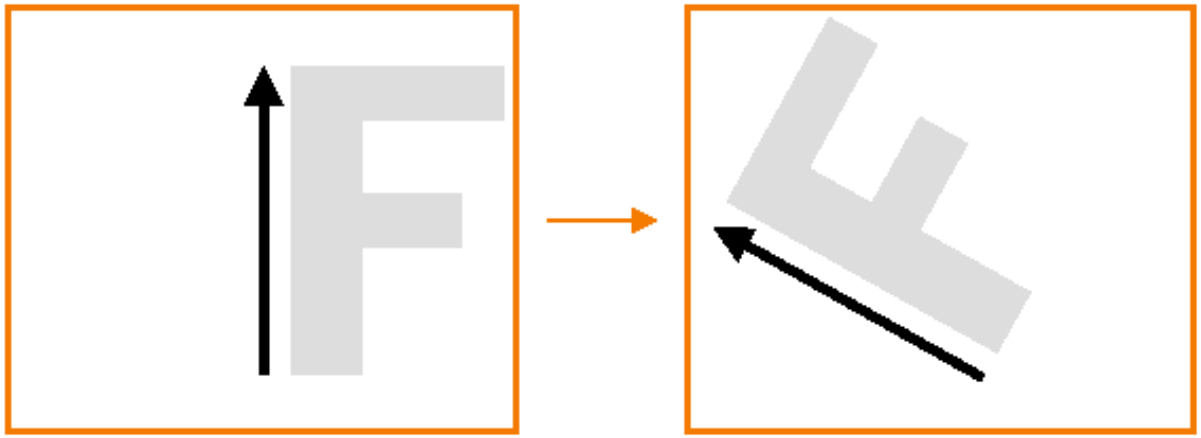
- What happens to the “F” ?



Scale !

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

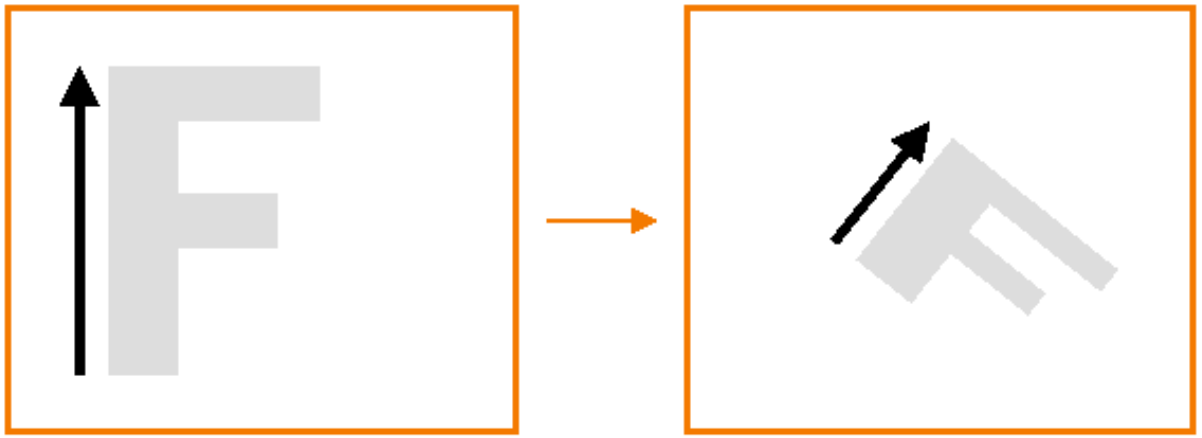
- What happens to the “F” ?



Rotation !

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

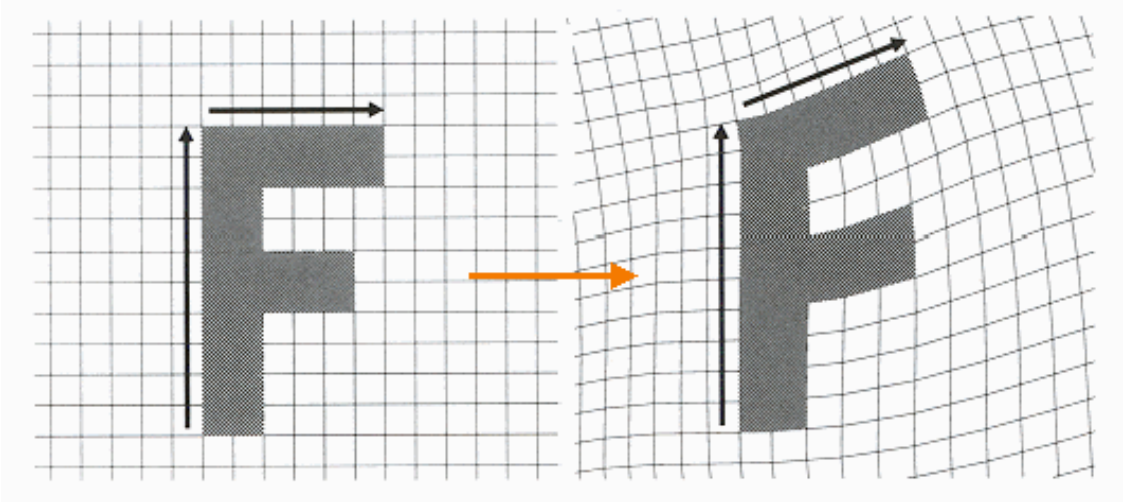
- What happens to the “F” ?



In general, similarity transformations

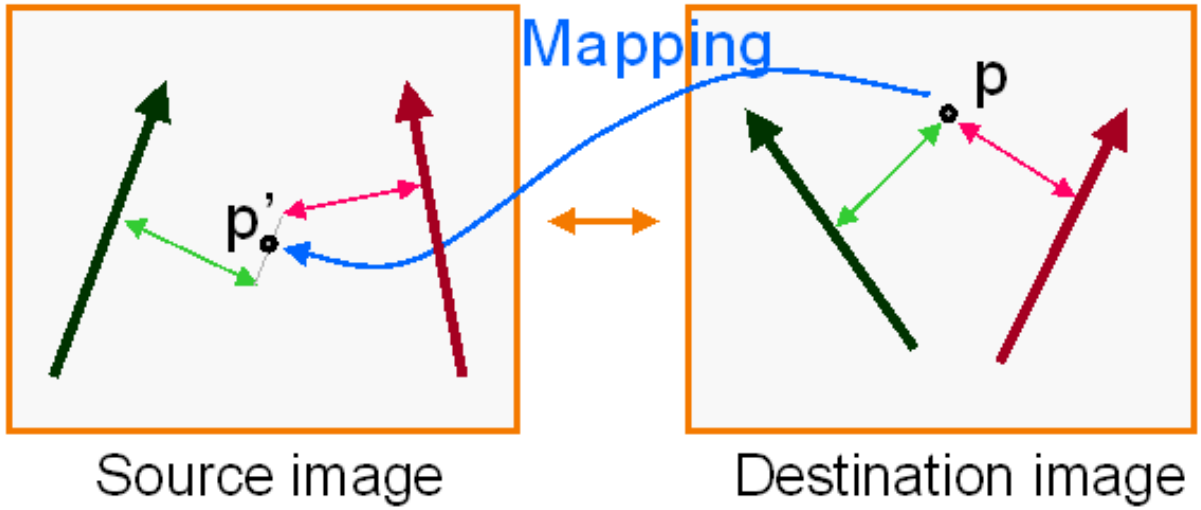
# *Warping with Multiple Line Pairs: Beier-Neeley*

- 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

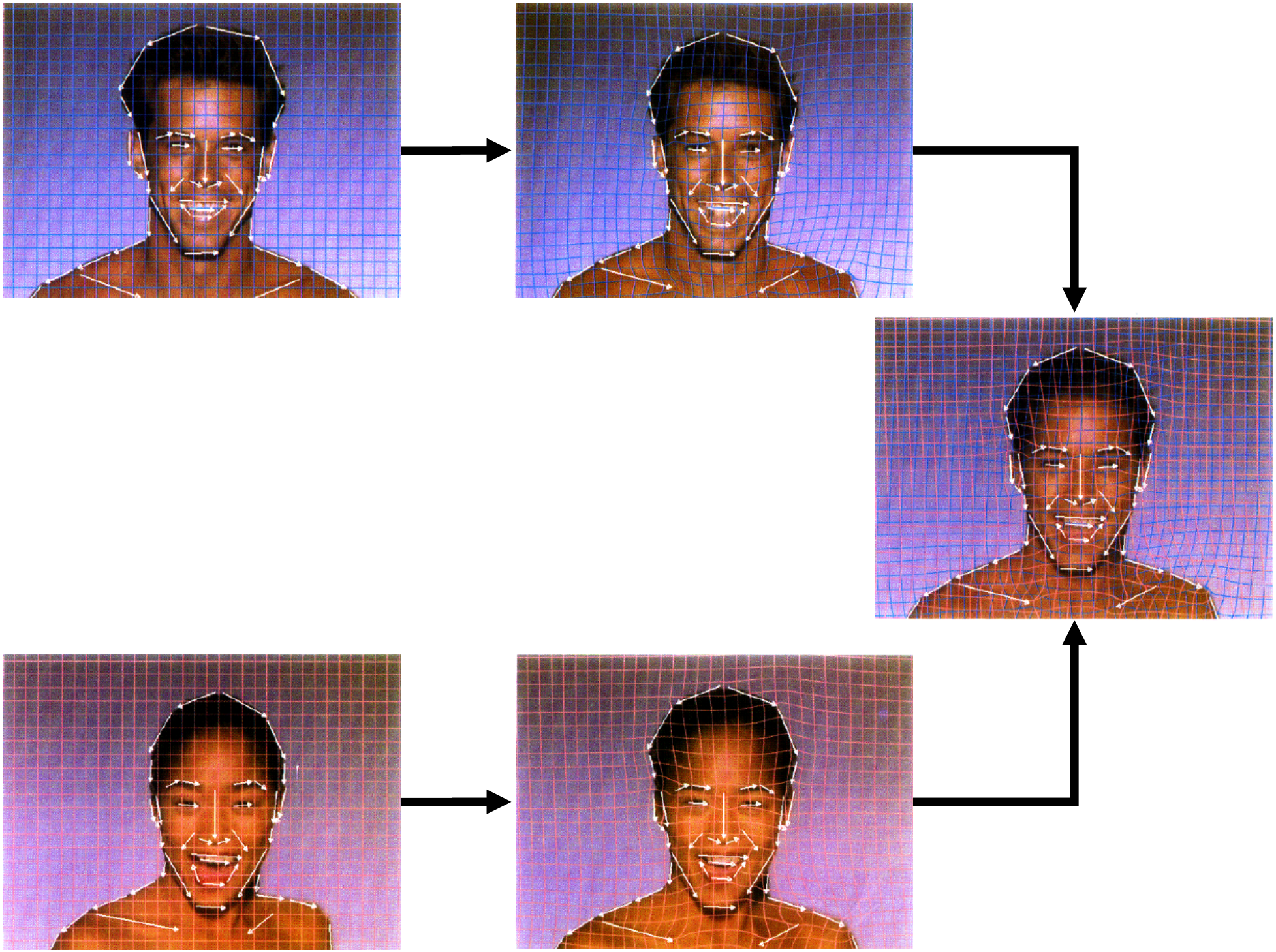
$$weight[i] = \left( \frac{length[i]^p}{a + dist[i]} \right)^b$$

where

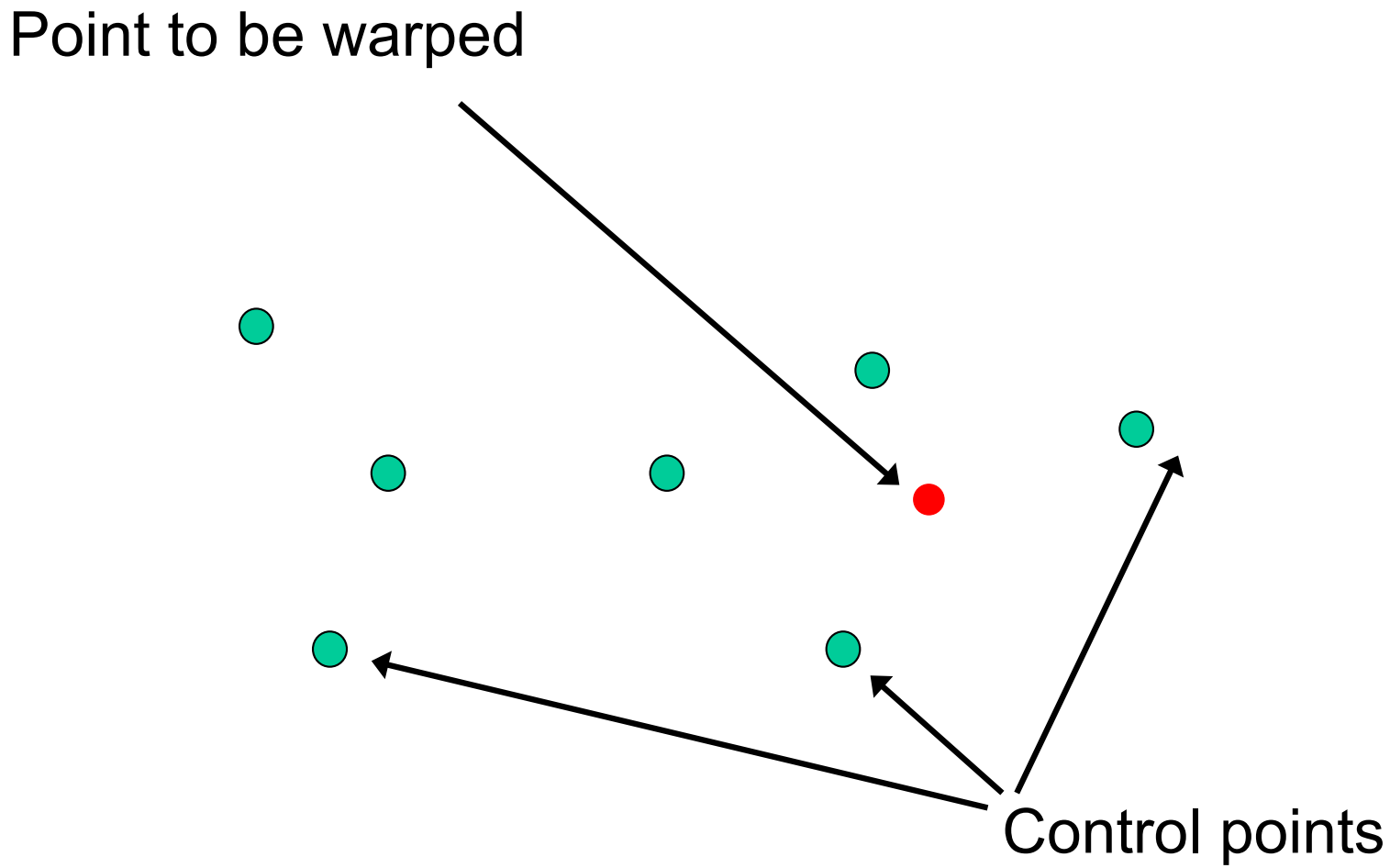
- $length[i]$ is the length of $L[i]$
- $dist[i]$ is the distance from $X$ to $L[i]$
- $a, b, p$ are constants that control the warp

# *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
```

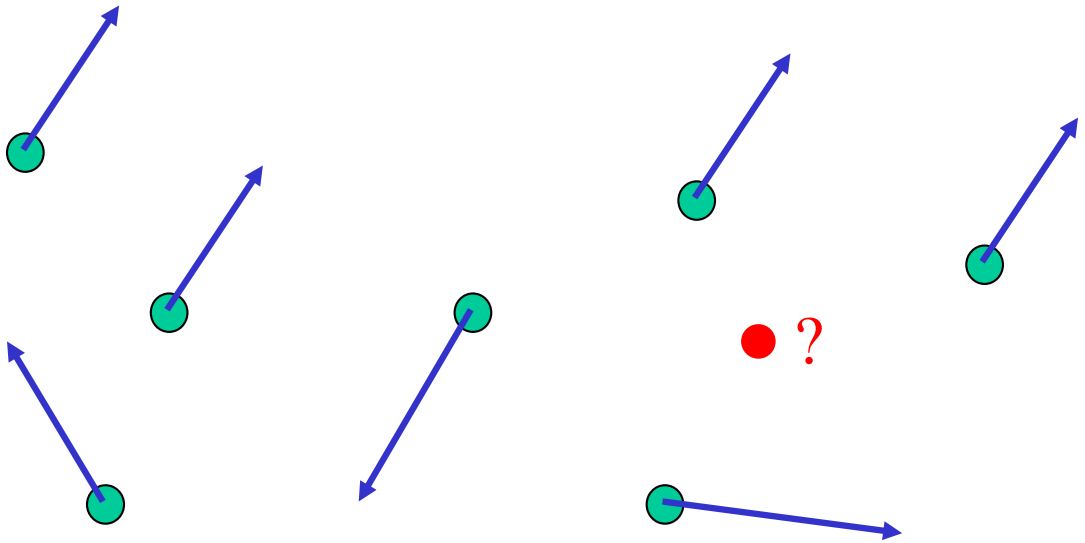


# *Warping using control points*



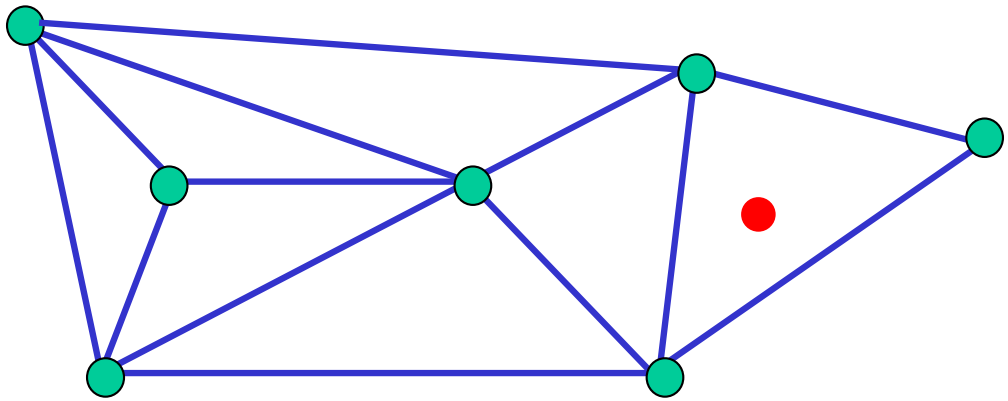
# *Warping using control points*

- For each control point we have a displacement vector
- How do we interpolate the displacement at a pixel?



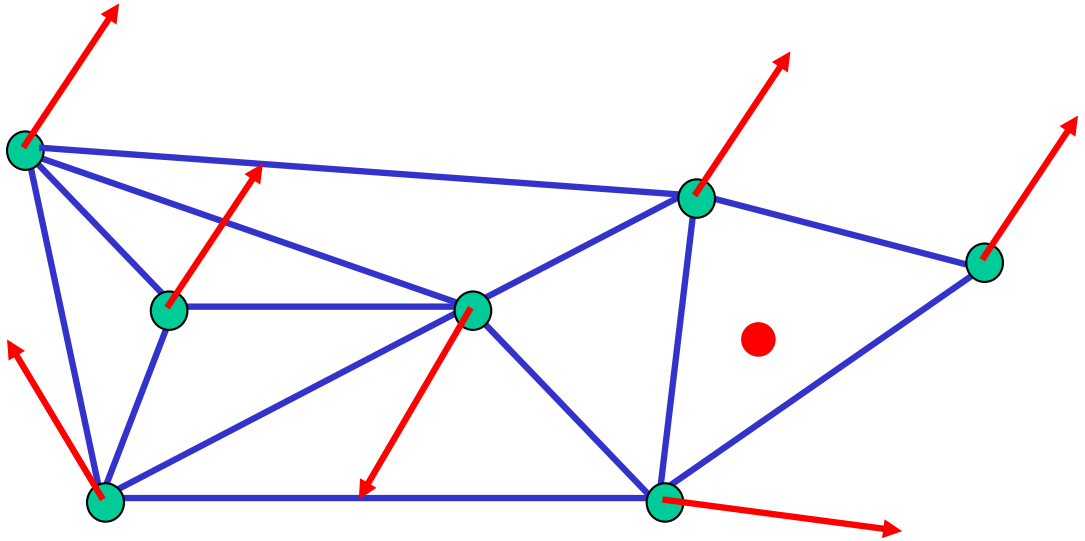
# *Warping using control points: Piecewise affine*

- Partition the convex hull of the control points into a set of triangles



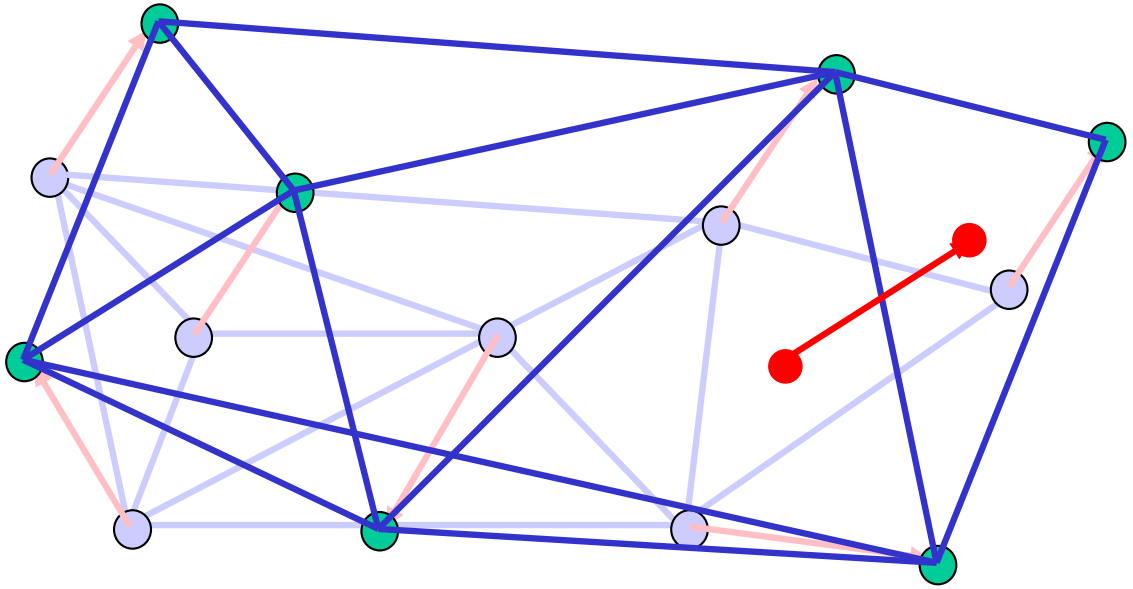
# *Warping using control points: Piecewise affine*

- Partition the convex hull of the control points into a set of triangles



# *Warping using control points: Piecewise affine*

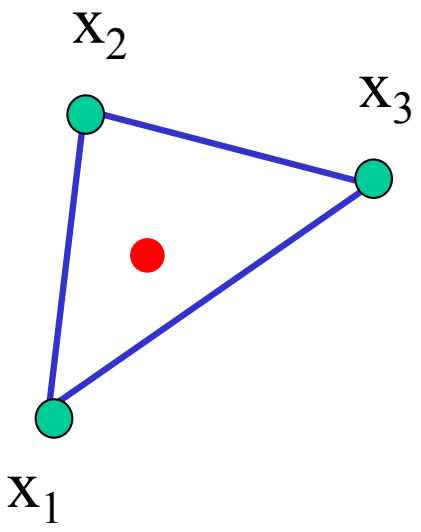
- Partition the convex hull of the control points into a set of triangles



# *Warping using control points: Piecewise affine*

- Find triangle which contains point **p** and express in terms of the vertices of the triangle:

$$\mathbf{p} = \mathbf{x}_1 + \alpha(\mathbf{x}_2 - \mathbf{x}_1) + \beta(\mathbf{x}_3 - \mathbf{x}_1)$$



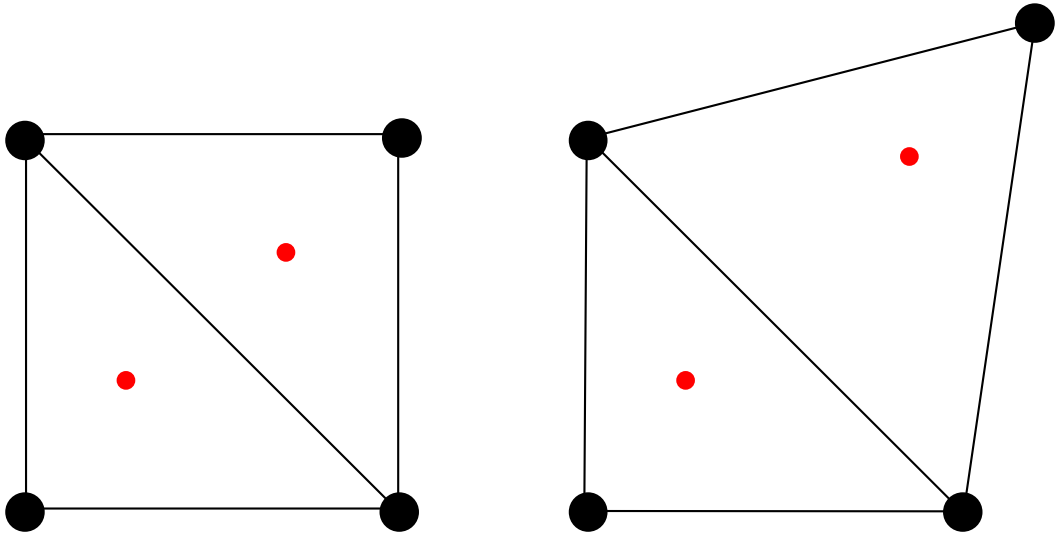
# *Warping using control points: Piecewise affine*

- Or $\mathbf{p} = \gamma\mathbf{x}_1 + \alpha\mathbf{x}_2 + \beta\mathbf{x}_3$ with $\gamma = 1 - (\alpha + \beta)$

- Under the affine transformation this point simply maps to

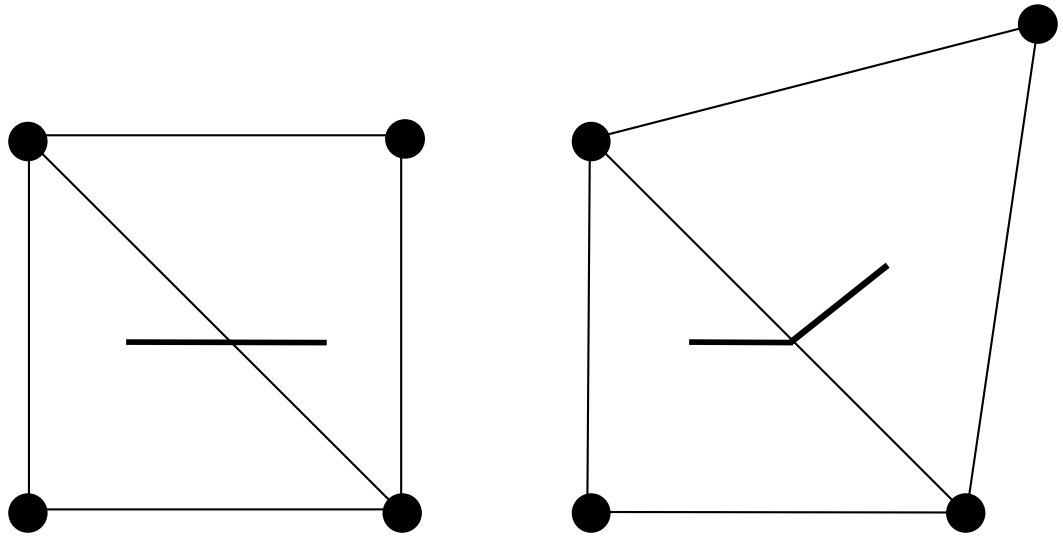
$$\mathbf{p}' = \gamma\mathbf{x}_1' + \alpha\mathbf{x}_2' + \beta\mathbf{x}_3'$$

# *Warping using control points: Piecewise affine*



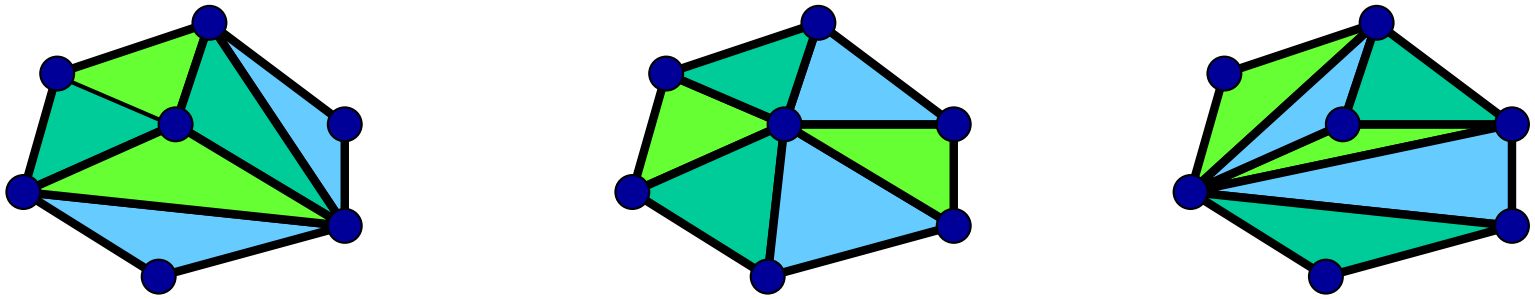
# *Warping using control points: Piecewise affine*

- Problem: Produces continuous deformations, but the deformation may not be smooth. Straight lines can be kinked across boundaries between triangles



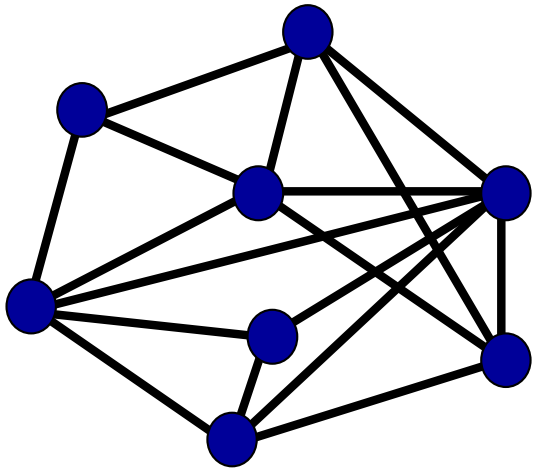
# *Triangulations*

- A *triangulation* of set of points in the plane is a *partition* of the convex hull to triangles whose vertices are the points, and do not contain other points.
- There are an exponential number of triangulations of a point set.



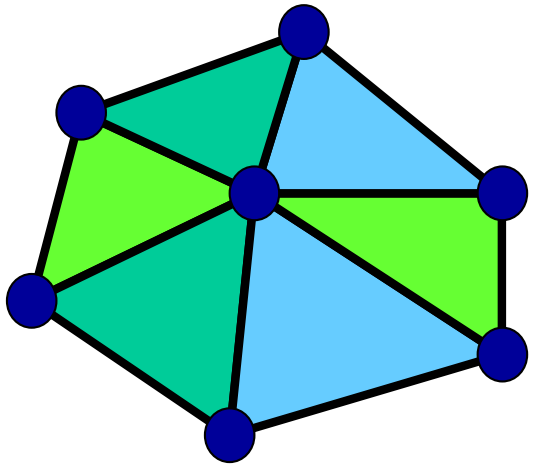
# *An O(n$^3$) Triangulation Algorithm*

- Repeat until impossible:
  - Select two sites.
  - If the edge connecting them does not intersect previous edges, keep it.

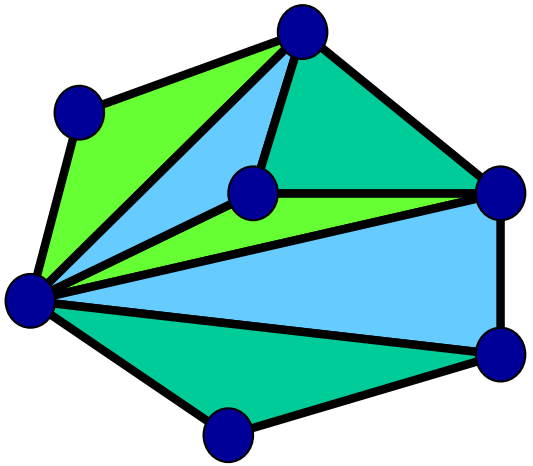


# *"Quality" Triangulations*

- Let $\alpha(T) = (\alpha_1, \alpha_2, .., \alpha_{3t})$ be the vector of angles in the triangulation $T$ in increasing order.
- A triangulation $T_1$ will be "better" than $T_2$ if $\alpha(T_1) > \alpha(T_2)$ lexicographically.
- The Delaunay triangulation is the "best"
  - Maximizes smallest angles



good



bad

# *Summary*

- Morphing = (warping)$^2$ + blending
- Blending
  - affects image attributes (intensity, colour, texture)
- Warping
  - affects geometry and is usally parameterized by controls
  - Beier-Neely: Controls are oriented lines
  - Piecewise affine warping: Controls are points (or landmarks)