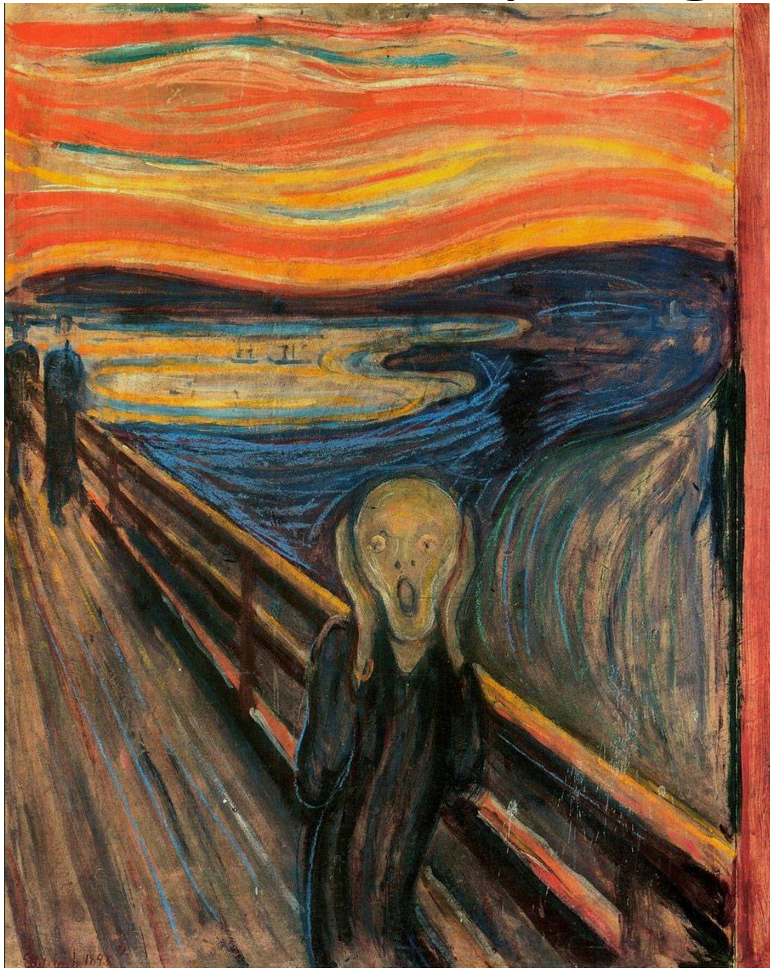
Review: Morphing and Warping



Edvard Munch, “The Scream”

CSC320: Introduction to Visual Computing
Michael Guerzhoy

Many slides borrowed
from Derek Hoeim, Alexei Efros

Homogeneous Coordinates

Q: How can we represent translation in matrix form?

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

Homogeneous Coordinates

Homogeneous coordinates

- represent coordinates in 2 dimensions with a 3-vector

$$\begin{bmatrix} x \\ y \end{bmatrix} \xrightarrow{\textbf{homogeneous coords}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Homogeneous Coordinates

Q: How can we represent translation in matrix form?

$$x' = x + t_x$$

$$y' = y + t_y$$

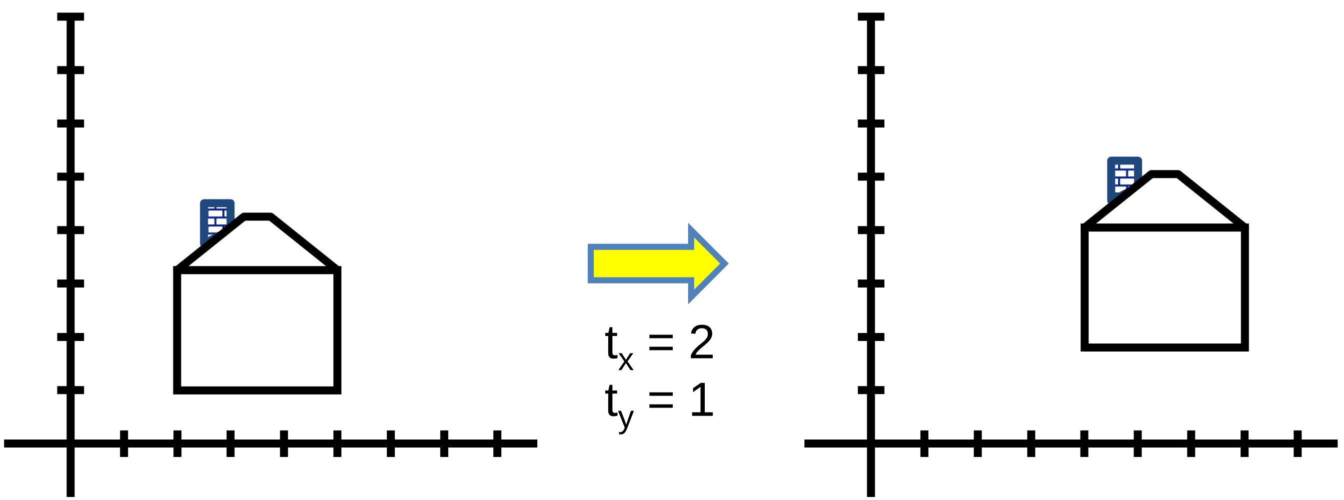
A: Using the rightmost column:

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

Translation Example

Homogeneous Coordinates

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



Homogeneous Coordinates

2D Points → Homogeneous Coordinates

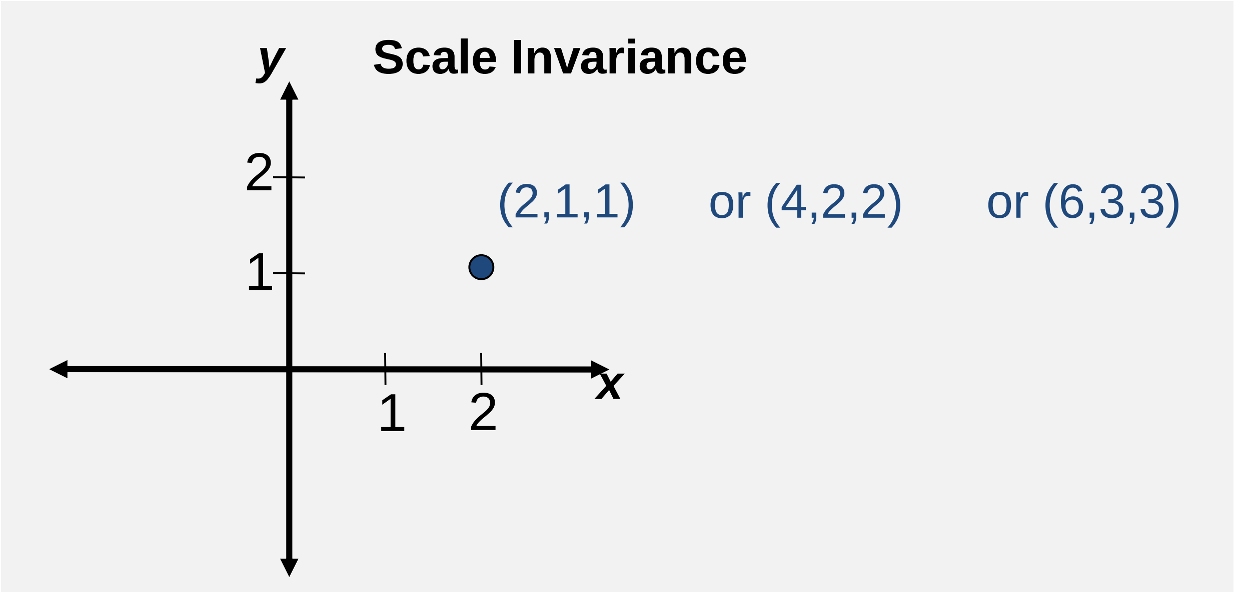
- Append 1 to every 2D point: (x y) → (x y 1)

Homogeneous coordinates → 2D Points

- Divide by third coordinate (x y w) → (x/w y/w)

Special properties

- Scale invariant: (x y w) = k * (x y w)
- (x, y, 0) represents a point at infinity
- (0, 0, 0) is not allowed



Basic 2D transformations as 3x3 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

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

Shear

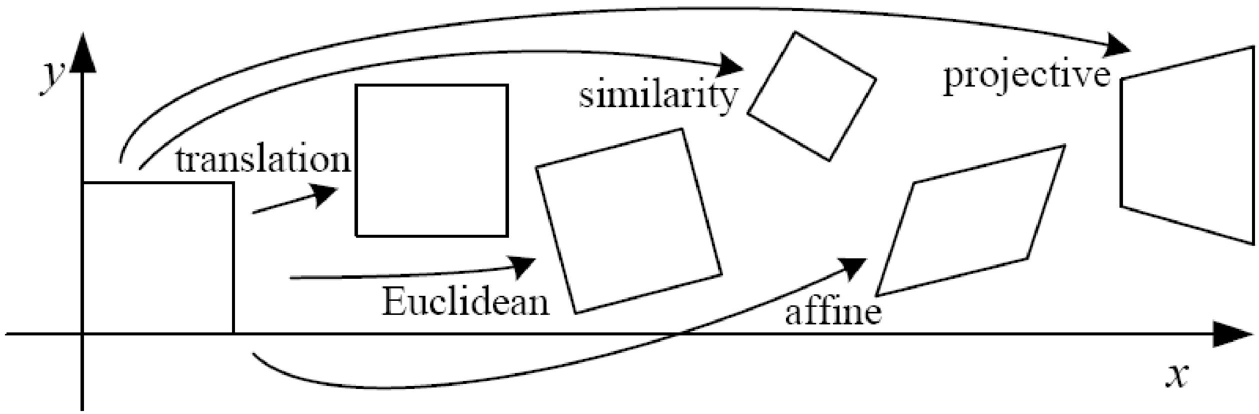
Matrix Composition

Transformations can be combined by matrix multiplication

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \left(\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

p' = $T(t_x,t_y)$ $R(\Theta)$ $S(s_x,s_y)$ **p**

2D image transformations

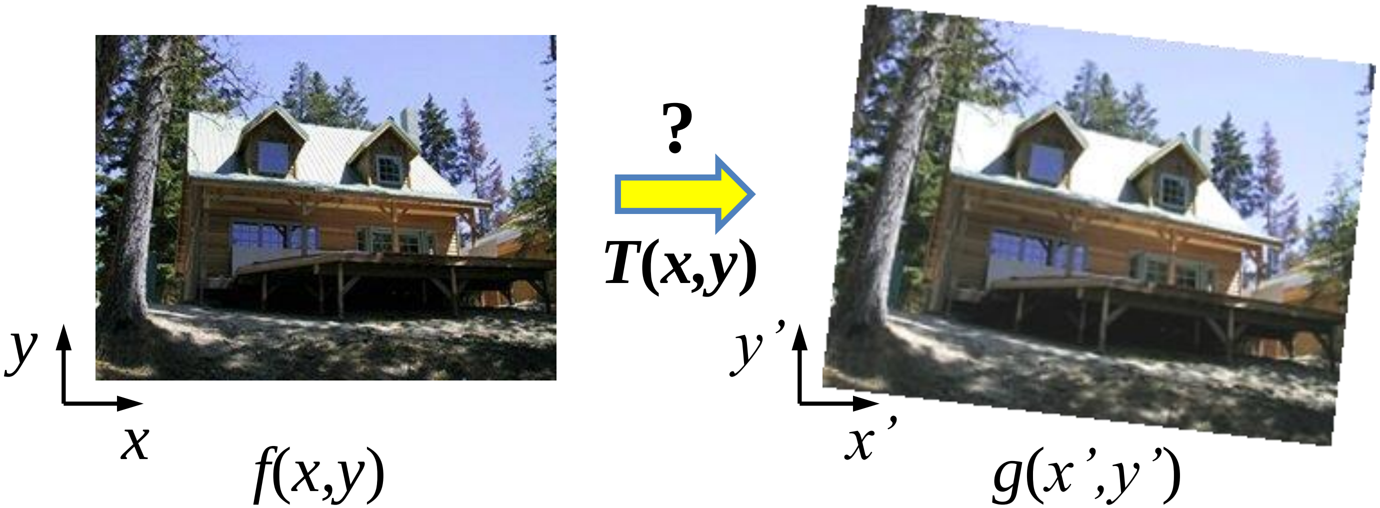


Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	2	orientation $+\cdots$	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	3	lengths $+\cdots$	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	4	angles $+\cdots$	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times 3}$	6	parallelism $+\cdots$	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times 3}$	8	straight lines	

These transformations are a nested set of groups

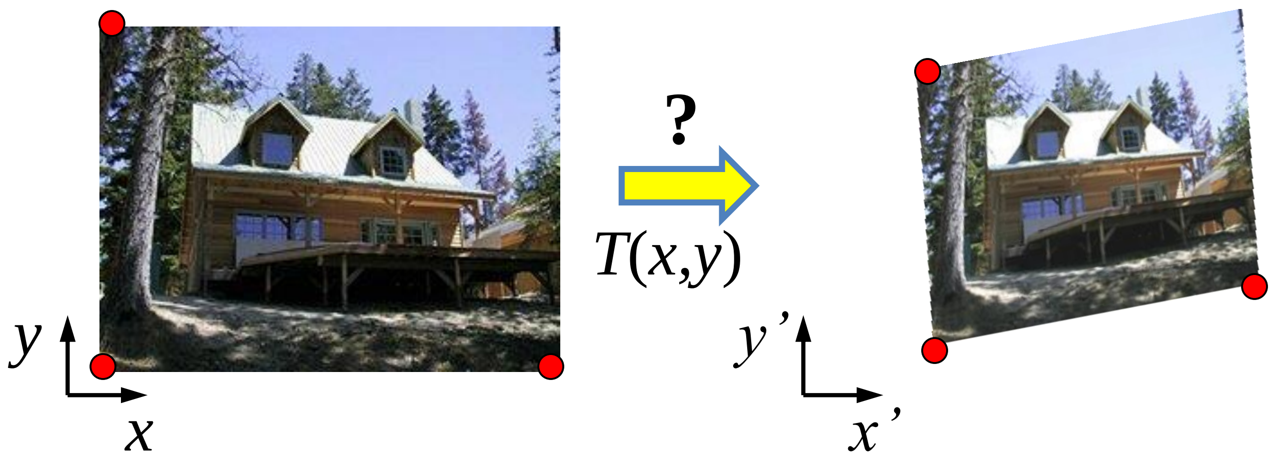
- Closed under composition and inverse is a member

Recovering Transformations



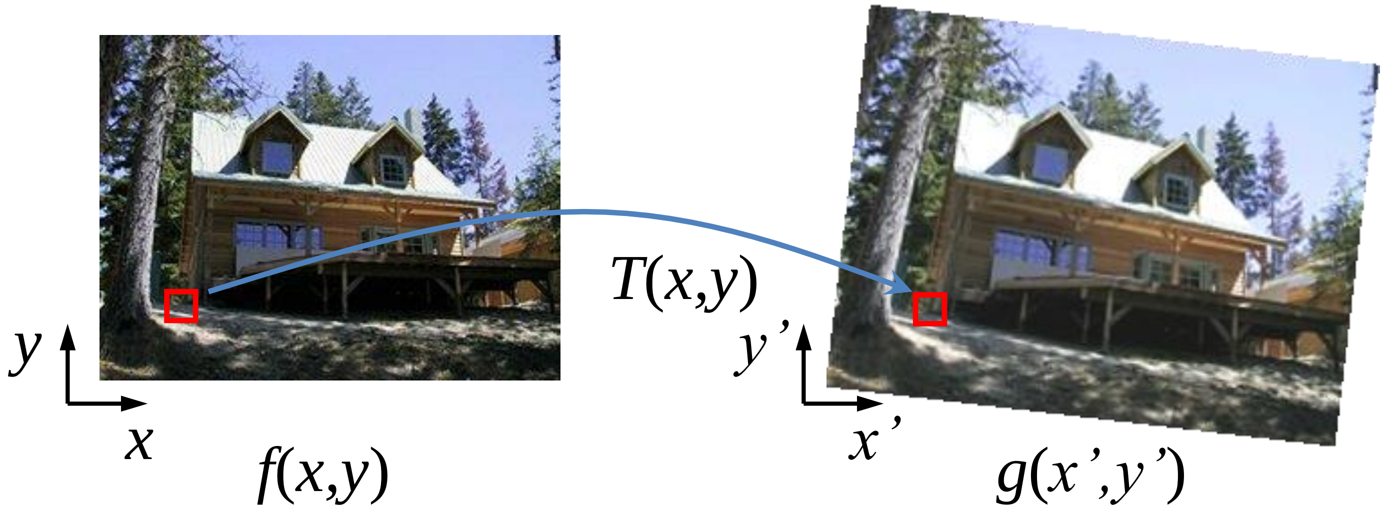
- What if we know *f* and *g* and want to recover the transform T?
  - willing to let user provide correspondences
    - How many do we need?

Affine: # correspondences?



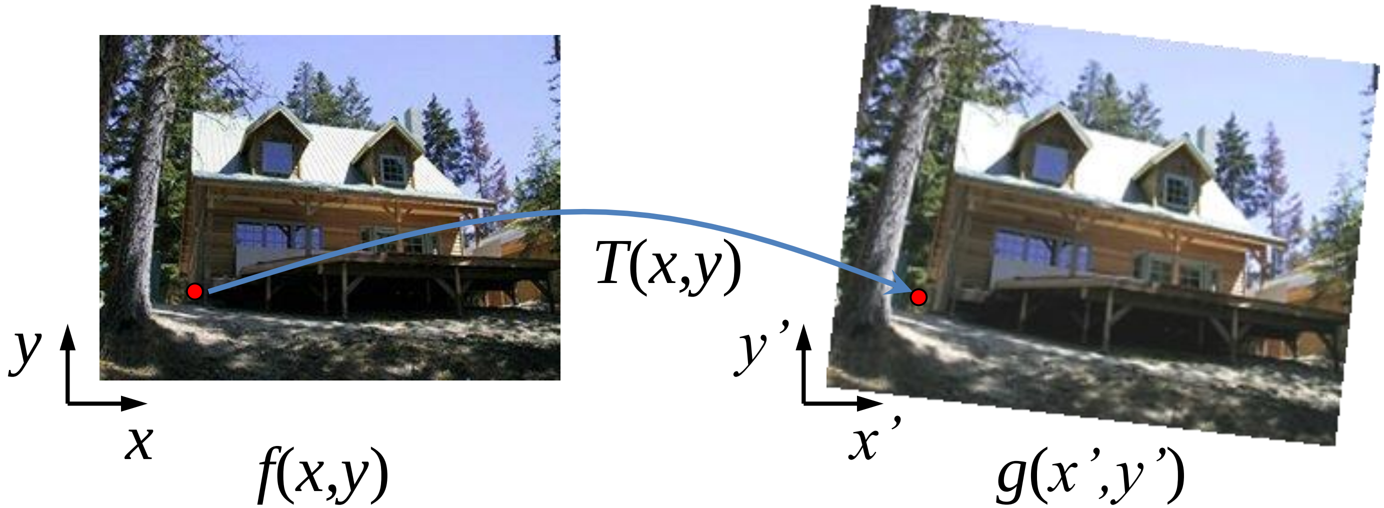
- How many DOF?
- How many correspondences needed for affine?

Image warping



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

Forward warping

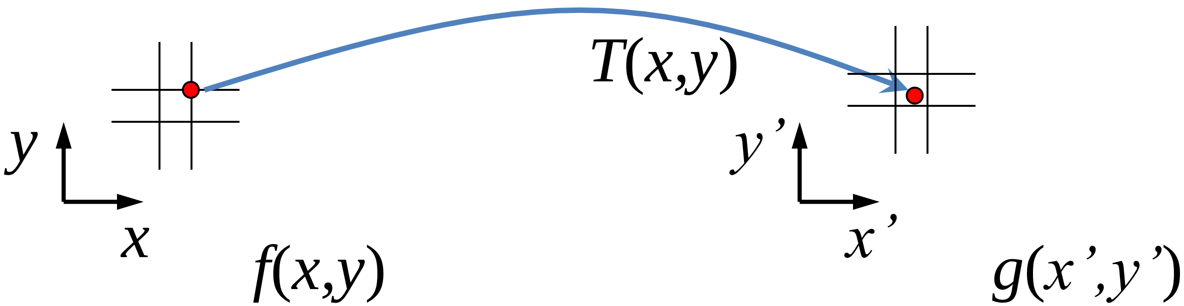


- Send each pixel $f(x,y)$ to its corresponding location
- $(x',y') = T(x,y)$ in the second image

Forward warping

$(x', y') = T(x, y)$ in the second image

What is the problem with this approach?



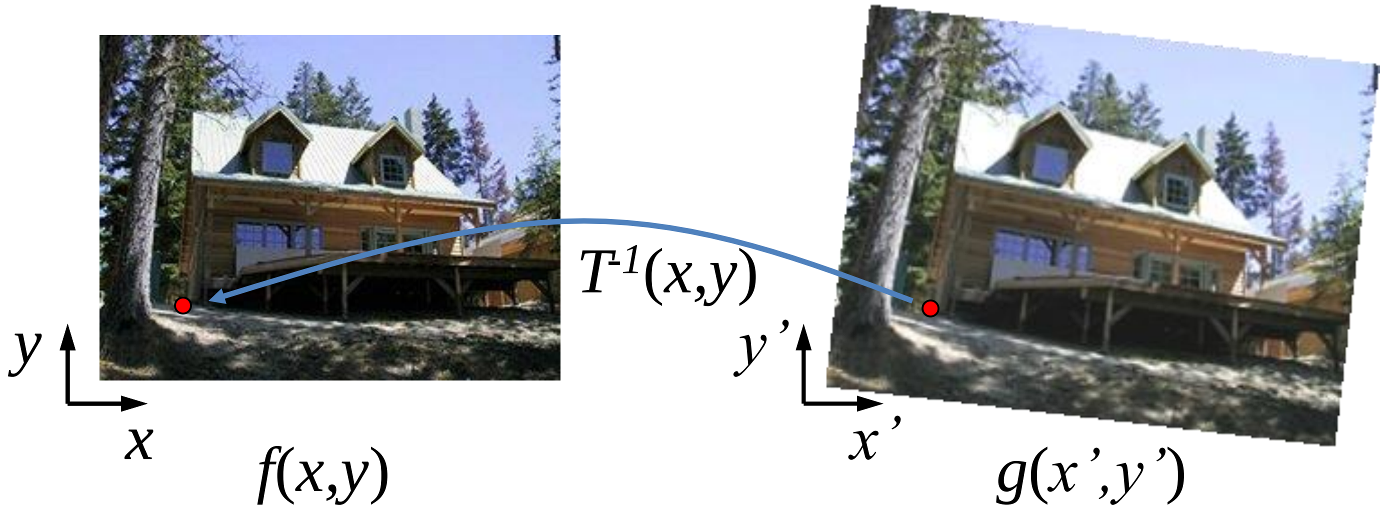
- Send each pixel $f(x,y)$ to its corresponding location

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

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

- Known as "splatting"

Inverse warping

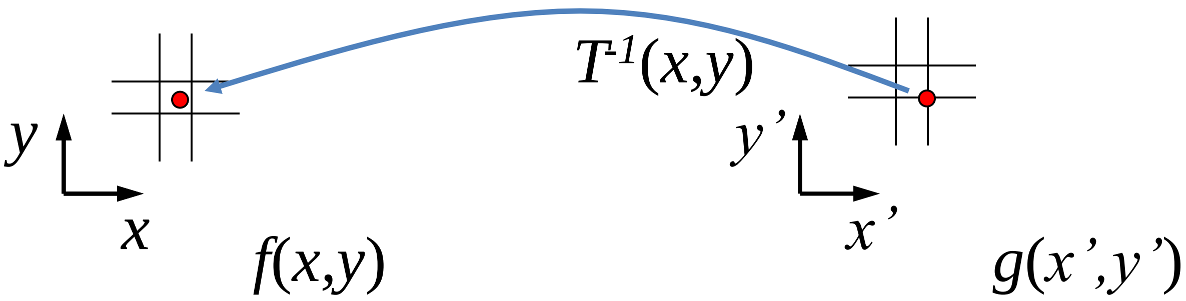


- Get each pixel $g(x',y')$ from its corresponding location
- $(x,y) = T^{-1}(x',y')$ in the first image

Q: what if pixel comes from "between" two pixels?

Inverse warping

$(x,y) = T^{-1}(x',y')$ in the first image



- Get each pixel $g(x',y')$ from its corresponding location

Q: what if pixel comes from "between" two pixels?

A: *Interpolate* color value from neighbors

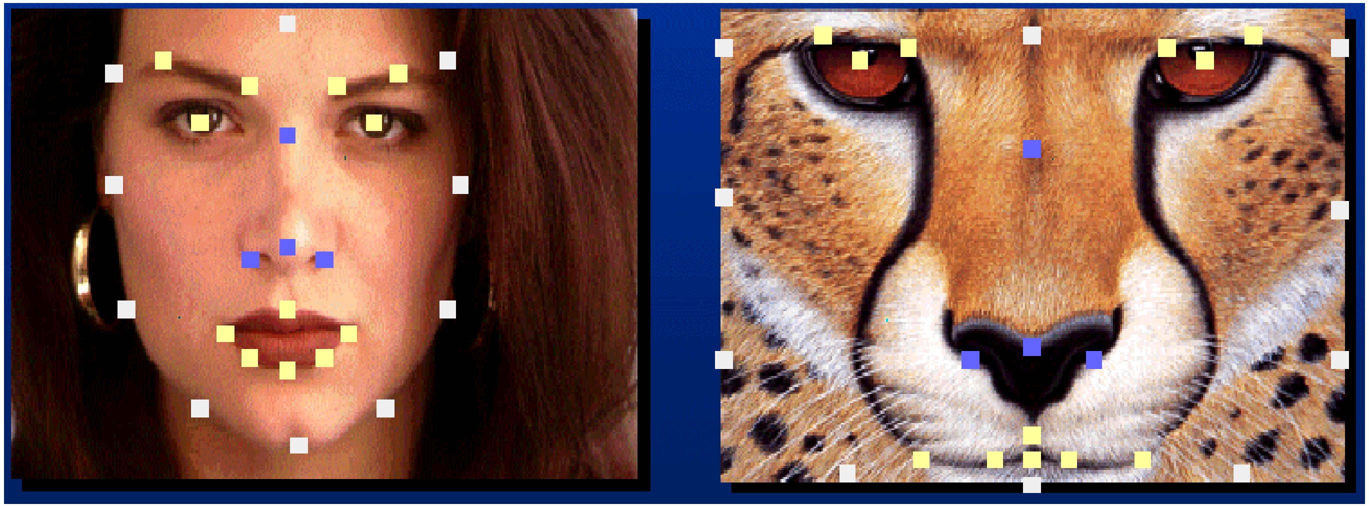
- – nearest neighbor, bilinear, Gaussian, bicubic
- – E.g. scipy.interpolate.interp2d

Warp specification - sparse

How can we specify the warp?

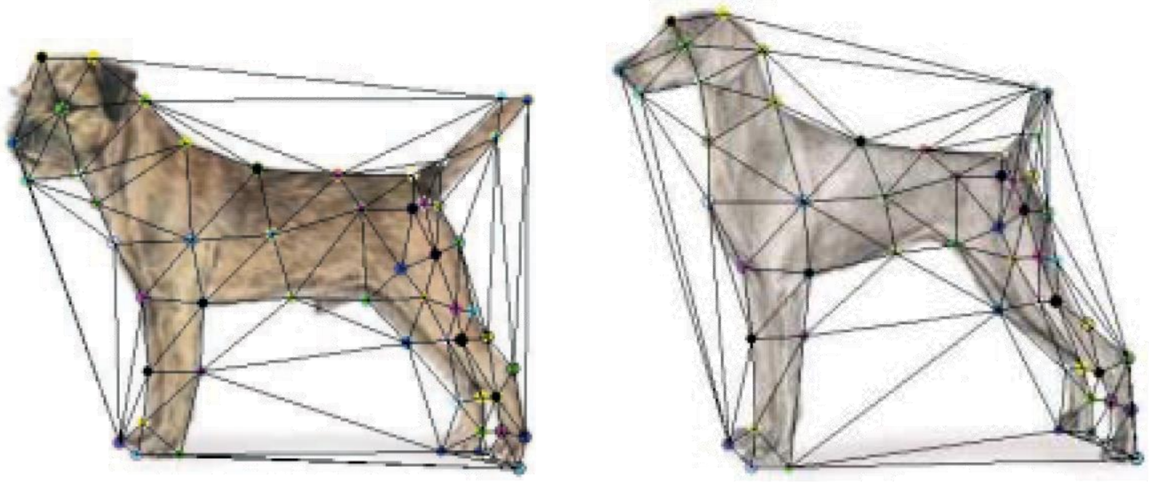
Specify corresponding *points*

- *interpolate* to a complete warping function
- How do we do it?



How do we go from feature points to pixels? Warping

Triangular Mesh

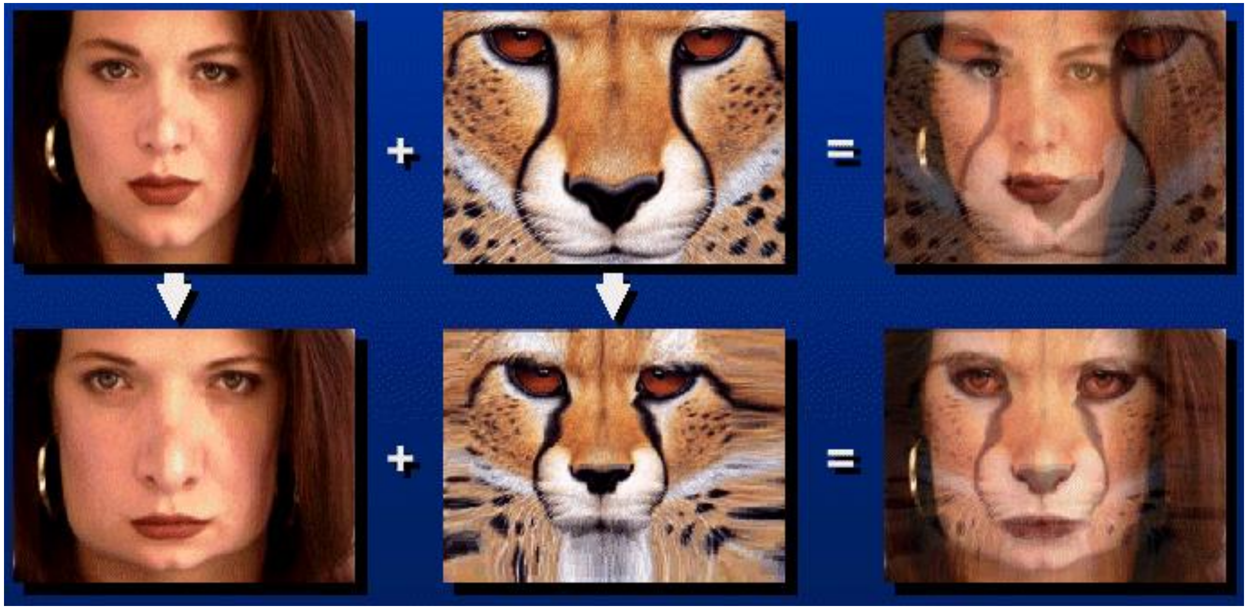


1. Input correspondences at key feature points
2. Define a triangular mesh over the points
   - Same mesh (triangulation) in both images!
   - Now we have triangle-to-triangle correspondences
3. Warp each triangle separately from source to destination
   - Affine warp with three corresponding points

Image Morphing

How do we create a morphing sequence?

1. Create an intermediate shape (by interpolation)
2. Warp both images towards it
3. Cross-dissolve the colors in the newly warped images



Summary of morphing

1. Define corresponding points
2. Define triangulation on points
   – Use same triangulation for both images
3. For each t in 0:step:1
   a. Compute the average shape (weighted average of points)
   b. For each triangle in the average shape
      - Get the affine projection to the corresponding triangles in each image
      - For each pixel in the triangle, find the corresponding points in each image and set value to weighted average (optionally use interpolation)
   c. Save the image as the next frame of the sequence