# 20 - Image Warping dan Image Morphing

Bahan Kuliah IF4073 Interpretasi dan Pengolahan Citra

# SUMBER (REFERENSI):


1. Alexei Efros, *Image Warping, 15-463: Computational Photography*, CMU, Fall 2008
2. Connelly Barnes, *Image Warping / Morphing, Computational Photography*.
3. Yao Wang, *EL512 Image Processing, Geometric Transformations: Warping, Registration, Morphing*, Polytchnic University, Brooklyn

# Image Warping

http://www.jeffrey-martin.com

15-463: Computational Photography
Alexei Efros, CMU, Fall 2008



Some slides from Steve Seitz

# Image Warping / Morphing

[Wolberg 1996, Recent Advances in Image Morphing]

Computational Photography

Connelly Barnes

Some slides from Fredo Durand, Bill Freeman, James Hays

# EL512 --- Image Processing

## Geometric Transformations: Warping, Registration, Morphing

Yao Wang
Polytechnic University, Brooklyn, NY 11201

With contribution from Zhu Liu, Onur Guleryuz, and
Partly based on
A. K. Jain, Fundamentals of Digital Image Processing

# What is Geometric Transformation?

- So far, the image processing operations we have discussed modify the color values of pixels in a given image
- With geometric transformation, we modify the positions of pixels in a image, but keep their colors unchanged
  - To create special effects
  - To register two images taken of the same scene at different times
  - To morph one image to another

# Image Transformations

image filtering: change ***range*** of image

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

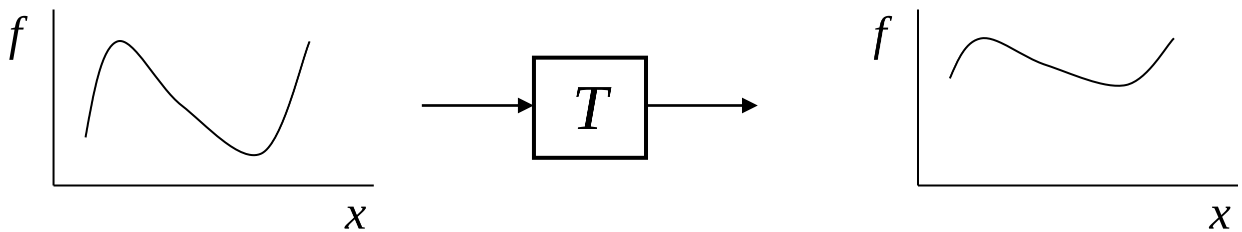
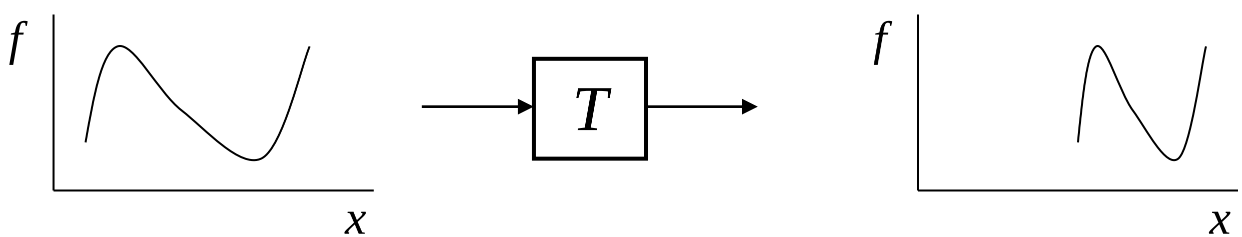



image warping: change ***domain*** of image

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



# Image Transformations

image filtering: change ***range*** of image

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

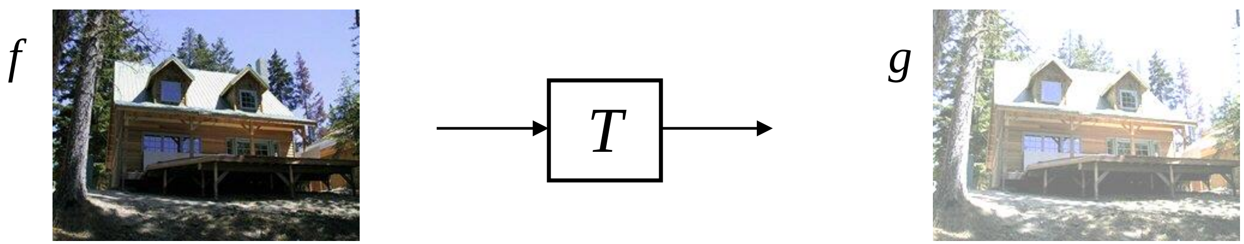
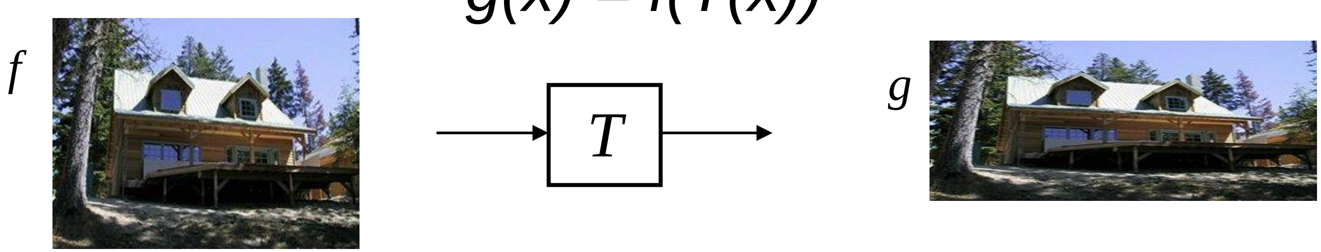
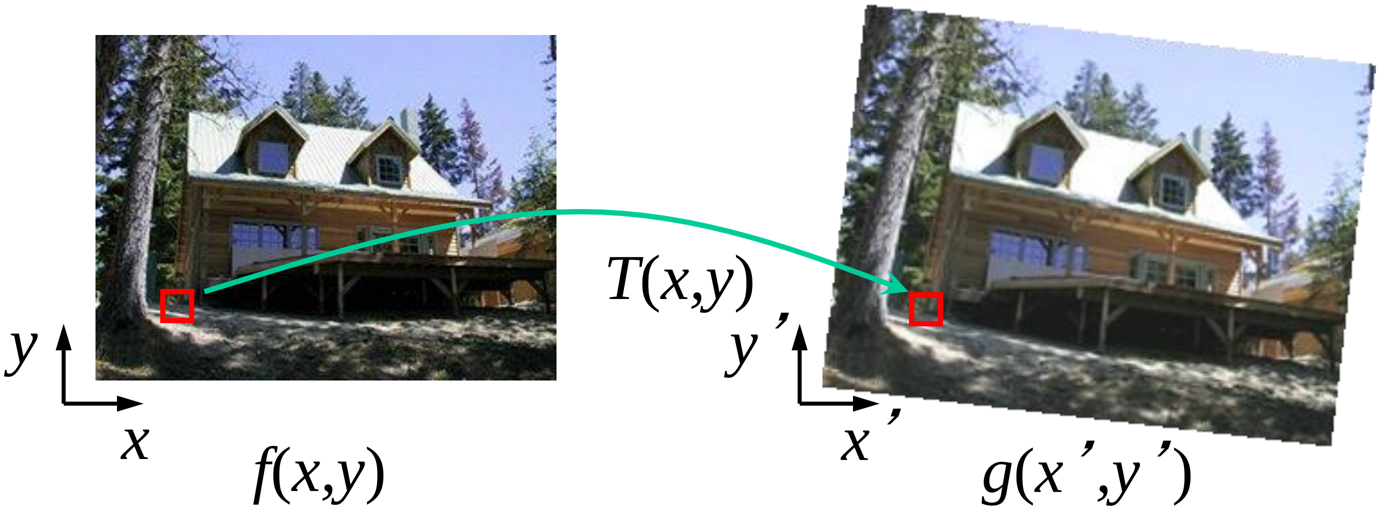



image warping: change ***domain*** of image

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



# 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 $g(x',y') = f(T(x,y))$?

# Parametric (global) warping

Examples of parametric warps:

translation

rotation

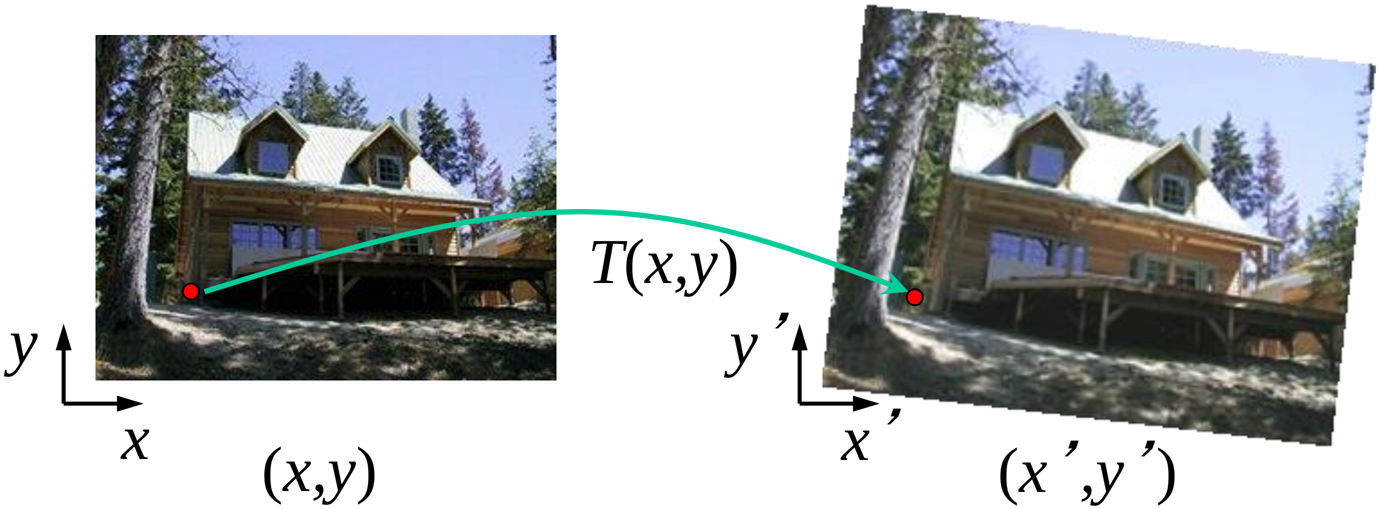
aspect

affine

perspective

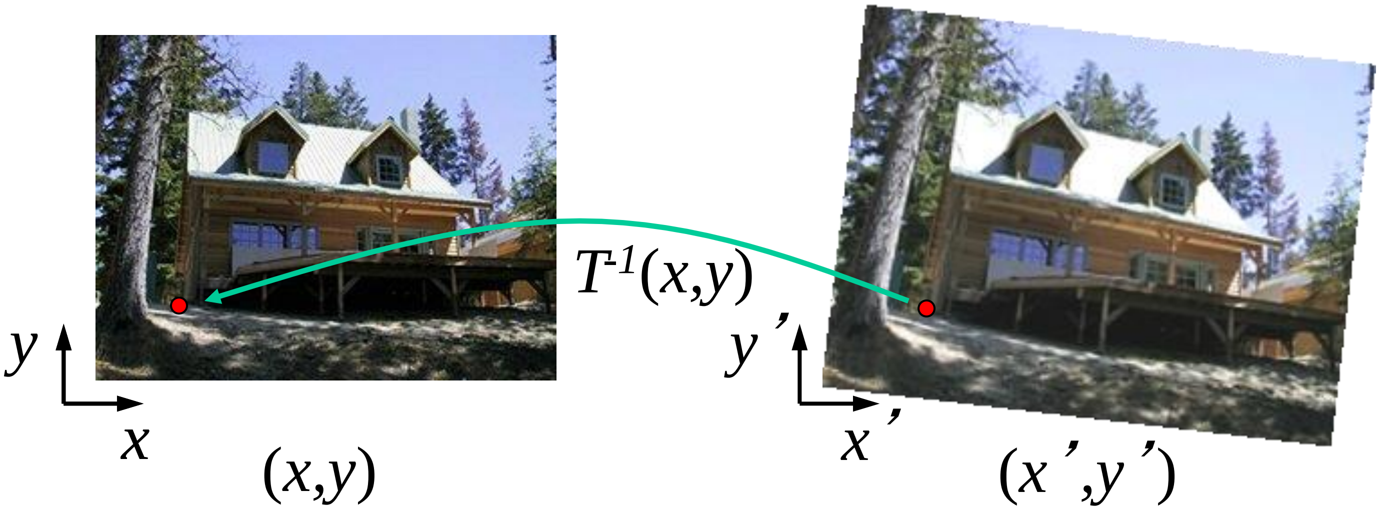
cylindrical

# Forward warping




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

# Inverse warping



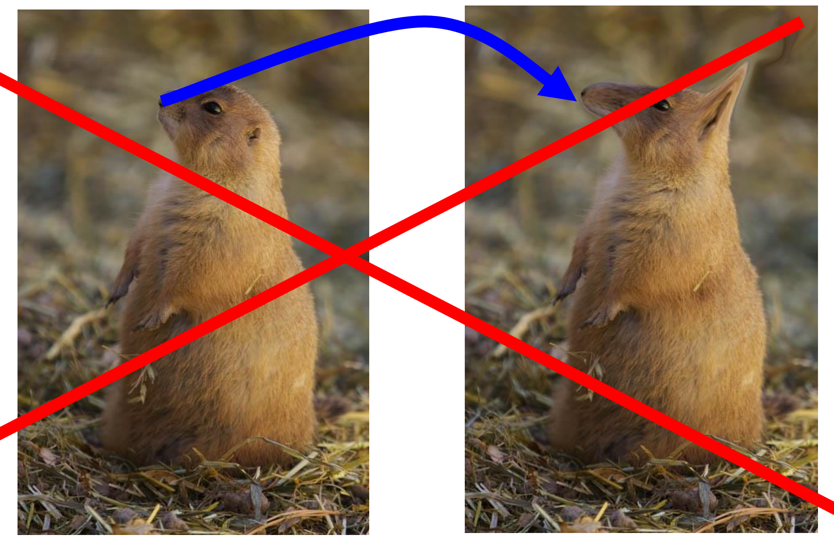

Get each pixel color $g(x',y')$ from its corresponding location

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

# Applying a warp: use inverse

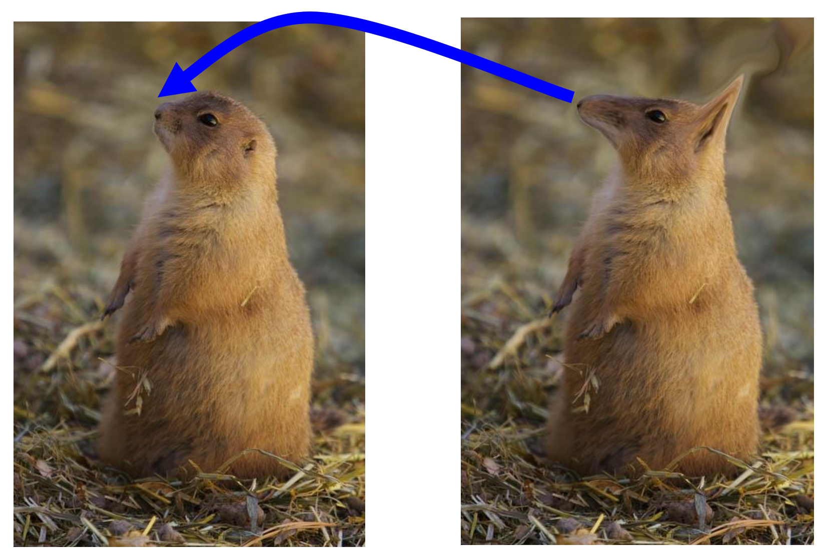
Forward warp:

- For each pixel in **input** image
  - Paste color **to warped** location in output
- Problem: gaps

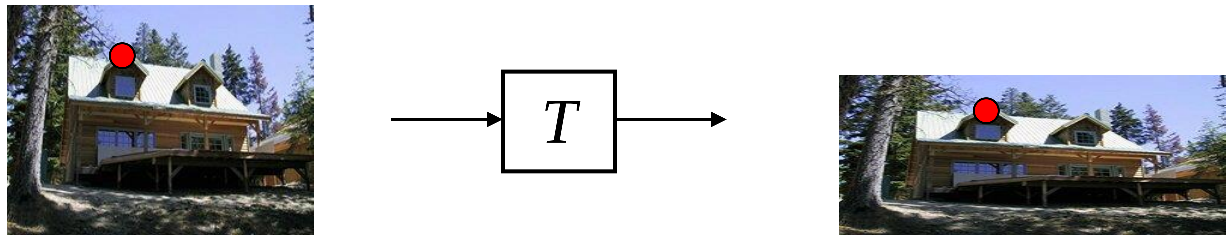


Inverse warp

- For each pixel in **output** image
  - Lookup color **from inverse-warped** location



# Parametric (global) warping




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

Transformation T is a coordinate-changing machine:

$$p' = T(p)$$

What does it mean that *T* is global?

- Is the same for any point p
- can be described by just a few numbers (parameters)

Let's represent *T* as a matrix:

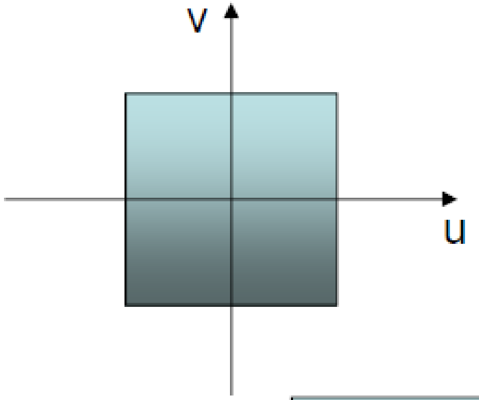
$$p' = \mathbf{M}p$$

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

# Translation

- Translation is defined by the following mapping functions:

$$\begin{cases} x = u + t_x \\ y = v + t_y \end{cases} \quad and \quad \begin{matrix} u = x - t_x \\ v = y - t_y \end{matrix}$$

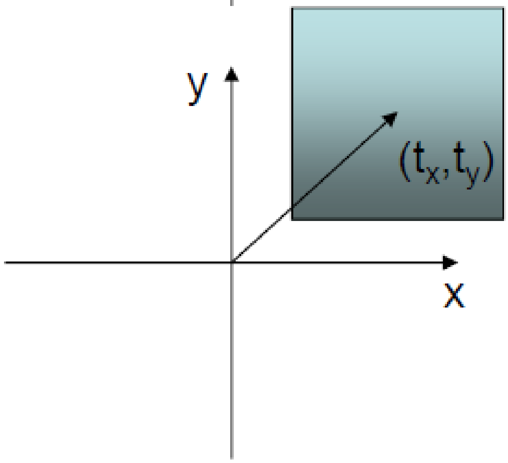



- In matrix notation

$$\mathbf{x} = \mathbf{u} + \mathbf{t}, \quad \mathbf{u} = \mathbf{x} - \mathbf{t}$$

*where*

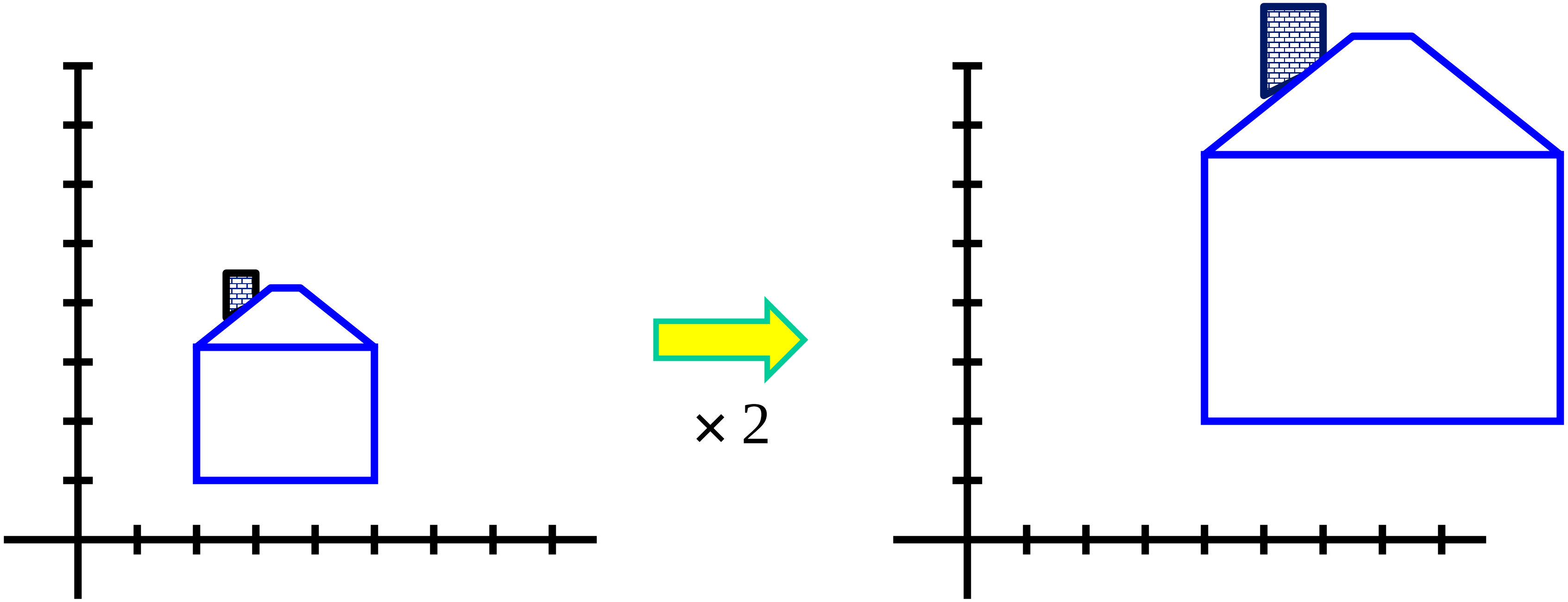
$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} u \\ v \end{bmatrix}, \quad \mathbf{t} = \begin{bmatrix} t_x \\ t_y \end{bmatrix}.$$



# Scaling

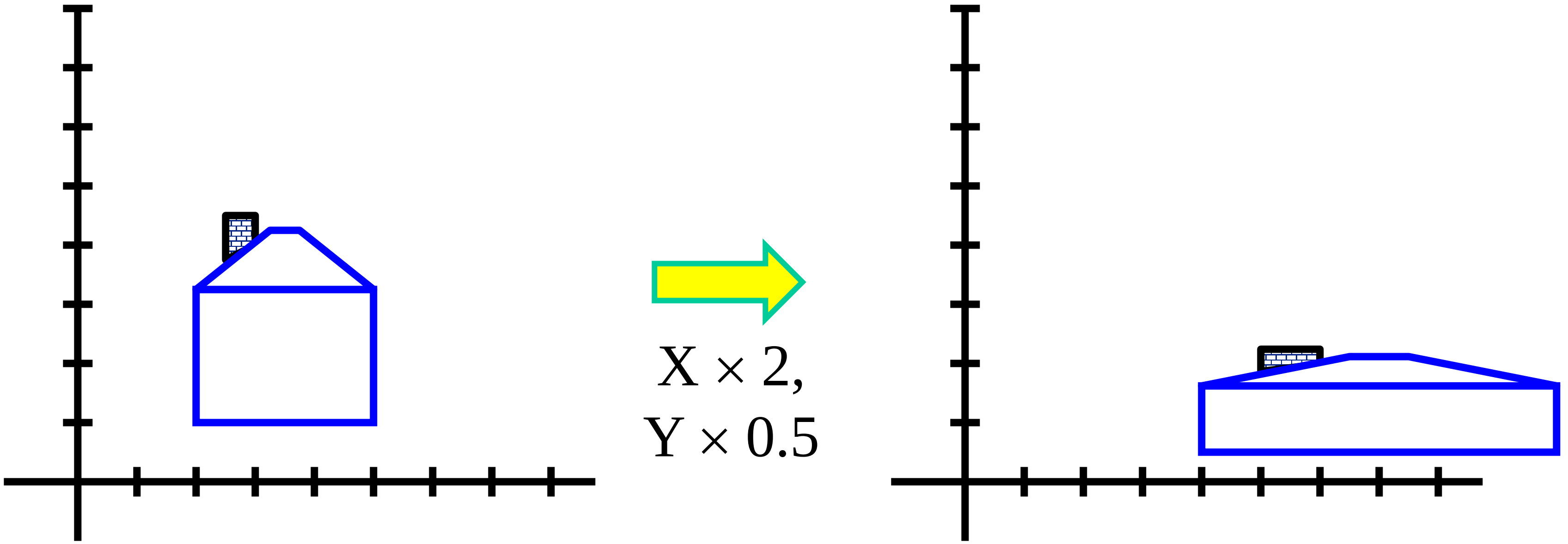
*Scaling* a coordinate means multiplying each of its components by a scalar

*Uniform scaling* means this scalar is the same for all components:



# Scaling

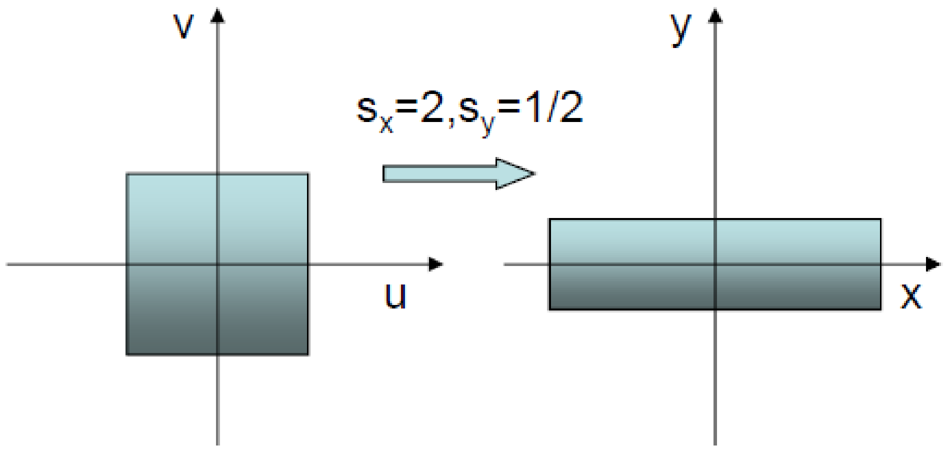
*Non-uniform scaling*: different scalars per component:



# Scaling

- Scaling is defined by

$$\begin{cases} x = s_x u \\ y = s_y v \end{cases} \quad and \quad \begin{cases} u = x / s_x \\ v = y / s_y \end{cases}$$




- Matrix notation

$$\mathbf{x} = \mathbf{S}\mathbf{u}, \quad \mathbf{u} = \mathbf{S}^{-1}\mathbf{x}$$

*where*

$$\mathbf{S} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

- If $s_x < 1$ and $s_y < 1$, this represents a minification or shrinking, if $s_x > 1$ and $s_y > 1$, it represents a magnification or zoom.

# Scaling

Scaling operation:

$$x' = ax$$

$$y' = by$$

Or, in matrix form:

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

What's inverse of S?

# Rotation

- Rotation by an angle of θ is defined by

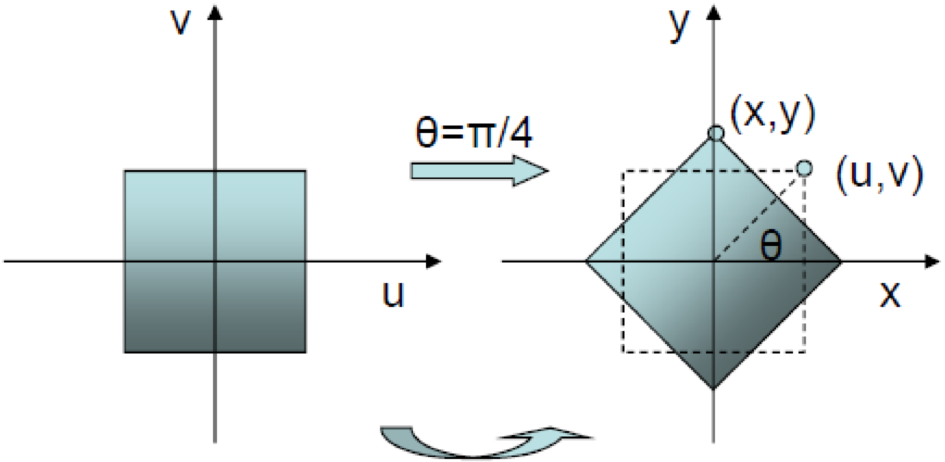
$$\begin{cases} x = u\cos\theta - v\sin\theta \\ y = u\sin\theta + v\cos\theta \end{cases} \quad and \quad \begin{cases} u = x\cos\theta + y\sin\theta \\ v = -x\sin\theta + y\cos\theta \end{cases}$$

- In matrix format

$$\mathbf{x} = \mathbf{R}\mathbf{u}, \quad \mathbf{u} = \mathbf{R}^T\mathbf{x}$$

*where*

$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$




- R is a unitary matrix: $\mathbf{R}^{-1}=\mathbf{R}^T$

B translation

B rotation

Translation: $x(k,l) = k + 50; y(k,l) = l;$

Rotation: $x(k,l) = (k - x_0)cos(\theta) + (l - y_0)sin(\theta) + x_0;$

$y(k,l) = -(k - x_0)sin(\theta) + (l - y_0)cos(\theta) + y_0;$

$x_0 = y_0 = 256.5$ the center of the image $\mathbf{A}$, $\theta = \pi/6$

By Onur Guleyuz

# 2x2 Matrices

What types of transformations can be represented with a 2x2 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 Scale around (0,0)?

$$\boldsymbol{x}' = \boldsymbol{s}_x * \boldsymbol{x}$$
$$\boldsymbol{y}' = \boldsymbol{s}_y * \boldsymbol{y}$$

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

# 2x2 Matrices

What types of transformations can be represented with a 2x2 matrix?

2D Rotate around (0,0)?

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

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

2D Shear?

$$\boldsymbol{x'} = \boldsymbol{x} + \boldsymbol{sh_x} * \boldsymbol{y}$$
$$\boldsymbol{y'} = \boldsymbol{sh_y} * \boldsymbol{x} + \boldsymbol{y}$$

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

# 2x2 Matrices

What types of transformations can be represented with a 2x2 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 over (0,0)?

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

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

# 2x2 Matrices

What types of transformations can be represented with a 2x2 matrix?

2D Translation?

$$x' = x + t_x$$

$$y' = y + t_y$$

NO!

Only linear 2D transformations can be represented with a 2x2 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

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

# Geometric Transformation

- A geometric transformation refers to a combination of translation, scaling, and rotation, with a general form of

$$\mathbf{x} = \mathbf{RS}(\mathbf{u}+\mathbf{t}) = \mathbf{Au}+\mathbf{b},$$
$$\mathbf{u} = \mathbf{A}^{-1}(\mathbf{x}-\mathbf{b}) = \mathbf{A}^{-1}\mathbf{x}+\mathbf{c},$$
$$\textit{with } \mathbf{A} = \mathbf{RS}, \mathbf{b} = \mathbf{RSt}, \mathbf{c} = -\mathbf{t}.$$

- Note that interchanging the order of operations will lead to different results.

# Affine Mapping

- All possible geometric transformations are special cases of the *Affine Mapping*:

$$\begin{cases} x = a_0 + a_1 u + a_2 v \\ y = b_0 + b_1 u + b_2 v \end{cases} \quad or \quad \mathbf{x} = \mathbf{A}\mathbf{u} + \mathbf{b}$$

$$\mathbf{A} = \begin{bmatrix} a_1 & a_2 \\ b_1 & b_2 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} a_0 \\ b_0 \end{bmatrix}$$

- When A is a orthonormal matrix, it corresponds to a rotation matrix, and the corresponding affine mapping reduces to a geometric mapping.

# Matlab Functions

- T = MAKETFORM('affine',U,X) builds a TFORM struct for a
- two-dimensional affine transformation that maps each row of U
- to the corresponding row of X. U and X are each 3-by-2 and
- define the corners of input and output triangles. The corners
- may not be collinear.
- Example
- -------
- Create an affine transformation that maps the triangle with vertices
- (0,0), (6,3), (-2,5) to the triangle with vertices (-1,-1), (0,-10),
- (4,4):
-
- u = [ 0   6  -2]';
- v = [ 0   3   5]';
- x = [-1   0   4]';
- y = [-1 -10   4]';
- tform = maketform('affine',[u v],[x y]);

- G = MAKETFORM('affine',T) builds a TFORM struct G for an N-dimensional affine transformation.  T defines a forward transformation such that TFORMFWD(U,T), where U is a 1-by-N vector, returns a 1-by-N vector X such that X = U * T(1:N,1:N) + T(N+1,1:N).T has both forward and inverse transformations.  N=2 for 2D image transformation

In MATLAB notation

$$T = \begin{bmatrix} a_1 & b_1 & 0 \\ a_2 & b_2 & 0 \\ a_0 & b_0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{A}^T & 0 \\ \mathbf{b}^T & 1 \end{bmatrix}$$

- B = IMTRANSFORM(A,TFORM, INTERP) transforms the image A according to the 2-D spatial transformation defined by TFORMB; INTERP specifies the interpolation filter
- Example 1
- ---------
- Apply a horizontal shear to an intensity image.
-
- I = imread('cameraman.tif');
- tform = maketform('affine',[1 0 0; .5 1 0; 0 0 1]);
- J = imtransform(I,tform);
- figure, imshow(I), figure, imshow(J)

- Show in class

# Horizontal Shear Example

tform = maketform('affine',[1 0 0; .5 1 0; 0 0 1]);
In MATLAB, 'affine' transform is defined by:
[a1,b1,0;a2,b2,0;a0,b0,1]

With notation used in this lecture note

$$\mathbf{A} = \begin{bmatrix} 1 & 0.5 \\ 0 & 1 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Note in this example, first coordinate indicates horizontal position, second coordinate indicate vertic

# MATLAB function for image warping

- B = IMTRANSFORM(A,TFORM, INTERP) transforms the image A according to the 2-D spatial transformation defined by TFORM
- INTERP specifies the interpolation filter
- Example 1
-     ---------
-     Apply a horizontal shear to an intensity image.
- 
-         I = imread('cameraman.tif');
-         tform = maketform('affine',[1 0 0; .5 1 0; 0 0 1]);
-         J = imtransform(I,tform);
-         figure, imshow(I), figure, imshow(J)

# Horizontal Shear Example

```
tform = maketform('affine',[1 0 0; .5 1 0; 0 0 1]);
```

In MATLAB, 'affine' transform is defined by:
[a1,b1,0;a2,b2,0;a0,b0,1]

With notation used in this lecture note

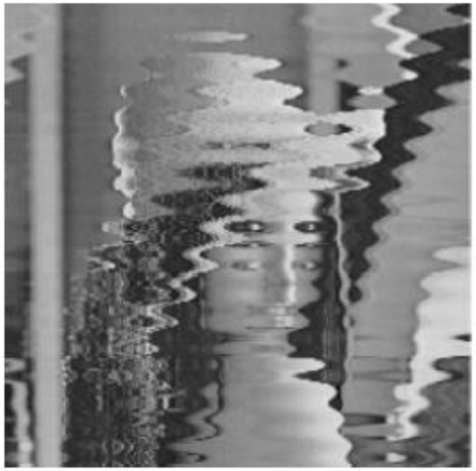
$$\mathbf{A} = \begin{bmatrix} 1 & 0.5 \\ 0 & 1 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Note in this example, x, u indicates vertical position, y, v indicate horizontal position

# Example of Image Warping (1)

WAVE1

WAVE2



wave1: $x(u,v)=u+20\sin(2\pi v/128); y(u,v)=v;$
wave2: $x(u,v)=u+20\sin(2\pi u/30); y(u,v)=v.$

By Onur Guleyuz

# Example of Image Warping (2)

WARP

SWIRL

WARP $x(u,v) = sign(u - x_0) * (u - x_0)^2 / x_0 + x_0; y(u,v) = v$

SWIRL

$$x(u,v) = (u - x_0)\cos(\theta) + (v - y_0)\sin(\theta) + x_0;$$

$$y(u,v) = -(u - x_0)\sin(\theta) + (v - y_0)\cos(\theta) + y_0;$$

$$r = ((u - x_0)^2 + (v - y_0)^2)^{1/2}, \theta = \pi r / 512.$$

By Onur Guleyuz

# Homogeneous Coordinates

**Q: How can we represent translation as a 3x3 matrix?**

$$x' = x + t_x$$

$$y' = y + t_y$$

# Homogeneous Coordinates

***Homogeneous coordinates***

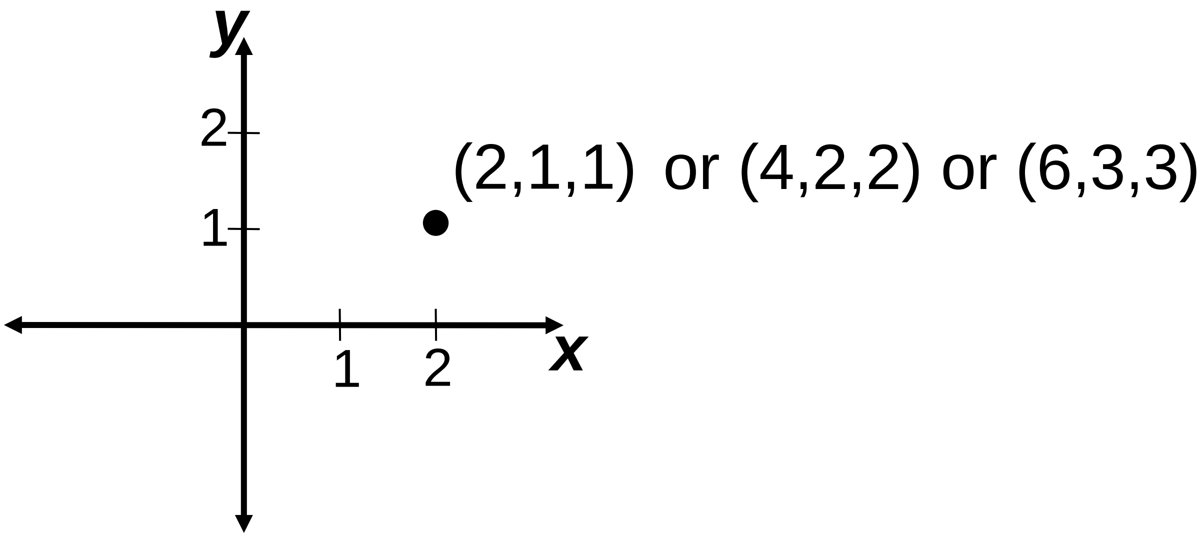
- represent coordinates in 2 dimensions with a 3-vector

$$\begin{bmatrix} x \\ y \end{bmatrix} \xrightarrow{\text{homogeneou s coords}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

# Homogeneous Coordinates

Add a 3rd coordinate to every 2D point

- (x, y, w) represents a point at location (x/w, y/w)
- (x, y, 0) represents a point at infinity
- (0, 0, 0) is not allowed




Convenient coordinate system to represent many useful transformations

# Homogeneous Coordinates

**Q: How can we represent translation as a 3x3 matrix?**

$$x' = x + t_x$$

$$y' = y + t_y$$

A: Using the rightmost column:

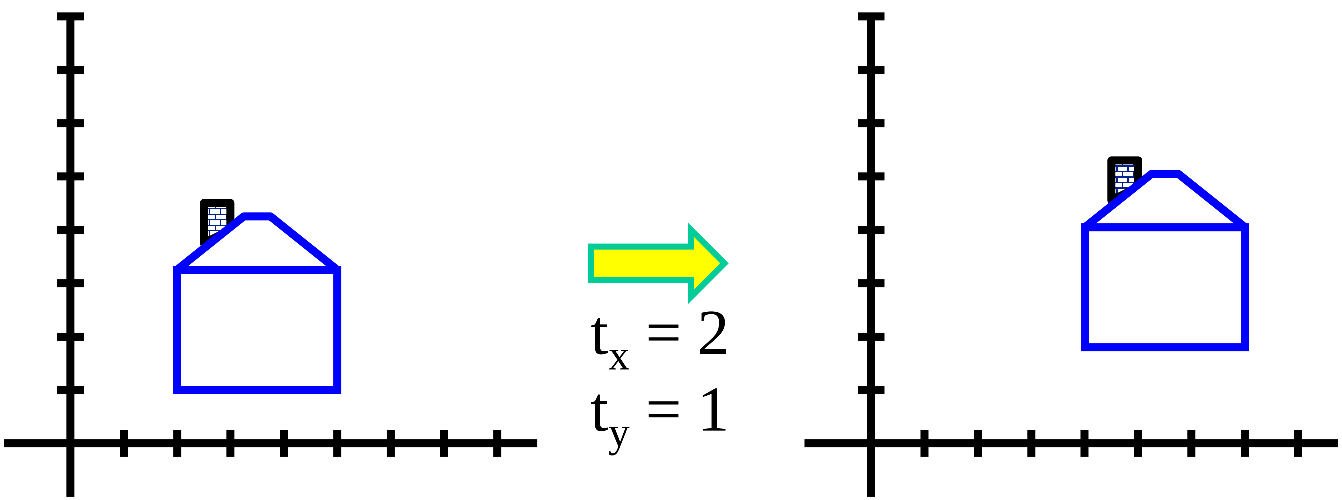
$$\mathbf{T}ranslation = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

# Translation

Example of translation

Homogeneous Coordinates

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



# Basic 2D Transformations

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

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

Shear

# 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**

# Affine Transformations

Affine transformations are combinations of …

- Linear transformations, and
- Translations

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Properties of affine transformations:

- Origin does not necessarily map to origin
- Lines map to lines
- Parallel lines remain parallel
- Ratios are preserved
- Closed under composition
- Models change of basis

Will the last coordinate *w* always be 1?

# Projective Transformations

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

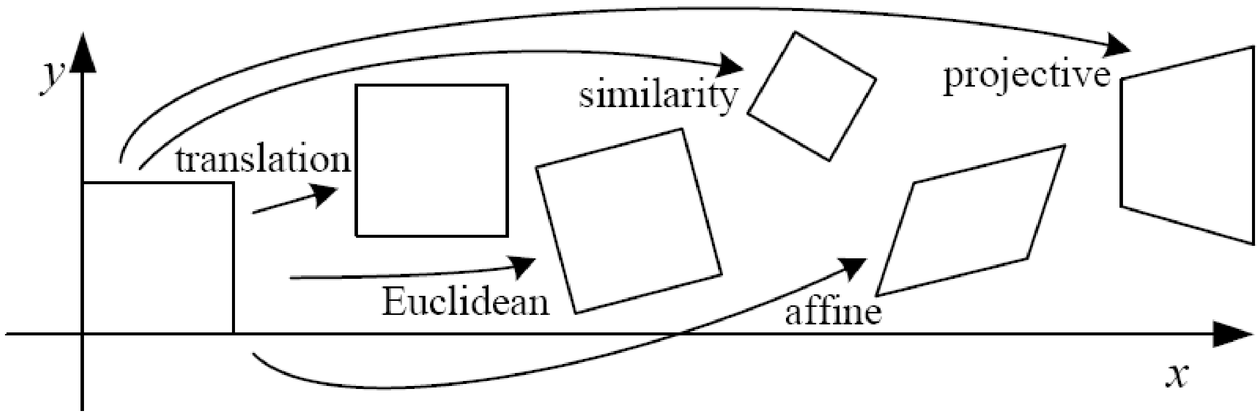
Projective transformations …

- Affine transformations, and
- Projective warps

Properties of projective transformations:

- Origin does not necessarily map to origin
- Lines map to lines
- Parallel lines do not necessarily remain parallel
- Ratios are not preserved
- Closed under composition
- Models change of basis

# 2D image transformations




| Name | Matrix | # D.O.F. | Preserves: | Icon |
|---|---|---|---|---|
| translation | $\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$ | | | |
| rigid (Euclidean) | $\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$ | | | |
| similarity | $\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$ | | | |
| affine | $\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times 3}$ | | | |
| projective | $\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times 3}$ | | | |

These transformations are a nested set of groups

- Closed under composition and inverse is a member

# Image Morphing

- Image morphing has been widely used in movies and commercials to create special visual effects. For example, changing a beauty gradually into a monster.
- The fundamental techniques behind image morphing is image warping.
- Let the original image be $f(\mathbf{u})$ and the final image be $g(\mathbf{x})$. In image warping, we create $g(\mathbf{x})$ from $f(\mathbf{u})$ by changing its shape. In image morphing, we use a combination of both $f(\mathbf{u})$ and $g(\mathbf{x})$ to create a series of intermediate images.

# Examples of Image Morphing

Cross Dissolve

$I(t) = (1-t)*S+t*T$

Mesh based

*George Wolberg,* "Recent Advances in Image Morphing", *Computer Graphics Intl. '96*, Pohang, Korea, June 1996.

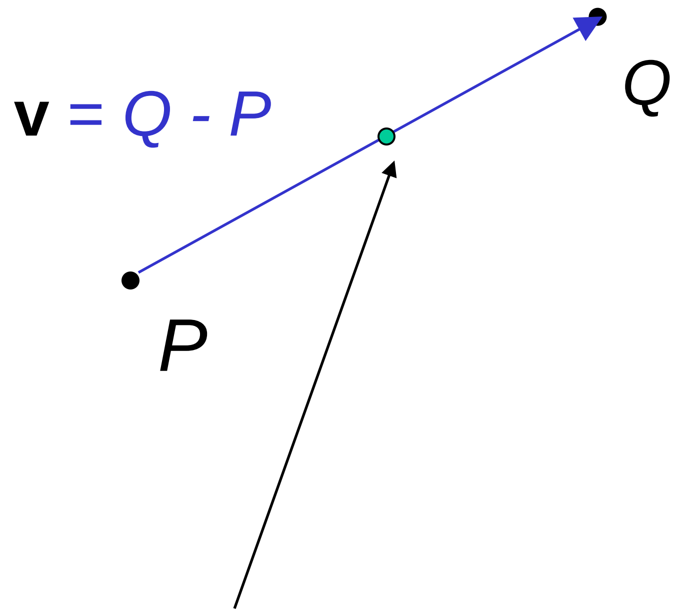
# Image Morphing Method

- Suppose the mapping function between the two end images is given as $\mathbf{x}=\mathbf{u}+\mathbf{d}(\mathbf{u})$. $\mathbf{d}(\mathbf{u})$ is the displacement between corresponding points in these two images.
- In image morphing, we create a series of images, starting with $f(\mathbf{u})$ at $k=0$, and ending at $g(\mathbf{x})$ at $k=K$. The intermediate images are a linear combination of the two end images:

$$h_k(\mathbf{u}+s_k\mathbf{d}) = (1-s_k)f(\mathbf{u}) + s_k g(\mathbf{u}+\mathbf{d}(\mathbf{u})), \quad k = 0,1,\ldots,K,$$
$$where\ s_k = k/K.$$

# Linear Interpolation

How can we linearly transition between point *P* and point *Q*?




$P + t\,\mathbf{v}$

$= (1-t)P + tQ$, e.g. t = 0.5

P and Q can be anything:

- points on a plane (2D) or in space (3D)
- Colors in RGB or HSV (3D)
- Whole images (m-by-n D)… etc.

# Idea #1: Cross-Dissolve




Interpolate whole images:

$\text{Image}_{\text{halfway}} = (1-t)*\text{Image}_1 + t*\text{image}_2$

This is called **cross-dissolve** in film industry

But what if the images are not aligned?

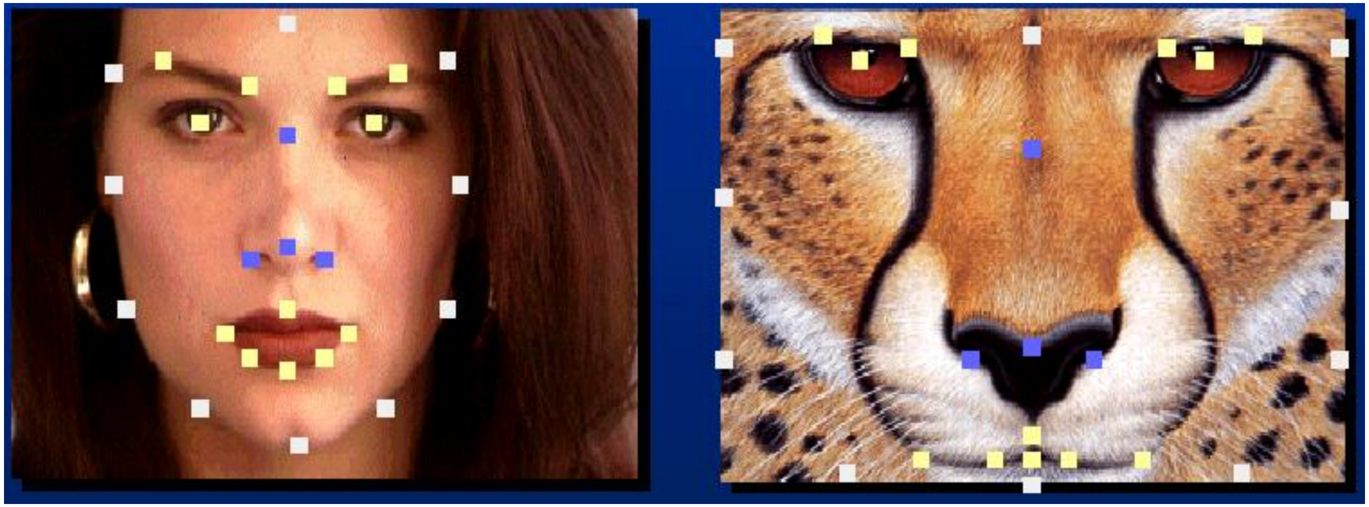
# Idea #2: Align, then cross-disolve


Align first, then cross-dissolve

- Alignment using global warp – picture still valid

# Full Morphing

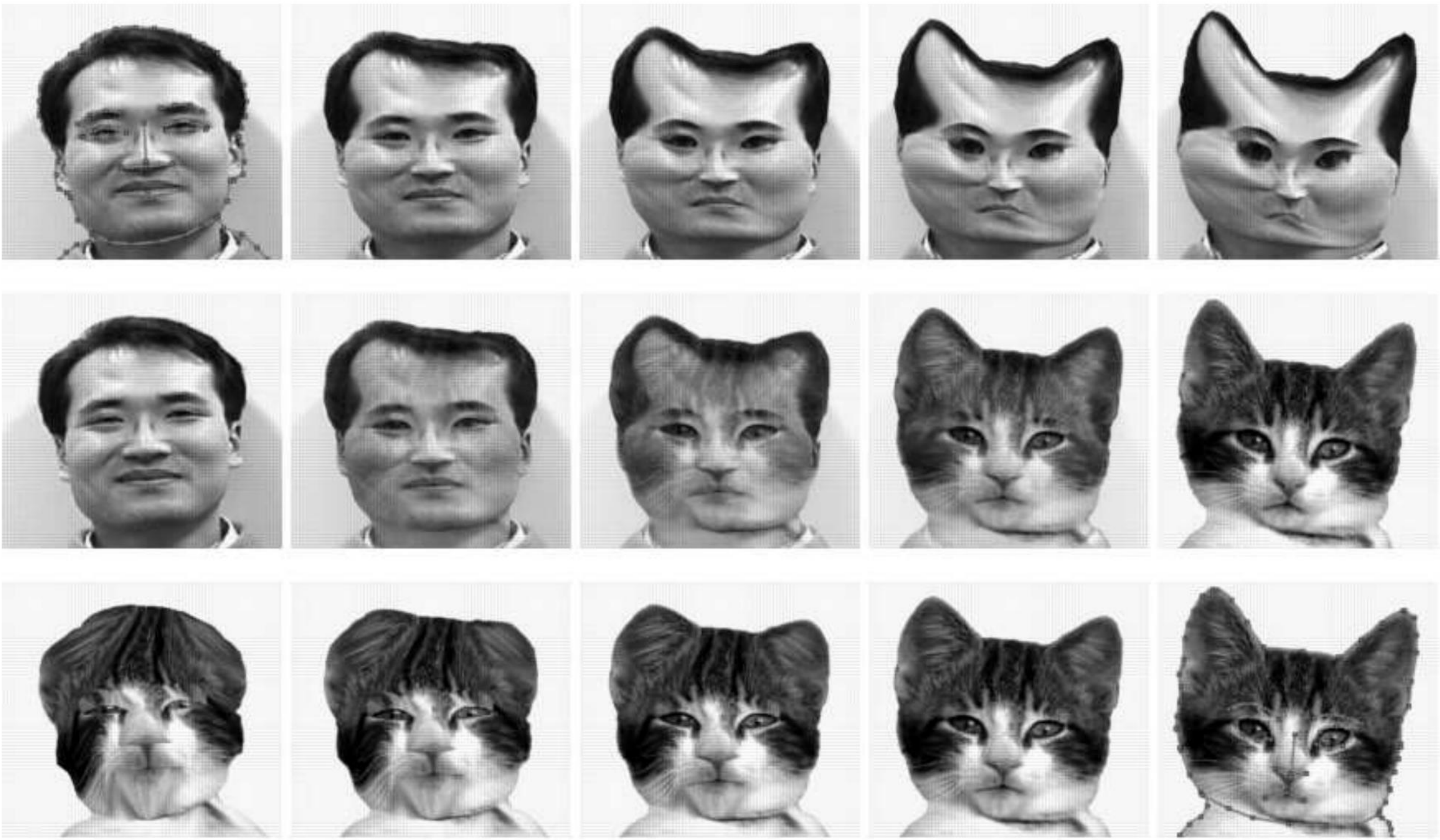
A



B

- What if there is no simple global function that aligns two images?
- User specifies corresponding feature points
- Construct warp animations A -> B and B -> A
- Cross dissolve these

# Full Morphing



# Full Morphing

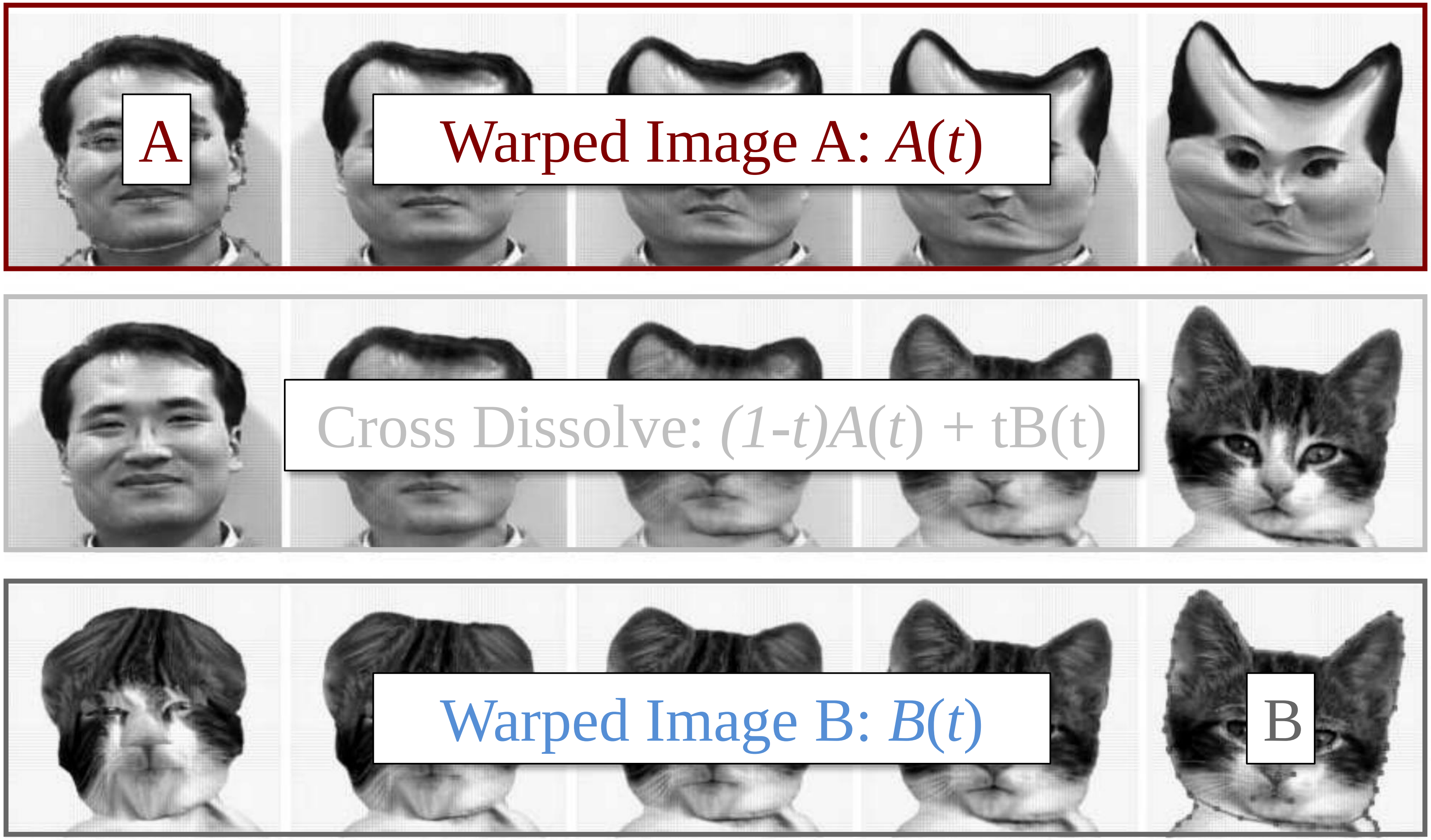
~~Image A~~ ~~Image B~~

1. Find warping fields from user constraints (points or lines):
   Warp field $T_{AB}$(x, y) that maps A pixel to B pixel
   Warp field $T_{BA}$(x, y) that maps B pixel to A pixel

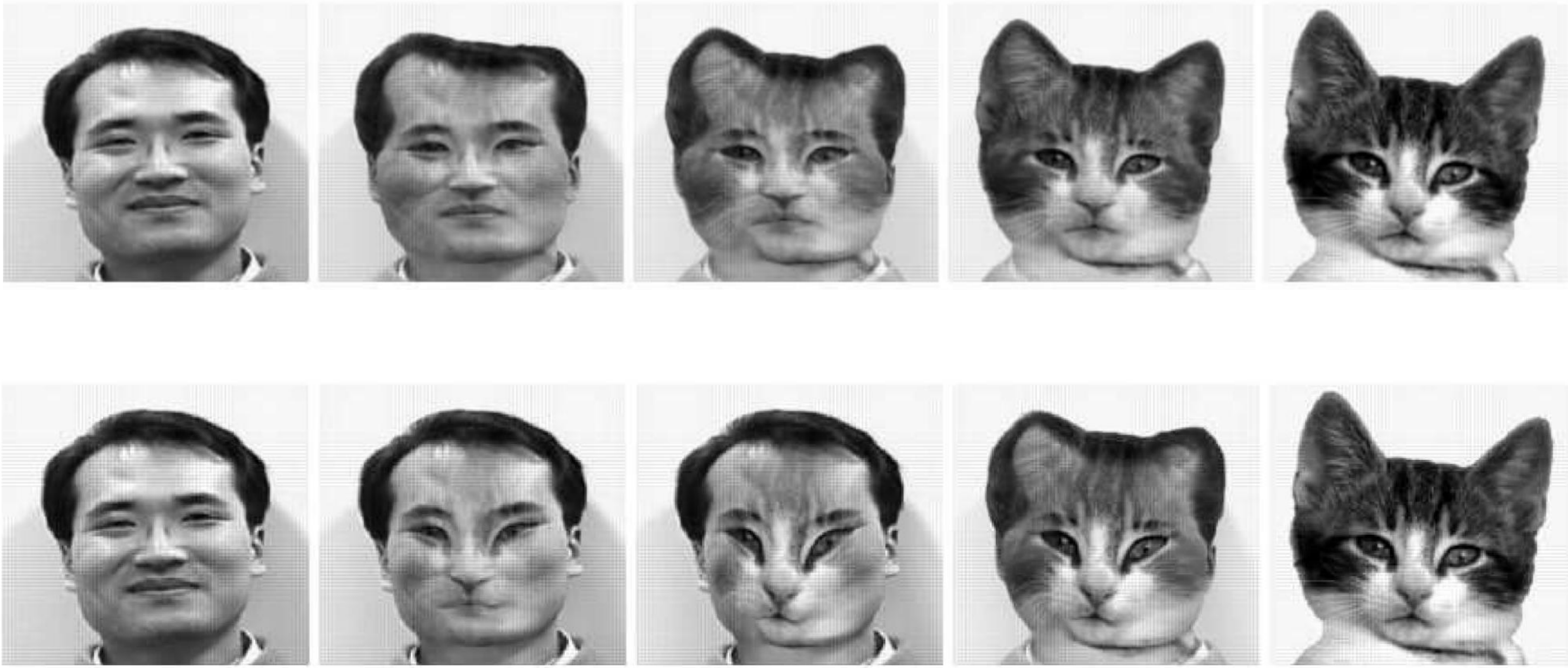
2. Make video $A(t)$ that warps $A$ over time to the shape of $B$
   Start warp field at identity and linearly interpolate to $T_{BA}$
   Construct video $B(t)$ that warps $B$ over time to shape of $A$

3. Cross dissolve these two videos.

# Full Morphing



# Catman!



# Conclusion

Illustrates general principle in graphics:

- First register, then blend

Avoids ghosting

[Michael Jackson - Black or White]