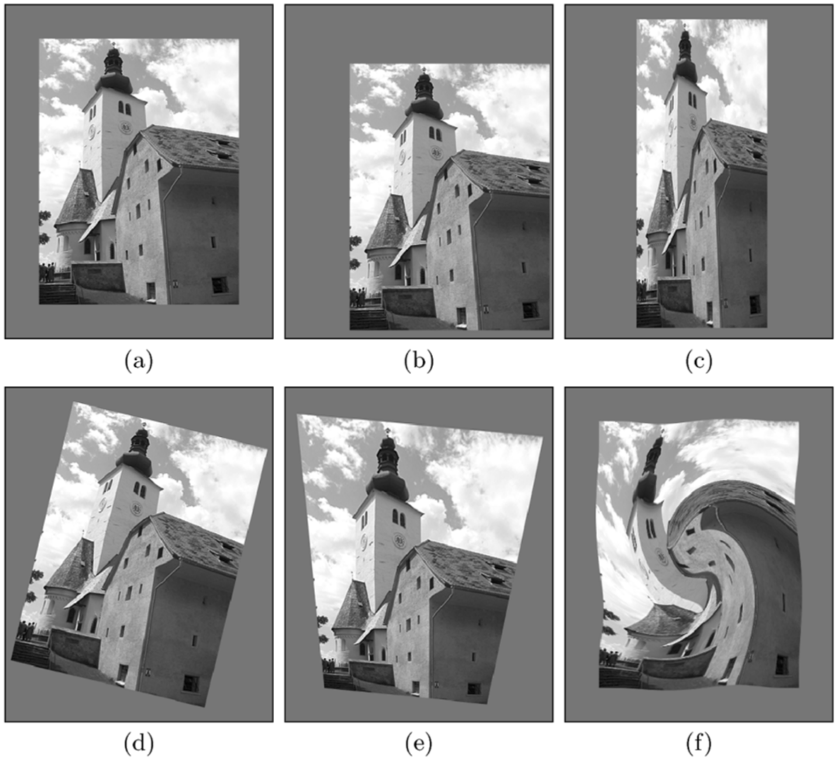
EIE4512 - Digital Image Processing
Geometric Operations, Image Warping and Image Registration

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Zhen Li
lizhen@cuhk.edu.cn
School of Science and Engineering
The Chinese University of Hong Kong, Shen Zhen
April 9-12, 2019

Geometric Operations

- Filters, point operations change intensity
- Pixel position (and geometry) unchanged
- Geometric operations: change image geometry
- **Examples:** translating, rotating, scaling an image



Examples of Geometric operations

Geometric Operations

- Example applications of geometric operations:
  - Zooming images, windows to arbitrary size
  - Computer graphics: deform textures and map to arbitrary surfaces
- **Definition:** Geometric operation transforms image *I* to new image *I'* by modifying **coordinates of image pixels**

$$I(x, y) \rightarrow I'(x', y')$$

- Intensity value originally at (x,y) moved to new position (x',y')

Example: Translation geometric operation moves value at (x,y) to (x + d_x, y + d_y)

Geometric Operations

- Since image coordinates can only be discrete values, some transformations may yield (x’,y’) that’s not discrete
- **Solution:** interpolate nearby values

Simple Mappings

- **Translation:** (shift) by a vector (d_x, d_y)

$$T_x : x' = x + d_x \qquad T_y : y' = y + d_y$$

or

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} d_x \\ d_y \end{pmatrix}$$

- **Scaling:** (contracting or stretching) along x or y axis by a factor s_x or s_y

$$T_x : x' = s_x \cdot x \qquad T_y : y' = s_y \cdot y$$

or

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

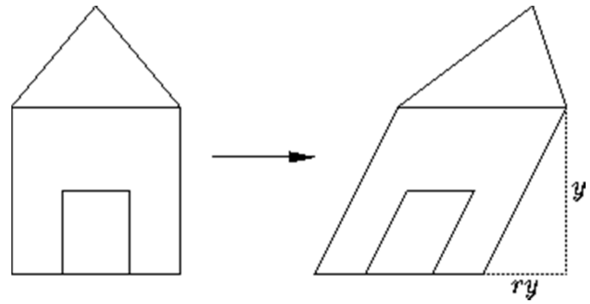
Simple Mappings

- **Shearing:** along x and y axis by factor b_x and b_y

$$T_x : x' = x + b_x \cdot y$$
$$T_y : y' = y + b_y \cdot x$$

or

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & b_x \\ b_y & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$



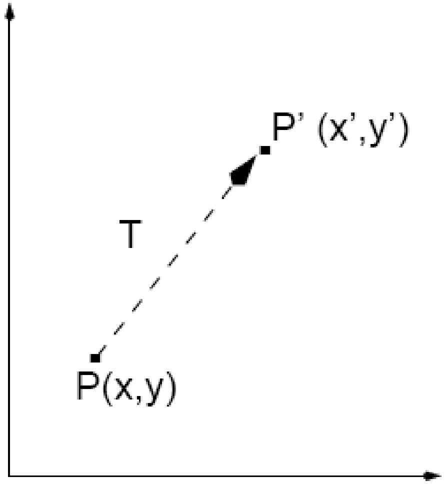
- **Rotation:** the image by an angle α

$$T_x : x' = x \cdot \cos\alpha - y \cdot \sin\alpha$$
$$T_y : y' = x \cdot \sin\alpha + y \cdot \cos\alpha$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

2D Translation

- Moves a point to a new location by adding translation amounts to the coordinates of the point.



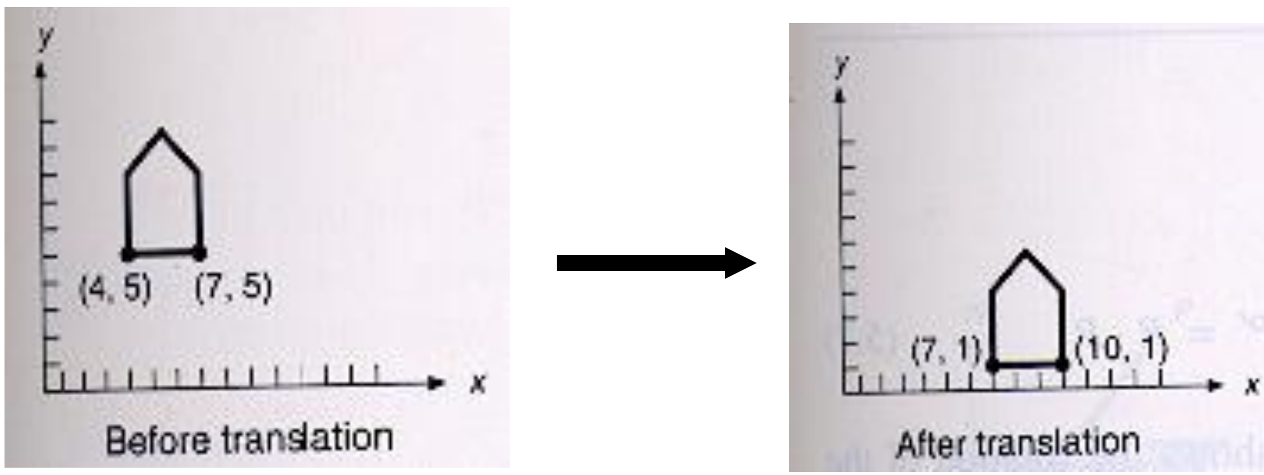
$$x' = x + dx, \quad y' = y + dy$$

or $$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} dx \\ dy \end{bmatrix}$$

or $$\underline{P' = P + T}$$

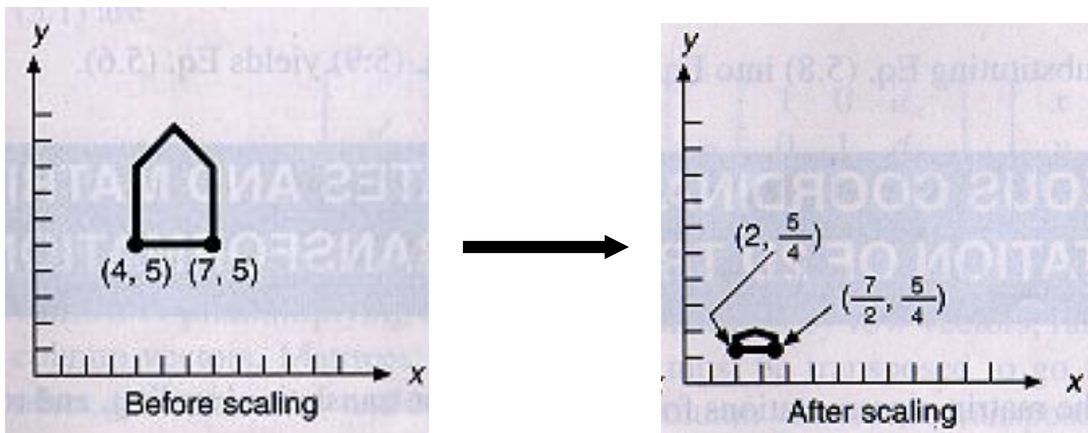
2D Translation (cont'd)

- To translate an object, translate every point of the object by the same amount.



2D Scaling

- Changes the size of the object by multiplying the coordinates of the points by scaling factors.



$x' = x\ s_x,\ \ y' = y\ s_y$ or $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix},\ \ s_x, s_y > 0$ or $\underline{P' = S\ P}$

2D Scaling (cont'd)

- Uniform vs non-uniform scaling

  If $s_x = s_y$ uniform scaling

  If $s_x \neq s_y$ nonuniform scaling

- Effect of scale factors:

If $s_x, s_y < 1$, size is reduced, object moves closer to origin

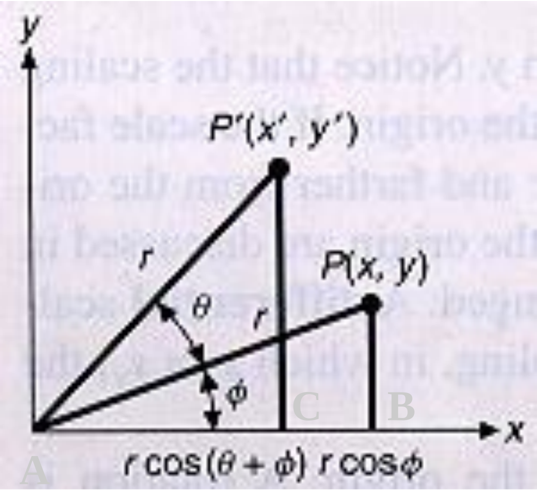
If $s_x, s_y > 1$, size is increased, object moves further from origin

If $s_x = s_y = 1$, size does not change

2D Rotation

- Rotates points by an angle θ about origin

  (θ >0: counterclockwise rotation)



- From *ABP* triangle:

$$\cos(\phi) = x/r \text{ or } x = r\cos(\phi)$$
$$\sin(\phi) = y/r \text{ or } y = r\sin(\phi)$$

- From *ACP*' triangle:

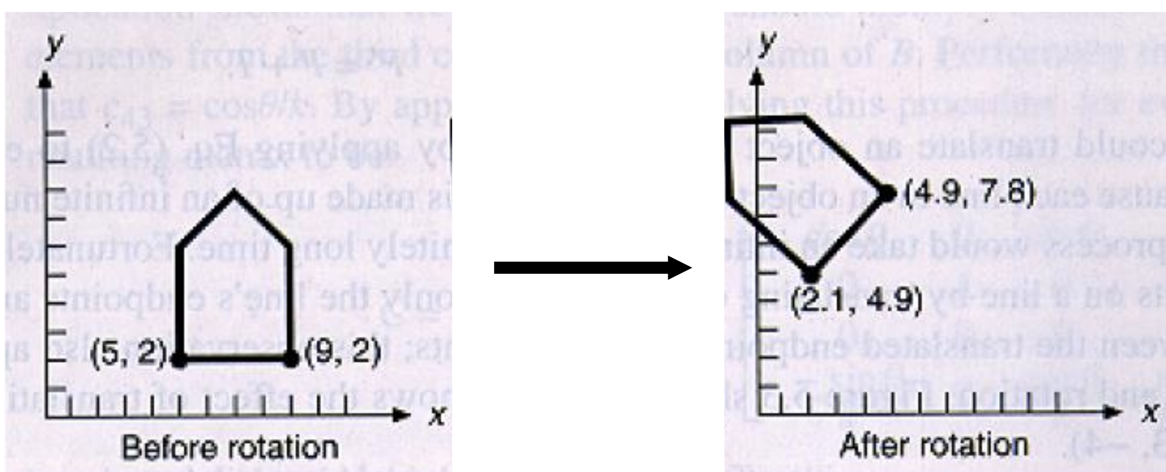
$$\cos(\phi+\theta) = x'/r \text{ or } x' = r\cos(\phi+\theta) = r\cos(\phi)\cos(\theta) - r\sin(\phi)\sin(\theta)$$
$$\sin(\phi+\theta) = y'/r \text{ or } y' = r\sin(\phi+\theta) = r\cos(\phi)\sin(\theta) + r\sin(\phi)\cos(\theta)$$

2D Rotation (cont'd)

- From the above equations we have:

$$x' = x\cos(\theta) - y\sin(\theta), \quad y' = x\sin(\theta) + y\cos(\theta) \qquad \text{or}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \qquad \text{or} \qquad \underline{P' = R\,P}$$



Summary of 2D transformations

Translation: $P' = P + T$

Scale: $P' = S\ P$

Rotation: $P' = R\ P$

- Use *homogeneous* coordinates to express translation as matrix multiplication

Homogeneous Coordinates

- Notation useful for converting scaling, translation, rotating into point-matrix multiplication
- To convert ordinary coordinates into homogeneous coordinates

$$\boldsymbol{x} = \begin{pmatrix} x \\ y \end{pmatrix} \quad \text{converts to} \quad \hat{\boldsymbol{x}} = \begin{pmatrix} \hat{x} \\ \hat{y} \\ h \end{pmatrix} = \begin{pmatrix} h\,x \\ h\,y \\ h \end{pmatrix}$$

Homogeneous coordinates

- Add one more coordinate: $(x,y) \rightarrow (x_h, y_h, w)$
- Recover (x,y) by homogenizing (x_h, y_h, w):

$$x = \frac{x_h}{w}, \quad y = \frac{y_h}{w}, \; w \neq 0$$

- So, $x_h = xw$, $y_h = yw$, $(w \neq 0)$

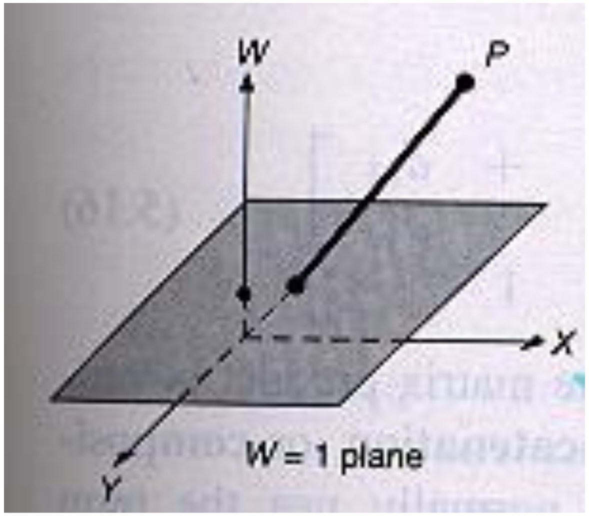
$$(x, y) \rightarrow (xw, yw, w)$$

Homogeneous coordinates (cont'd)

- (x, y) has multiple representations in homogeneous coordinates:
  - $w=1$ $(x,y) \rightarrow (x,y,1)$
  - $w=2$ $(x,y) \rightarrow (2x,2y,2)$

$(w \neq 0)$

- All these points lie on a line in the space of homogeneous coordinates !!



projective space

2D Translation using homogeneous coordinates

w=1

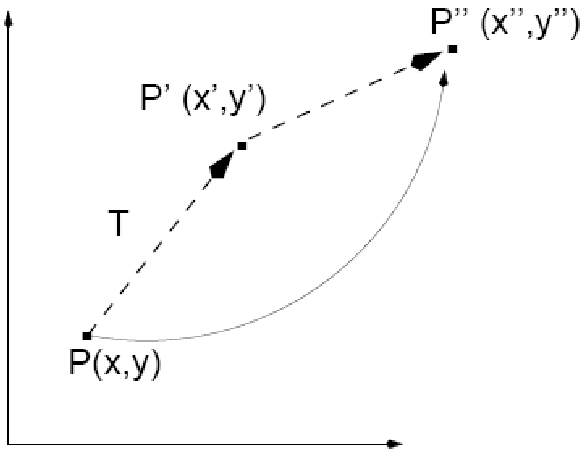
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$x' = x + dx, \quad y' = y + dy$$

$$\underline{P' = T(dx, dy)\ P}$$

2D Translation using homogeneous coordinates (cont'd)

- Successive translations:



$$P' = T(dx_1, dy_1)\ P, \quad P'' = T(dx_2, dy_2)\ P'$$

$$P'' = T(dx_2, dy_2)T(dx_1, dy_1)\ P = T(dx_1 + dx_2, dy_1 + dy_2)\ P$$

$$\begin{bmatrix} 1 & 0 & dx_2 \\ 0 & 1 & dy_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & dx_1 \\ 0 & 1 & dy_1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & dx_1 + dx_2 \\ 0 & 1 & dy_1 + dy_2 \\ 0 & 0 & 1 \end{bmatrix}$$

2D Scaling using homogeneous coordinates

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

w=1

$$x' = x\ s_x,\ \ y' = y\ s_y$$

$$\underline{P' = S(s_x, s_y)\ P}$$

2D Scaling using homogeneous coordinates (cont'd)

- Successive scalings:

$$P' = S(s_{x_1}, s_{y_1})\ P, \quad P'' = S(s_{x_2}, s_{y_2})\ P'$$

$$P'' = S(s_{x_2}, s_{y_2})S(s_{x_1}, s_{y_1})\ P = S(s_{x_1}s_{x_2}, s_{y_1}s_{y_2})\ P$$

$$\begin{bmatrix} s_{x_2} & 0 & 0 \\ 0 & s_{y_2} & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} s_{x_1} & 0 & 0 \\ 0 & s_{y_1} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{x_2}s_{x_1} & 0 & 0 \\ 0 & s_{y_2}s_{y_1} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2D Rotation using homogeneous coordinates

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

w=1

$$x' = xcos(\theta) - ysin(\theta), \quad y' = xsin(\theta) + ycos(\theta)$$

$$\underline{P' = R(\theta)\ P}$$

2D Rotation using homogeneous coordinates (cont'd)

- Successive rotations:

$$P' = R(\theta_1)\ P, \quad P'' = R(\theta_2)\ P'$$

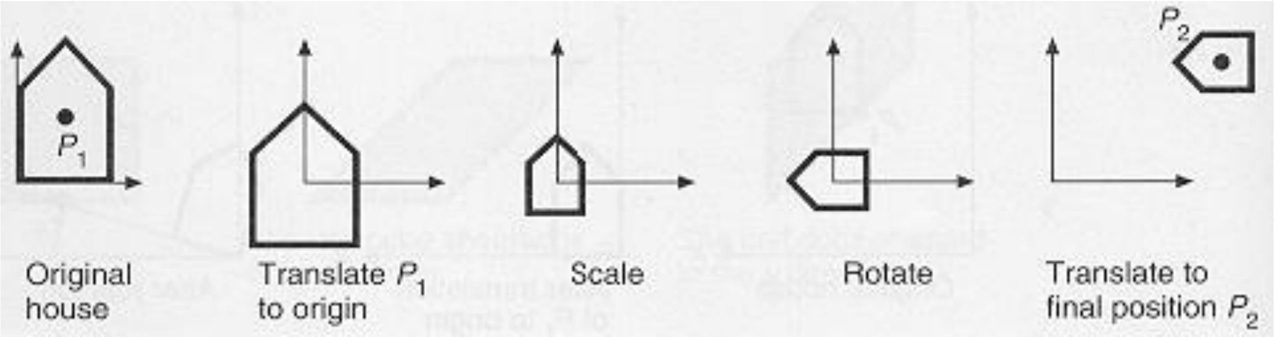
or $$P'' = R(\theta_1)R(\theta_2)\ P = R(\theta_1 + \theta_2)\ P$$

Composition of transformations

- The transformation matrices of a series of transformations can be concatenated into a single transformation matrix.

Example:

* Translate P_1 to origin
* Perform scaling and rotation
* Translate to P_2

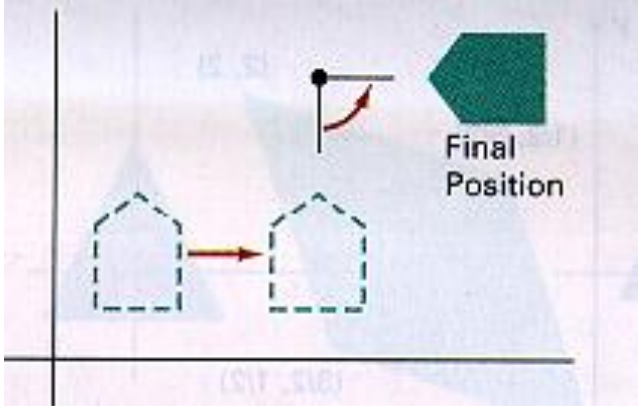


$$M = T(x_2, y_2)R(\theta)S(s_x, s_y)T(-x_1, -y_1)$$

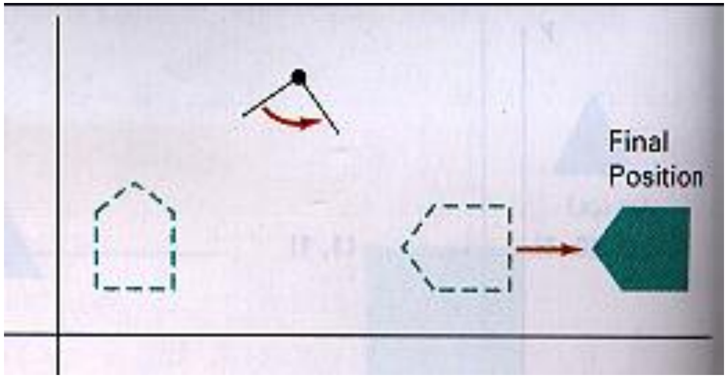
Composition of transformations (cont'd)

- <u>Important:</u> preserve the order of transformations!

translation + rotation



rotation + translation



General form of transformation matrix

$$\begin{array}{c} \\ \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \end{array} = \begin{array}{cc} \text{rotation, scale} & \text{translation} \\ \left[\begin{array}{cc} \boxed{\begin{matrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{matrix}} \\ 0 \quad 0 \end{array}\right. & \left.\begin{array}{c} \boxed{\begin{matrix} a_{13} \\ a_{23} \end{matrix}} \\ 1 \end{array}\right] \end{array} \begin{array}{c} \\ \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \end{array}$$

• Representing a sequence of transformations as a single transformation matrix is more efficient!

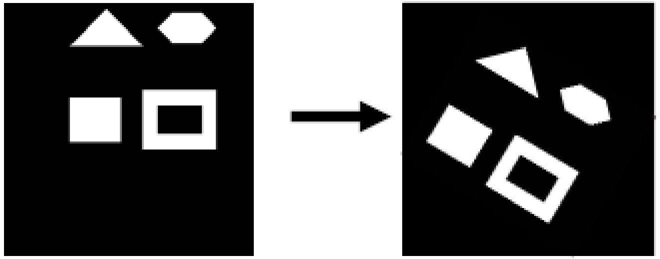
$$x' = a_{11}x + a_{12}y + a_{13}$$

$$y' = a_{21}x + a_{22}y + a_{23}$$

(only 4 multiplications and 4 additions)

Special cases of transformations

- Rigid transformations
  - Involves only translation and rotation (3 parameters)
  - Preserve angles and lengths



$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

upper 2x2 submatrix is ortonormal

$$u_1 = (r_{11}, r_{12}),\ u_2 = (r_{21}, r_{22})$$

$$u_1 . u_1 = \|u_1\|^2 = r_{11}^2 + r_{12}^2 = 1$$

$$u_2 . u_2 = \|u_2\|^2 = r_{21}^2 + r_{22}^2 = 1$$

$$u_1 . u_2 = r_{11} r_{21} + r_{12} r_{22} = 0$$

Example: rotation matrix

$$\begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

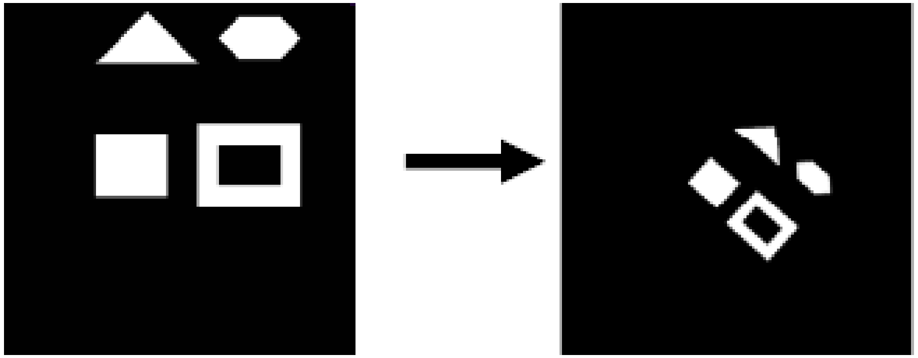
$$u_1 . u_1 = cos(\theta)^2 + sin(-\theta)^2 = 1$$

$$u_2 . u_2 = cos(\theta)^2 + sin(\theta)^2 = 1$$

$$u_1 . u_2 = cos(\theta)sin(\theta) - sin(\theta)cos(\theta) = 0$$

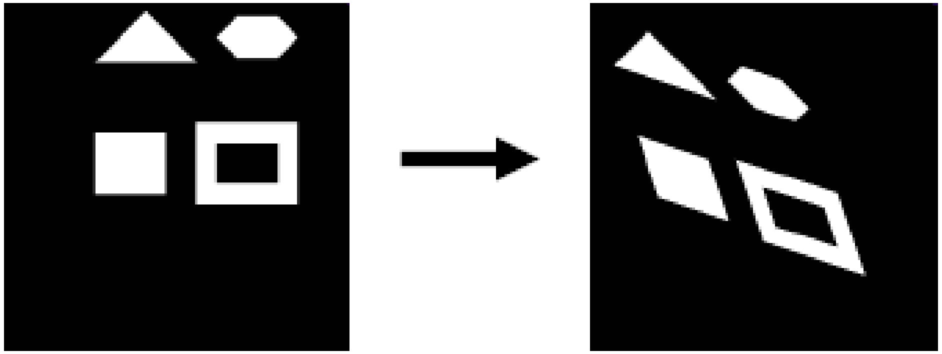
Special cases of transformations

- Similarity transformations
  - Involve rotation, translation, scaling (4 parameters)
  - Preserve angles but not lengths



Affine transformations

- Involve translation, rotation, scale, and <u>shear</u> (6 parameters)
- Preserve parallelism of lines but <u>not</u> lengths and angles.



2D shear transformation

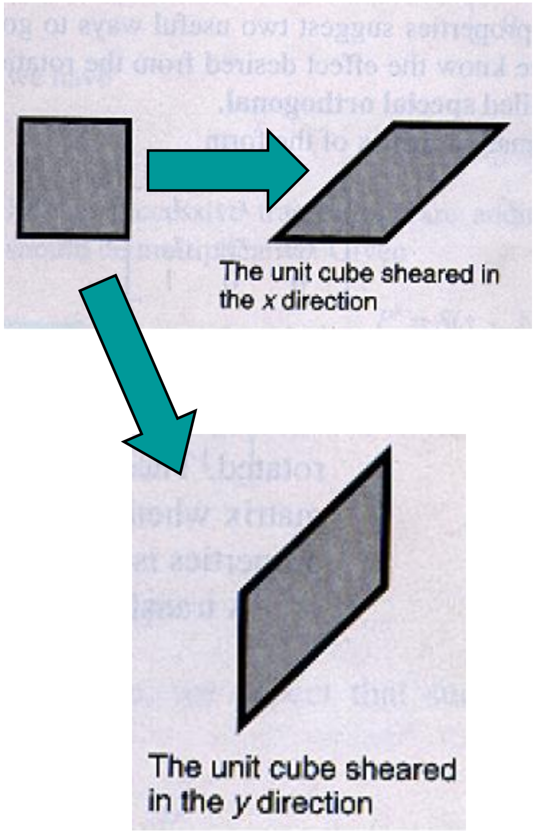
- Shearing along x-axis:

$$x' = x + ay,\ y' = y \qquad SH_x = \begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Shearing along y-axis

$$x' = x,\ y' = bx + y \qquad SH_y = \begin{bmatrix} 1 & 0 & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

changes object shape!



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]

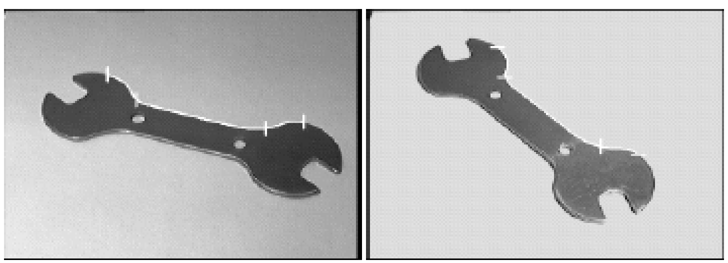
With notation used in this lecture note

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

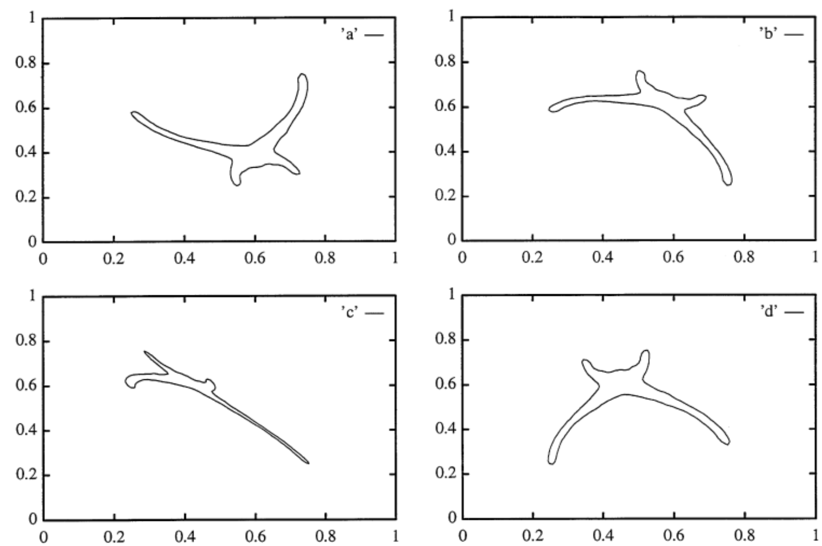
Note in this example, first coordinate indicates horizontal position, second coordinate indicate vertic

Affine Transformations

- Under certain assumptions, affine transformations can be used to approximate the effects of perspective projection!



affine transformed object



G. Bebis, M. Georgiopoulos, N. da Vitoria Lobo, and M. Shah, " Recognition by learning affine transformations", **Pattern Recognition**, Vol. 32, No. 10, pp. 1783-1799, 1999.

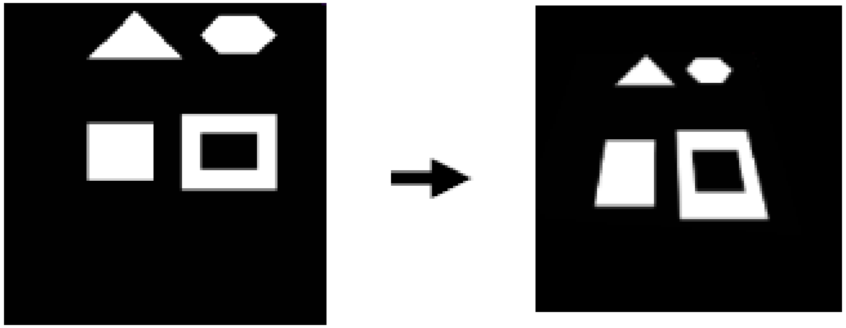
Projective Transformations

affine (6 parameters)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} A & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

projective (8 parameters)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} A & t \\ \boxed{v} & b \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

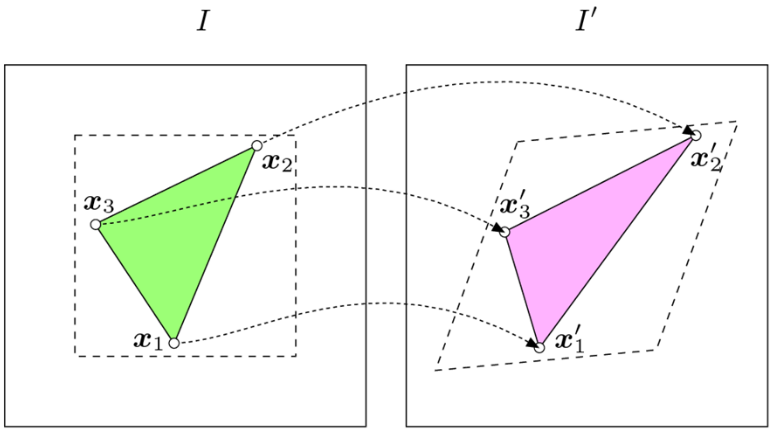


Affine (3-Point) Mapping

- Can use homogeneous coordinates to rewrite translation, rotation, scaling, etc as vector-matrix multiplication

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

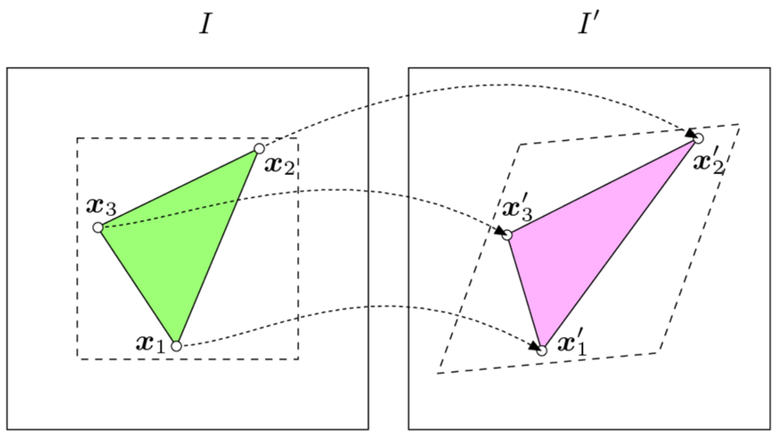
- **Affine mapping:** Can then derive values of matrix that achieve desired transformation (or combination of transformations)



- Inverse of transform matrix is **inverse mapping**

Affine (3-Point) Mapping

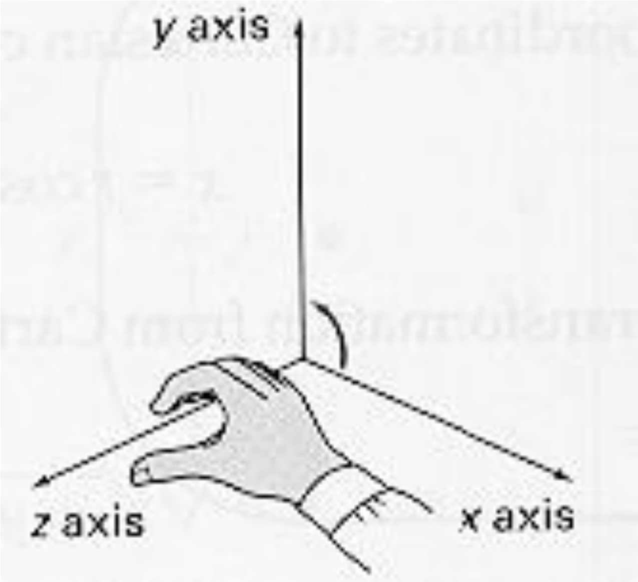
- What's so special about affine mapping?

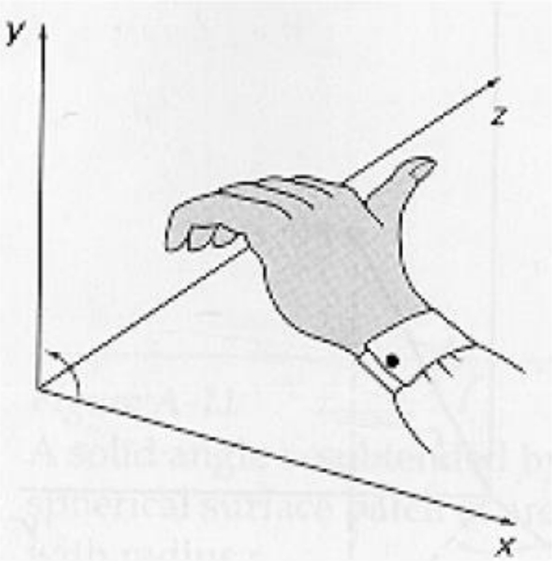


- Maps
  - straight lines -> straight lines,
  - triangles -> triangles
  - rectangles -> parallelograms
  - Parallel lines -> parallel lines
- Distance ratio on lines do not change

3D Transformations

- **Right-handed / left-handed** systems

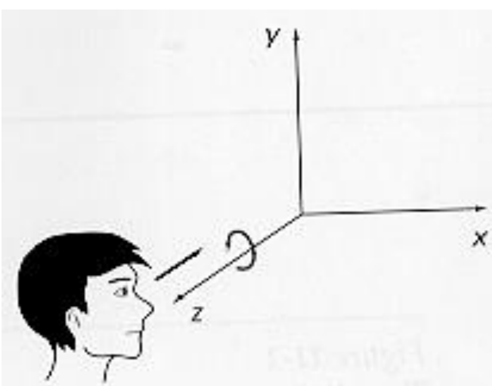


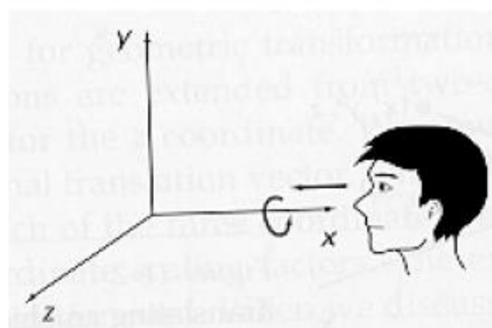


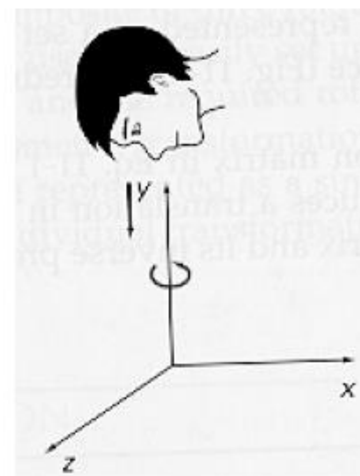
3D Transformations (cont'd)

- Positive rotation angles for right-handed systems:

(counter-clockwise rotations)







Homogeneous coordinates

- Add one more coordinate: $(x,y,z) \rightarrow (x_h, y_h, z_h, w)$
- Recover (x,y,z) by homogenizing (x_h, y_h, z_h, w):

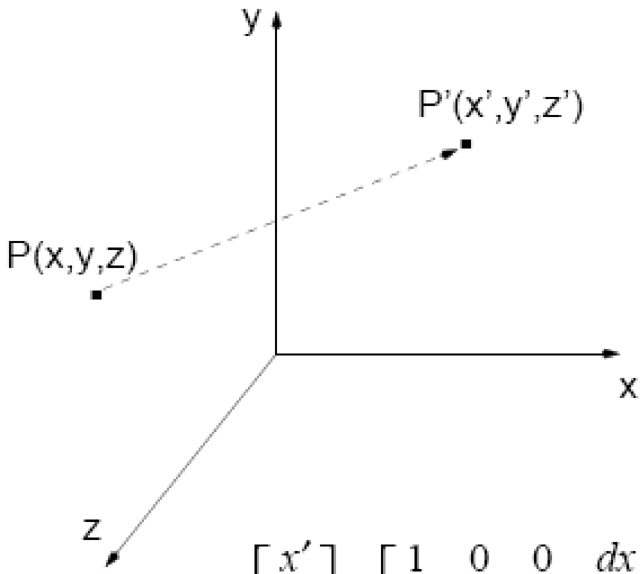
$$x = \frac{x_h}{w},\ \ y = \frac{y_h}{w},\ z = \frac{z_h}{w},\ w \neq 0$$

- In general, $x_h = xw$, $y_h = yw$, $z_h = zw$

$$(x, y, z) \rightarrow (xw, yw, zw, w) \quad (w \neq 0)$$

- Each point (x, y, z) corresponds to a line in the 4D-space of homogeneous coordinates.

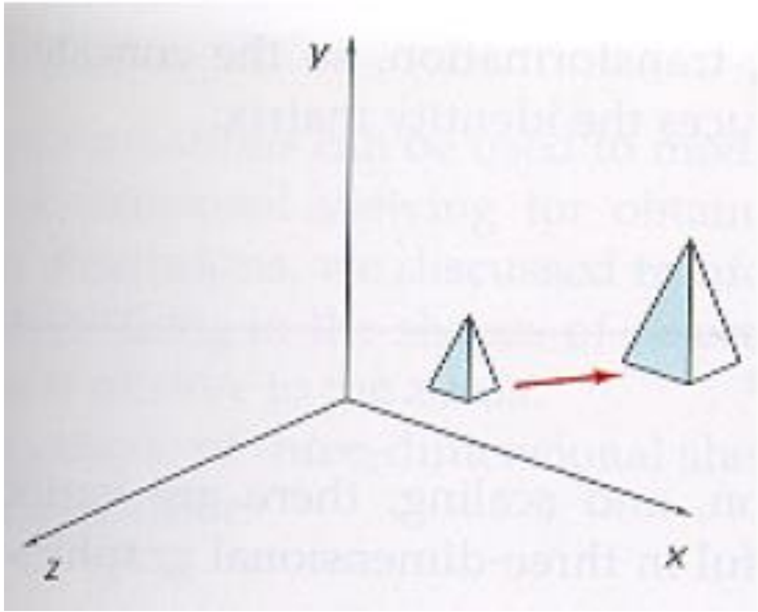
3D Translation



$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\underline{P' = T(dx, dy, dz)\ P}$$

3D Scaling

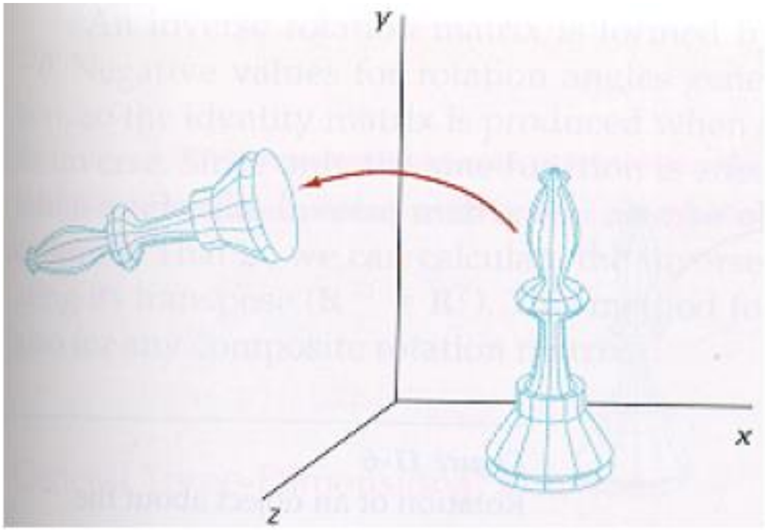


$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\underline{P' = S(s_x, s_y, s_z)\ P}$$

3D Rotation

- Rotation about the *z*-axis:



$$x' = x\cos(\theta) - y\sin(\theta)$$

$$y' = x\sin(\theta) + y\cos(\theta)$$

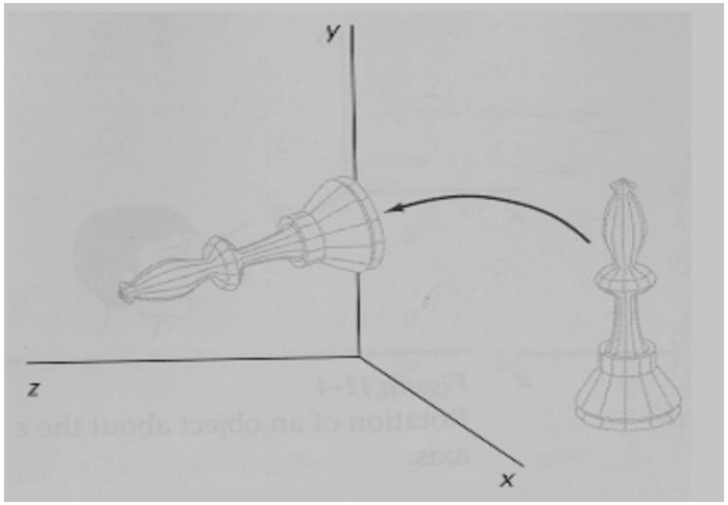
$$z' = z$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\underline{P' = R_z(\theta)\ P}$$

3D Rotation (cont'd)

- Rotation about the *x*-axis:



$$x' = x$$

$$y' = y\cos(\theta) - z\sin(\theta)$$

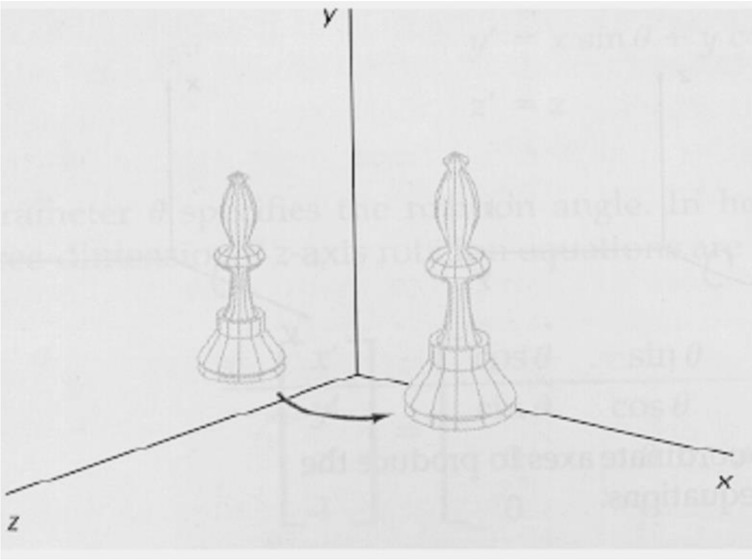
$$z' = y\sin(\theta) + z\cos(\theta)$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\underline{P' = R_x(\theta)\ P}$$

3D Rotation (cont'd)

- Rotation about the *y*-axis



$$x' = z\sin(\theta) + x\cos(\theta)$$

$$y' = y$$

$$z' = z\cos(\theta) - x\sin(\theta)$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

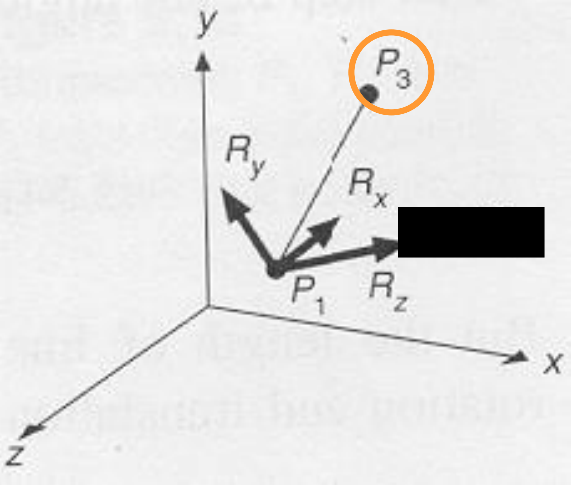
$$\underline{P' = R_y(\theta)\ P}$$

Change of coordinate systems

- Suppose that the coordinates of P_3 are given in the *xyz* coordinate system
- How can you compute its coordinates in the $R_xR_yR_z$ coordinate system?

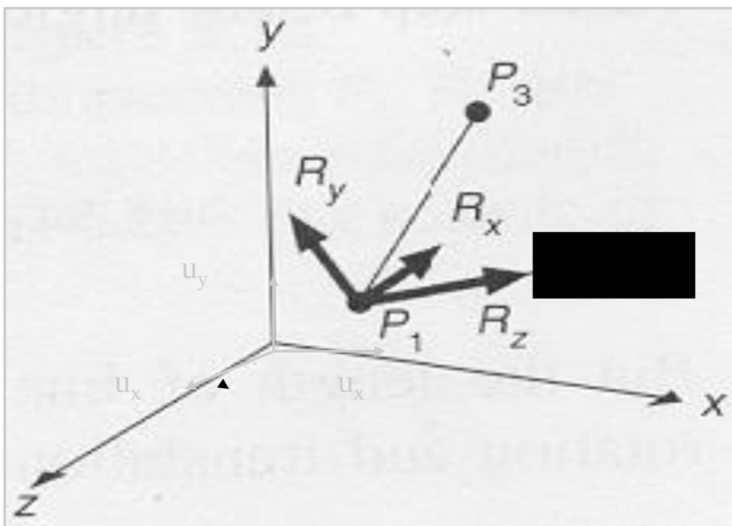
(1) Recover the translation T and rotation R from $R_xR_yR_z$ to *xyz*. that aligns $R_xR_yR_z$ with xyz

(2) Apply T and R on P_3 to compute its coordinates in the $R_xR_yR_z$ system.



(1.1) Recover translation T

- If we know the coordinates of P_1 (i.e., origin of $R_x R_y R_z$) in the xyz coordinate system, then T is:



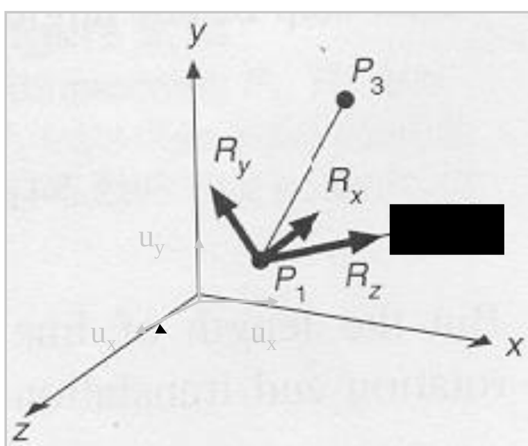
$$T=\begin{pmatrix} 1 & 0 & 0 & -P_{1x} \\ 0 & 1 & 0 & -P_{1y} \\ 0 & 0 & 1 & -P_{1z} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(1.2) Recover rotation R

- u_x, u_y, u_z are <u>unit vectors</u> in the *xyz* coordinate system.

- r_x, r_y, r_z are <u>unit vectors</u> in the $R_xR_yR_z$ coordinate system
  (r_x, r_y, r_z are represented in the *xyz* coordinate system)

- Find rotation R: $r_z \rightarrow u_z$, $r_x \rightarrow u_x$, and $r_y \rightarrow u_y$



R

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Change of coordinate systems: recover rotation R (cont'd)

$$u_z = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{z_x} \\ r_{z_y} \\ r_{z_z} \\ 1 \end{bmatrix} \quad (r_z \to u_z) \qquad (1)$$

$$u_x = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{x_x} \\ r_{x_y} \\ r_{x_z} \\ 1 \end{bmatrix} \quad (r_x \to u_x) \qquad (2)$$

$$u_y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{y_x} \\ r_{y_y} \\ r_{y_z} \\ 1 \end{bmatrix} \quad (r_y \to u_y) \qquad (3)$$

Change of coordinate systems: recover rotation R (cont'd)

From (1): $a_3^T r_z = 1$ or $a_3 = r_z$

From (2): $a_1^T r_x = 1$ or $a_1 = r_x$

From (3): $a_2^T r_y = 1$ or $a_2 = r_y$

Thus, the rotation matrix R is given by:

$$R = \begin{bmatrix} & r_x^T & & 0 \\ & r_y^T & & 0 \\ & r_z^T & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

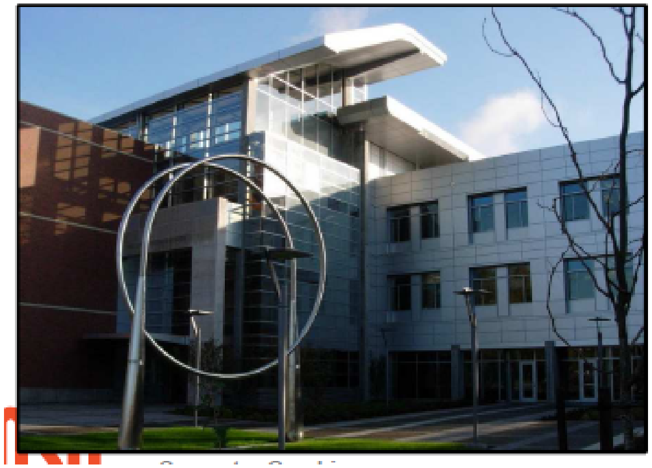
Change of coordinate systems: recover rotation R (cont'd)

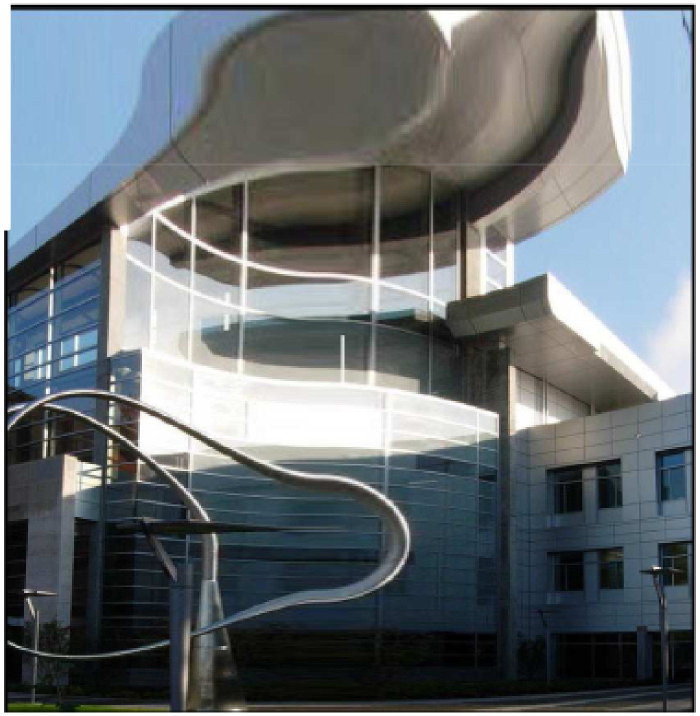
- Verify that it performs the correct mapping:

$r_x \rightarrow u_x$ $\quad$ $r_y \rightarrow u_y$ $\quad$ $r_z \rightarrow u_z$

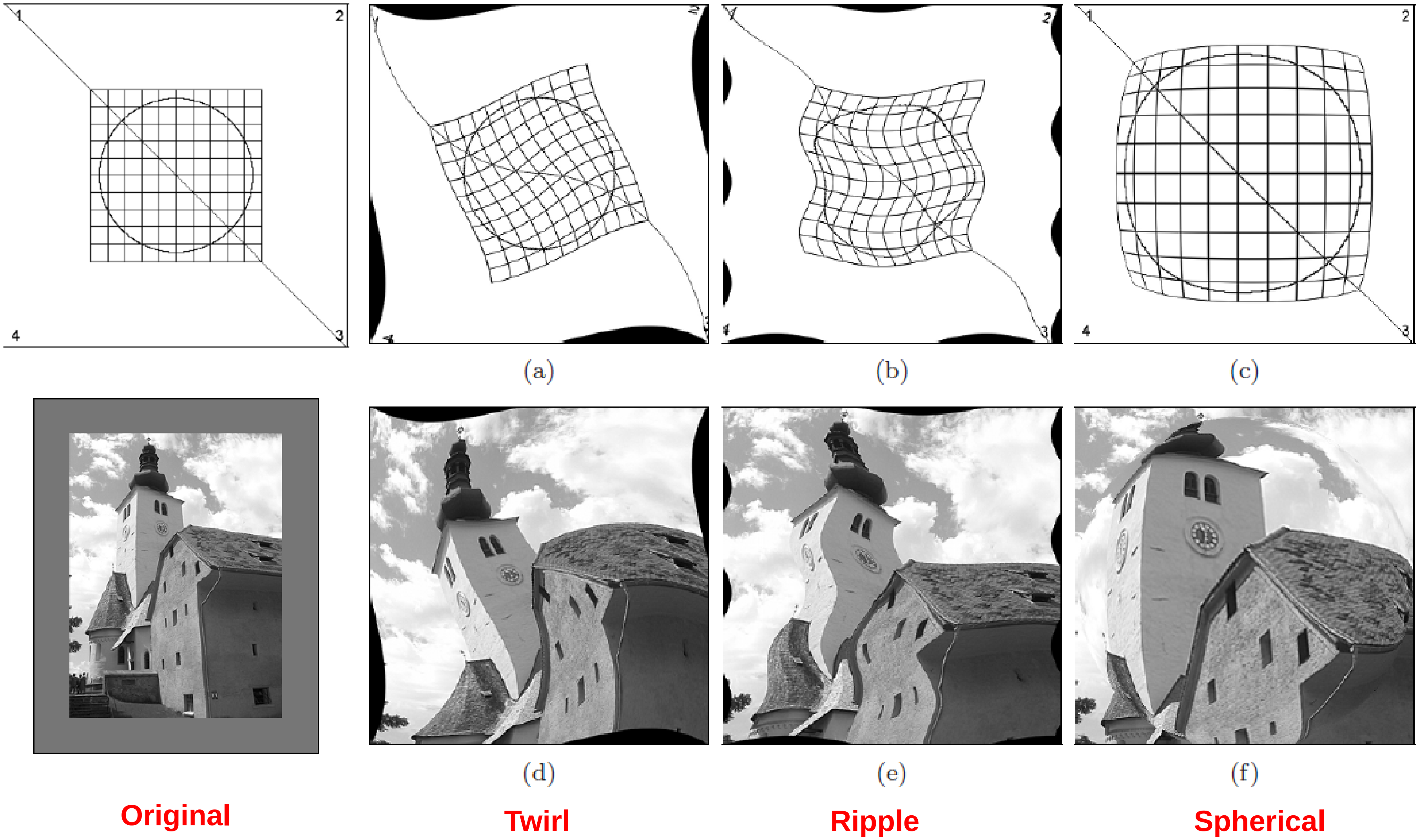
$$R\begin{bmatrix} r_x \\ \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad R\begin{bmatrix} r_y \\ \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \qquad R\begin{bmatrix} r_z \\ \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Image Warping





Non-Linear Image Warps



Twirl

- **Notation:** Instead using texture colors at (x',y'), use texture colors at twirled (x,y) location
- Twirl?
  - Rotate image by angle $\boldsymbol{\alpha}$ at center or anchor point (x_c, y_c)
  - Increasingly rotate image as radial distance $\boldsymbol{r}$ from center increases (up to $\boldsymbol{r_{max}}$)
  - Image unchanged outside radial distance $\boldsymbol{r_{max}}$

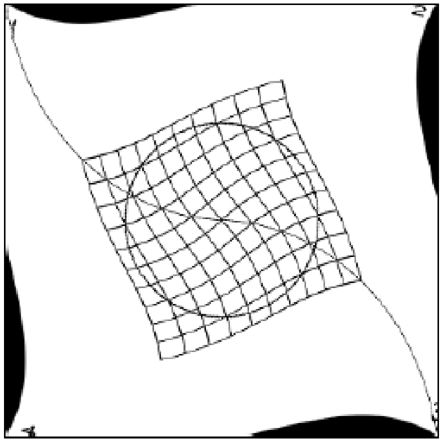
$$T_x^{-1}: \quad x = \begin{cases} x_c + r \cdot \cos(\beta) & \text{for } \ r \le r_{\max} \\ x' & \text{for } \ r > r_{\max}, \end{cases}$$

$$T_y^{-1}: \quad y = \begin{cases} y_c + r \cdot \sin(\beta) & \text{for } \ r \le r_{\max} \\ y' & \text{for } \ r > r_{\max}, \end{cases}$$

with

$$d_x = x' - x_c, \qquad r = \sqrt{d_x^2 + d_y^2},$$

$$d_y = y' - y_c, \qquad \beta = \operatorname{Arctan}(d_y, d_x) + \alpha \cdot \left(\tfrac{r_{\max} - r}{r_{\max}}\right).$$



(a)

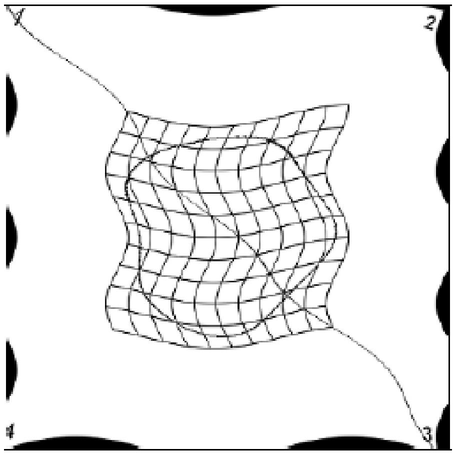
(d)

Ripple

- Ripple causes wavelike displacement of image along both the x and y directions

$$T_x^{-1}: \quad x = x' + a_x \cdot \sin\left(\frac{2\pi \cdot y'}{\tau_x}\right),$$
$$T_y^{-1}: \quad y = y' + a_y \cdot \sin\left(\frac{2\pi \cdot x'}{\tau_y}\right).$$

- Sample values for parameters (in pixels) are
  - τ_x= 120
  - τ_y= 250
  - a_x= 10
  - a_y= 15



(b)


(e)

Spherical Transformation

- Imitates viewing image through a lens placed over image
- Lens parameters: center (x_c, y_c), lens radius r_{max} and refraction index ρ
- Sample values ρ = 1.8 and r_{max} = half image width

$$T_x^{-1}: \quad x = x' - \begin{cases} z \cdot \tan(\beta_x) & \text{for } r \le r_{\max} \\ 0 & \text{for } r > r_{\max}, \end{cases}$$

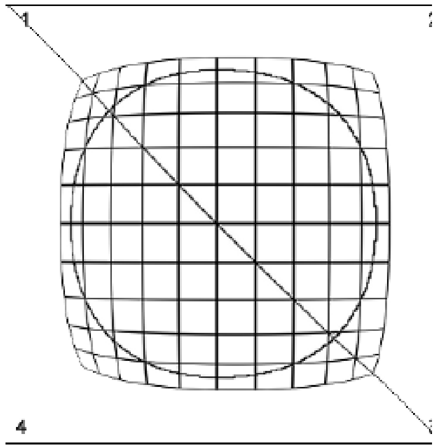
$$T_y^{-1}: \quad y = y' - \begin{cases} z \cdot \tan(\beta_y) & \text{for } r \le r_{\max} \\ 0 & \text{for } r > r_{\max}, \end{cases}$$

$$d_x = x' - x_c, \qquad r = \sqrt{d_x^2 + d_y^2}\ ,$$

$$d_y = y' - y_c, \qquad z = \sqrt{r_{\max}^2 - r^2}\ ,$$

$$\beta_x = \left(1 - \tfrac{1}{\rho}\right) \cdot \sin^{-1}\left(\frac{d_x}{\sqrt{(d_x^2 + z^2)}}\right),$$

$$\beta_y = \left(1 - \tfrac{1}{\rho}\right) \cdot \sin^{-1}\left(\frac{d_y}{\sqrt{(d_y^2 + z^2)}}\right).$$



(c)

(f)

Polynomial Warping

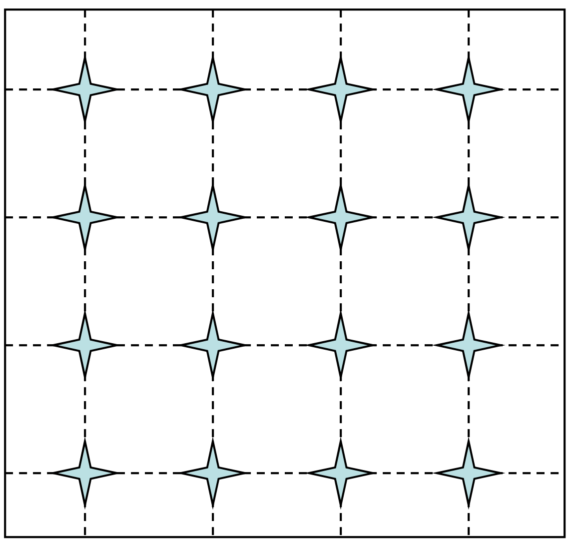
- The polynomial warping includes all deformations that can be modeled by polynomial transformations:

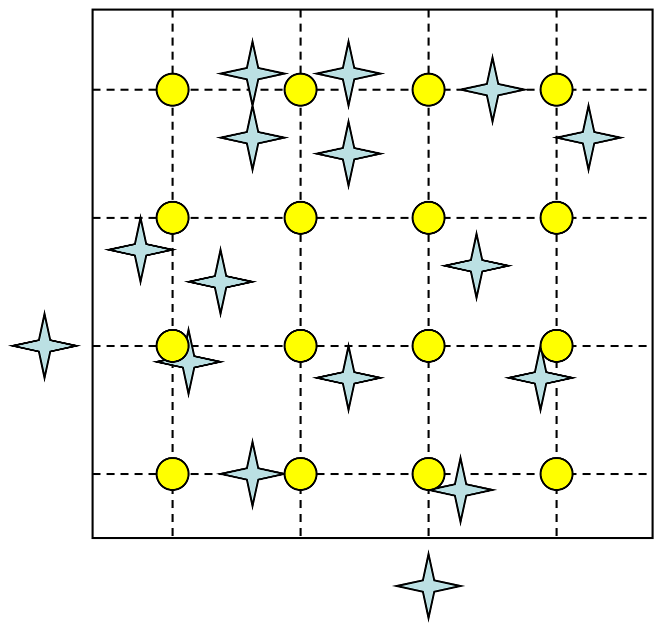
$$\begin{cases} x = a_0 + a_1 u + a_2 v + a_3 uv + a_4 u^2 + a_5 v^2 + \cdots \\ y = b_0 + b_1 u + b_2 v + b_3 uv + b_4 u^2 + b_5 v^2 + \cdots \end{cases}$$

- Includes affine and bilinear mapping as special cases

Image Warping by Forward Mapping

- Mapping image f(u, v) to g(x, y) based on a given mapping function: x(u, v), y(u, v).
- Forward Mapping
  - For each point (u, v) in the original image, find the corresponding position (x, y) in the deformed image by the forward mapping function, and let g(x,y)=f(u,v).
  - What if the mapped position (x,y) is not an integer sample in the desired image?



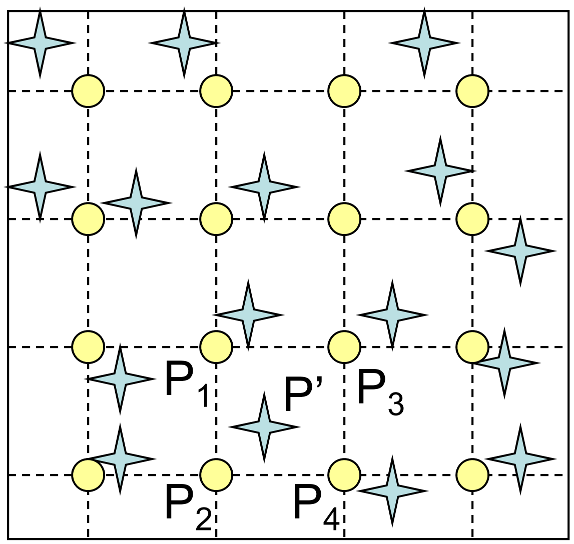


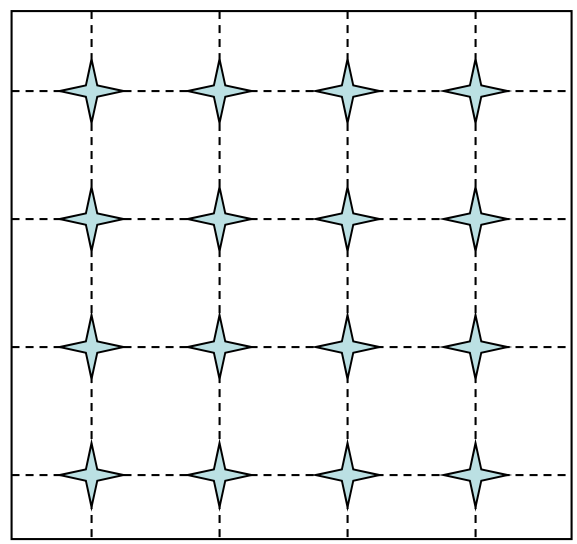
Warping points are often non-integer samples

Many integer samples "o" are not assigned Values

Image Warping by Inverse Mapping

- For each point (x, y) in the image to be obtained, find its corresponding point (u, v) in the original image using the inverse mapping function, and let g(x, y) = f(u, v).
- What if the mapped point (u,v) is not an integer sample?
  - Interpolate from nearby integer samples!





P' will be interpolated from P_1, P_2, P_3, and P_4

Interpolation Method

- Nearest neighbor:
  – Round (u,v) to the nearest integer samples
- Bilinear interpolation:
  – find four integer samples nearest to (u,v), apply bilinear interpolation
- Other higher order interpolation methods can also be used
  – Requiring more than 4 nearest integer samples!

How to find inverse mapping

- Invert the forward mapping

If $\mathbf{x} = \mathbf{A}\mathbf{u} + \mathbf{b}$

Then $\mathbf{u} = \mathbf{A}^{-1}(\mathbf{x} - \mathbf{b})$

- Directly finding inverse mapping from point correspondence

MATLAB function: interp2

- ZI = INTERP2(X,Y,Z,XI,YI, METHOD) interpolates to find ZI, the values of the underlying 2-D function Z at the points in matrices XI and YI.
  - Matrices X and Y specify the points at which the data Z is given.
  - METHOD specifies interpolation filter
    - 'nearest' - nearest neighbor interpolation
    - 'linear' - bilinear interpolation
    - 'spline' - spline interpolation
    - 'cubic' - bicubic interpolation as long as the data is uniformly spaced, otherwise the same as 'spline'

Using ‘interp2’ to realize image warping

- Use inverse mapping
- Step 1: For all possible pixels in output image (x,y), find corresponding points in the input image (u,v)
  - (X,Y)=meshgrid(1:M,1:N)
  - Apply inverse mapping function to find corresponding (u,v), for every (x,y), store in (UI,VI)
    - Can use tforminv( ) function if you derived the transformation using maketform().
    - Or write your own code using the specified mapping
- Step 2: Use interp2 to interpret the value of the input image at (UI,VI) from their values at regularly sampled points (U,V)
  - (U,V)=meshgrid(1:M,1:N)
  - Outimg=interp2(U,V,inimg,UI,VI,’linear’);

MATLAB function for image warping

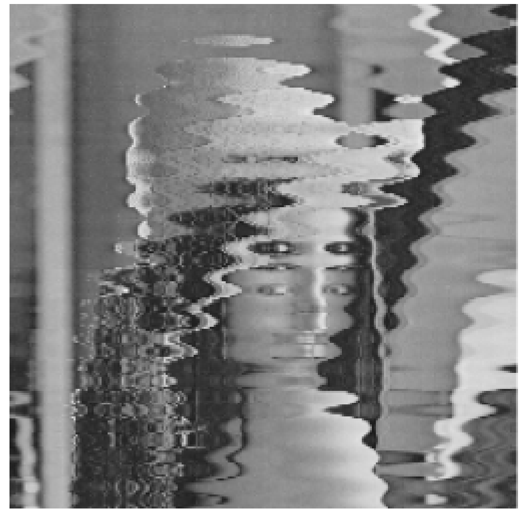
- B = IMTRANSFORM(A,TFORM, INTERP) transforms the image A according to the 2-D spatial transformation defined by TFORM
- INTERP specifies the interpolation filter
- Example 1
- ---------
- Apply a horizontal shear to an intensity image.
-
- I = imread('cameraman.tif');
- tform = maketform('affine',[1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

https://www.mathworks.com/help/images/ref/imtransform.html?searchHighlight=IMTRANSFORM&s_tid=doc_srchtitle

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

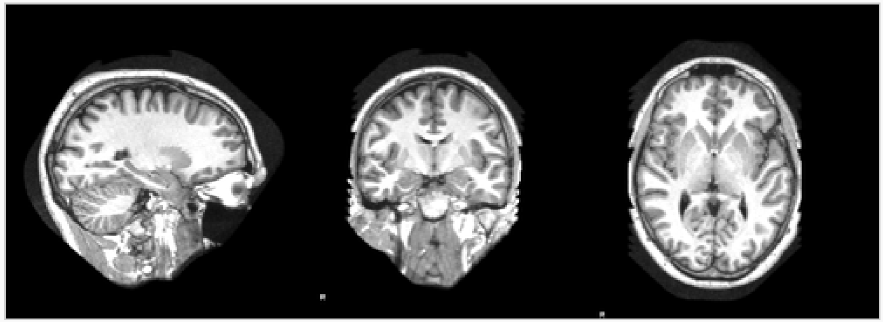
SWIRL
$$x(u,v) = (u - x_0)\cos(\theta) + (v - y_0)\sin(\theta) + x_0;$$
$$y(u,v) = -(u - x_0)\sin(\theta) + (v - y_0)\cos(\theta) + y_0;$$
$$r = ((u - x_0)^2 + (v - y_0)^2)^{1/2}, \theta = \pi r / 512.$$

By Onur Guleyuz

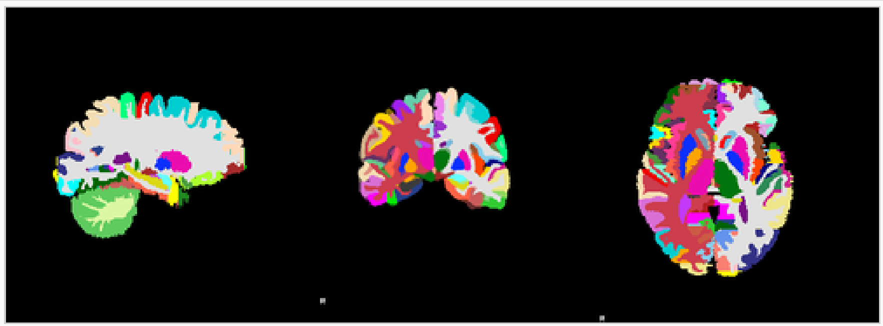
Image Registration

- Suppose we are given two images taken at different times of the same object. To observe the changes between these two images, we need to make sure that they are aligned properly. To obtain this goal, we need to find the correct mapping function between the two. The determination of the mapping functions between two images is known as the registration problem.
- Once the mapping function is determined, the alignment step can be accomplished using the warping methods.

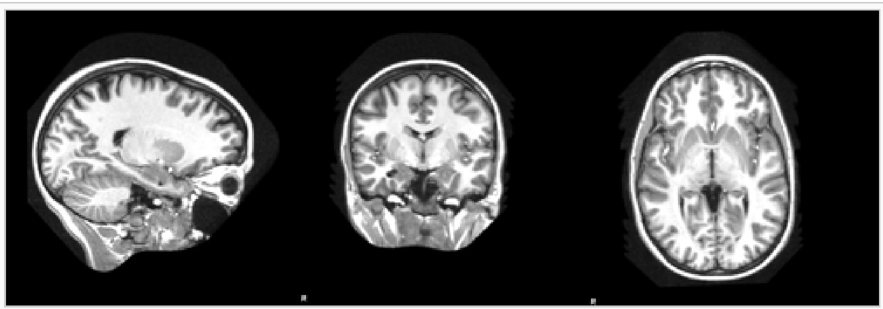
Image Registration



template.nii



labels.nii



new_image.nii

Assuming a template image (template.nii) and its associated segmentation (labels.nii), one can transfer the label information into the space of another image (new_image.nii).

<u>http://cmictig.cs.ucl.ac.uk/wiki/index.php/NiftyReg_Segmentation_Propagation_Tutorial</u>

How to find the mapping function?

- Assume the mapping function is a polynomial of order N
- Step 1: Identify K≥N corresponding points between two images, i.e.

$$(u_i, v_i) \leftrightarrow (x_i, y_i), i = 1,2...,K.$$

- Step 2: Determine the coefficients a_i, b_i, i = 0,…,N-1 by solving

$$\begin{cases} x(u_i, v_i) = a_0 + a_1 u_i + a_2 v_i + \cdots = x_i, \\ y(u_i, v_i) = b_0 + b_1 u_i + b_2 v_i + \cdots = y_i, \end{cases} \quad i = 1,2,...,K$$

- How to solve this?

How to Solve the Previous Equations?

- Convert to matrix equation:

$$\mathbf{A}\mathbf{a} = \mathbf{x}, \quad \mathbf{A}\mathbf{b} = \mathbf{y}$$

$$where$$

$$\mathbf{A} = \begin{bmatrix} 1 & u_1 & v_1 & \cdots \\ 1 & u_2 & v_2 & \cdots \\ \vdots & \vdots & \vdots & \ddots \\ 1 & u_K & v_K & \cdots \end{bmatrix}, \mathbf{a} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{bmatrix}, \mathbf{b} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{N-1} \end{bmatrix}, \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_K \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_K \end{bmatrix}$$

If K = N, and the matrix **A** is non-singular, then

$$\mathbf{a} = \mathbf{A}^{-1}\mathbf{x}, \quad \mathbf{b} = \mathbf{A}^{-1}\mathbf{y}$$

If K > N, then we can use a least square solution

$$\mathbf{a} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{x}, \quad b = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T y$$

If K < N, or **A** is singular, then more corresponding feature points must be identified.

Examples

- If we want to use an affine mapping to register to images, we need to find 3 or more pairs of corresponding points
- If we have only 3 pairs, we can solve the mapping parameters exactly as before
- If we have more than 3 pairs, these pairs may not all be related by an affine mapping. We find the “least squares fit” by solving an over-determined system of equations

MATLAB function: cp2tform()

TFORM=CP2TFORM(INPUT_POINTS,BASE_POINTS,TRANSFORM TYPE)

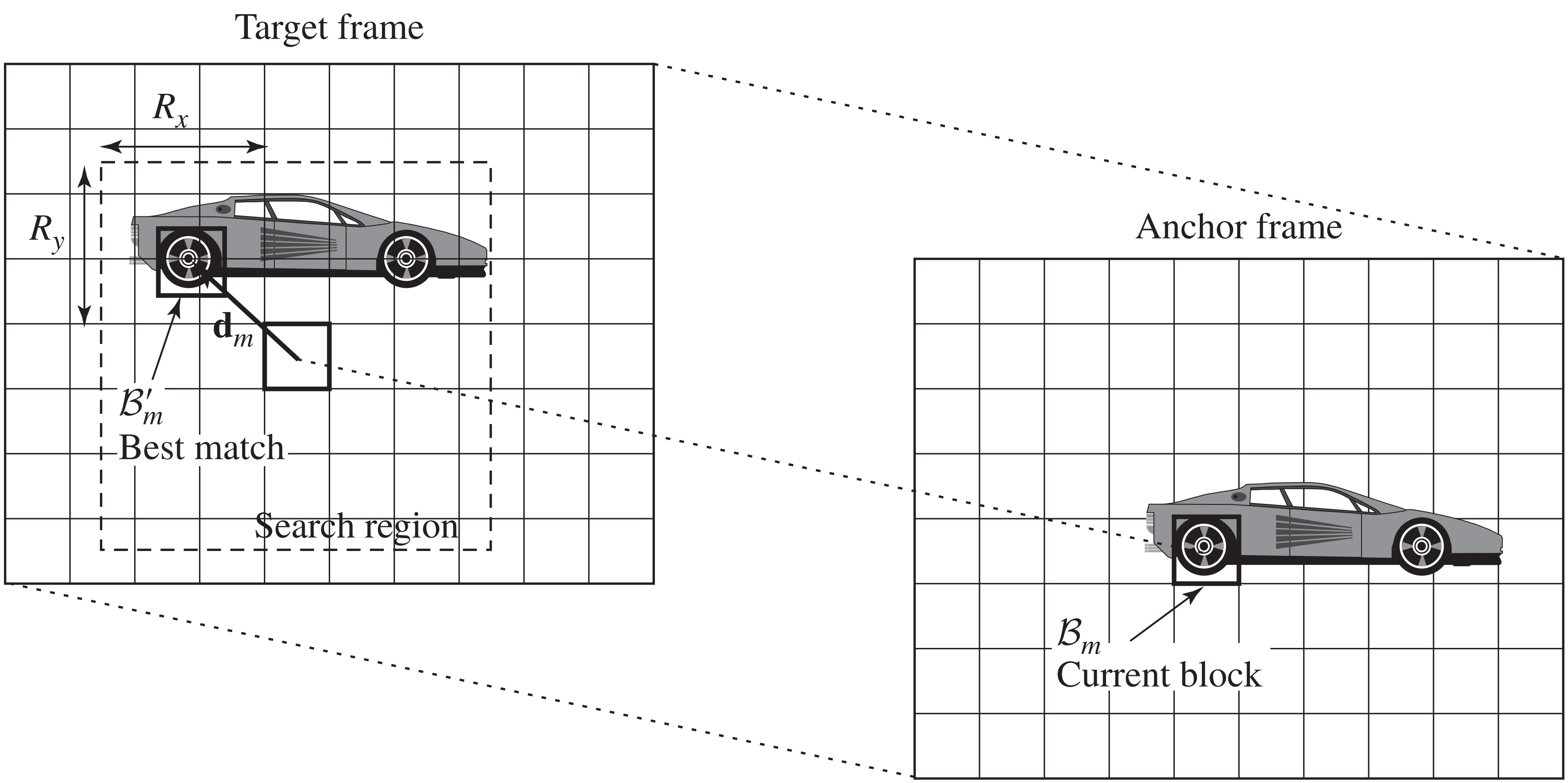
- returns a TFORM structure containing a spatial transformation.
- INPUT_POINTS is an M-by-2 double matrix containing the X and Y coordinates of control points in the image you want to transform.
- BASE_POINTS is an M-by-2 double matrix containing the X and Y coordinates of control points in the base image.
- TRANSFORMTYPE can be 'nonreflective similarity', 'similarity', 'affine', 'projective', 'polynomial', 'piecewise linear' or 'lwm'.

https://www.mathworks.com/help/images/ref/cp2tform.html?s_tid=doc_ta

How to find corresponding points in two images?

- Which points to select in one image (image 1)?
  - Ideally choose “interesting points”: corners, special features
  - Can also use points over a regular grid
- How to find the corresponding point in the other image (image 2)?
  - Put a small block around the point in image 1
  - Find a block in image 2 that matches the block pattern the best
  - Exhaustive search within a certain range.

Exhaustive Block Matching AlgorithmAlgorithm (EBMA)



MATLAB Example

- Register an aerial photo to an orthophoto.

```
unregistered = imread('westconcordaerial.png');
figure, imshow(unregistered)
figure, imshow('westconcordorthophoto.png')
load westconcordpoints % load some points that were already picked
t_concord = cp2tform(input_points,base_points,'projective');
info = imfinfo('westconcordorthophoto.png');
registered = imtransform(unregistered,t_concord,...
                  'XData',[1 info.Width], 'YData',[1 info.Height]);
figure, imshow(registered)
```

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.

Image Morphing

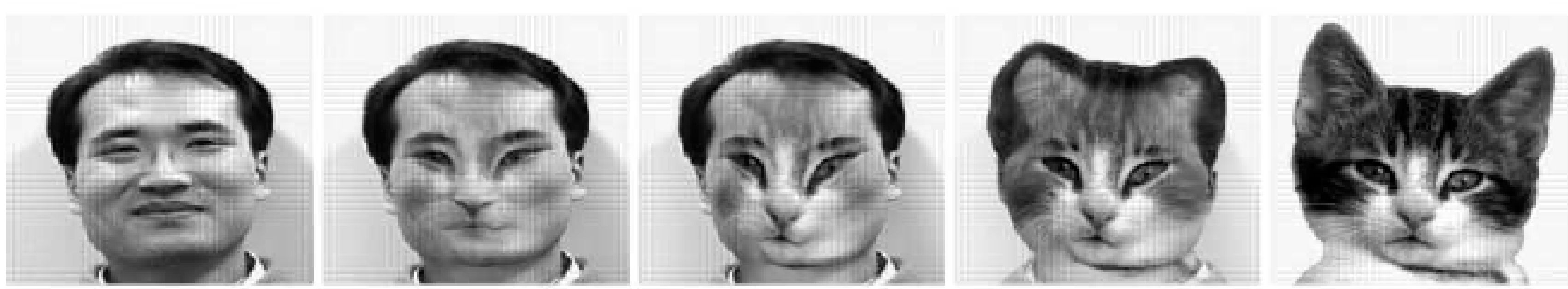
The procedure of a PhD student

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

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

MATLAB function for selecting control points

- CPSELECT(INPUT,BASE) returns control points in CPSTRUCT. INPUT is the image that needs to be warped to bring it into the coordinate system of the BASE image.
- Example
- cpselect('westconcordaerial.png','westconcordorthophoto.png')

https://www.mathworks.com/help/images/ref/cpselect.html