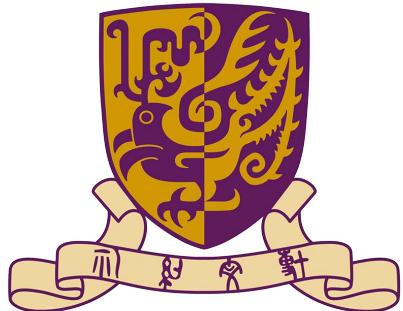


EIE4512 - Digital Image Processing

Geometric Operations, Image Warpping and Image Registration



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

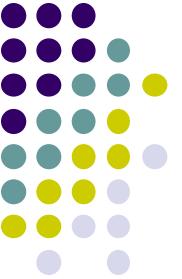
Zhen Li

lizhen@cuhk.edu.cn

School of Science and Engineering

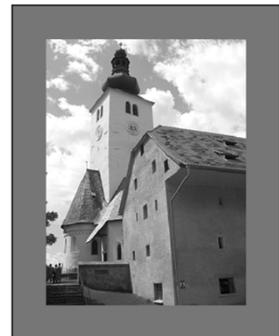
The Chinese University of Hong Kong, Shen Zhen

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



(a)



(b)



(c)



(d)



(e)



(f)

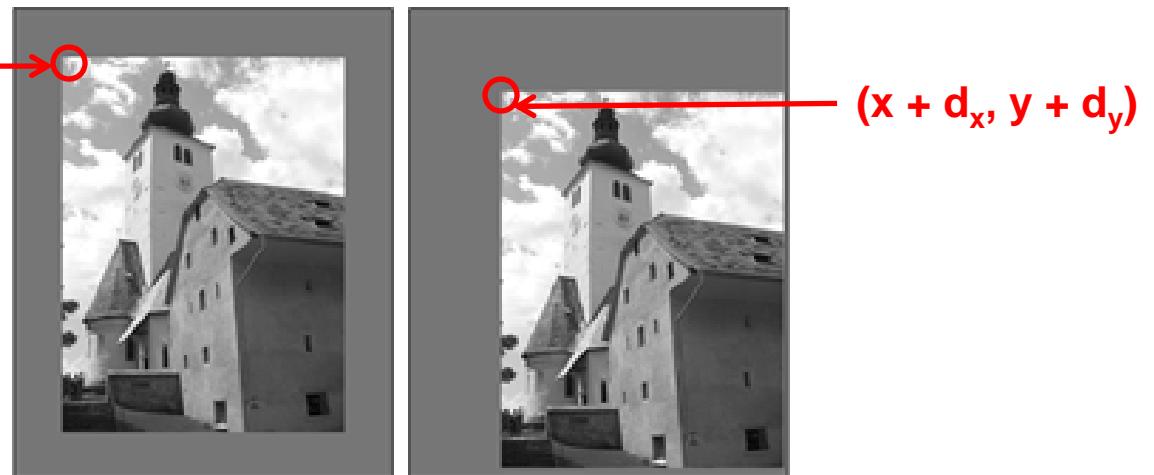
Examples of
Geometric
operations



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

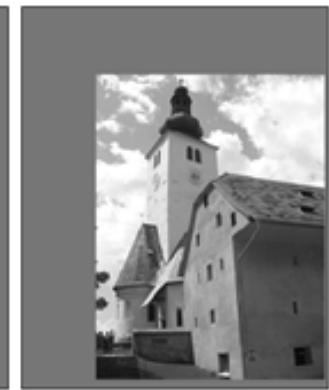
- Since image coordinates can only be discrete values, some transformations may yield (x',y') that's not discrete
- **Solution:** interpolate nearby values



Simple Mappings

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

$$\begin{aligned} T_x : x' &= x + d_x \\ T_y : y' &= y + d_y \end{aligned} \quad \text{or} \quad \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} d_x \\ d_y \end{pmatrix}$$



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

s_x or s_y

$$\begin{aligned} T_x : x' &= s_x \cdot x \\ T_y : y' &= s_y \cdot y \end{aligned} \quad \text{or} \quad \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

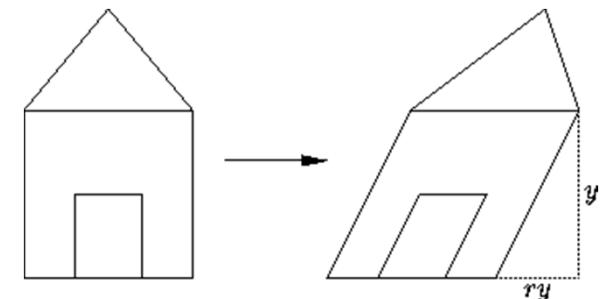




Simple Mappings

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

$$T_x : x' = x + b_x \cdot y \quad \text{or} \quad \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & b_x \\ b_y & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$



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

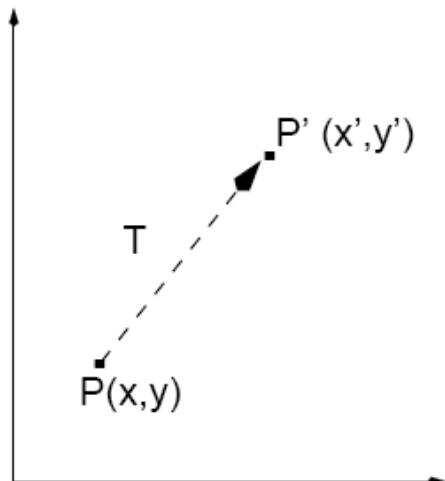
$$T_x : x' = x \cdot \cos \alpha - y \cdot \sin \alpha$$
$$T_y : y' = x \cdot \sin \alpha + y \cdot \cos \alpha$$

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



2D Translation

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



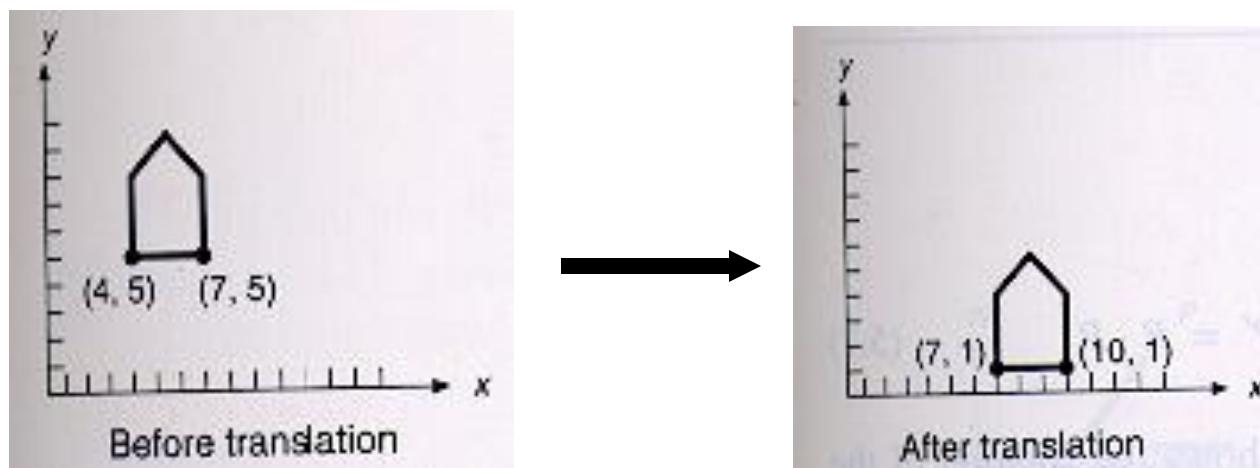
$$x' = x + dx, \quad y' = y + dy$$

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

or
$$P' = P + T$$

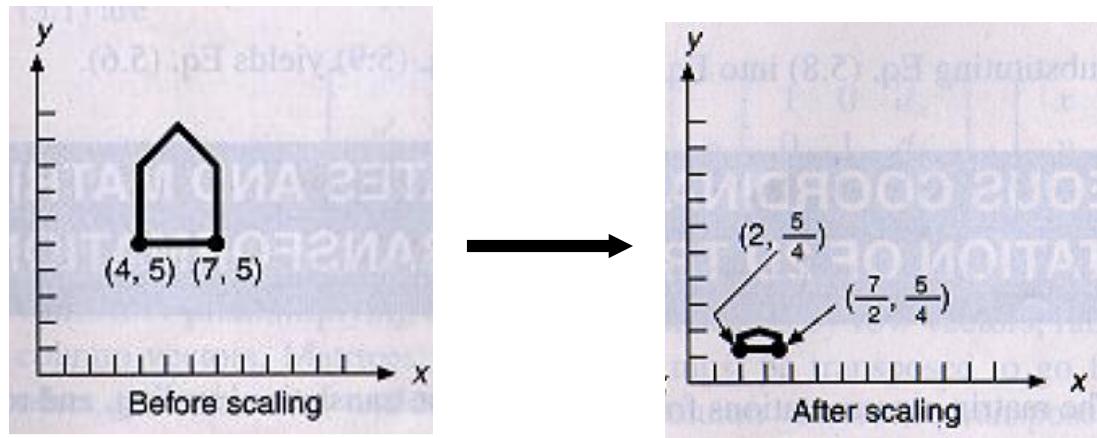
2D Translation (cont'd)

- 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.



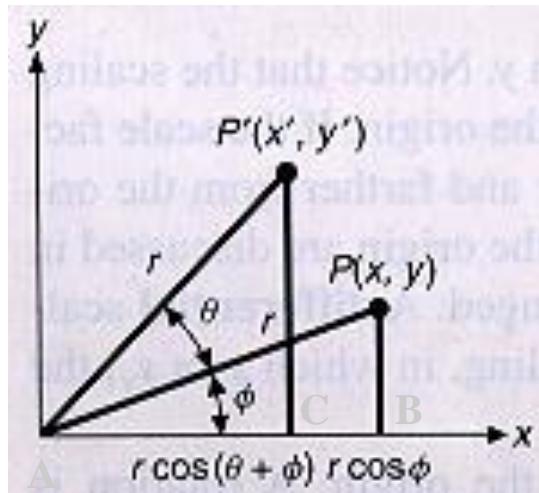
$$x' = x s_x, \quad y' = y s_y \quad \text{or} \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}, \quad s_x, s_y > 0 \quad \text{or} \quad \underline{P' = S P}$$

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
($\theta > 0$: 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:

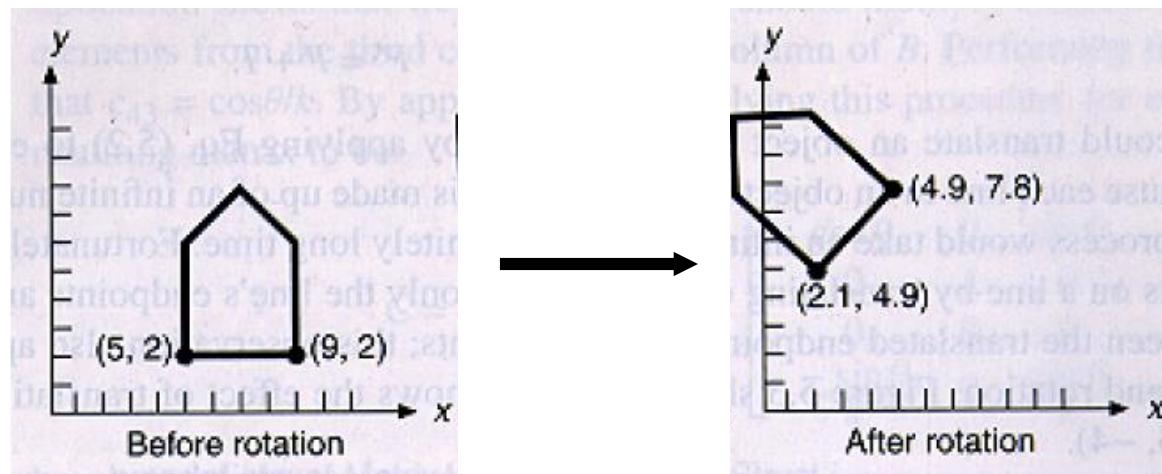
$$\cos(\phi + \theta) = x'/r \text{ or } x' = r\cos(\phi + \theta) = r\cos(\phi)\cos(\theta) - r\sin(\phi)\sin(\theta)$$
$$\sin(\phi + \theta) = y'/r \text{ or } y' = r\sin(\phi + \theta) = r\cos(\phi)\sin(\theta) + r\sin(\phi)\cos(\theta)$$

2D Rotation (cont'd)

- From the above equations we have:

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

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



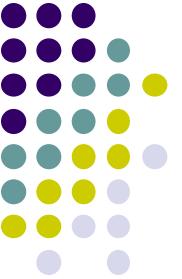
Summary of 2D transformations

Translation: $P' = P + T$

Scale: $P' = S \cdot P$

Rotation: $P' = R \cdot P$

- 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

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

Homogeneous coordinates

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

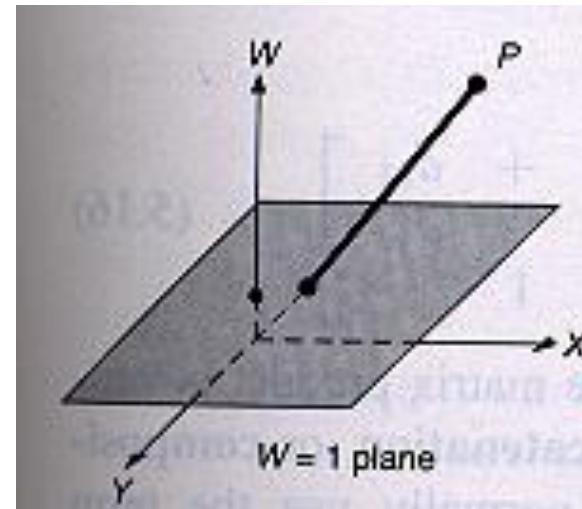
$$x = \frac{x_h}{w}, \quad y = \frac{y_h}{w}, \quad w \neq 0$$

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

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

Homogeneous coordinates (cont'd)

- (x, y) has multiple representations in homogeneous coordinates:
 - $w=1$ $(x,y) \rightarrow (x,y,1)$ $(w \neq 0)$
 - $w=2$ $(x,y) \rightarrow (2x,2y,2)$
- All these points lie on a line in the space of homogeneous coordinates !!



projective
space

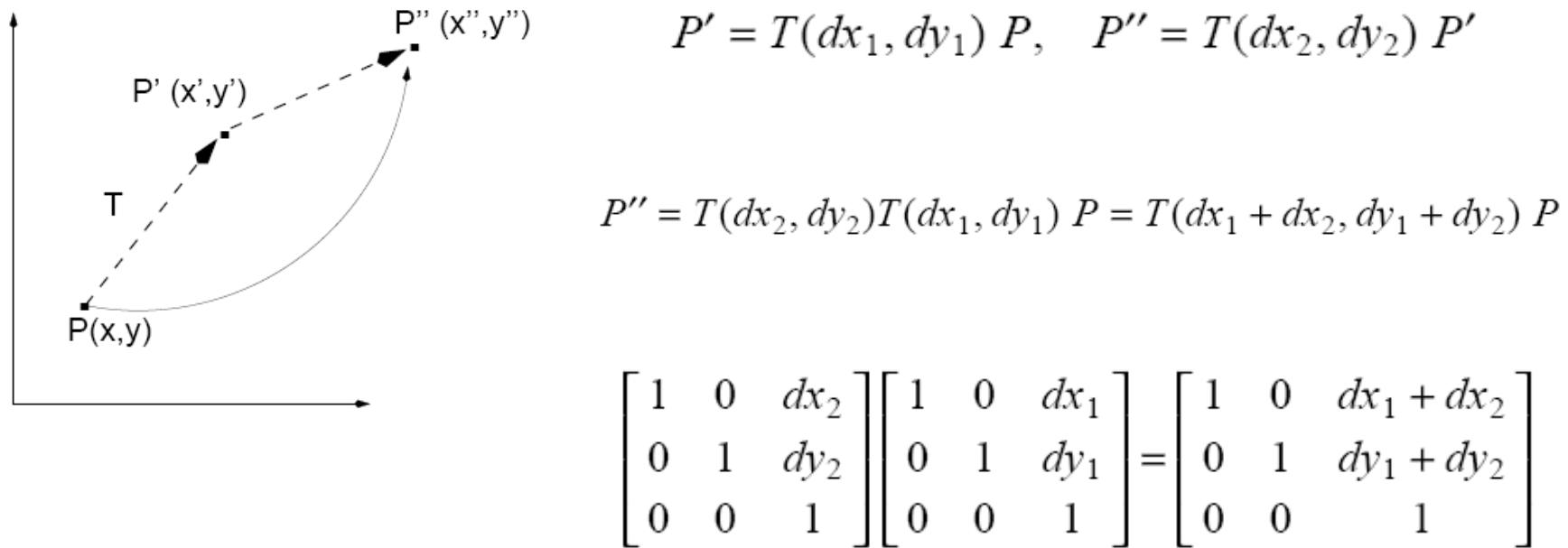
2D Translation using homogeneous coordinates

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

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

2D Translation using homogeneous coordinates (cont'd)

- Successive translations:



2D Scaling using homogeneous coordinates

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

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

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

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

$$w=1$$

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

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

2D Rotation using homogeneous coordinates (cont'd)

- Successive rotations:

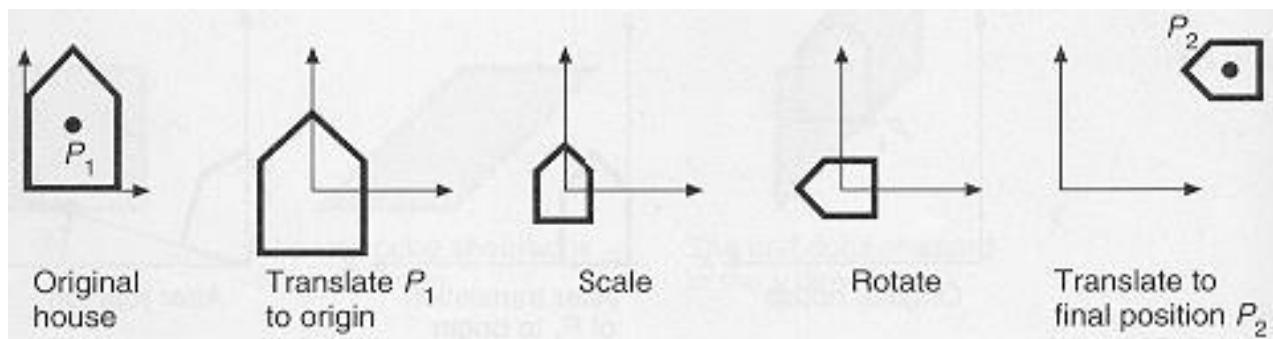
$$P' = R(\theta_1) P, \quad P'' = R(\theta_2) P'$$

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

Example:

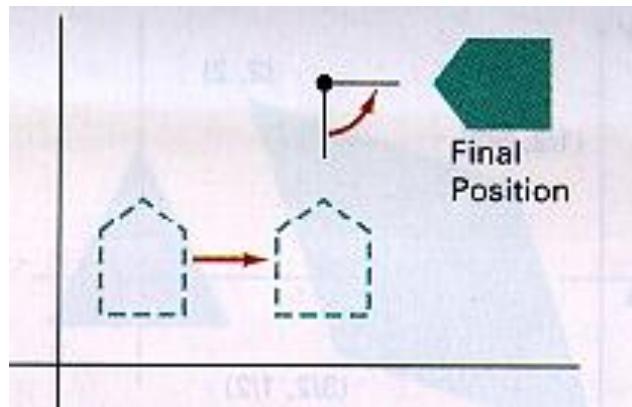


$$M = T(x_2, y_2)R(\theta)S(s_x, s_y)T(-x_1, -y_1)$$

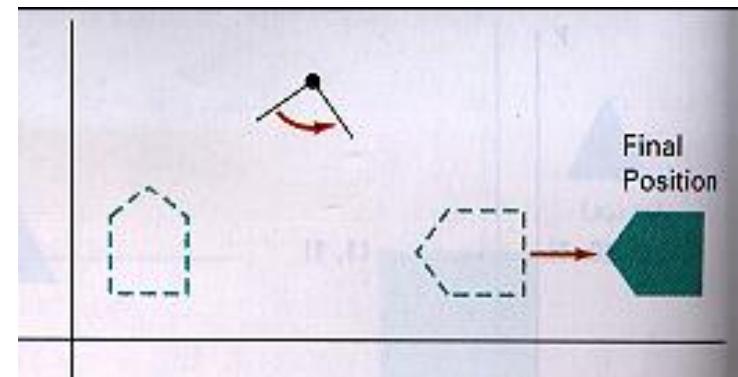
Composition of transformations (cont'd)

- Important: preserve the order of transformations!

translation + rotation



rotation + translation



General form of transformation matrix

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

rotation, scale translation

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

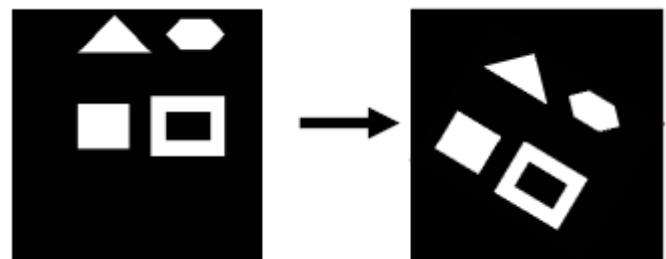
$$x' = a_{11}x + a_{12}y + a_{13}$$

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

(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 2×2 submatrix is orthonormal

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

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

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

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

Example: rotation matrix

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

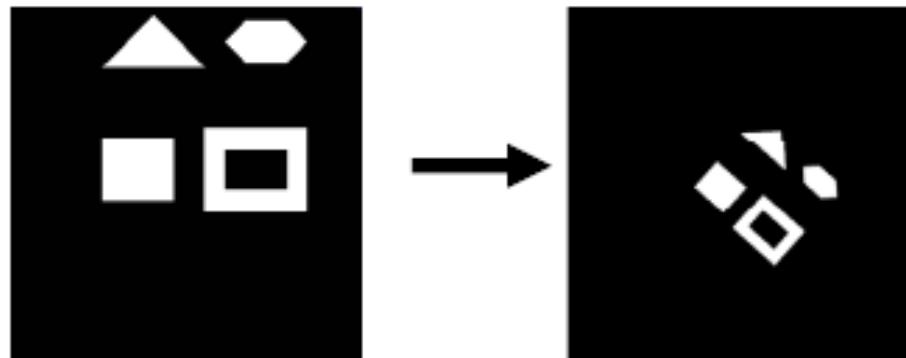
$$u_1 \cdot u_1 = \cos(\theta)^2 + \sin(-\theta)^2 = 1$$

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

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

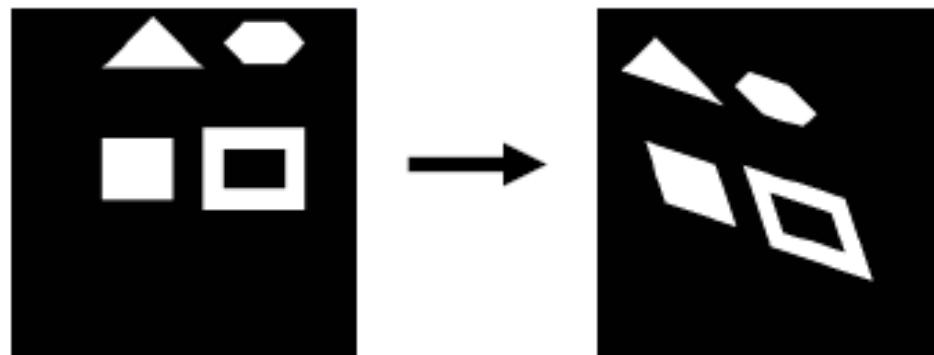
Special cases of transformations

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



Affine transformations

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

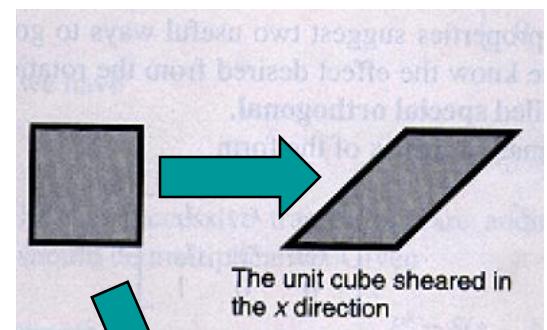


2D shear transformation

- Shearing along x-axis:

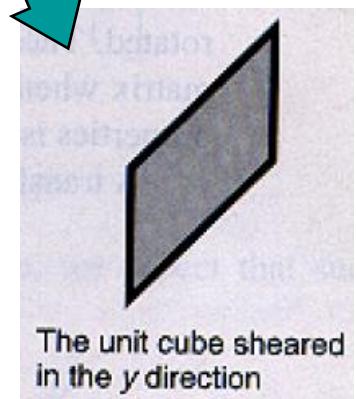
$$x' = x + ay, y' = y \quad SH_x = \begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

changes object shape!



- Shearing along y-axis

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



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

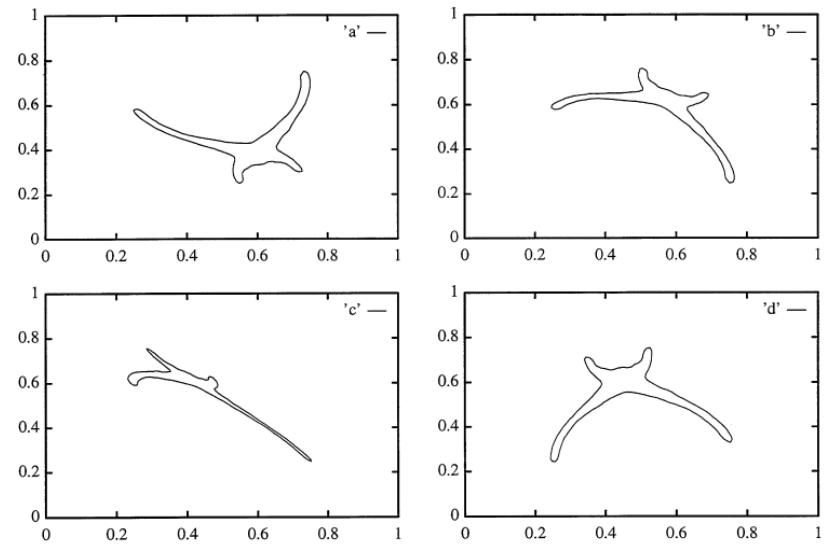
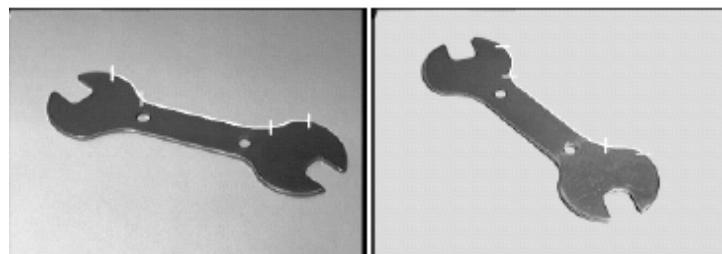
$$\mathbf{A} = \begin{bmatrix} 1 & 0.5 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

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

Affine Transformations

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

affine transformed object



G. Bebis, M. Georgopoulos, 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)

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



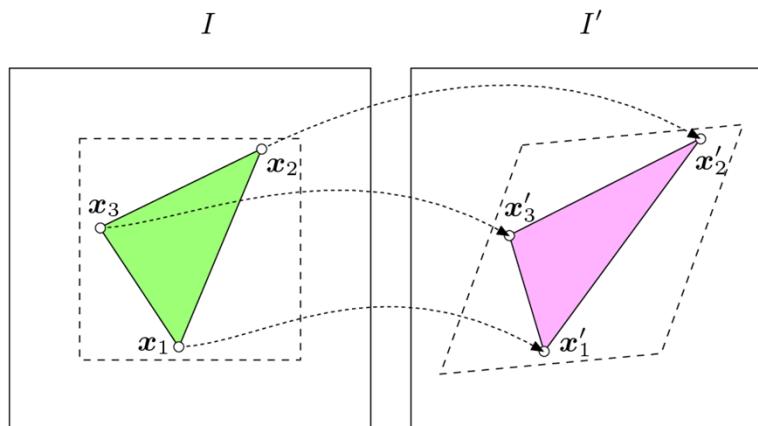


Affine (3-Point) Mapping

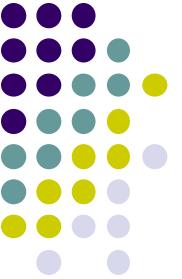
- Can use homogeneous coordinates to rewrite translation, rotation, scaling, etc as vector-matrix multiplication

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

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

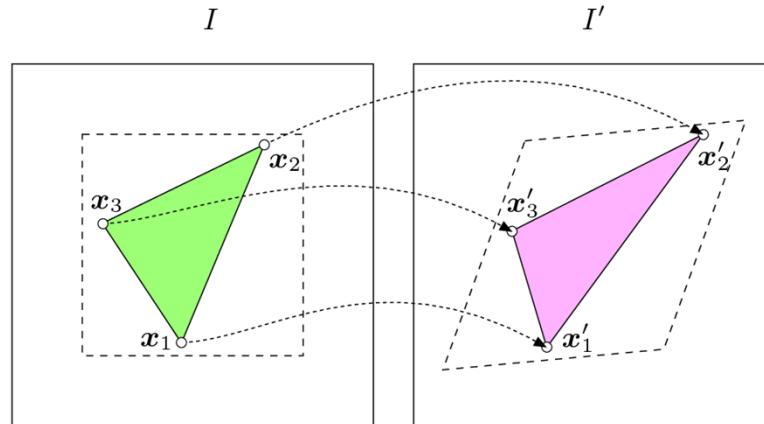


- Inverse of transform matrix is **inverse mapping**



Affine (3-Point) Mapping

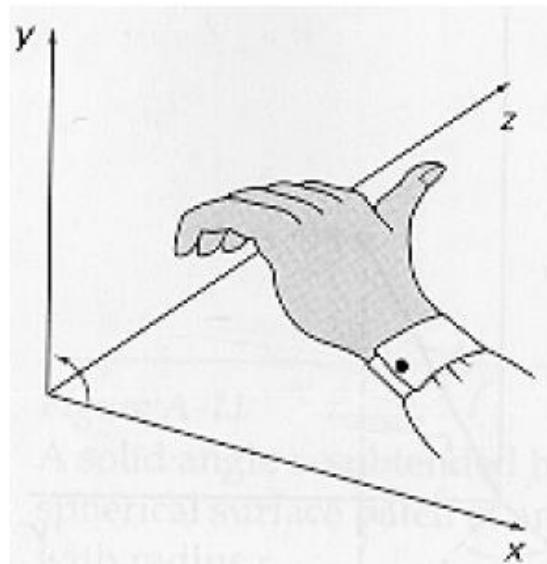
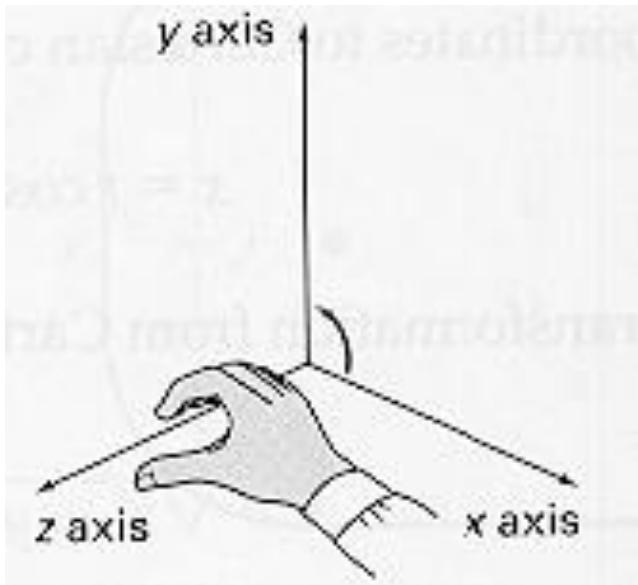
- What's so special about affine mapping?



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

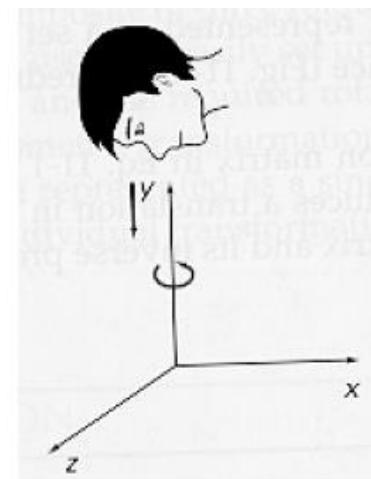
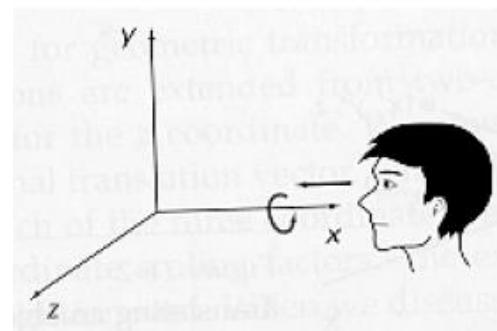
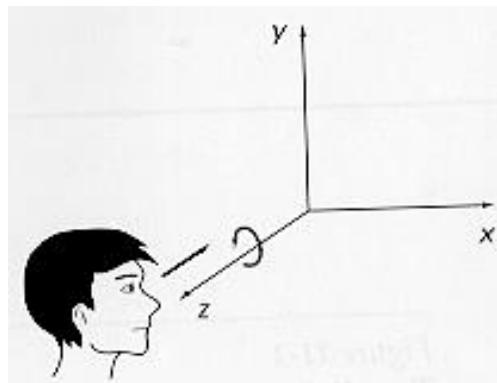
3D Transformations

- 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) :

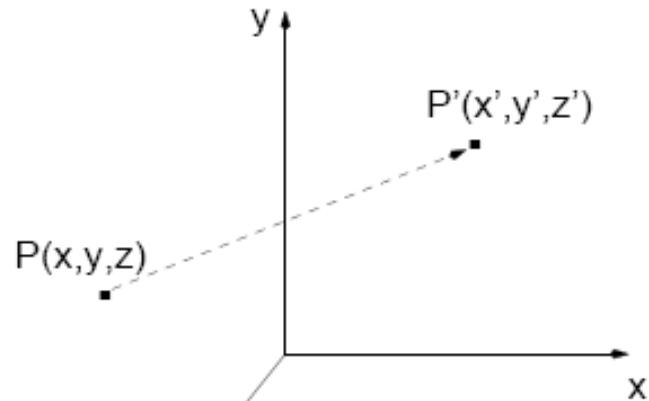
$$x = \frac{x_h}{w}, \quad y = \frac{y_h}{w}, \quad z = \frac{z_h}{w}, \quad w \neq 0$$

- 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.

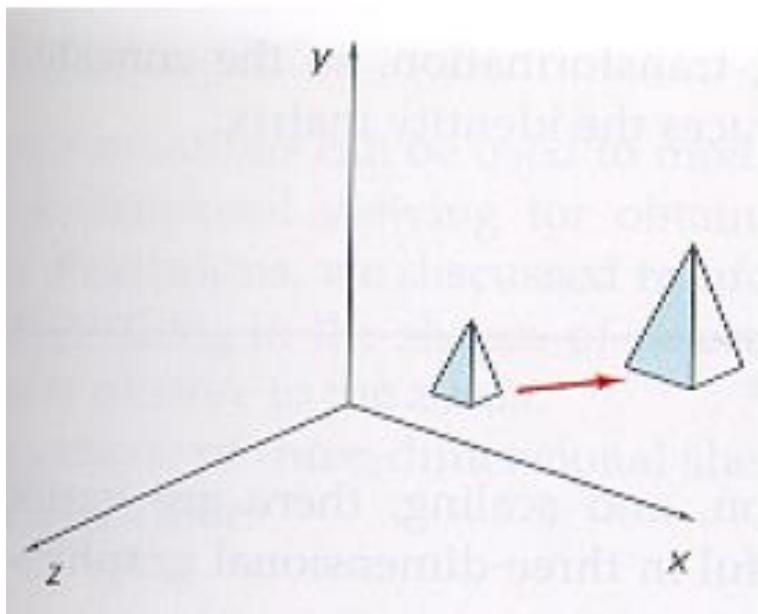
3D Translation



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

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

3D Scaling

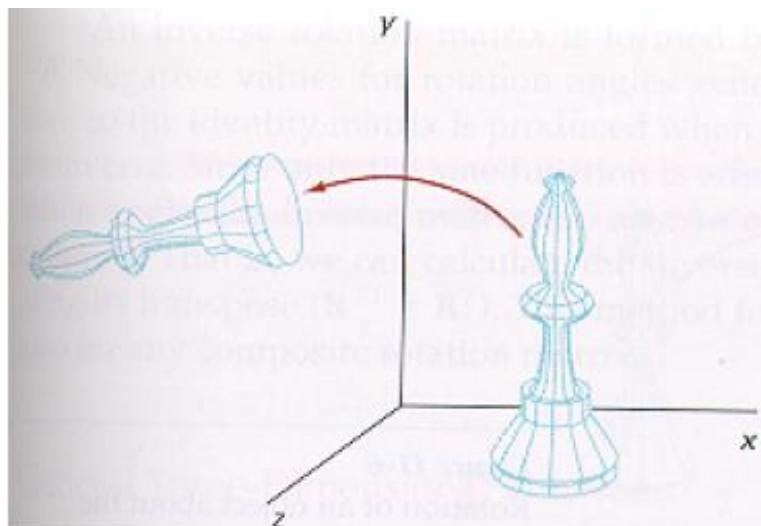


$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

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

3D Rotation

- Rotation about the z -axis:



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

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

$$z' = z$$

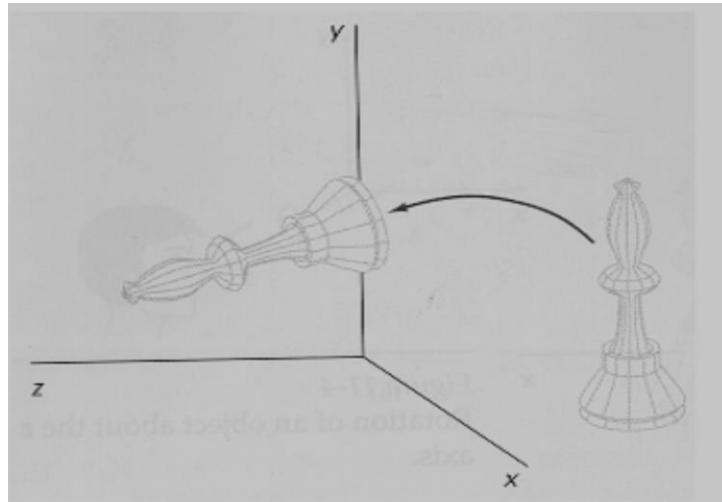
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

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

3D Rotation (cont'd)

- Rotation about the x -axis:

$$x' = x$$



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

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

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

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

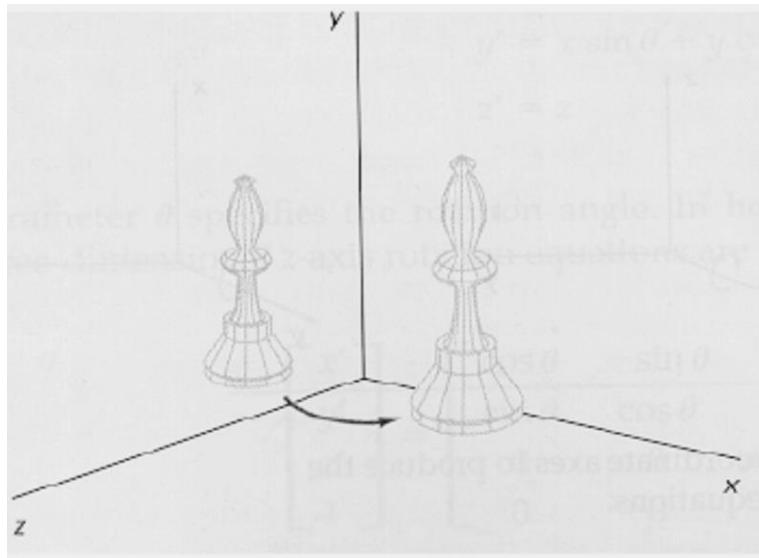
3D Rotation (cont'd)

- Rotation about the y-axis

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

$$y' = y$$

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



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

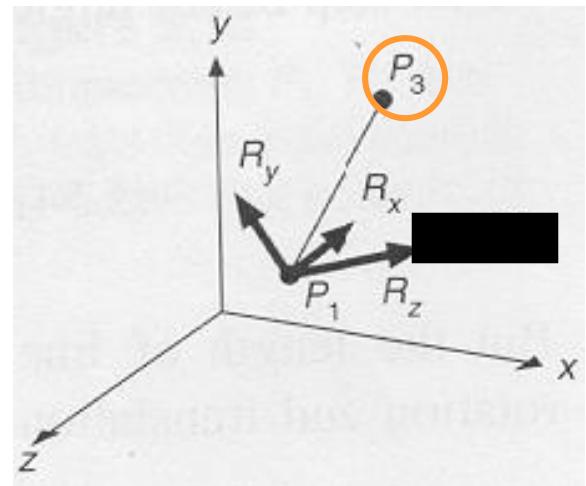
$$\underline{P' = R_y(\theta) P}$$

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?

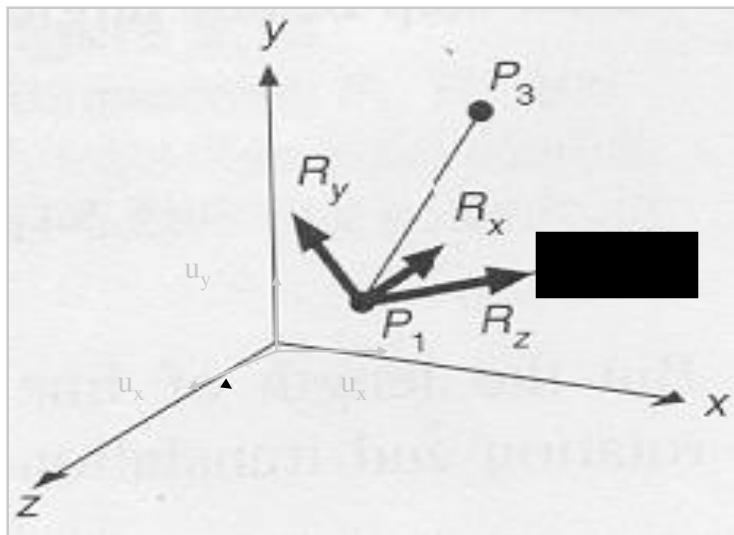
(1) Recover the translation T and rotation R from $R_xR_yR_z$ to xyz .
that aligns $R_xR_yR_z$ with xyz

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



(1.1) Recover translation T

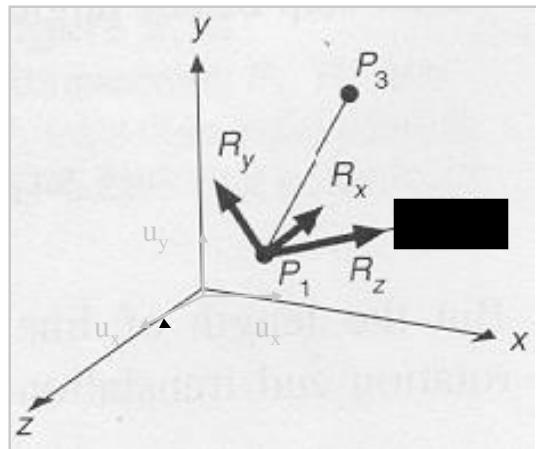
- If we know the coordinates of P_1 (i.e., origin of $R_xR_yR_z$) in the xyz coordinate system, then T is:



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

(1.2) Recover rotation R

- u_x, u_y, u_z are unit vectors in the xyz coordinate system.
- r_x, r_y, r_z are unit vectors 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

$$\left[\begin{array}{cccc} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ 0 & 0 & 0 & 1 \end{array} \right]$$

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

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

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

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

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:

$$r_x \rightarrow u_x$$

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

$$r_y \rightarrow u_y$$

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

$$r_z \rightarrow u_z$$

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

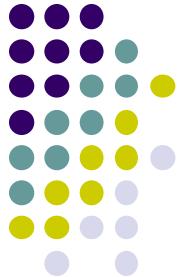
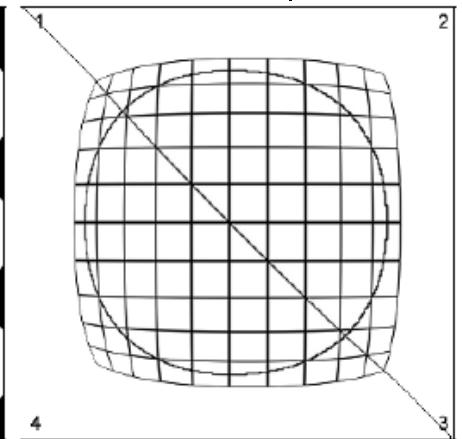
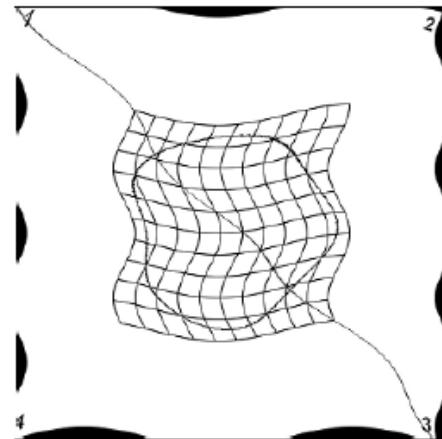
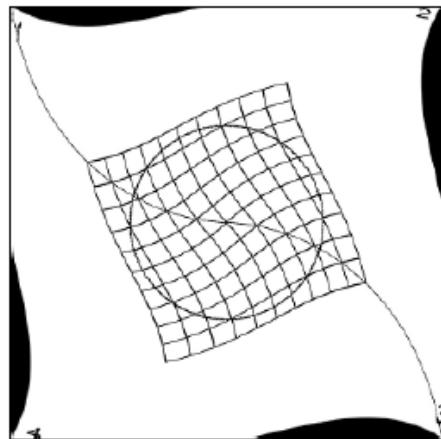
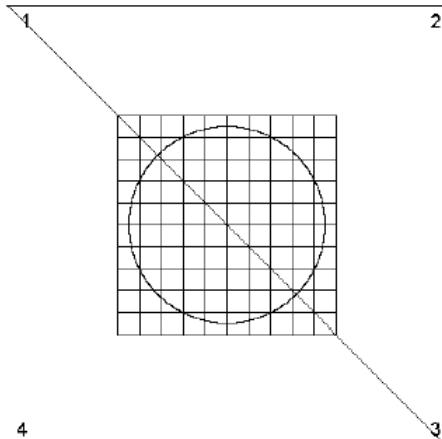


Image Warping





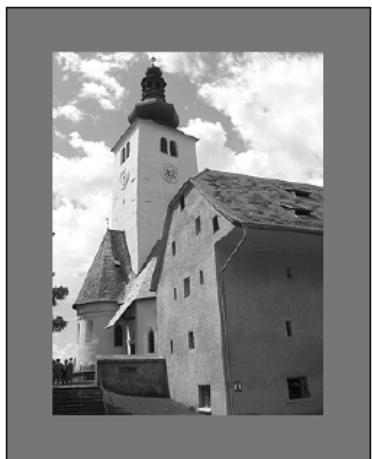
Non-Linear Image Warps



(a)

(b)

(c)



Original

Twirl

Ripple

Spherical

Twirl

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

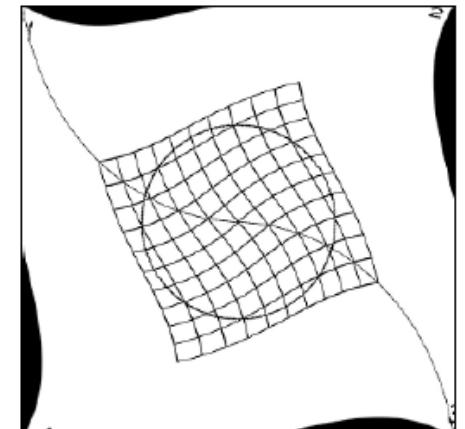
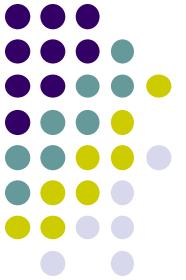
$$T_x^{-1} : \quad x = \begin{cases} x_c + r \cdot \cos(\beta) & \text{for } r \leq r_{max} \\ x' & \text{for } r > r_{max}, \end{cases}$$

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

with

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

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



(a)



(d)

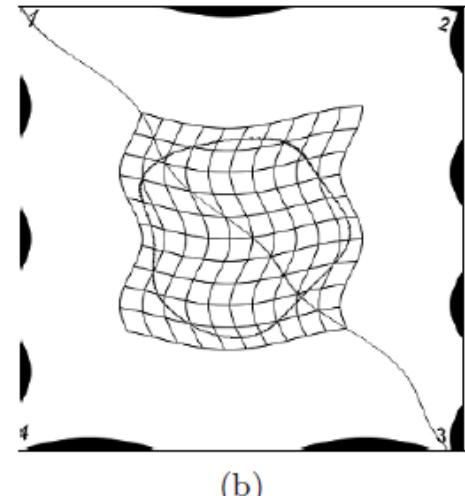
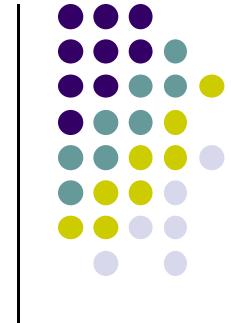
Ripple

- 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
 - $\tau_x = 120$
 - $\tau_y = 250$
 - $a_x = 10$
 - $a_y = 15$

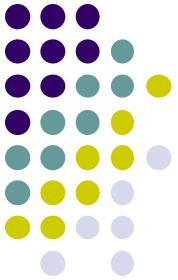


(b)



(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 $\rho = 1.8$ and $r_{max} = \text{half image width}$

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

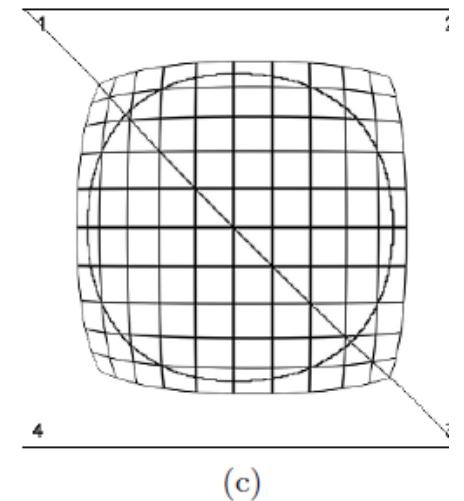
$$T_y^{-1} : y = y' - \begin{cases} z \cdot \tan(\beta_y) & \text{for } r \leq r_{max} \\ 0 & \text{for } r > r_{max}, \end{cases}$$

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

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

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

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



Polynomial Warping

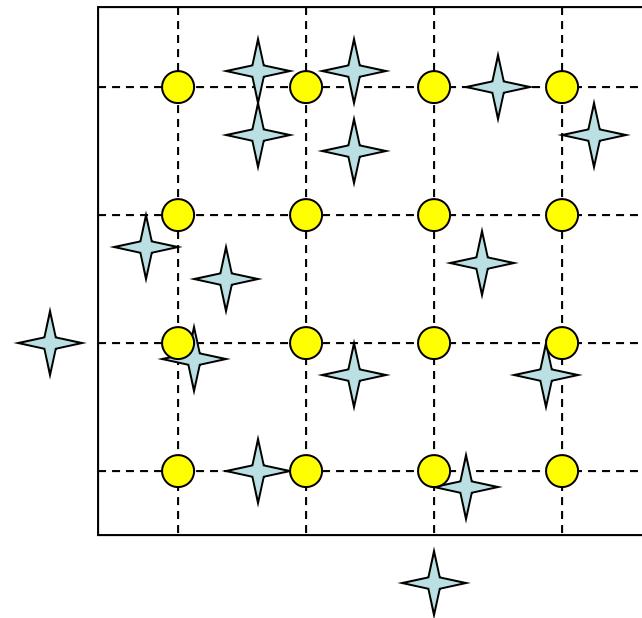
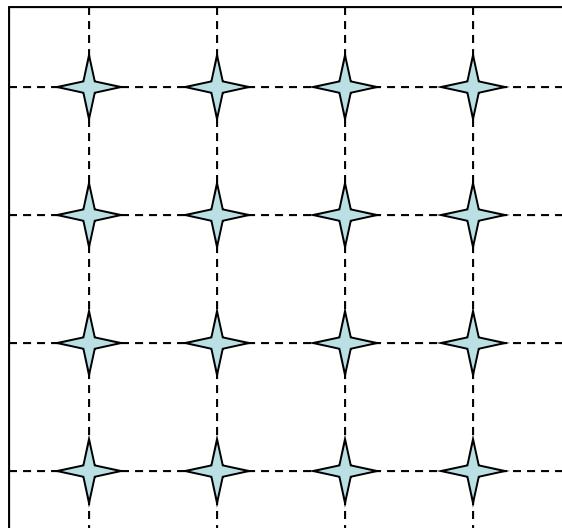
- The **polynomial warping** includes all deformations that can be modeled by **polynomial transformations**:

$$\begin{cases} x = a_0 + a_1u + a_2v + a_3uv + a_4u^2 + a_5v^2 + \dots \\ y = b_0 + b_1u + b_2v + b_3uv + b_4u^2 + b_5v^2 + \dots \end{cases}$$

- 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?

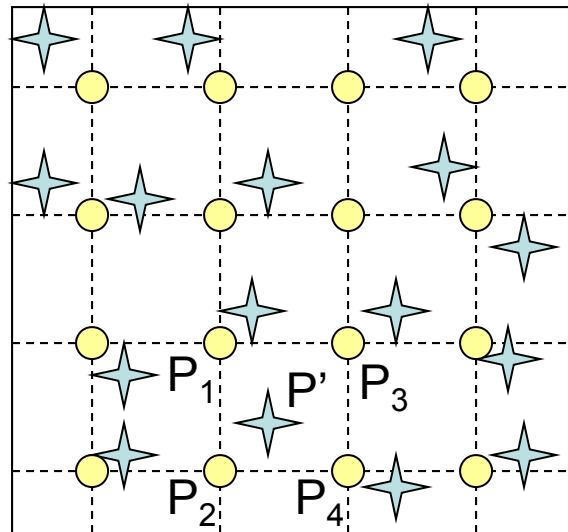


Warping points
are often non-
integer samples

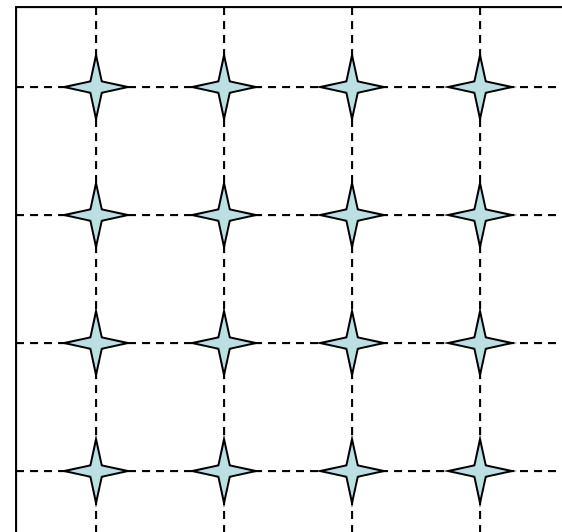
Many integer
samples “o”
are not assigned
Values

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

- $ZI = \text{INTERP2}(X, Y, Z, XI, YI, \text{METHOD})$ 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)
 - $(X,Y) = \text{meshgrid}(1:M, 1:N)$
 - 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)
 - $(U,V) = \text{meshgrid}(1:M, 1:N)$
 - `Outimg = interp2(U,V,inimg,UI,VI,'linear');`

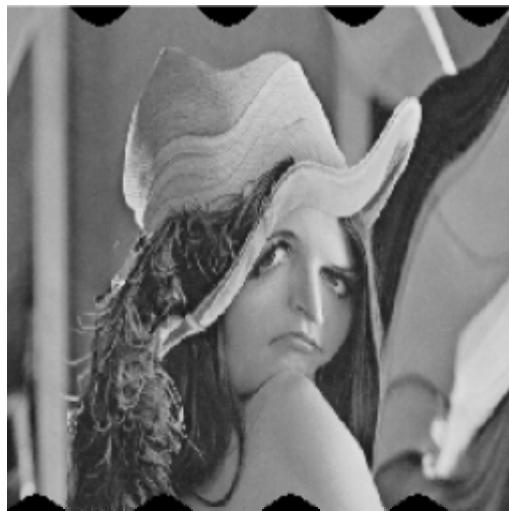
MATLAB function for image warping

- $B = \text{IMTRANSFORM}(A, TFORM, \text{INTERP})$ 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.
 -
 - $I = \text{imread}('cameraman.tif');$
 - $tform = \text{maketform}('affine',[1 0 0; .5 1 0; 0 0 1]);$
 - $J = \text{imtransform}(I, tform);$
 - $\text{figure, imshow}(I), \text{figure, imshow}(J)$

[https://www.mathworks.com/help/images/ref/imtransform.html?
searchHighlight=IMTRANSFORM&s tid=doc srcTitle](https://www.mathworks.com/help/images/ref/imtransform.html?searchHighlight=IMTRANSFORM&s_tid=doc_srcTitle)

Example of Image Warping (1)

WAVE1



WAVE2



```
wave1:x(u,v)=u+20sin(2πv/128);y(u,v)=v;  
wave2:x(u,v)=u+20sin(2πu/30);y(u,v)=v.
```

By Onur Guleyuz

Example of Image Warping (2)

WARP



SWIRL



WARP

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

SWIRL

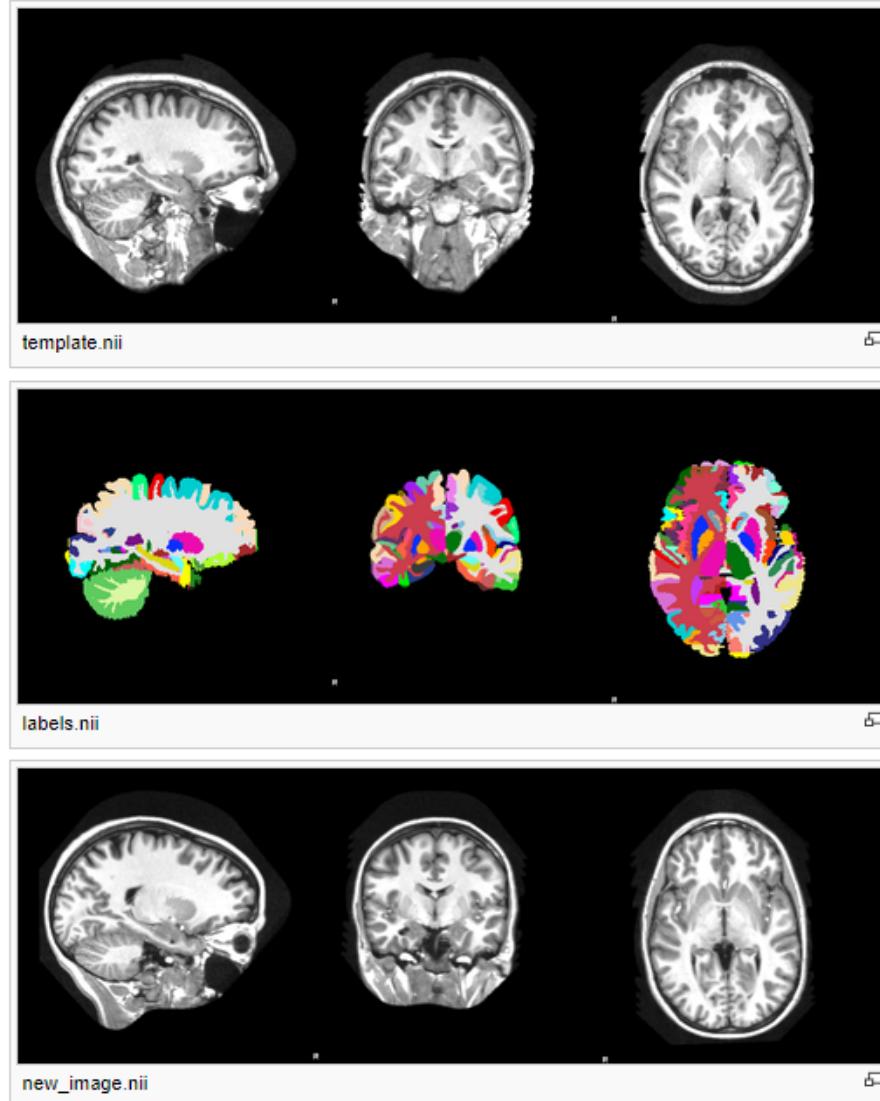
$$\begin{aligned} x(u, v) &= (u - x_0) \cos(\theta) + (v - y_0) \sin(\theta) + x_0; \\ y(u, v) &= -(u - x_0) \sin(\theta) + (v - y_0) \cos(\theta) + y_0; \\ r &= ((u - x_0)^2 + (v - y_0)^2)^{1/2}, \theta = \pi r / 512. \end{aligned}$$

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



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).

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

How to find the mapping function?

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

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

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

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

- 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 \mathbf{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

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

If $K < N$, or \mathbf{A} is singular, then more corresponding feature points must be identified.

Examples

- If we want to use an affine mapping to register two 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)

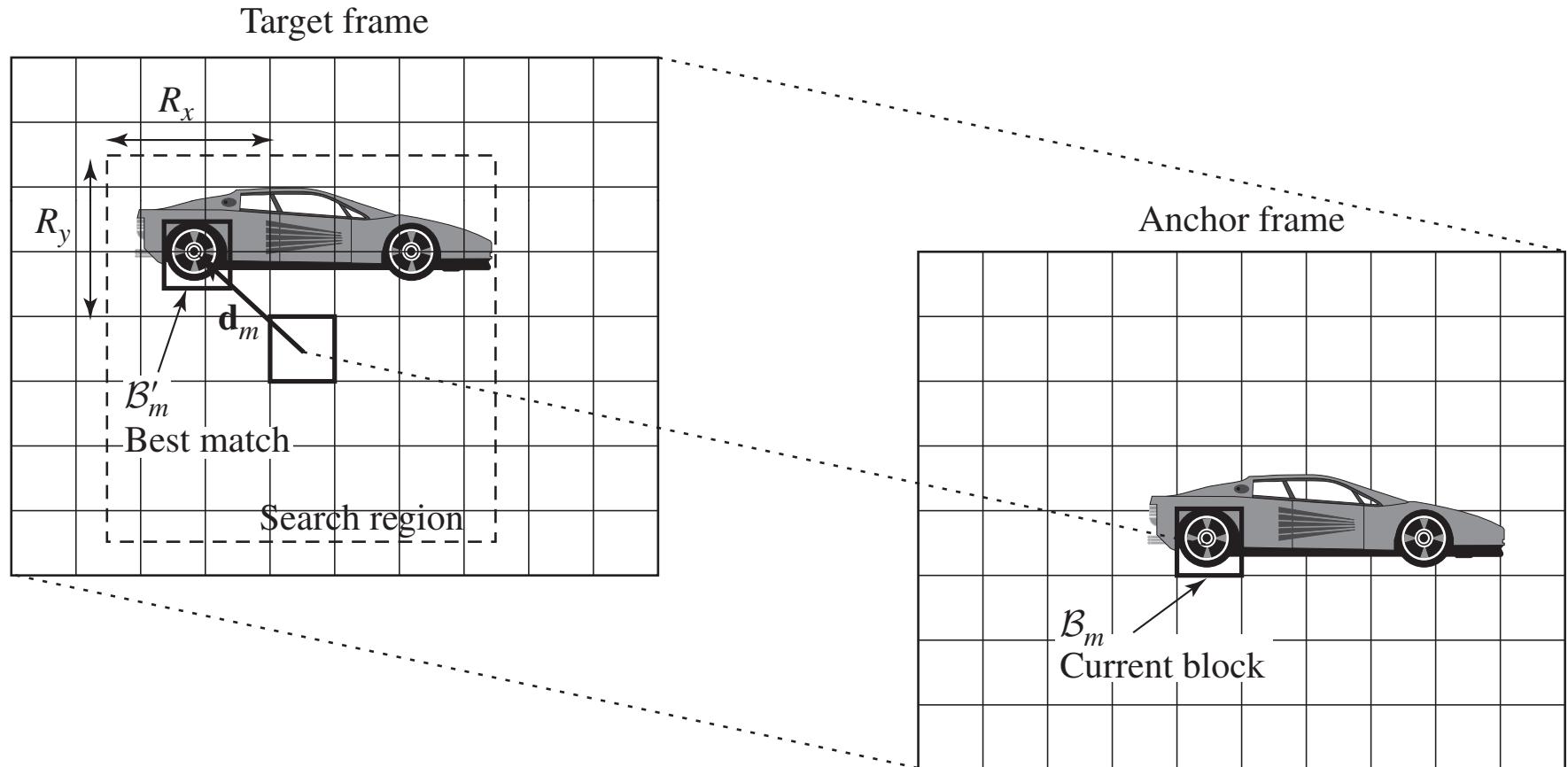
- returns a TFORM structure containing a spatial transformation.
- INPUT_POINTS is an M-by-2 double matrix containing the X and Y coordinates of control points in the image you want to transform.
- BASE_POINTS is an M-by-2 double matrix containing the X and Y coordinates of control points in the base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

https://www.mathworks.com/help/images/ref/cp2tform.html?secid=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 Algorithm (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

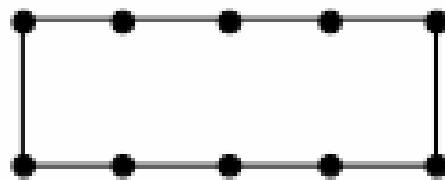
Morphing: Deformations in 2D and 3D

- Parametric Deformations
- Cross-Dissolve
- Mesh Warping
- Control Points

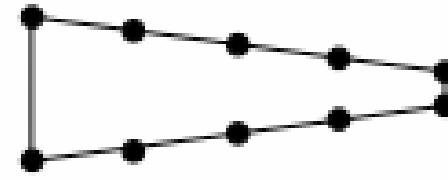
Human conducting morphing: <https://www.youtube.com/watch?v=sNjE2EVYSq0>

Parametric Deformations

Parametric Deformations - Taper



a) original object



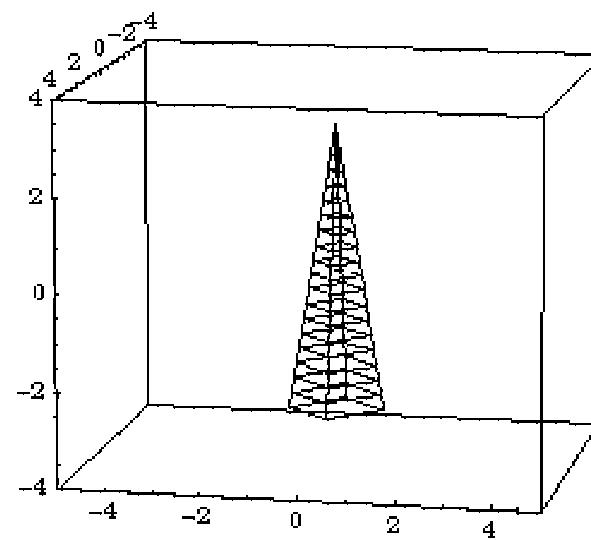
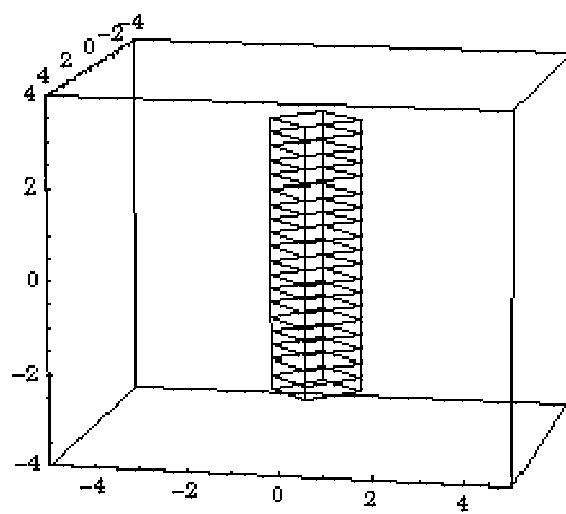
b) tapered object

$$\begin{aligned}x' &= x \\y' &= f(x)\end{aligned}$$

$$\begin{bmatrix}x' \\ y'\end{bmatrix} = \begin{bmatrix}1 & 0 \\ 0 & f(x)\end{bmatrix} \cdot \begin{bmatrix}x \\ y\end{bmatrix}$$

$$P' = M(P) \cdot P$$

Parametric Deformations - Taper



Parametric Deformations - Twist

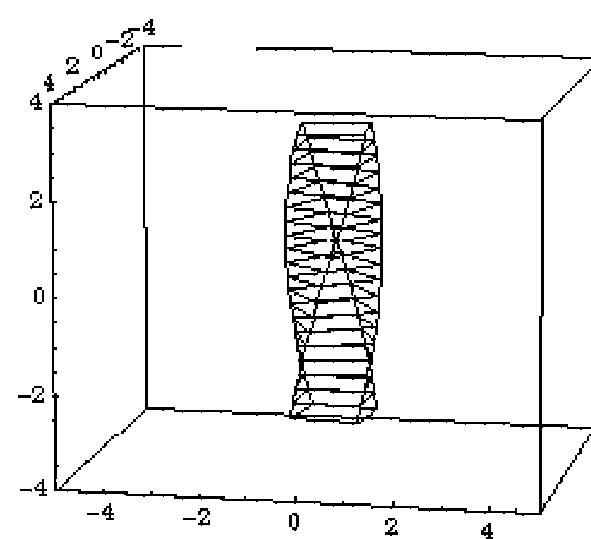
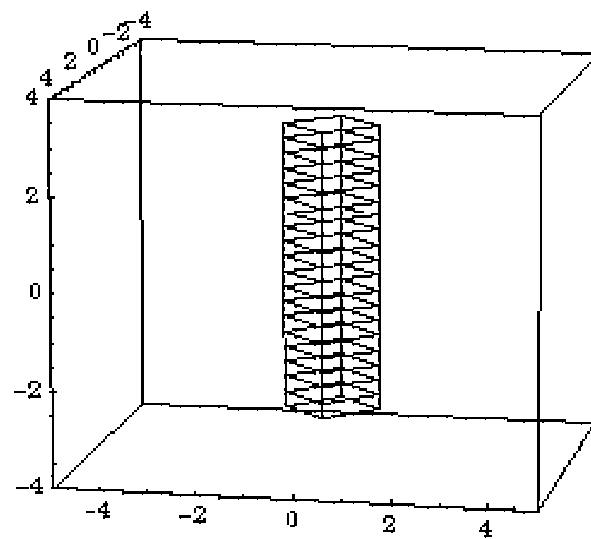
$$x' = s(z) \cdot x$$

$$y' = s(z) \cdot y$$

$$z' = z$$

$$\text{Where } s(z) = \frac{(maxz - z)}{(maxz - minz)}$$

Parametric Deformations - Twist



Parametric Deformations - Bend

y_0 - center of bend
 $1/k$ - radius of bend
 $y_{min}:y_{max}$ - bend region

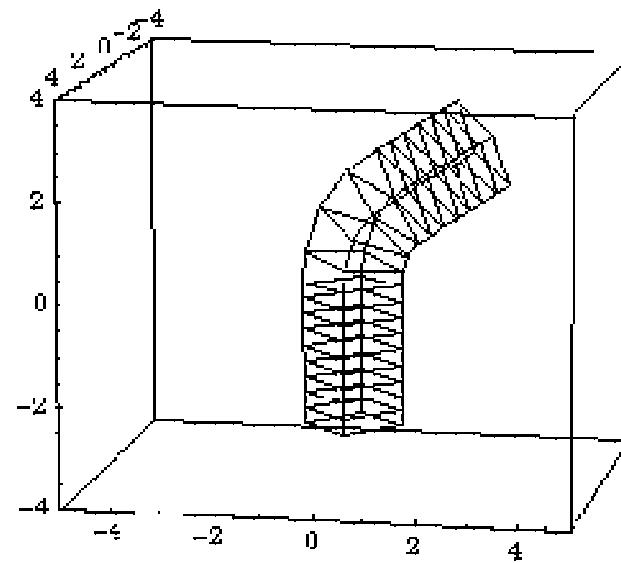
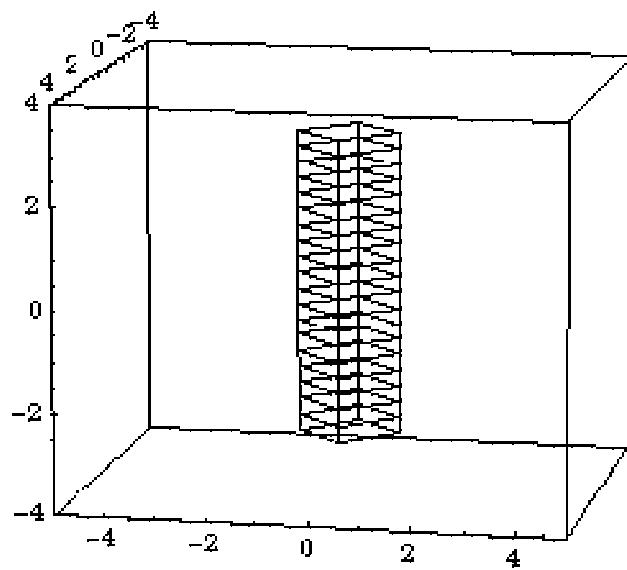
$$\begin{array}{lll}
 y_{min} & y \leq y_{min} & \theta = k \cdot (\hat{y} - y_0) \\
 \hat{y} = & y & C_\theta = \cos \theta \\
 y_{min} < y < y_{max} & & S_\theta = \sin \theta \\
 y_{max} & y \geq y_{max} &
 \end{array}$$

$$x' = x$$

$$y' = \begin{cases} -S_\theta \cdot z - \frac{1}{k} + y_0 & y_{min} \leq y \leq y_{max} \\ -\left(S_\theta \cdot \left(z - \frac{1}{k}\right)\right) + y_0 + C_\theta \cdot (y - y_{min}) & y < y_{min} \\ \left(-\left(S_\theta \cdot \left(z - \frac{1}{k}\right)\right) + y_0 + C_\theta \cdot (y - y_{max})\right) & y > y_{max} \end{cases}$$

$$z' = \begin{cases} -C_\theta \cdot z - \frac{1}{k} + \frac{1}{k} & y_{min} \leq y \leq y_{max} \\ -\left(C_\theta \cdot \left(z - \frac{1}{k}\right)\right) + \frac{1}{k} + S_\theta \cdot (y - y_{min}) & y < y_{min} \\ \left(-\left(C_\theta \cdot \left(z - \frac{1}{k}\right)\right) + \frac{1}{k} + S_\theta \cdot (y - y_{max})\right) & y > y_{max} \end{cases}$$

Parametric Deformations - Bend



Parametric Deformations - Compound

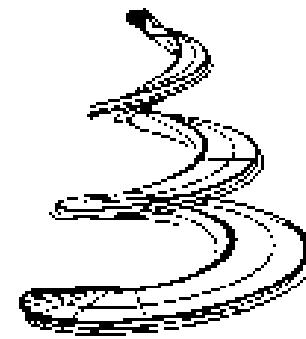
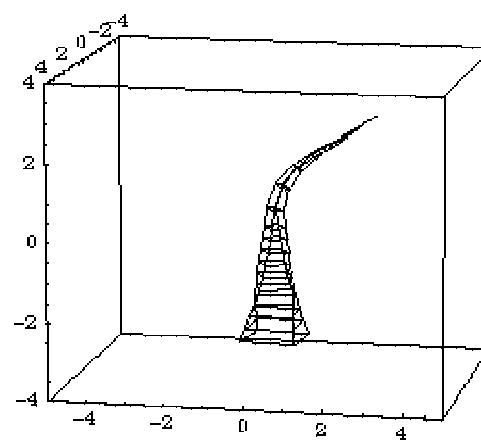
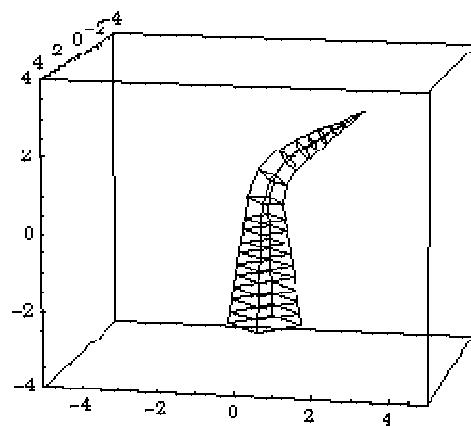


Image Blending

Image Blending

- Goal is smooth transformation between image of one object and another
- The idea is to get a sequence of intermediate images which when put together with the original images would represent the change from one image to the other
- Realized by
 - Image warping
 - Color blending
- Image blending has been widely used in creating movies, music videos and television commercials
 - Terminator 2

Cross-Dissolve (Cross-Fading)

- Simplest approach is **cross-dissolve**:
 - linear interpolation to fade from one image (or volume) to another
- No geometrical alignment between images (or volumes)
- Pixel-by-pixel (or voxel by voxel) interpolation
- No smooth transitions, intermediate states not realistic



from G. Wolberg, CGI '96

Problems

- Problem with cross-dissolve is that if features don't line up exactly, we get a double image
- Can try shifting/scaling/etc. one **entire** image to get better alignment, but this doesn't always fix problem
- Can handle more situations by applying different warps to different **pieces** of image
 - Manually chosen
 - Takes care of feature correspondences

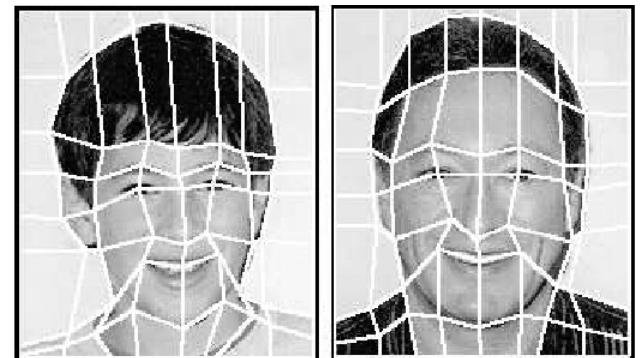


Image \mathbf{I}_S with mesh
 \mathbf{M}_S defining pieces

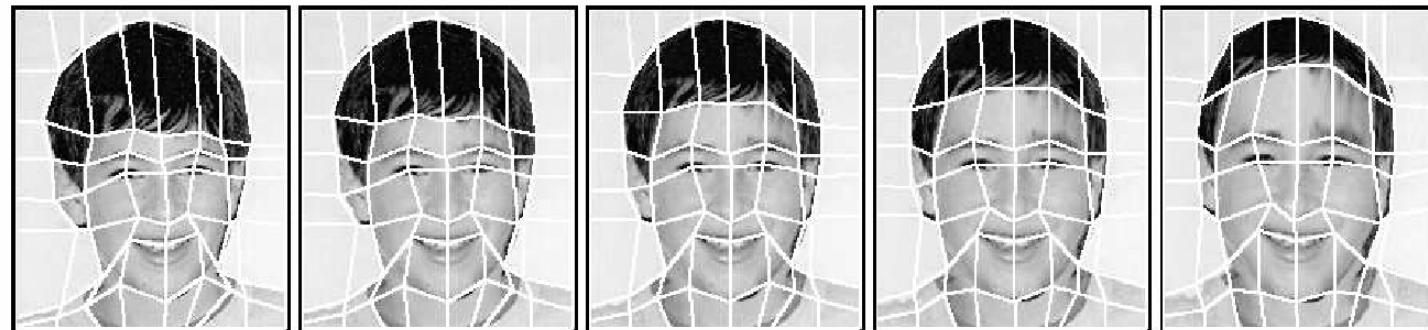
Image \mathbf{I}_T , mesh \mathbf{M}_T

from G. Wolberg, CGI '96

Mesh Warping

Mesh Warping Application

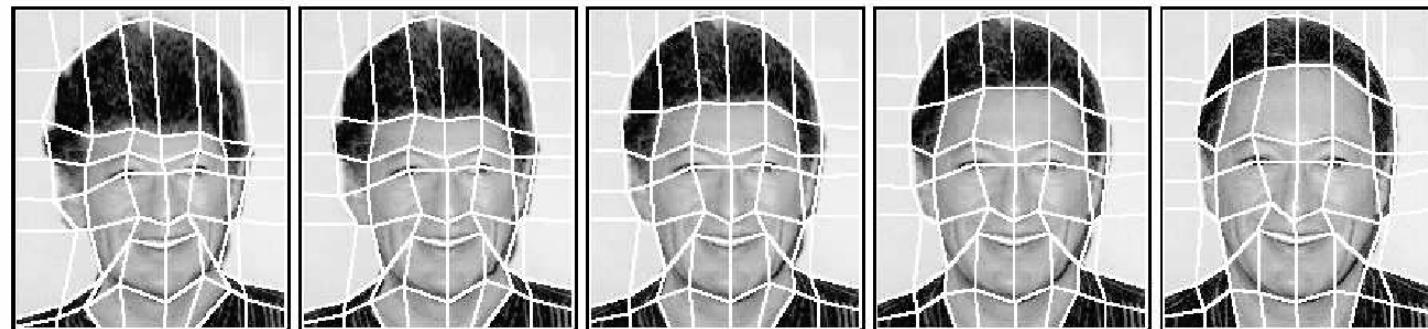
Images I_S
& meshes M_S



Morphed
images I^f

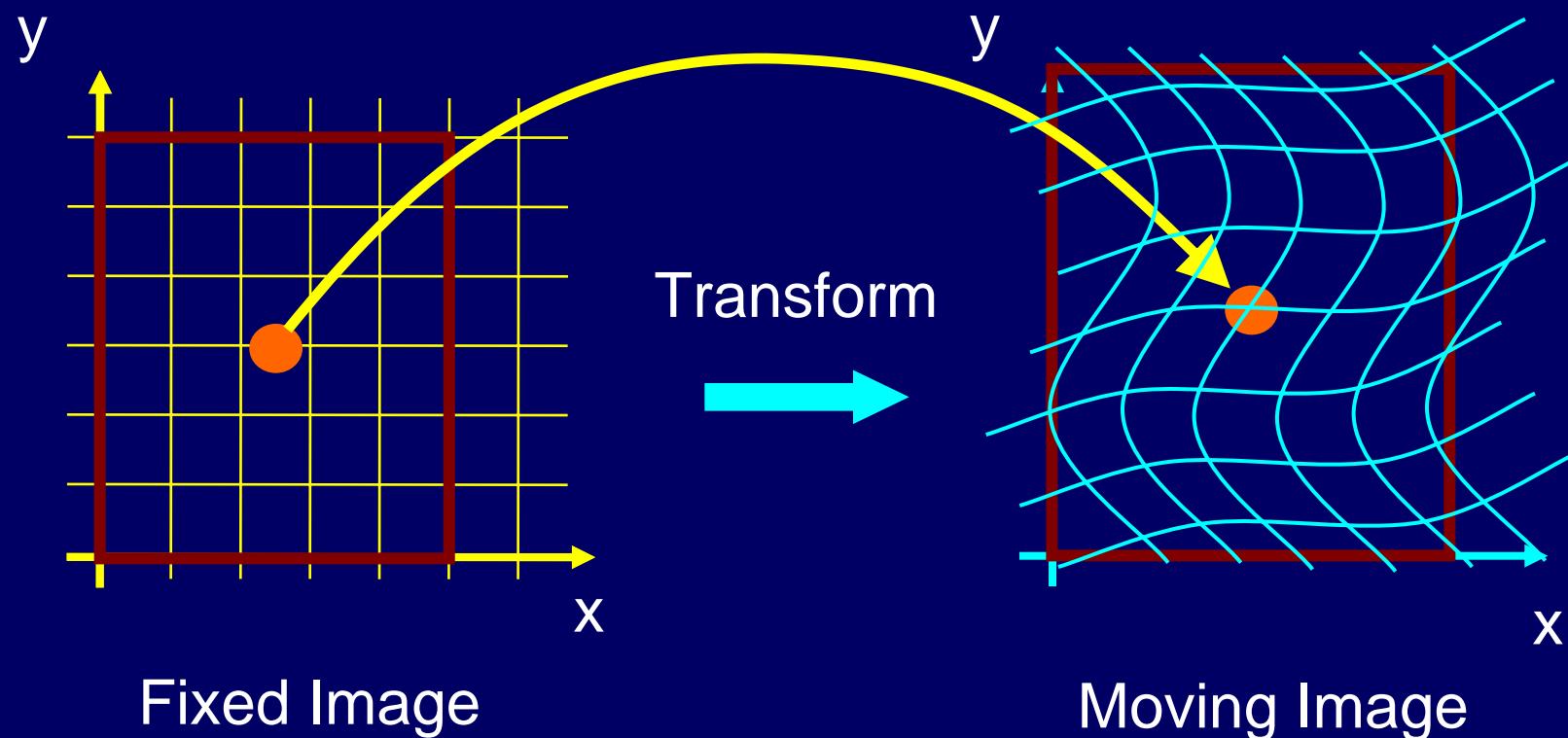


Images I_T
& meshes M_S

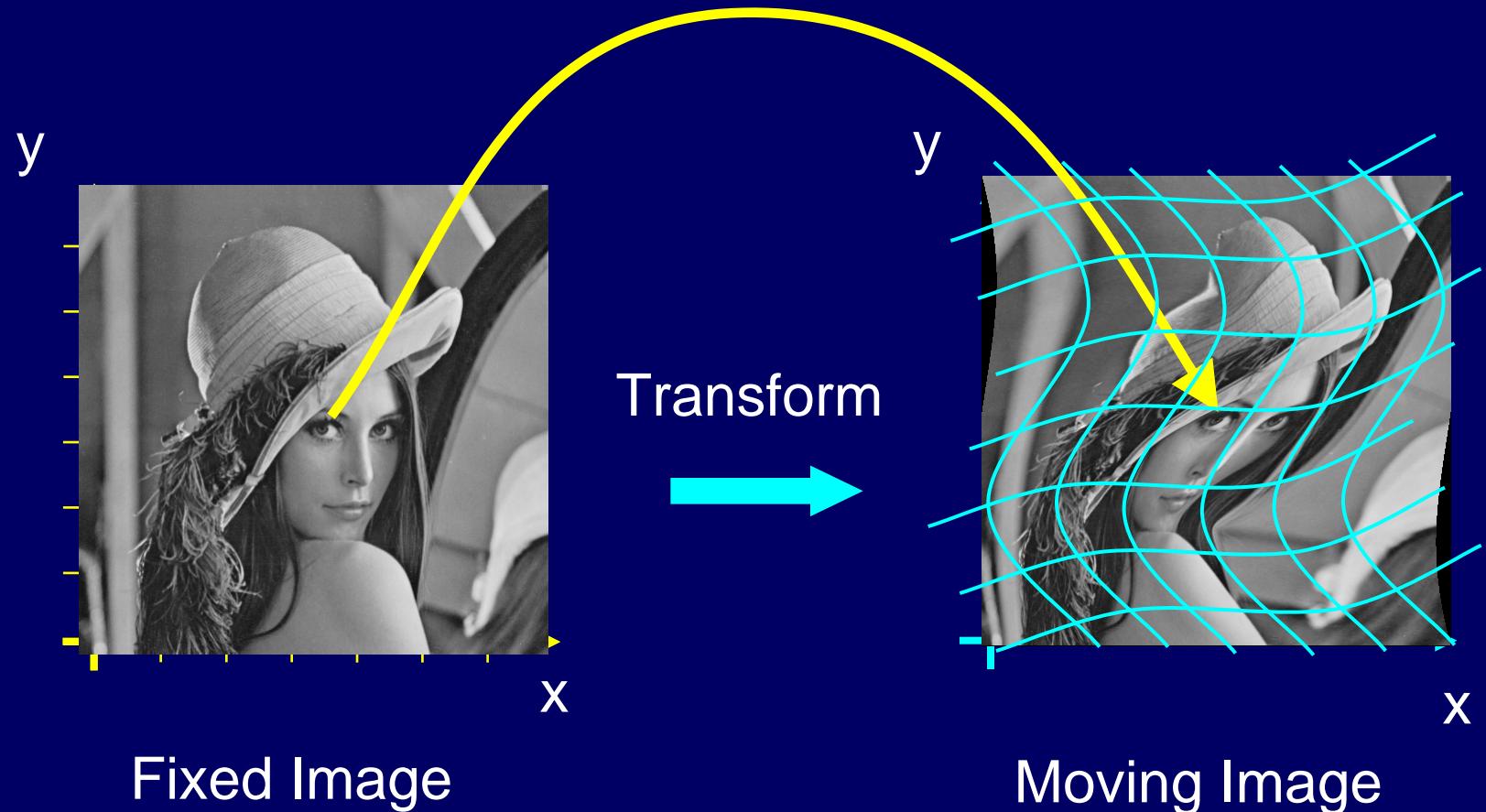


from G. Wolberg, CGI '96

Mesh Warping



Mesh Warping



Mesh Warping

- Source and target images are meshed
- The meshes for both images are interpolated
- The intermediate images are cross-dissolved
- Here, we look at 2D example

Mesh Warping Algorithm

- Algorithm

for each frame f **do**

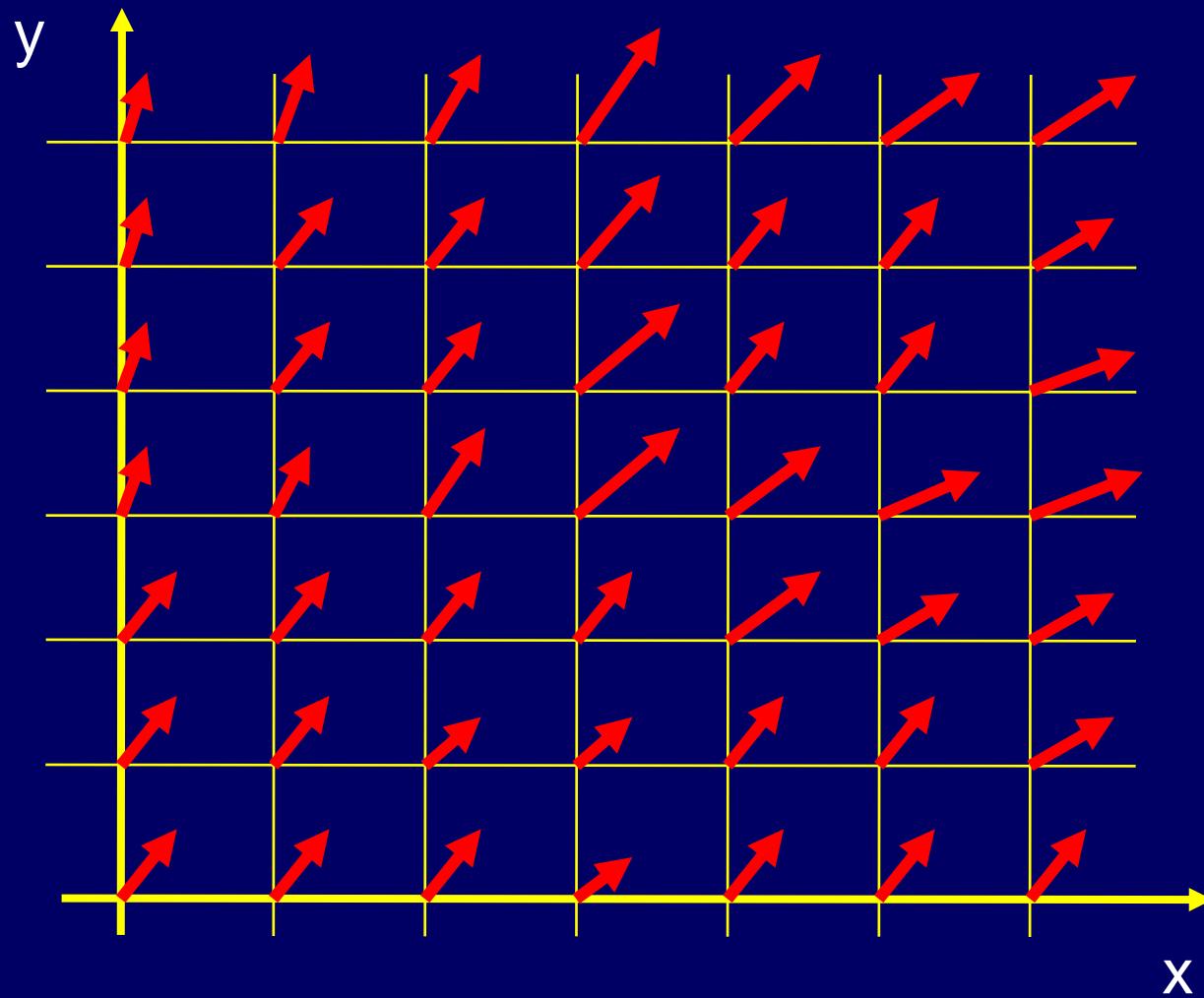
- interpolate mesh M , between M_s and M_t
- warp Image I_s to I_1 , using meshes M_s and M
- warp Image I_t to I_2 , using meshes M_t and M
- interpolate image I_1 and I_2

end

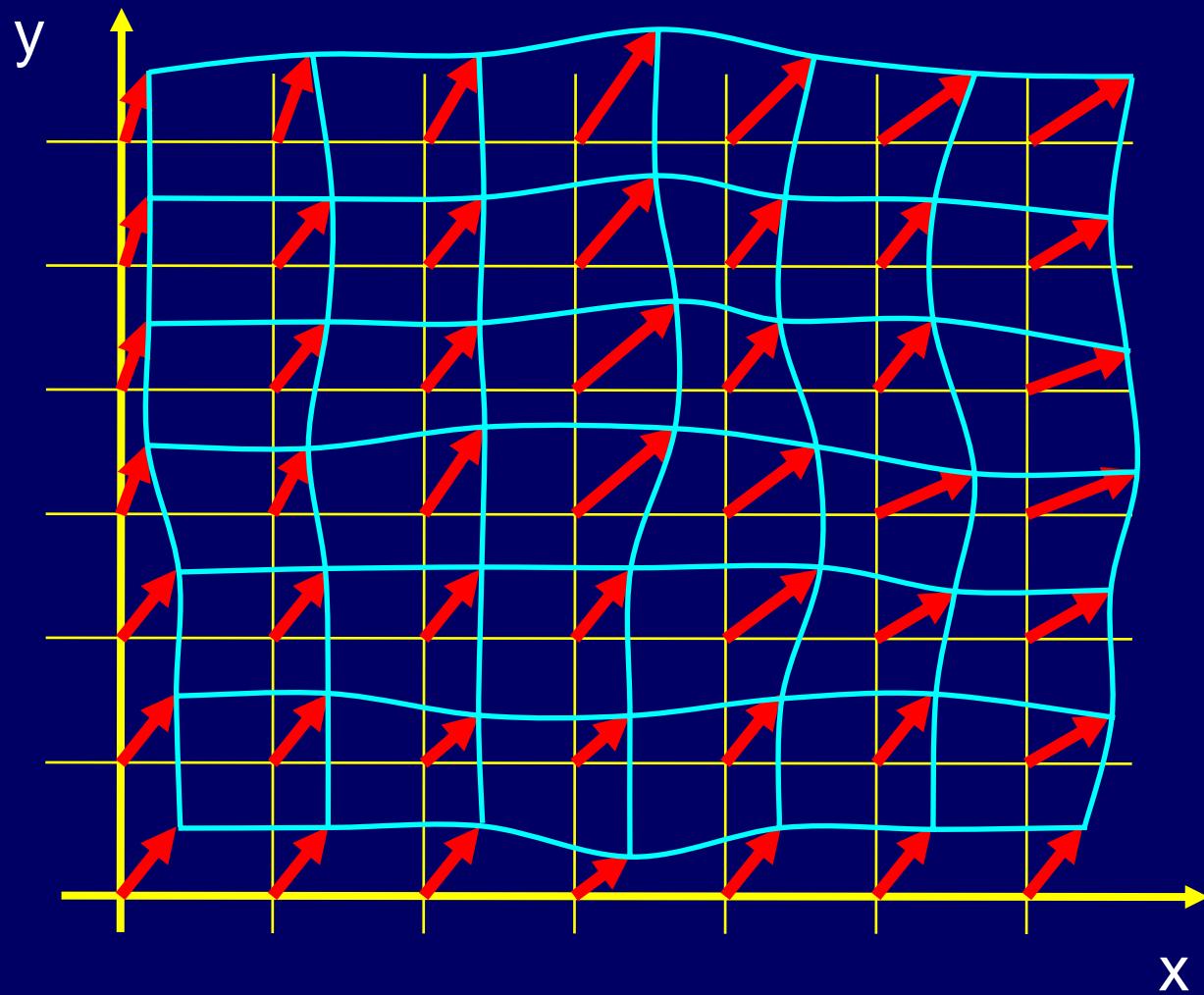
– I_s : source image, I_t : target image

– source image has mesh M_s , target image has mesh M_t

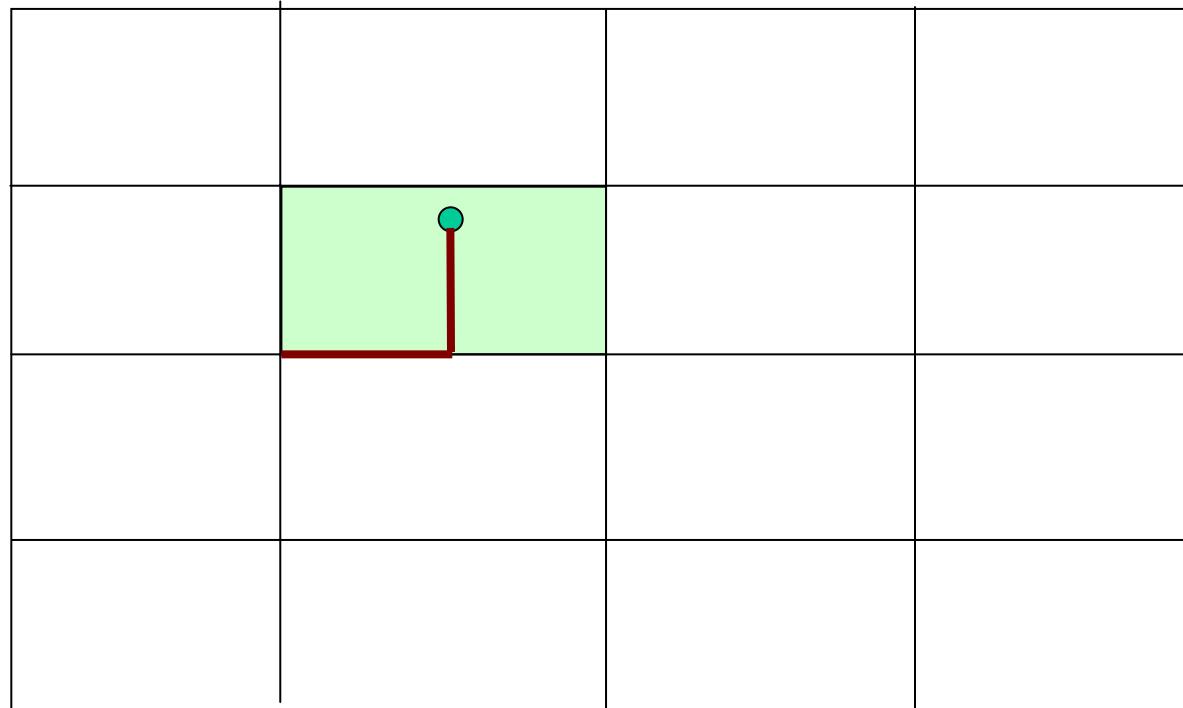
Mesh Deformation



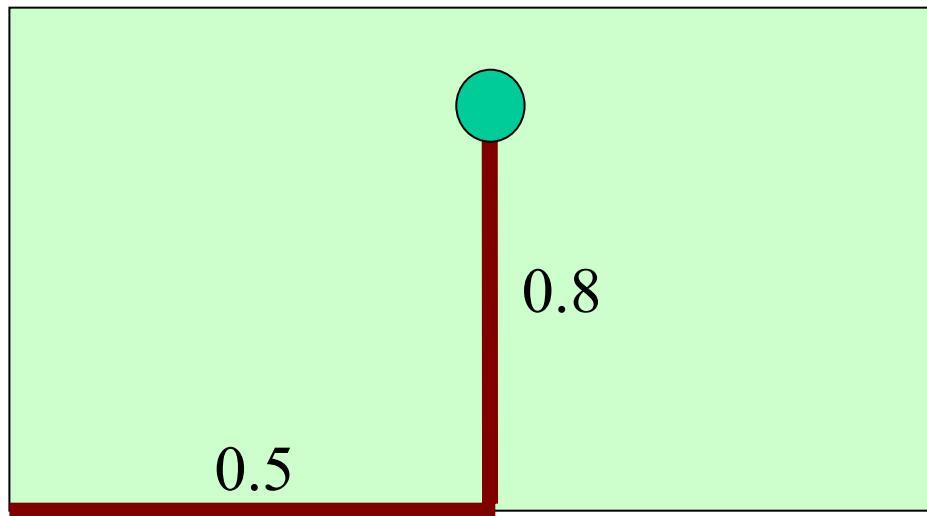
Mesh Deformation



Mesh Deformation

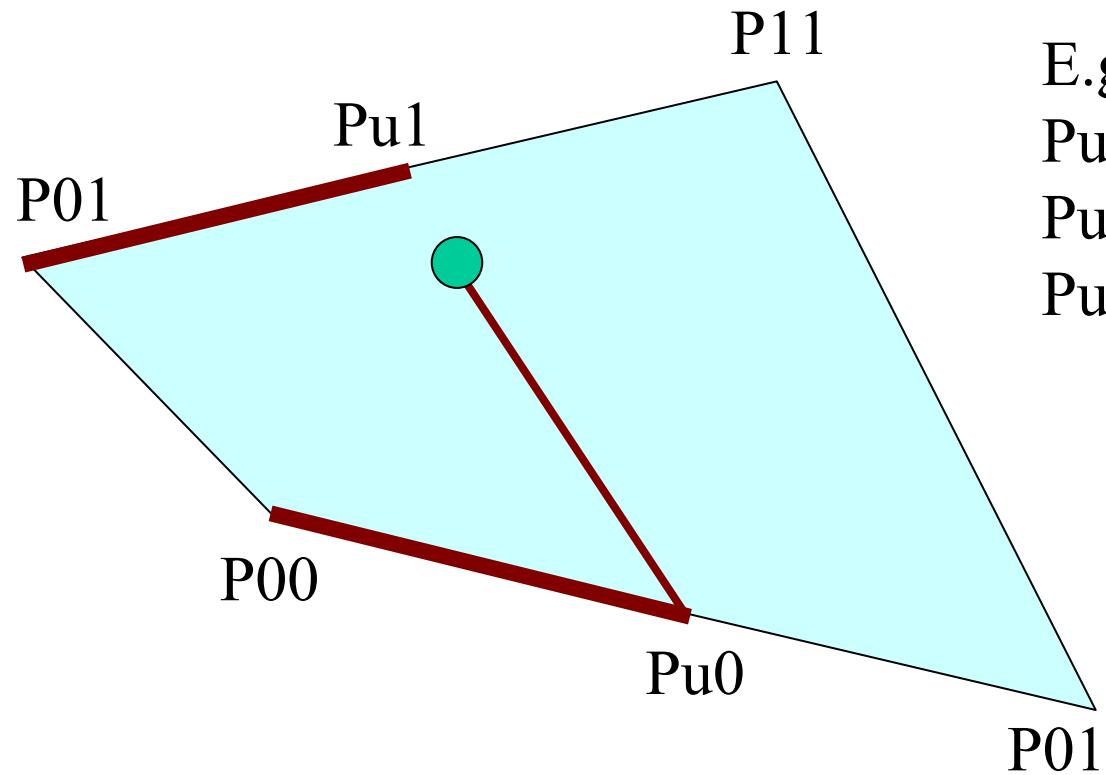


Mesh Deformation



For each vertex
identify cell,
fractional u,v
coordinate
in unit cell

Mesh Deformation



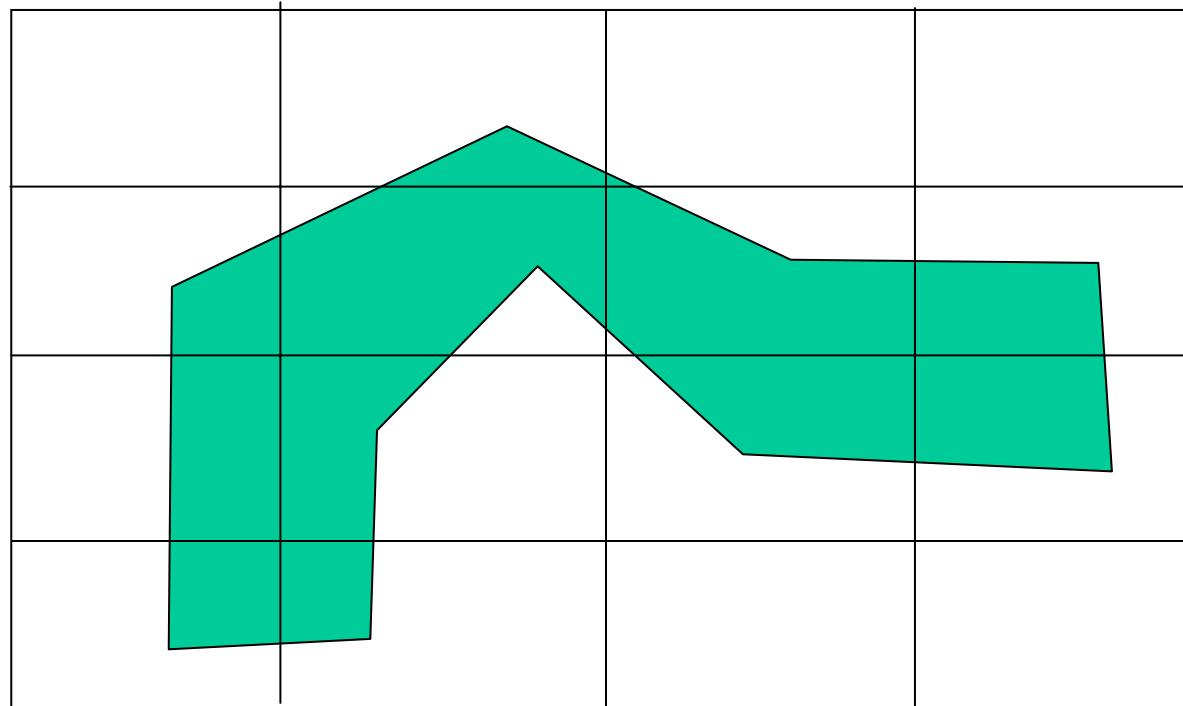
E.g. bilinear interpolation

$$Pu0 = (1-u)*P00 + u*P01$$

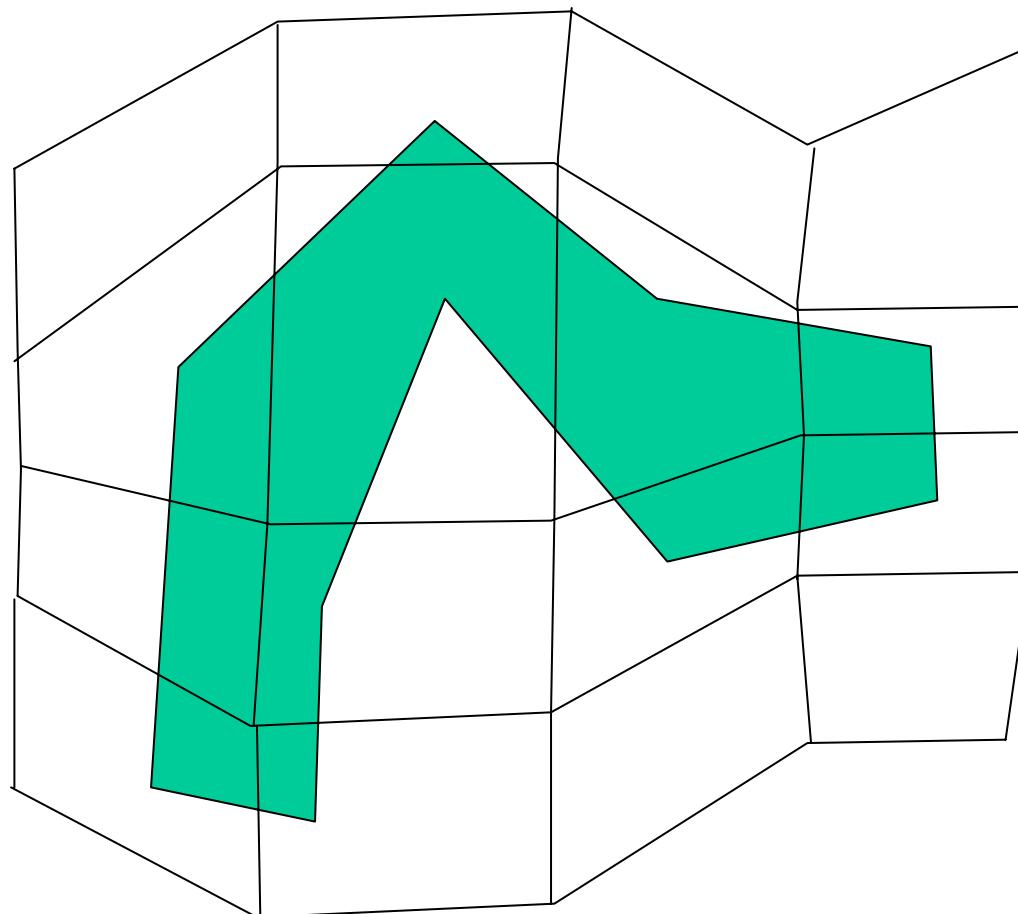
$$Pu1 = (1-u)*P01 + u*P11$$

$$Puv = (1-v)*Pu0 + v*Pu1$$

Mesh Deformation

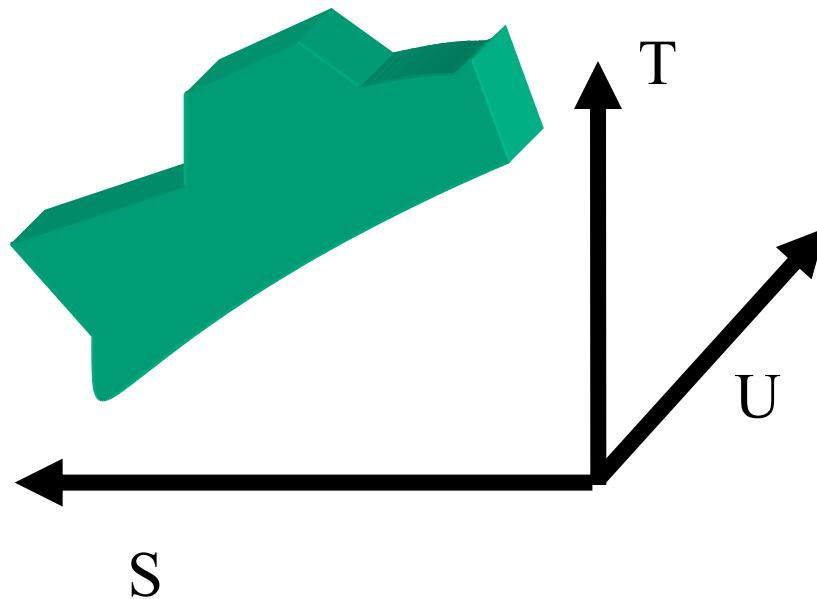


Mesh Deformation



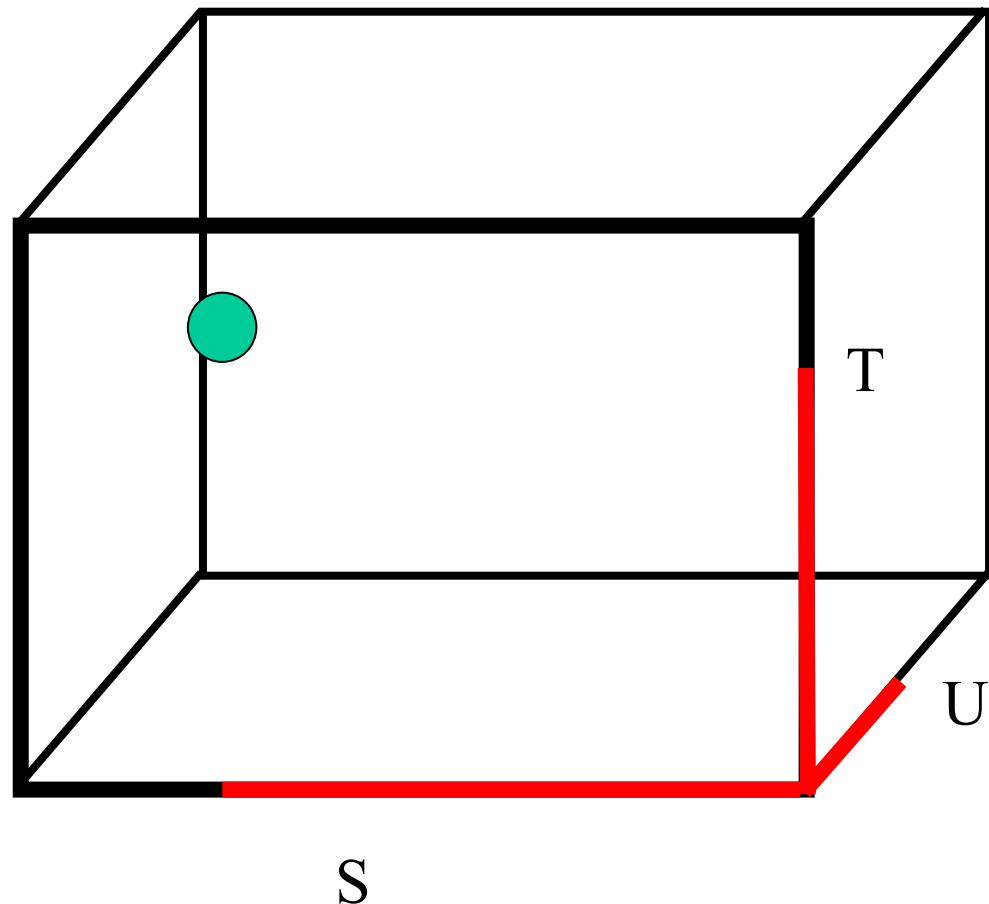
Free-Form Deformations

Define local coordinate system for deformation

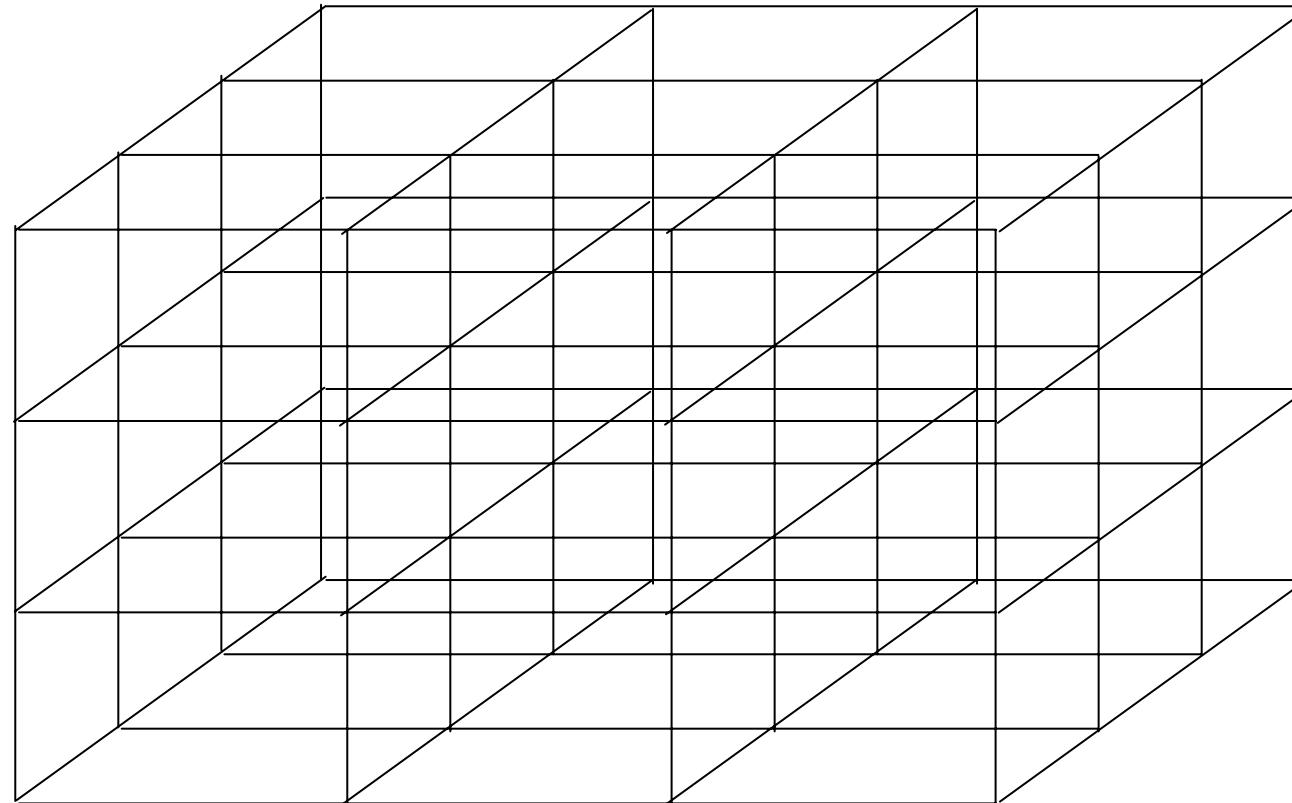


(not necessarily orthogonal)

FFD – Register Point in Cell



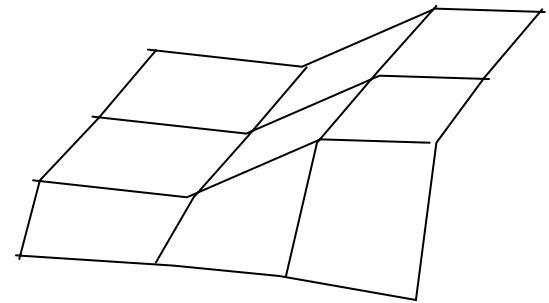
FFD – Create Control Grid



(not necessarily orthogonal)

FFD – Move and Reposition

Move control grid points



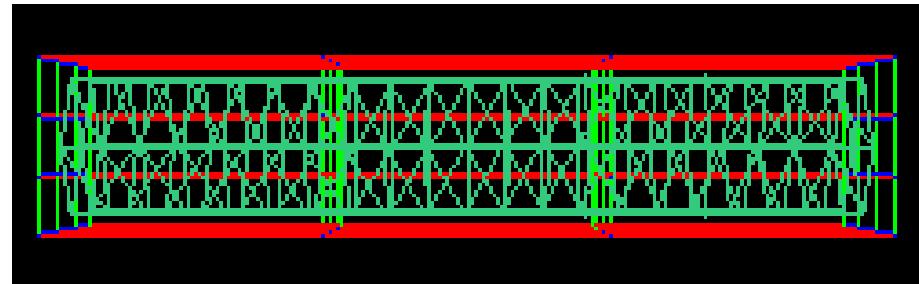
Usually tri-cubic interpolation is used with FFDs

Originally, Bezier interpolation was used.

B-spline and Catmull-Romm interpolation have also been used (as well as tri-linear interpolation)

FFD Example

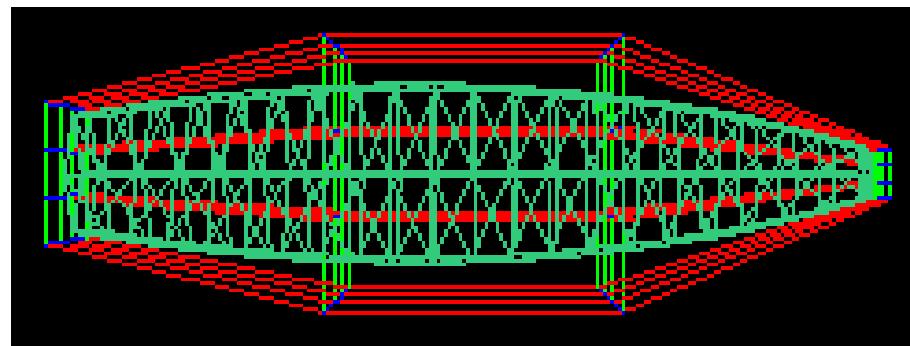
Step1



It is originally a cylinder.

Red boundary is FFD block embedded with that cylinder.

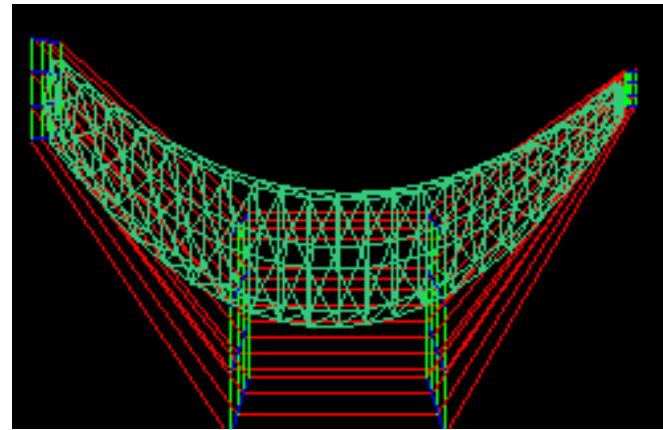
Step2



Move control points of each end, and you can see cylinder inside also changes.

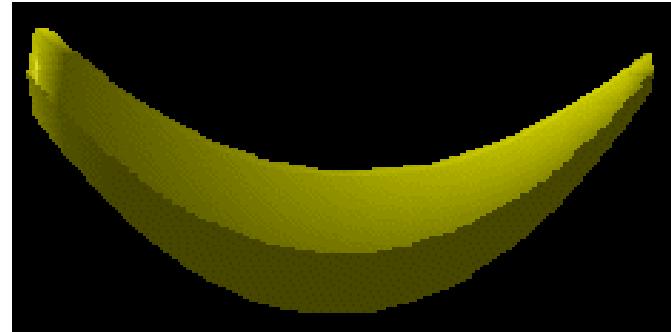
FFD Example

step3



Move inner control points downwards.

step4



Finally, get a shaded version of banana!

BSplines (Cubic) Interpolation



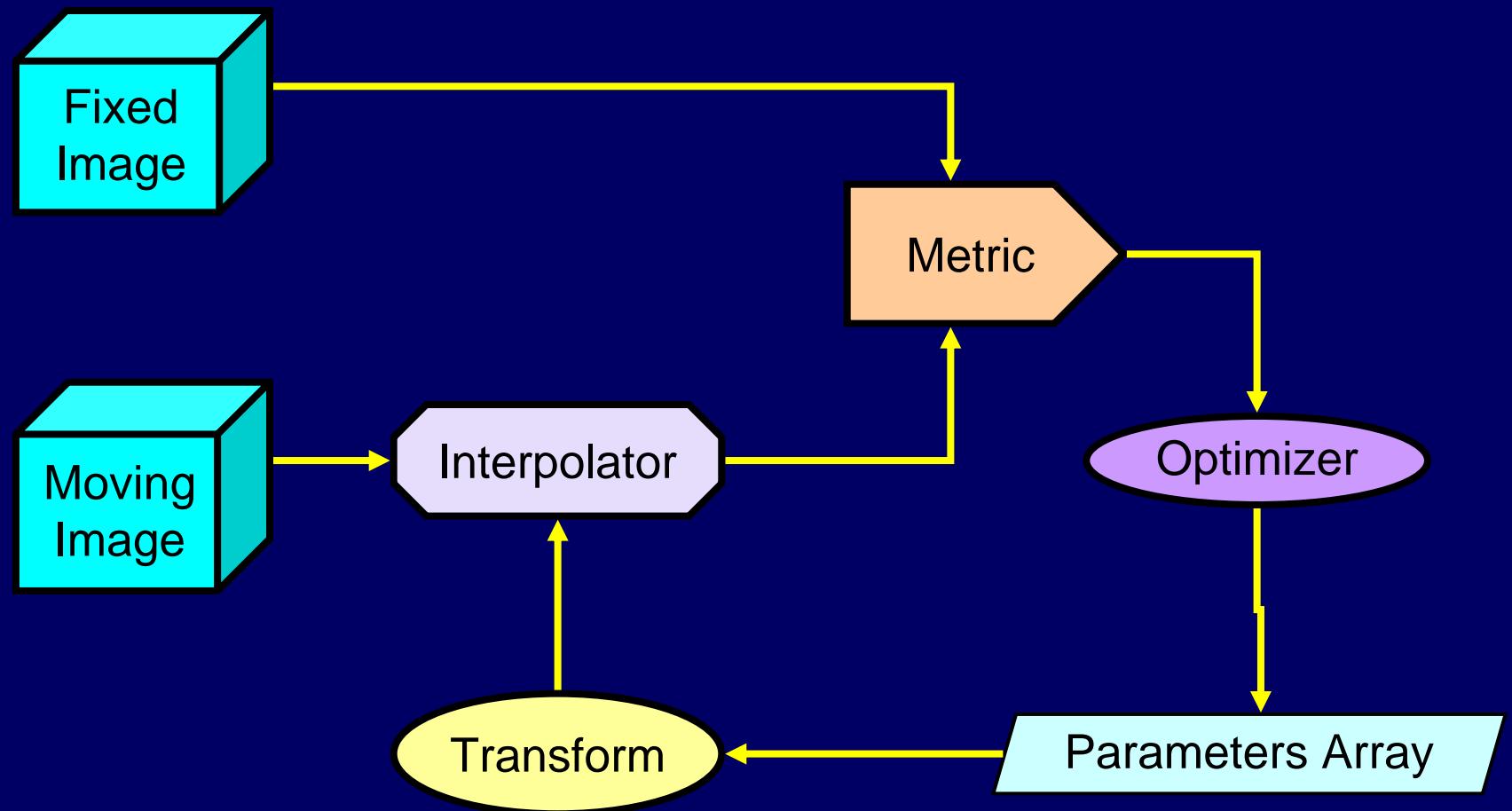
Original Lena

BSplines (Cubic) Interpolation



Deformed with BSpline Transform

Deformable Registration Framework



Deformable Registration



Deformed with BSpline Transform

Deformable Registration



Registered with BSpline Transform

Deformable Registration

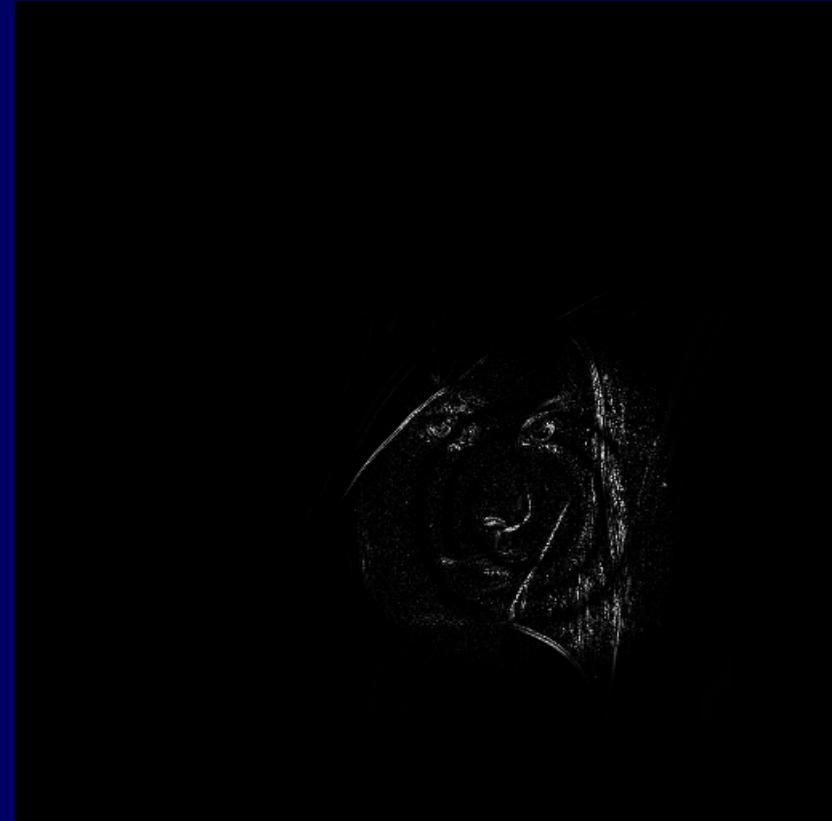


Original Lena

Deformable Registration



Difference Before
Registration



Difference After
Registration

Control Point Warping

Control Point Warping

Instead of a warping mesh, use arbitrary correspondence points:

Tip of one person's nose to the tip of another, eyes to eyes, etc.

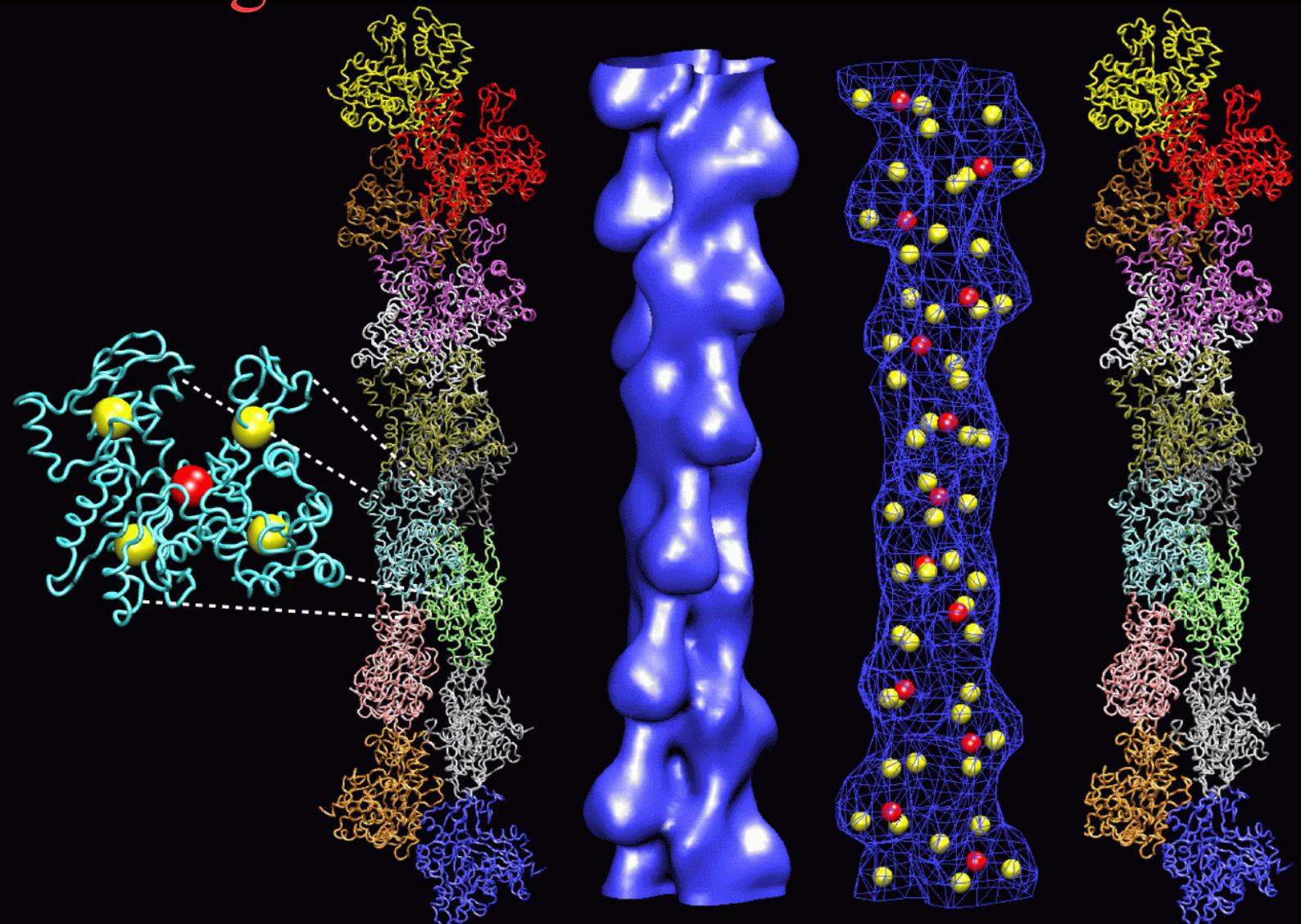
Interpolate between correspondence points to determine how points move

Apply standard warping (forward or backward):

In-between image is a weighted average of the source and destination corresponding pixels

Here we look at 3D example...

Finding Control Points in 3D Structures



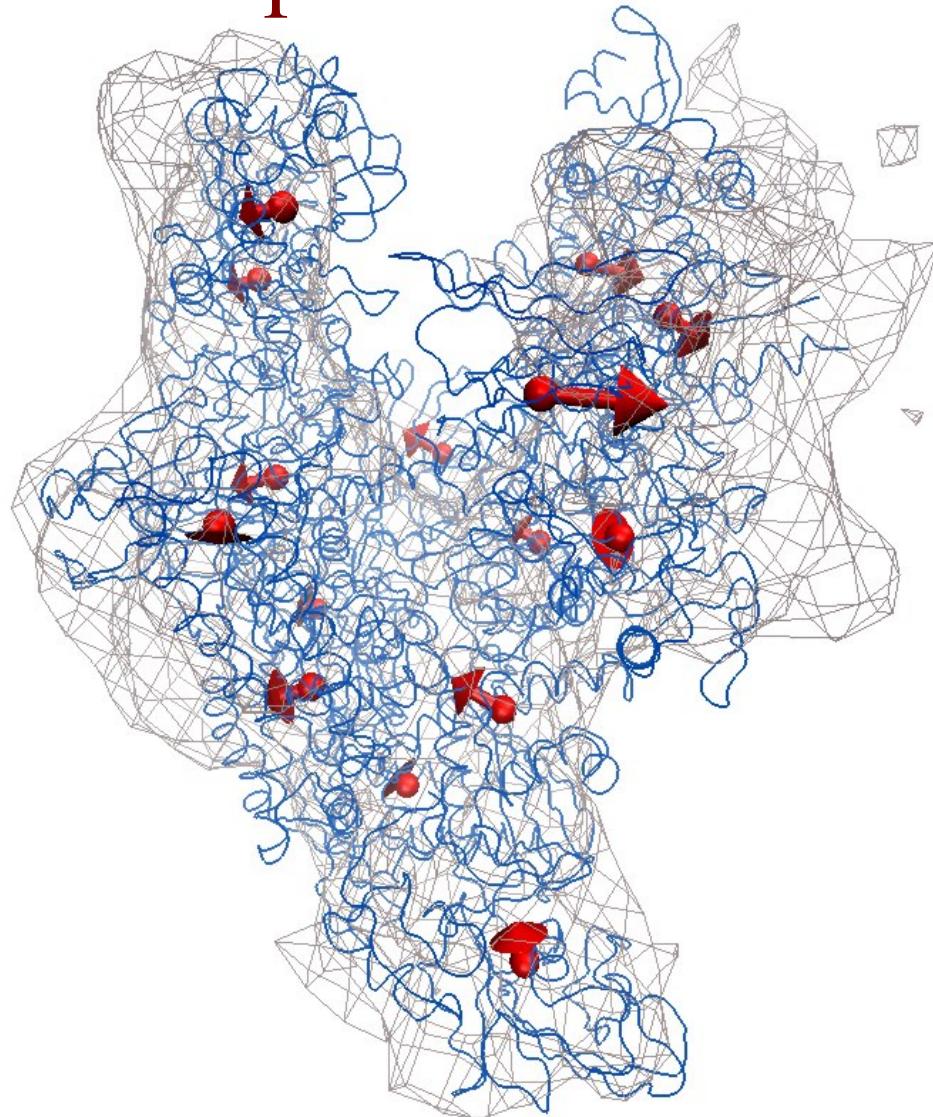
Actin filament: Reconstruction from EM data at 20 \AA resolution

rmsd: 1.1 \AA

Wriggers et al., J. Molecular Biology, 1998, 284: 1247-1254

Control Point Displacements

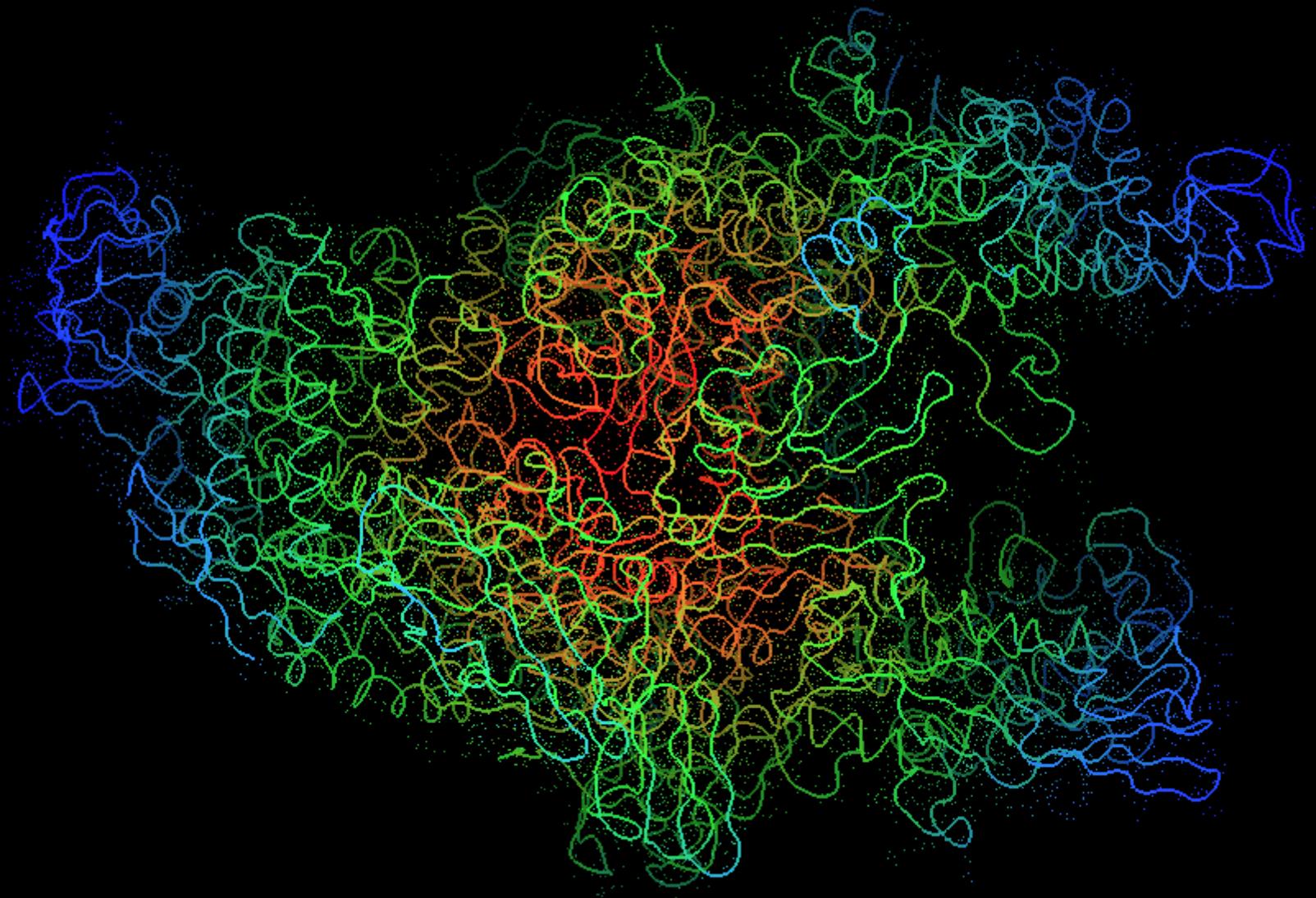
Have 2 conformations,
both source and target
characterized by
control points



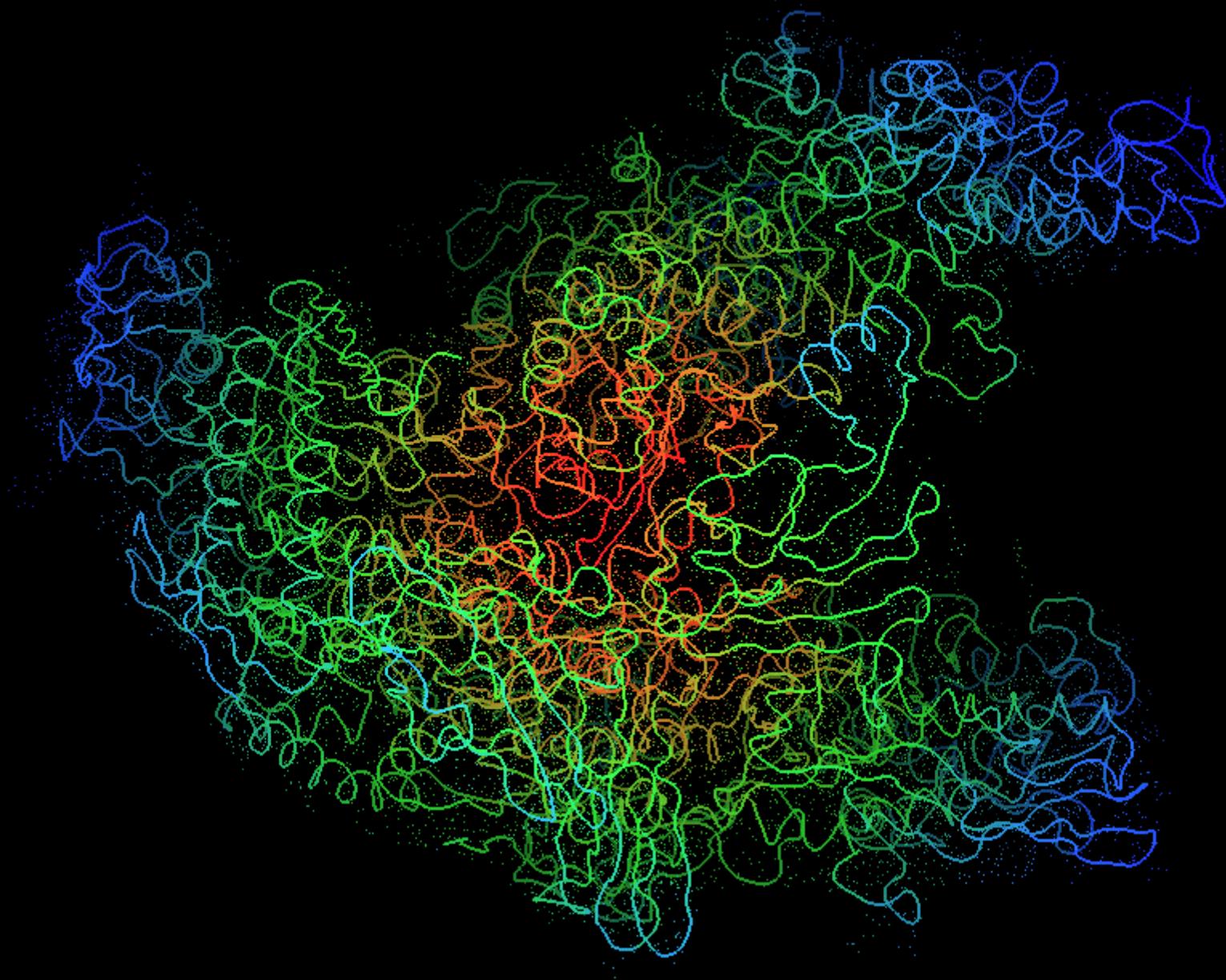
RNA Polymerase, Wriggers, Structure, 2004, Vol. 12, pp. 1-2.

Automating image morphing: <https://www.youtube.com/watch?v=ICitcimKZZk>

RNAP Example: Source Structure



Control: Molecular Dynamics



Manipulating Facial Appearance through Shape and Color

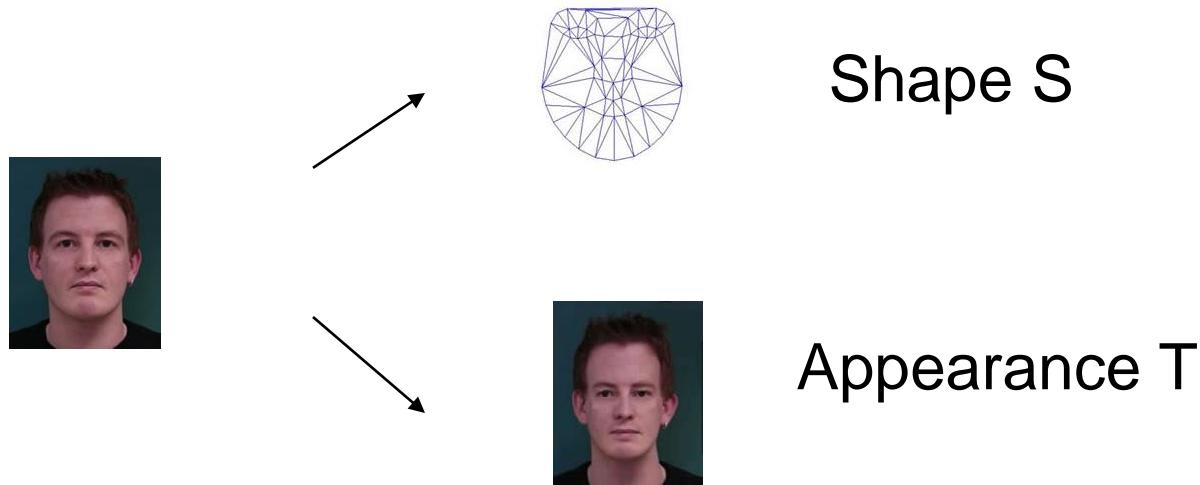
Duncan A. Rowland and David I.
Perrett

St Andrews University

IEEE CG&A, September 1995

The Morphable Face Model

- shape vector $\mathbf{S} = (x_1, y_1, x_2, \dots, y_n)^T$
- appearance (texture) vector $\mathbf{T} = (R_1, G_1, B_1, R_2, \dots, G_n, B_n)^T$



The Morphable face model

- Assuming that we have m such vector pairs in full correspondence, we can form new shapes \mathbf{S}_{model} and new appearances \mathbf{T}_{model} as:

$$\mathbf{S}_{model} = \sum_{i=1}^m a_i \mathbf{S}_i \quad \mathbf{T}_{model} = \sum_{i=1}^m b_i \mathbf{T}_i$$

$$s = \alpha_1 \cdot \text{[face]} + \alpha_2 \cdot \text{[face]} + \alpha_3 \cdot \text{[face]} + \alpha_4 \cdot \text{[face]} + \dots = \mathbf{S} \cdot \mathbf{a}$$

$$t = \beta_1 \cdot \text{[face]} + \beta_2 \cdot \text{[face]} + \beta_3 \cdot \text{[face]} + \beta_4 \cdot \text{[face]} + \dots = \mathbf{T} \cdot \mathbf{b}$$

- If number of basis faces m is large enough to span the face subspace then:
- Any new face can be represented as a pair of vectors

Face averaging by morphing

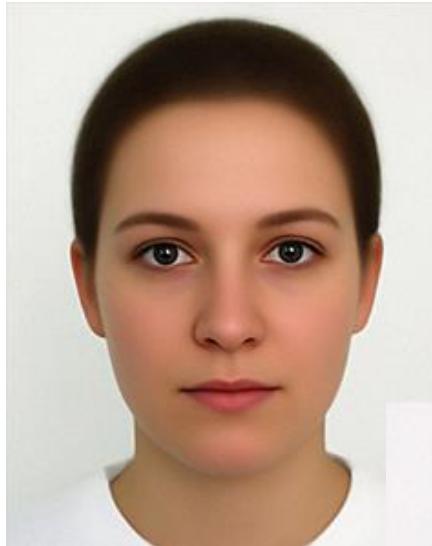


average faces

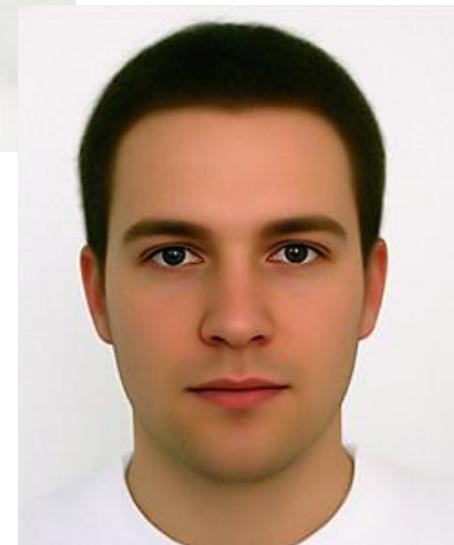
<http://www.beautycheck.de>

Subpopulation means

- Examples:
 - Happy faces
 - Young faces
 - Asian faces
 - Etc.
 - Sunny days
 - Rainy days
 - Etc.
 - Etc.



Average female



Average male

The average face



On the left: the “real” Miss Germany 2002 (= Miss Berlin) and on the right: the “virtual” Miss Germany, which was computed by blending together all contestants of the final round and was rated as being much more attractive.

Women In Arts

http://www.youtube.com/watch?v=nUDIoN-_Hxs

References

- Thaddeus Beier, Shawn Neely, [Feature-Based Image Metamorphosis](#), SIGGRAPH 1992, pp35-42.
- Detlef Ruprecht, Heinrich Muller, [Image Warping with Scattered Data Interpolation](#), IEEE Computer Graphics and Applications, March 1995, pp37-43.
- Seung-Yong Lee, Kyung-Yong Chwa, Sung Yong Shin, [Image Metamorphosis Using Snakes and Free-Form Deformations](#), SIGGRAPH 1995.
- Seungyong Lee, Wolberg, G., Sung Yong Shin, [Polymorph: morphing among multiple images](#), IEEE Computer Graphics and Applications, Vol. 18, No. 1, 1998, pp58-71.
- Peinsheng Gao, Thomas Sederberg, [A work minimization approach to image morphing](#), The Visual Computer, 1998, pp390-400.
- George Wolberg, [Image morphing: a survey](#), The Visual Computer, 1998, pp360-372.

Overview of Morphing Methods

- Mesh Warping
- Field Morphing
- Radial Basis Function
- Energy minimization
- Multilevel Free-Form Deformation
- Work minimization

Image Morphing: A Survey

George Wolberg 1998