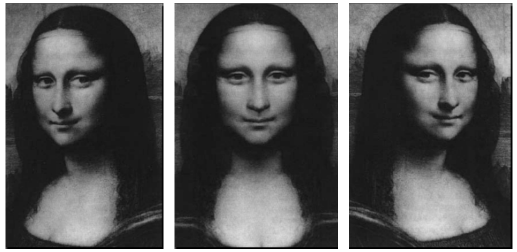
Interactive Computer Graphics

- Lecture 14+15: Warping and Morphing

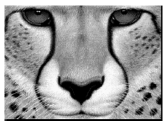
Warping and morphing



Warping and Morphing

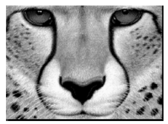
- What is
  - warping ?
  - morphing ?

?



Warping and Morphing

- What is
  - warping ?
  - morphing ?



Warping

- The term warping refers to the geometric transformation of graphical objects (images, surfaces or volumes) from one coordinate system to another coordinate system.
- Warping does not affect the attributes of the underlying graphical objects.
- Attributes may be
  - color (RGB, HSV)
  - texture maps and coordinates
  - normals, etc.

Morphing

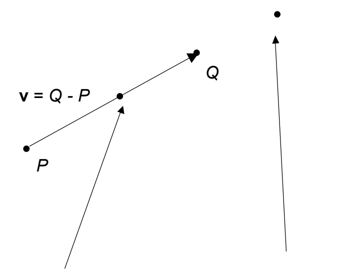
- The term morphing stands for metamorphosing and refers to an animation technique in which one graphical object is gradually turned into another.
- Morphing can affect both the shape and attributes of the graphical objects.

Morphing = Object Averaging

- The aim is to find "an average" between two objects
  - Not an average of two images of objects…
  - …but an image of the average object!
  - How can we make a smooth transition in time?
    - Do a "weighted average" over time t
- How do we know what the average object looks like?
  - Need an algorithm to compute the average geometry and appearance

Averaging

What's the average of P and Q?



Linear Interpolation
(Affine Combination):
New point *aP* + *bQ*,
defined only when *a*+*b* = 1
So aP+bQ = aP+(1-a)Q

P + 0.5v
= *P* + 0.5(*Q* – *P*)
= 0.5*P* + 0.5 *Q*

P + 1.5v
= *P* + 1.5(*Q* – *P*)
= -0.5*P* + 1.5 *Q*
(extrapolation)

Morphing using cross-dissolve

- Interpolate whole images:

$$I(t) = t*I_1 + (1-t)*I_2$$

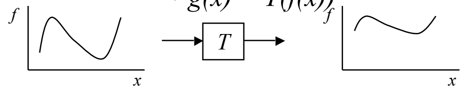
- This is called **cross-dissolve**
- But what is the images are not aligned?

Morphing using warping and cross-dissolve

- Align first, then cross-dissolve

Image warping

- image filtering: change ***range*** of image
  - *g(x) = T(f(x))*



- image warping: change ***domain*** of image
  - *g(x) = f(T(x))*

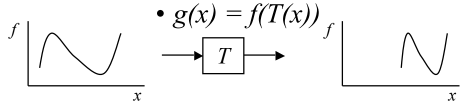
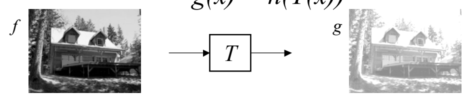
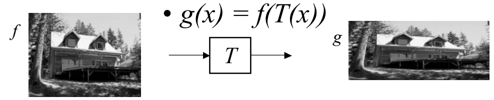


Image warping

- image filtering: change ***range*** of image
  - *g(x) = h(T(x))*

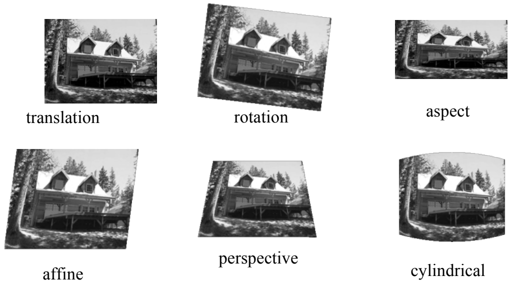


- image warping: change ***domain*** of image
  - *g(x) = f(T(x))*

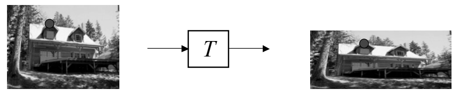


Parametric (global) warping

- Examples of parametric warps:



Parametric (global) warping



p = (x,y) **p’** = (x’,y’)

- Transformation T can be expressed as a mapping:

$$p' = T(p)$$

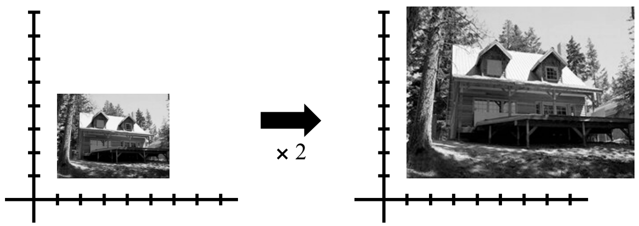
- Transformation T can be expressed as a matrix:

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

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

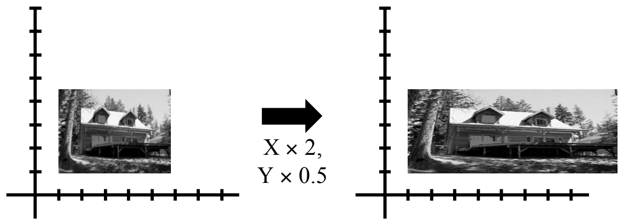
Scaling

- *Scaling* a coordinate means multiplying each of its components by a scalar
- *Uniform scaling* means this scalar is the same for all components:



Scaling

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



Scaling

- Scaling operation:

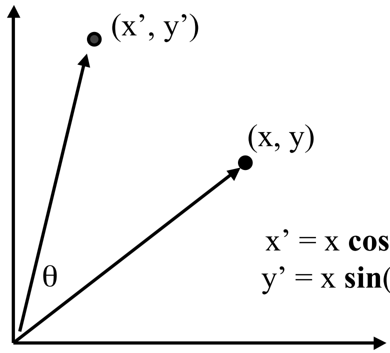
$$x' = ax$$
$$y' = by$$

- Or, in matrix form:

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

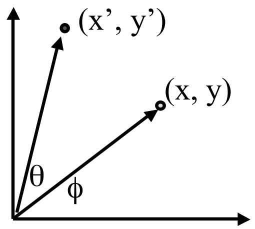
What is the inverse of S?

2-D Rotation



x' = x **cos**(θ) - y **sin**(θ)
y' = x **sin**(θ) + y **cos**(θ)

2-D Rotation



x = r cos (ϕ)
y = r sin (ϕ)
x' = r cos (ϕ + θ)
y' = r sin (ϕ + θ)

Trig Identity…
x' = r cos(ϕ) cos(θ) – r sin(ϕ) sin(θ)
y' = r sin(ϕ) sin(θ) + r cos(ϕ) cos(θ)

Substitute…
x' = x **cos**(θ) - y **sin**(θ)
y' = x **sin**(θ) + y **cos**(θ)

2-D Rotation

- This is easy to capture in matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}}_{\mathbf{R}} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Even though sin(θ) and cos(θ) are nonlinear functions of θ,
  - x' is a linear combination of x and y
  - y' is a linear combination of x and y
- What is the inverse transformation?
  - Rotation by –θ
  - For rotation matrices, det(R) = 1 so $\mathbf{R}^{-1} = \mathbf{R}^{T}$

2x2 Matrices

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

2D Identity?

$$x' = x \qquad y' = y$$

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

2D Scale around (0,0)?

$$\boldsymbol{x' = s_x * x} \qquad \boldsymbol{y' = s_y * y}$$

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

2x2 Matrices

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

2D Rotate around (0,0)?

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

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

2D Shear?

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

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

2x2 Matrices

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

2D Mirror about Y axis?

$$x' = -x \qquad y' = y$$

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

2D Mirror over (0,0)?

$$x' = -x \qquad y' = -y$$

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

2x2 Matrices

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

2D Translation?

$$\boldsymbol{x' = x + t_x} \qquad \boldsymbol{y' = y + t_y}$$

NO!

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

All 2D Linear Transformations

- Linear transformations are combinations of …
  - Scale,
  - Rotation,
  - Shear, and
  - Mirror

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Properties of linear transformations:
  - Origin maps to origin
  - Lines map to lines
  - Parallel lines remain parallel
  - Ratios are preserved
  - Closed under composition

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

Homogeneous Coordinates

- Q: How can we represent translation as a matrix transformation?

$$\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{t}_x$$
$$\boldsymbol{y}' = \boldsymbol{y} + \boldsymbol{t}_y$$

- A: Using the translation parameters as the rightmost column:

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

Basic 2D Transformations

- Basic 2D transformations as 3x3 matrices

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

Translate

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

Scale

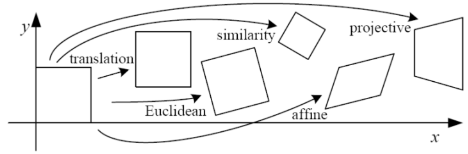
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotate

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

Shear

2D image transformations



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

Transformations

- Dimensions of transformation
  - 1D: curves
  - 2D: images
  - 3D: volumes
- Types of transformations
  - rigid
  - affine
  - polynomial
    - quadratic
    - cubic
  - splines

Transformations in 3D: Rigid

- Rigid transformation (6 degrees of freedom)

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} r_{01} & r_{02} & r_{03} & t_x \\ r_{11} & r_{12} & r_{13} & t_y \\ r_{21} & r_{22} & r_{23} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = T^x_{rigid} \cdot T^y_{rigid} \cdot T^z_{rigid} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \\ 0 \end{pmatrix}$$

- t_x, t_y, t_z describe the 3 translations in x, y and z
- r_{11}, ..., r_{33} describe the 3 rotations around x, y, z

Transformations in 3D: Rigid

$$\mathbf{T}^x_{rigid} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{T}^y_{rigid} = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{T}^z_{rigid} = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Transformations in 3D: Affine

- Affine transformations (12 degrees of freedom)

$$\mathbf{T}_{scale} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{T}^{xy}_{shear} = \begin{pmatrix} 1 & 0 & sh_x & 0 \\ 0 & 1 & sh_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{T}(x, y, z) = \mathbf{T}_{shear} \cdot \mathbf{T}_{scale} \cdot \mathbf{T}_{rigid} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Non-rigid transformations

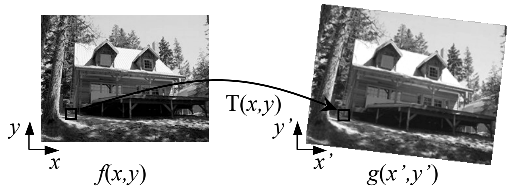
- Quadratic transformation (30 degrees of freedom)

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} r_{00} & \cdots & r_{08} & r_{09} \\ r_{10} & \cdots & r_{18} & r_{19} \\ r_{20} & \cdots & r_{28} & r_{29} \\ 0 & \cdots & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x^2 \\ y^2 \\ \vdots \\ 1 \end{pmatrix}$$

Non-rigid transformations

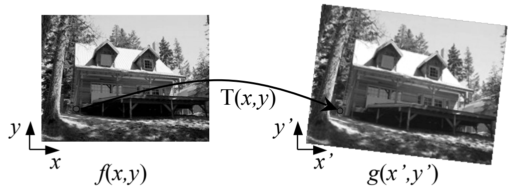
- Can be extended to other higher-order polynomials:
  - 3rd order (60 DOF)
  - 4th order (105 DOF)
  - 5th order (168 DOF)
- Problems:
  - can model only global shape changes, not local shape changes
  - higher order polynomials introduce artifacts such as oscillations

Image warping



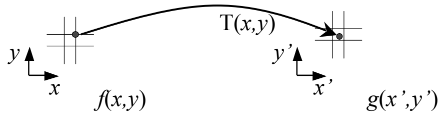
- Given a coordinate transform $(x',y') = T(x,y)$ and a source image $f(x,y)$, how do we compute a transformed image $g(x',y') = f(T(x,y))$?

Forward warping



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

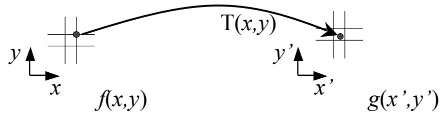
Forward warping



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

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

Forward warping



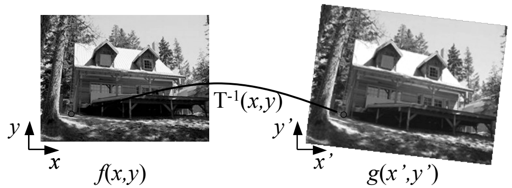
- Send each pixel $f(x,y)$ to its corresponding location $(x',y') = T(x,y)$ in the second image

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

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

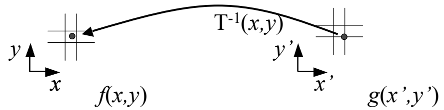
  - known as "splatting"

Inverse warping



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

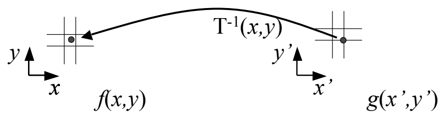
Inverse warping



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

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

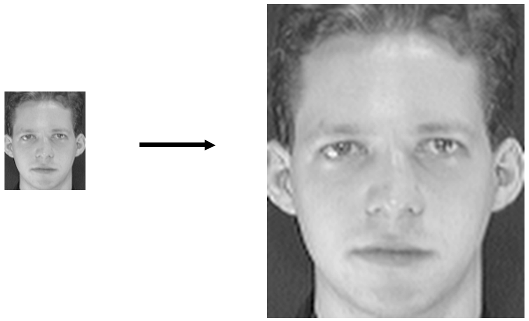
Inverse warping



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

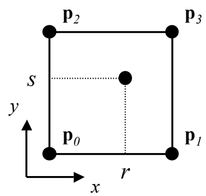
Interpolation

Interpolation



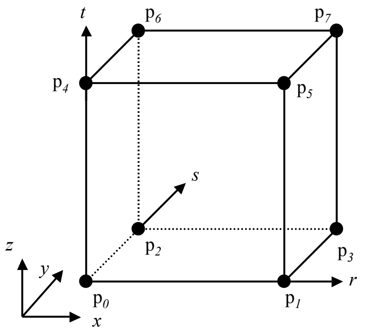
Interpolation: Linear, 2D

$$f(p) = \sum_{i=0}^{n-1} w_i f(p_i)$$



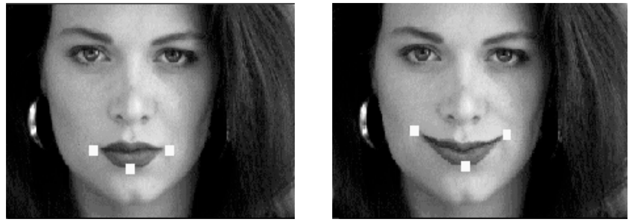
$$w_0 = (1-r)(1-s)$$
$$w_1 = r(1-s)$$
$$w_2 = (1-r)s$$
$$w_3 = rs$$

Interpolation: Linear, 3D

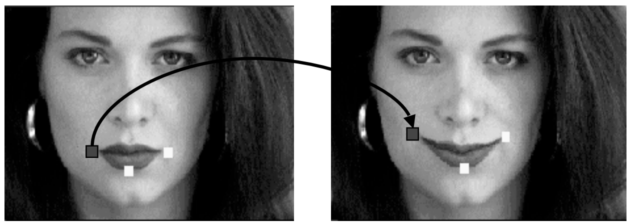


$$w_0 = (1-r)(1-s)(1-t)$$
$$w_1 = r(1-s)(1-t)$$
$$w_2 = (1-r)s(1-t)$$
$$w_3 = rs(1-t)$$
$$w_4 = (1-r)(1-s)t$$
$$w_5 = r(1-s)t$$
$$w_6 = (1-r)st$$
$$w_7 = rst$$

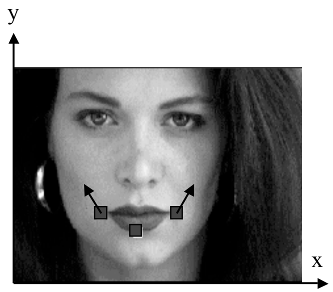
Non-rigid transformations



Non-rigid transformations: Correspondences

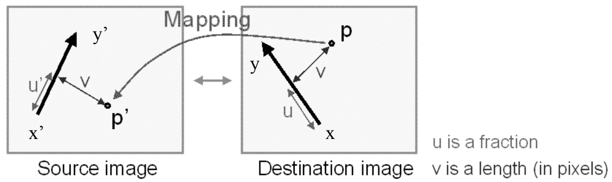


Non-rigid transformations: Correspondences



Feature-Based Warping: Beier-Neeley

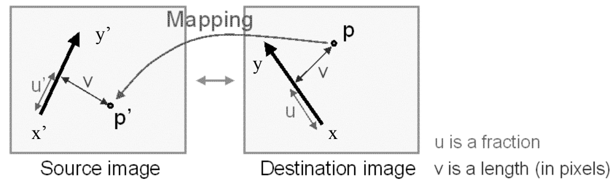
- Beier & Neeley use pairs of lines to specify warp
  - Given **p** in destination image, where is **p’** in source image?



Feature-Based Warping: Beier-Neeley

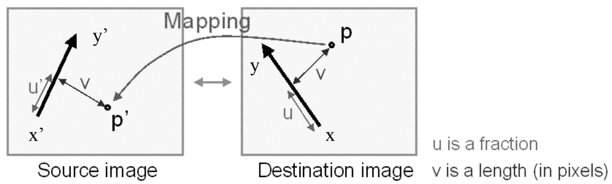
$$u = \frac{(p-x)\cdot(y-x)}{\|y-x\|^2} \qquad v = \frac{(p-x)\cdot Perpendicular(y-x)}{\|y-x\|}$$

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



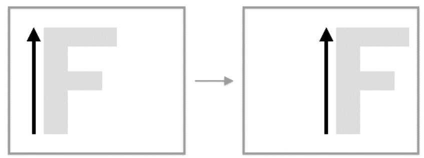
Feature-Based Warping: Beier-Neeley

- For each pixel p in the destination image
  - find the corresponding u,v
  - find the p’ in the source image for that u,v
  - destination(p) = source(p’)



Warping with One Line Pair: Beier-Neeley

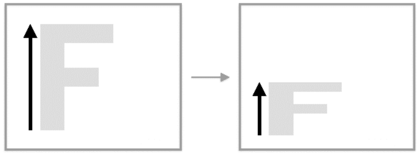
- What happens to the “F” ?



Translation !

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

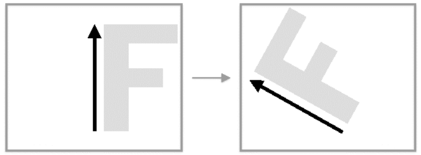
- What happens to the “F” ?



Scale !

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

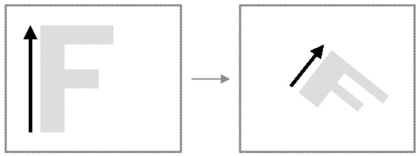
- What happens to the “F” ?



Rotation !

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

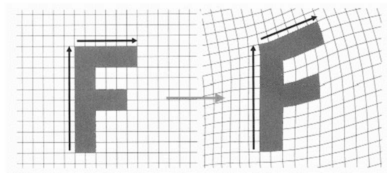
- What happens to the “F” ?



In general, similarity transformations

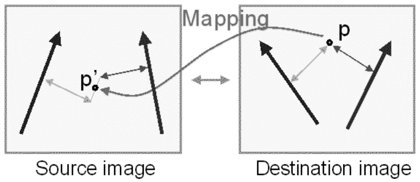
Warping with Multiple Line Pairs: Beier-Neeley

- Use weighted combination of points defined each pair of corresponding lines



Warping with Multiple Line Pairs: Beier-Neeley

- Use weighted combination of points defined by each pair corresponding lines



p' is a weighted average

Weighting Effect of Each Line Pair: Beier-Neeley

- To weight the contribution of each line pair

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

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

Warping Pseudocode: Beier-Neeley

```
foreach destination pixel p do
    psum = (0, 0)
    wsum = (0, 0)
    foreach line L[i] in destination do
        p'[i] = p transformed by (L[i], L'[i])
        psum = psum + p'[i] * weight[i]
        wsum += weight[i]
    end
    p' = psum / wsum
    destination(p) = source(p')
end
```