Geometric Operations and Morphing

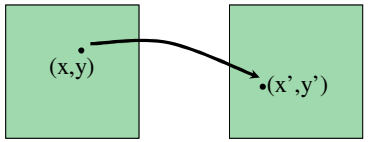
1

Geometric Transformation

- Operations depend on pixel's Coordinates.
- Context free.
- Independent of pixel values.

$$x \to f_x(x,y) = x'$$
$$y \to f_y(x,y) = y'$$

$$I(x,y) = I'\left(f_x(x,y), f_y(x,y)\right)$$

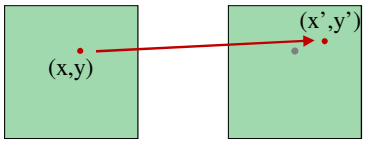


I(x,y) I'(x',y')

- Example: Translation

$$x' = f_x(x,y) = x+3$$
$$y' = f_y(x,y) = y-1$$

$$I'(x+3, y-1) = I(x,y)$$

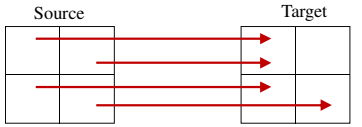


I(x,y) I'(x',y')

Forward Mapping

- Forward mapping:

$$x \to f_x(x,y) = x'$$
$$y \to f_y(x,y) = y'$$

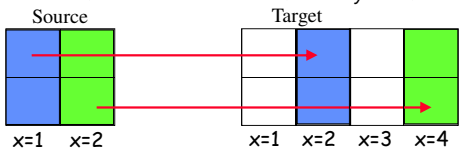


Forward Mapping

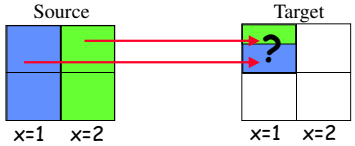
- Problems with forward mapping due to sampling:
  - Holes (some target pixels are not populated)
  - Overlaps (some target pixels assigned few colors)

Forward Mapping

$x' = f_x(x, y) = 2x$ $\quad y' = f_y(x, y) = y$

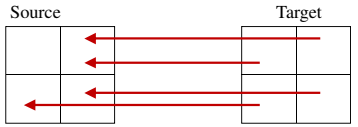


$x' = f_x(x, y) = 0.7x$ $\quad y' = f_y(x, y) = y$



Inverse Mapping

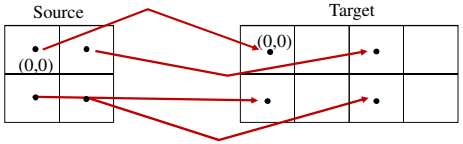
- Inverse mapping: $x' \rightarrow f_x^{-1}(x', y') = x$
  $y' \rightarrow f_y^{-1}(x', y') = y$



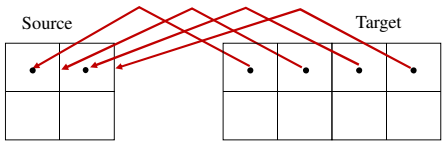
Inverse Mapping

- Each target pixel assigned a single color.
- Color Interpolation is required.

- Example: Scaling along X
  - Forward mapping: $x' = 2x\ ;\ \ y' = y$



  - Inverse mapping: $x = x'/2\ ;\ \ y = y'$



Interpolation

- What happens when a mapping function calculates a fractional pixel location?

- **Interpolation**: generates a new pixel by analyzing the surrounding pixels.

Interpolation

- Good interpolation techniques attempt to find an optimal balance between three undesirable artifacts: aliasing, blurring, and edge halos.

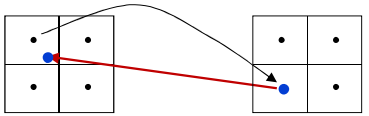
9

Nearest Neighbor Interpolation

- The assign value is taken from the pixel closest to the generated location:

$$I'(x', y') = I\left(round\left\{f_x^{-1}(x', y')\right\}, round\left\{f_y^{-1}(x', y')\right\}\right)$$

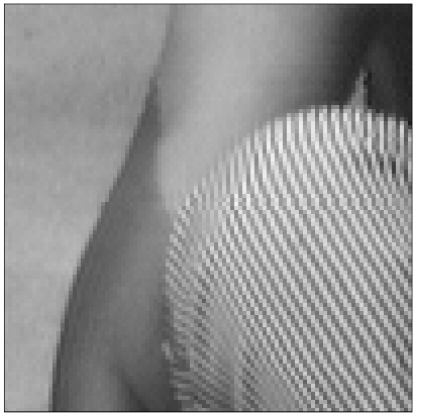
- Advantage:
  - Fast
- Disadvantage:
  - Jagged results
  - Aliasing near edges



Original Image

Nearest N. Interpolation

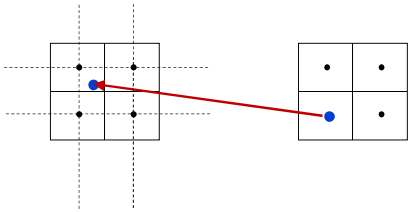
Original Image



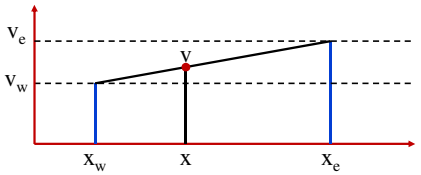
Nearest N. Interpolation

Bilinear Interpolation

- The assign value is a weighted sum of the four nearest pixels.
- Each weight is proportional to the distance from each existing pixel.



Linear Interpolation

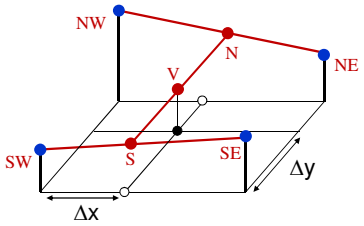


$$\frac{x - x_w}{x_e - x_w} = \frac{v - v_w}{v_e - v_w}$$

- Isolating v in the above equation:

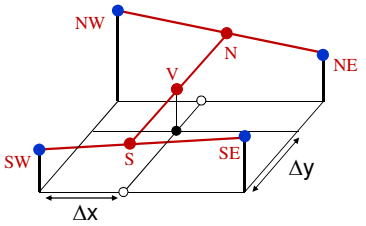
$$v = \alpha v_e + (1-\alpha) v_w \qquad where\ \alpha = \frac{x - x_w}{x_e - x_w}$$

Bilinear Interpolation



$$S = SE \cdot \Delta x + SW \cdot (1 - \Delta x)$$
$$N = NE \cdot \Delta x + NW \cdot (1 - \Delta x)$$
$$V = N \cdot \Delta y + S \cdot (1 - \Delta y)$$

Bilinear Interpolation

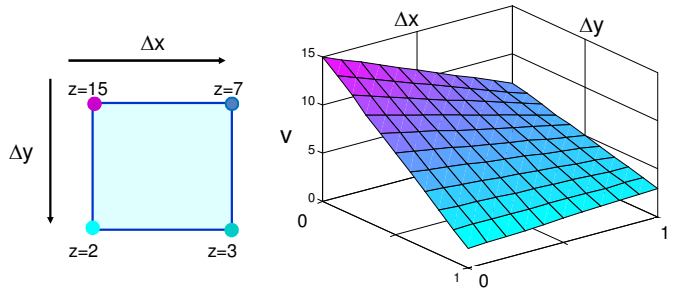


- The bilinear interpolation is the best fit low-degree polynomial of the form:

$$v(\Delta x, \Delta y) = \sum_{i,j=0}^{1} a_{ij} \Delta x^i \Delta y^j$$

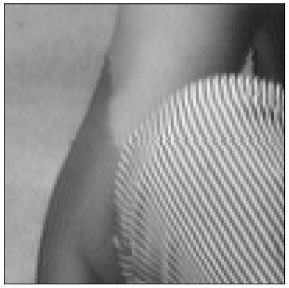
- The pixel's boundaries are C_0 continuous (continuous values across boundaries).

Bilinear example

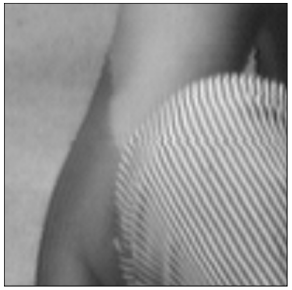


Nearest N. Interpolation

Bilinear Interpolation



Nearest N. Interpolation

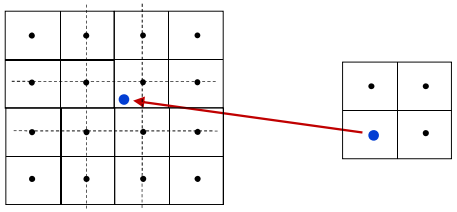


Bilinear Interpolation

Bicubic Interpolation

- The assign value is a weighted sum of the 4x4 nearest pixels:

$$v(\Delta x, \Delta y) = \sum_{i,j=0}^{3} a_{ij} \Delta x^i \Delta y^j$$

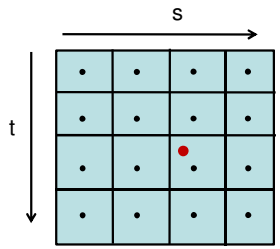


How can we find the right coefficients?

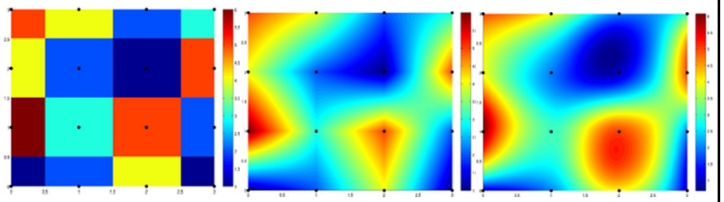
- Denote the pixel values V_{pq} {p,q=0..3}
- The unknown coefficients are a_{ij} {i,j=0..3}

$$v_{pq} = \sum_{i,j=0}^{3} a_{ij} \Delta x^i \Delta y^j \quad \text{for } p,q = \{0..3\}, \quad \Delta x, \Delta y \in [-1,2]$$

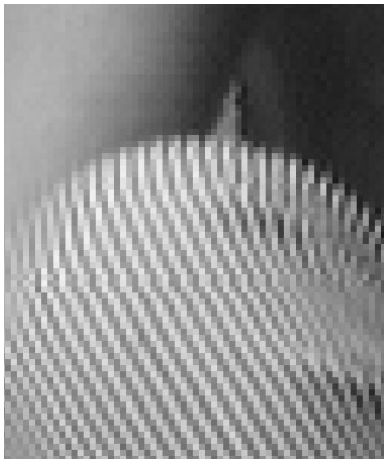
- We have a linear system of 16 equations with 16 coefficients.
- The pixel's boundaries are C_1 continuous (continuous derivatives across boundaries).



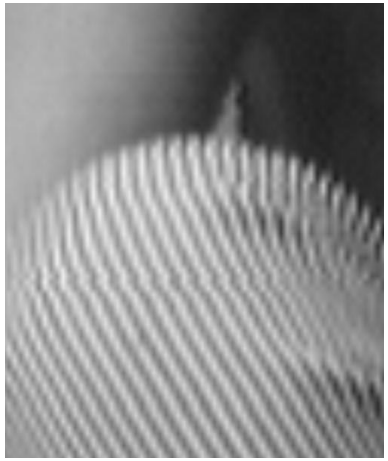
21



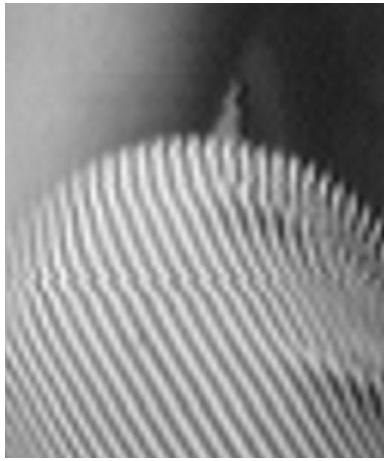
N.N Bilinear Bicubic



N.N



Bilinear



Bicubic

Applying the Transformation

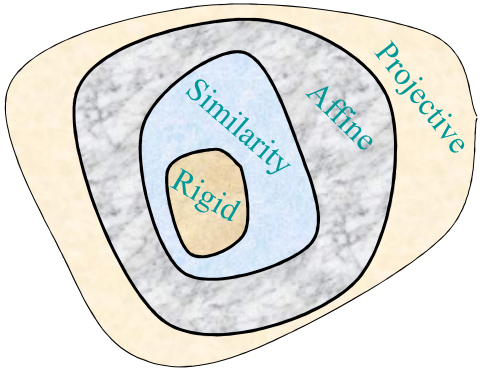
```
T = ……  % 2x2 transformation matrix
[r,c] = size(img)

% create array of destination x,y coordinates
[X,Y]=meshgrid(1:c,1:r);

% calculate source coordinates
sourceCoor = inv(T) * [X(:) Y(:) ]' ;

% calculate nearest neighbor interpolation
Xs = round(sourceCoor(1,:));
Ys = round(sourceCoor(2,:));

indx=find(Xs<1 | Xs>r);   %out of range pixels
Xs(indx)=1; Ys(indx)=1;

indy=find(Ys<1 | Ys>c);   %out of range pixels
Xs(indy)=1; Ys(indy)=1;

% calculate new image
newImage = img((Xs-1).*r+Ys);
newImage(indx)=0; newImage(indx)=0;
newImage = reshape(newImage,r,c);
```

Types of linear 2D transformations

- Rigid (Euclidean) transformation:
  - Translation + Rotation (distance preserving).
- Similarity transformation:
  - Translation + Rotation + Uniform Scale (angle preserving).
- Affine transformation:
  - Translation + Rotation + Scale + Shear (parallelism preserving).
- Projective transformation
  - Cross-ratio preserving
- All above transformations are groups where Rigid ⊂ Similarity ⊂ Affine ⊂ Projective

Types of linear 2D transformations

- All above transformations are groups where Rigid ⊂ Similarity ⊂ Affine ⊂ Projective



Matrix Notation

- Every location (x,y) is treated as a column vector:

$$\begin{bmatrix} x \\ y \end{bmatrix}$$

- Coordinate transformation is obtained by multiplying with a 2x2 matrix?

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}=\begin{bmatrix} ax+by \\ cx+dy \end{bmatrix}$$

Matrix Notation - Scale

- Scale(a,b): (x,y) ⟶ (ax,by)

$$\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}=\begin{bmatrix} ax \\ by \end{bmatrix}$$

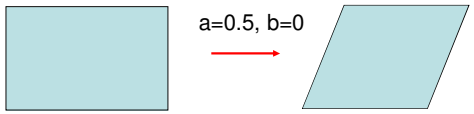
- If *a* or *b* are negative, we get reflection.
- Inverse: $S^{-1}(a,b)=S(1/a,1/b)$

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

Matrix Notation - Shear

- Shear(a,b): (x,y) ⟶ (x+ay,y+bx)

$$\begin{bmatrix} 1 & a \\ b & 1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}=\begin{bmatrix} x+ay \\ y+bx \end{bmatrix}$$



Matrix Notation - Rotation

- Rotate(θ):

(x,y) ⟶ ($x\cos\theta+y\sin\theta$, $-x\sin\theta + y\cos\theta$)

$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}=\begin{bmatrix} x\cos\theta+y\sin\theta \\ -x\sin\theta+y\cos\theta \end{bmatrix}$$

- Inverse: $R^{-1}(\theta)=R^{T}(\theta)=R(-\theta)$

Matrix Notation - Translation

- Translation(a,b): $\begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x+a \\ y+b \end{bmatrix}$
- Cannot represent translation using 2x2 matrices.
- Inverse: $\begin{bmatrix} x' \\ y' \end{bmatrix} \rightarrow \begin{bmatrix} x'-a \\ y'-b \end{bmatrix}$

Homogeneous Coordinates

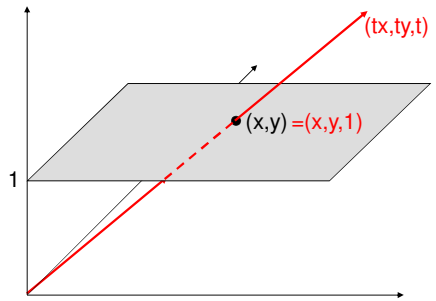
- Homogeneous Coordinates is a mapping from R^n to R^{n+1}:

$$(x, y) \rightarrow (X, Y, W) \equiv (tx, ty, t)$$

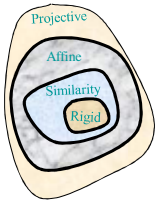
- Note: (tx,ty,t) all correspond to the same non-homogeneous point (x,y). E.g. (2,3,1)≡(6,9,3) ≡(4,6,2).
- Inverse mapping:

$$(X, Y, W) \rightarrow \left(\frac{X}{W}, \frac{Y}{W}\right) = (x, y)$$

Homogeneous Coordinates



Some 2D Transformations

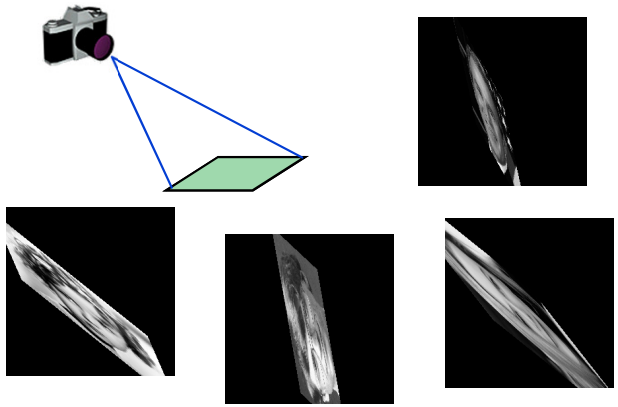


- Translation : $\begin{bmatrix} X' \\ Y' \\ W' \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x+t_x \\ y+t_y \\ 1 \end{bmatrix}$
- Affine transformation: $\begin{bmatrix} X' \\ Y' \\ W' \end{bmatrix} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$
- Projective transformation: $\begin{bmatrix} X' \\ Y' \\ W' \end{bmatrix} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ e & f & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

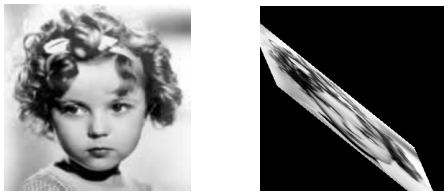
Hierarchy of Linear 2D Transformations

Group	Matrix	Distortion	Invariant properties
Projective 8 dof	$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$		Concurrency, collinearity, **order of contact**: intersection (1 pt contact); tangency (2 pt contact); inflections (3 pt contact with line); tangent discontinuities and cusps. cross ratio (ratio of ratio of lengths).
Affine 6 dof	$\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Parallelism, ratio of areas, ratio of lengths on collinear or parallel lines (e.g. midpoints), linear combinations of vectors (e.g. centroids). The line at infinity, $\mathbf{l}_\infty$.
Similarity 4 dof	$\begin{bmatrix} sr_{11} & sr_{12} & t_x \\ sr_{21} & sr_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Ratio of lengths, angle. The circular points, $\mathbf{I}, \mathbf{J}$ (see section 1.7.3).
Euclidean 3 dof	$\begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Length, area

Global Transformations – Image Rectification



Global Transformations – Global Warping



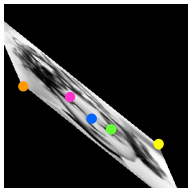
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = A \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Global Transformations – Global Warping

$$\begin{bmatrix} q_1^x & q_2^x & q_3^x \dots \\ q_1^y & q_2^y & q_3^y \dots \\ 1 & 1 & 1 \end{bmatrix} = A \begin{bmatrix} p_1^x & p_2^x & p_3^x \dots \\ p_1^y & p_2^y & p_3^Y \dots \\ 1 & 1 & 1 \end{bmatrix}$$

Global Transformations – Global Warping

Inverse Mapping:



points p_i ← match points q_i

$$\begin{bmatrix} p_1^x & p_2^x & p_3^x \dots \\ p_1^y & p_2^y & p_3^Y \dots \\ 1 & 1 & 1 \end{bmatrix} = A^{-1} \begin{bmatrix} q_1^x & q_2^x & q_3^x \dots \\ q_1^y & q_2^y & q_3^y \dots \\ 1 & 1 & 1 \end{bmatrix}$$

Global Transformations – Global Warping

Solve for A^{-1} in terms of the least mean square.
i.e. find A^{-1} which minimizes:

$$\sum_i \| p_i - A^{-1} q_i \|_2$$

Global Transformations – Global Warping

solution:

$$A^{-1} = \begin{bmatrix} p_1^x & p_2^x & p_3^x \dots \\ p_1^y & p_2^y & p_3^Y \dots \\ 1 & 1 & 1 \end{bmatrix} pinv \left(\begin{bmatrix} q_1^x & q_2^x & q_3^x \dots \\ q_1^y & q_2^y & q_3^y \dots \\ 1 & 1 & 1 \end{bmatrix} \right)$$

$$pinv(X) = X^T \left(X X^T \right)^{-1}$$

Global Transformations – Global Warping

Alternative representation:

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

Rearrange:

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

Global Transformations – Global Warping

$$\begin{pmatrix} p_x^1 \\ p_y^1 \end{pmatrix} = \begin{pmatrix} q_x^1 & q_y^1 & 0 & 0 & 1 & 0 \\ 0 & 0 & q_x^1 & q_y^1 & 0 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix}$$

Global Transformations – Global Warping

$$\begin{pmatrix} p_x^1 \\ p_y^1 \\ p_x^2 \\ p_y^2 \\ p_x^3 \\ p_y^3 \end{pmatrix} = \begin{pmatrix} q_x^1 & q_y^1 & 0 & 0 & 1 & 0 \\ 0 & 0 & q_x^1 & q_y^1 & 0 & 1 \\ q_x^2 & q_y^2 & 0 & 0 & 1 & 0 \\ 0 & 0 & q_x^2 & q_y^2 & 0 & 1 \\ q_x^3 & q_y^3 & 0 & 0 & 1 & 0 \\ 0 & 0 & q_x^3 & q_y^3 & 0 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix}$$

Global Transformations – Global Warping

solution:

$$\begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix} = pinv' \left(\begin{pmatrix} q_x^1 & q_y^1 & 0 & 0 & 1 & 0 \\ 0 & 0 & q_x^1 & q_y^1 & 0 & 1 \\ q_x^2 & q_y^2 & 0 & 0 & 1 & 0 \\ 0 & 0 & q_x^2 & q_y^2 & 0 & 1 \\ q_x^3 & q_y^3 & 0 & 0 & 1 & 0 \\ 0 & 0 & q_x^3 & q_y^3 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x^1 \\ p_y^1 \\ p_x^2 \\ p_y^2 \\ p_x^3 \\ p_y^3 \end{pmatrix} \right)$$

$$pinv'(Q) = \left(Q^T Q\right)^{-1} Q^T$$

Global Transformations – Global Warping

What about Projective Transformations?

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

Homogeneity must be preserved!

$$x' = \frac{ax + by + e}{gx + hy + 1} \quad ; \quad y' = \frac{cx + dy + f}{gx + hy + 1}$$

Global Transformations – Global Warping

What about Projective Transformations?

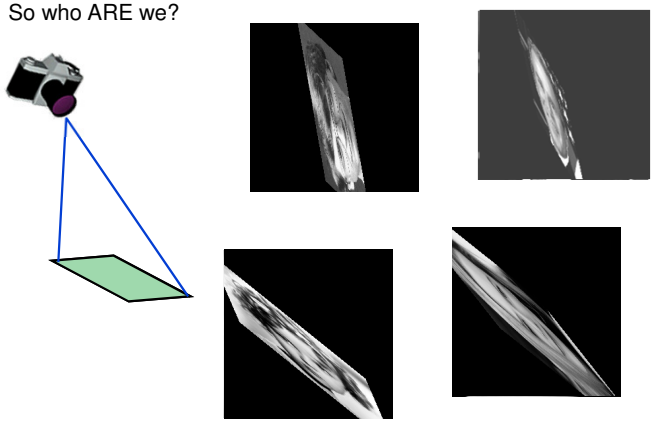
$$x' = \frac{ax + by + e}{gx + hy + 1} \quad ; \quad y' = \frac{cx + dy + f}{gx + hy + 1}$$

$$x' = \begin{pmatrix} x & y & 0 & 0 & 1 & 0 & -xx' & -yx' \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{pmatrix}$$

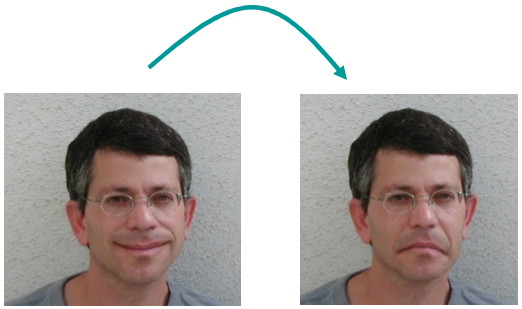
And similarly for y'

Global Transformations – Image Rectification

So who ARE we?

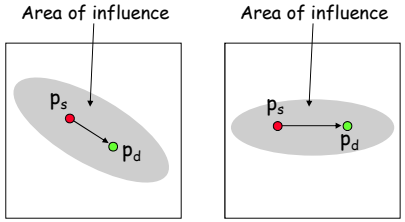


Local Transformations – Image Warping



Demo
alex

Local Transformations – Image Warping



p_s = source point
p_d = destination point

Local Transformations – Image Warping

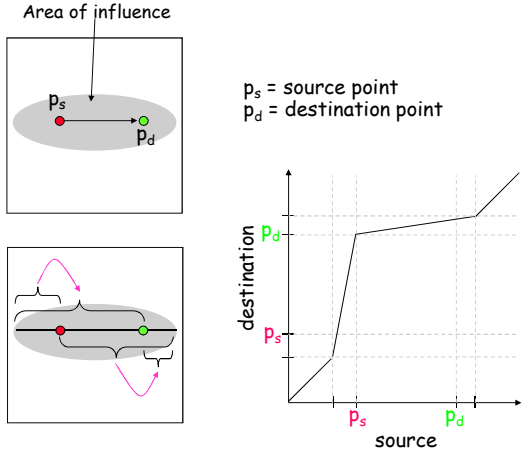
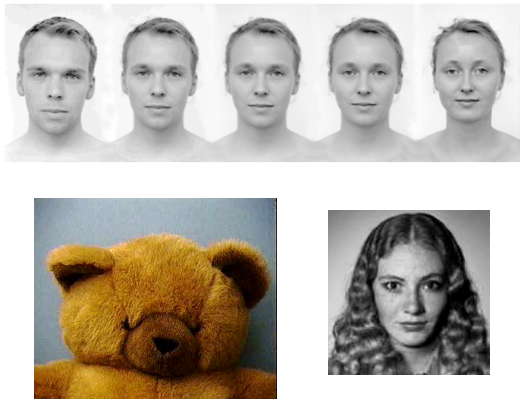
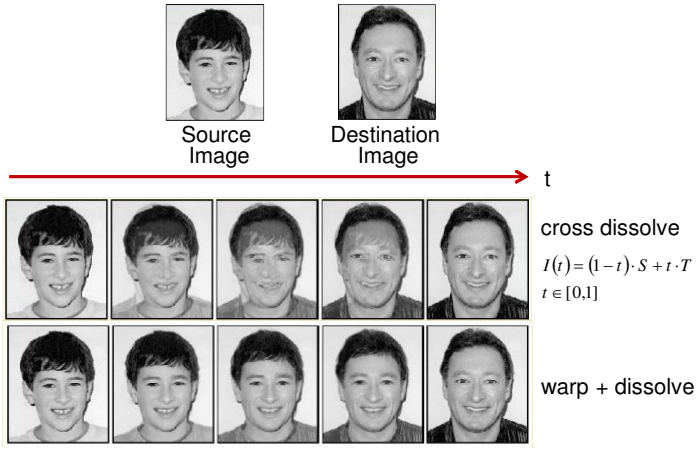


Image Morphing (Image Metamorphosis)



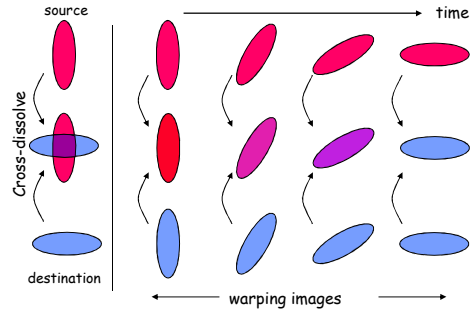
Demo bw

Cross Dissolve (pixel operations)



Warping + Cross Dissolve

- Warp source image towards intermediate image.
- Warp destination image towards intermediate image.
- Cross-dissolve the two images by taking the weighted average at each pixel.



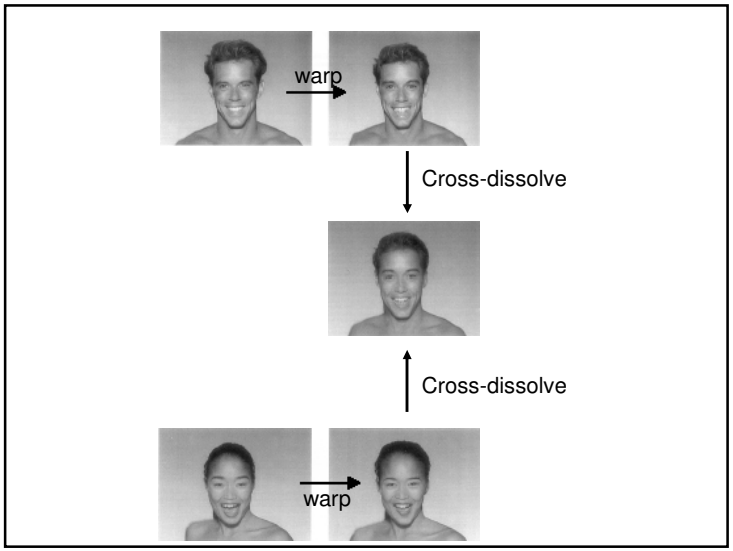
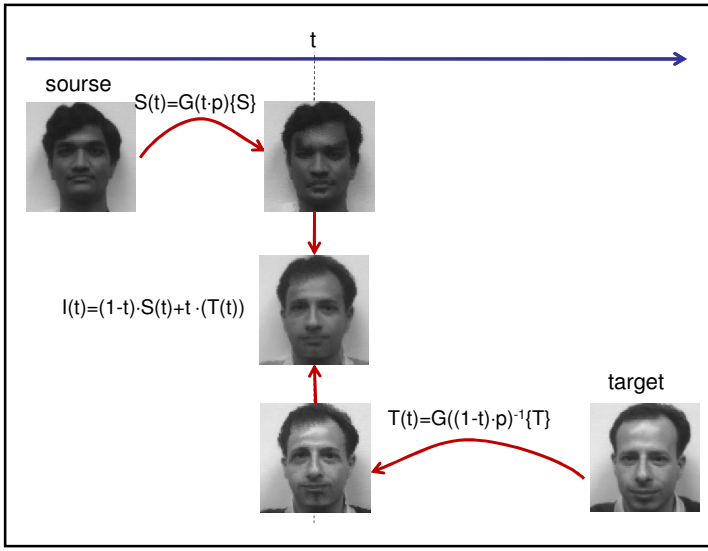


Image Metamorphosis

- Let S,T be the source and the target images
- Let G(p) be the transformation from S towards T, where G(0)=I (the identity)
- Let $t \in [0,1]$ the time step to be synthesized

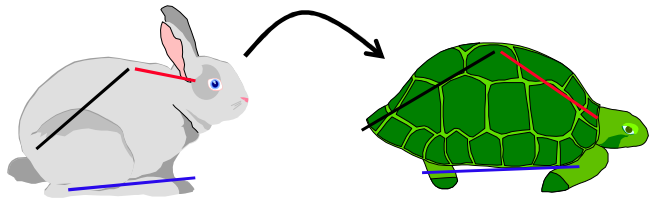
Algorithm:

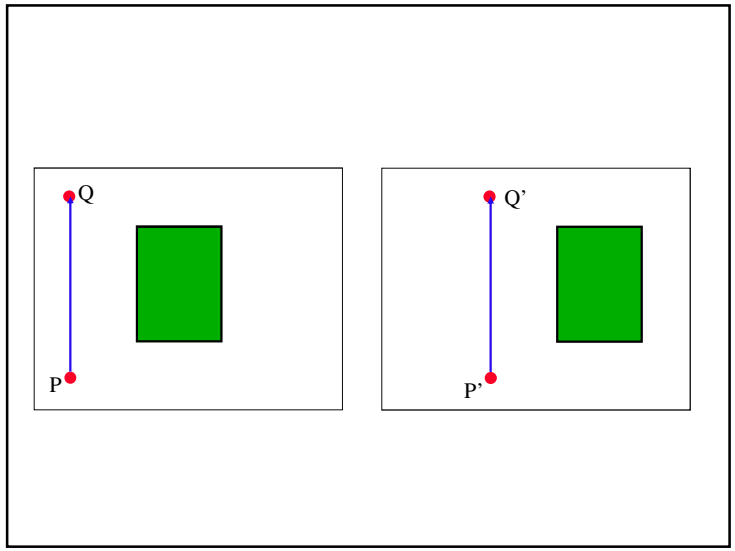
1. Warp S towards T: $S(t) = G(t \cdot p)\{S\}$
2. Warp T toward S: $T(t) = G((1-t) \cdot p)^{-1}\{T\}$
3. Cross dissolve: $I(t) = (1-t) \cdot S(t) + t \cdot T(t)$

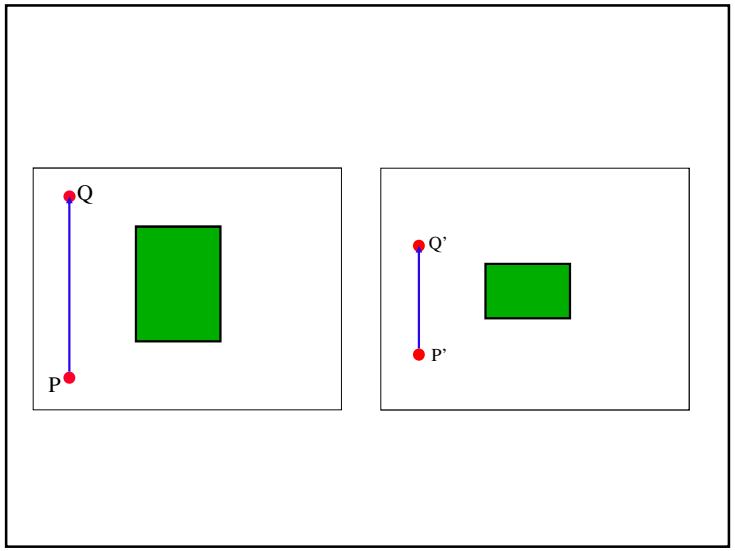


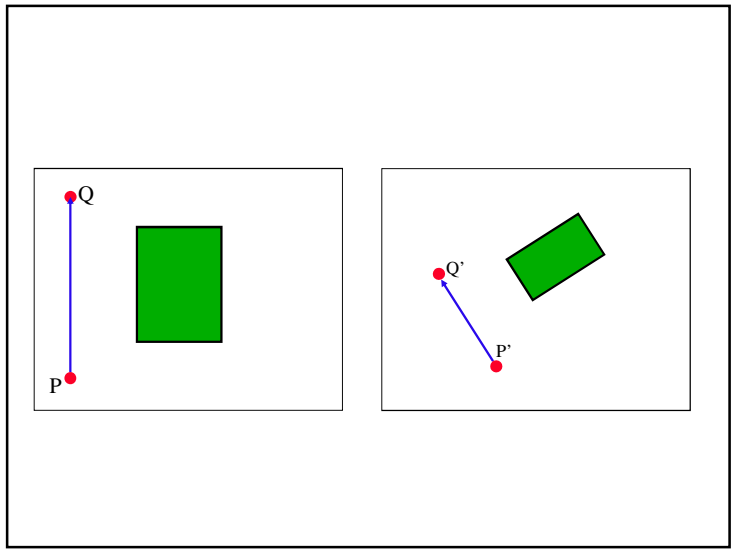
Feature Based Morphing

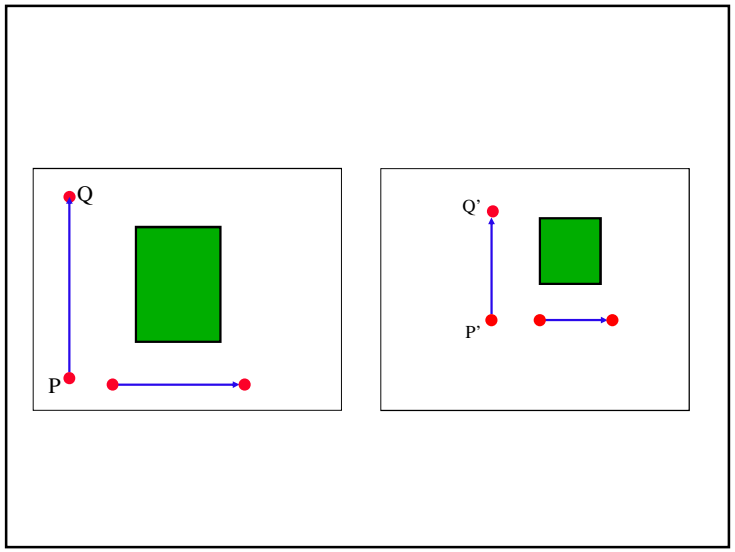
- Morph one shape into another shape
- Use local features to define the geometric warping

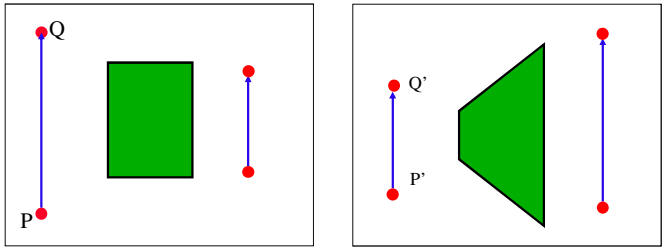


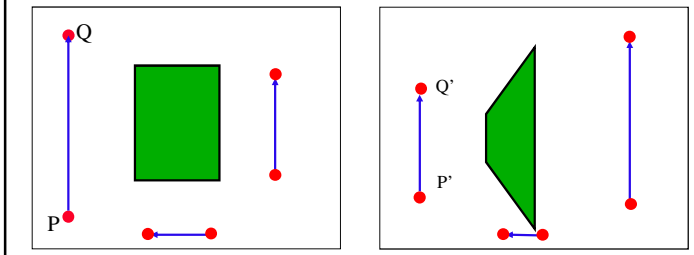

Q
P
Q'
P'


Q
P
Q'
P'

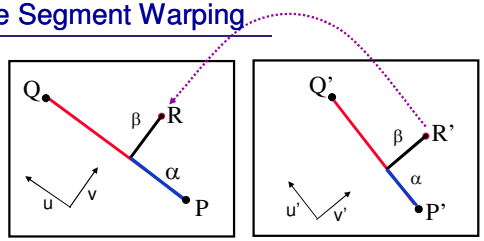

Q
P
Q'
P'


Q
P
Q'
P'





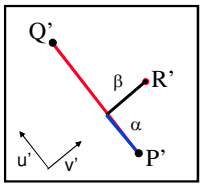
One Segment Warping



Source Image Dest Image

- $\alpha \in [0,1]$ is the **relative** position along the segment (P',Q').
- β is the **actual** perpendicular distance to the segment.
- **(u',v')** is the local coordinates of the segment (P',Q'):
  - **u'** is a unit vector parallel to Q'-P'
  - **v'** is the unit vector perpendicular to Q'-P'

$$u' = \frac{(Q'-P')}{\|Q'-P'\|} \qquad v' = \perp u' = \begin{pmatrix} u'_y \\ -u'_x \end{pmatrix}$$

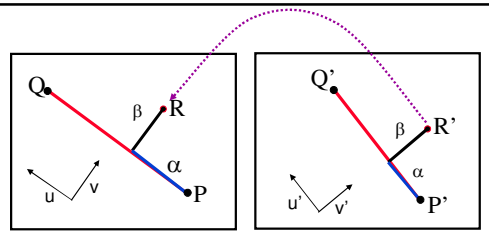


- The point R' is mapped into (α,β) :

$$\alpha = \frac{(R'-P')\cdot u'}{\|Q'-P'\|} \quad ; \quad \beta = (R'-P')\cdot v'$$

where

$$R' = P' + \alpha \|Q'-P'\| u' + \beta v'$$



Source Image Dest Image

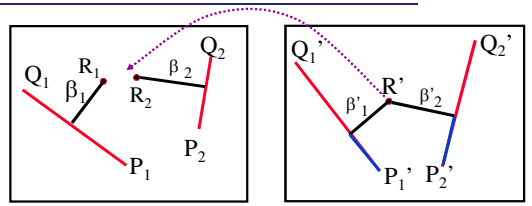
Inverse Mapping:

$$R(\alpha,\beta) = P + \alpha\|Q-P\|u + \beta v$$

where **(u,v)** is the local coordinates of the segment (P,Q):

$$u = \frac{(Q-P)}{\|Q-P\|} \qquad v = \perp u = \begin{pmatrix} u_y \\ -u_x \end{pmatrix}$$

Multiple Segment Warping



- In multiple segment warping the point R' is influenced by multiple segments.
- The influence **strength** of each segments is proportional to:
  - Segment length
  - The distance from the point R'

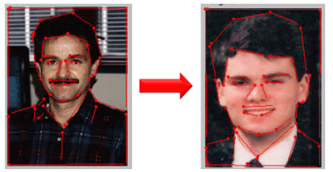
- The influence of each segments is:

$$W_i = \left(\frac{|Q_i - P_i|^p}{a + \beta_i} \right)^b$$

- The value $p \in [0,1]$ controls the influence of the line length.
- The value *a* is a small number avoiding division by zero.
- The value *b* determines how the relative weight diminish as the β increases
- The final mapping is:

$$R = \frac{\sum_k W_k R_k}{\sum_k W_k}$$

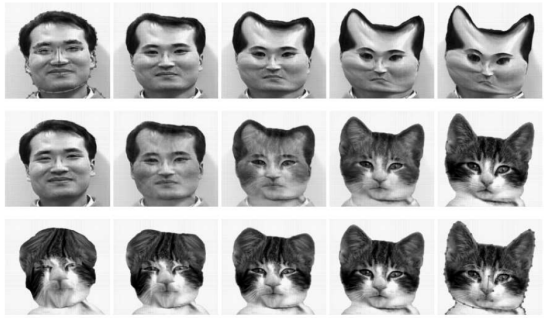
Example:



Example images from:
http://www.cc.gatech.edu/classes/AY2001/cs4451_spring/projects/Seven/

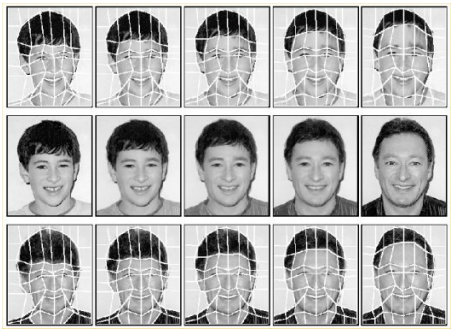
For more details see:
Thaddeus Beier & Shawn Neely / Feature-Based Image Metamorphosis Siggraph '92
http://www.hammerhead.com/thad/morph.html

Another Example:



73

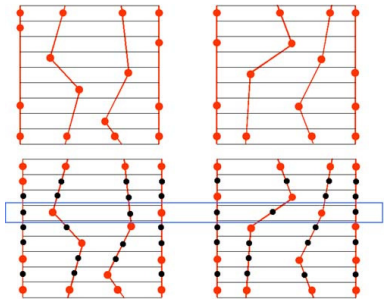
Mesh Warping



From:
http://www.cs.utk.edu/~huangj/CS594F01/imageMorph.ppt

2-Pass Mesh Warping Algorithm

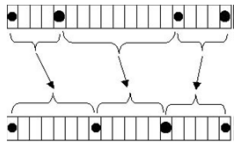
The **first pass** warps the rows of the image:



For each column of the mesh determine the x-coordinates at which the mesh column crosses each image row.

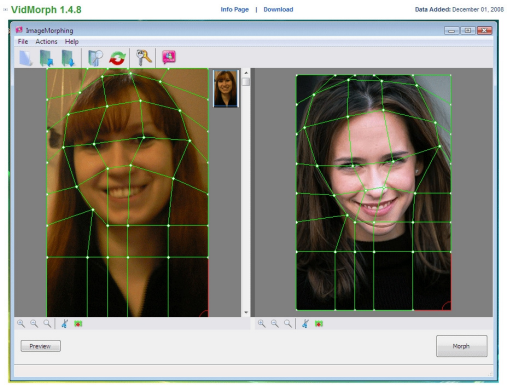
2-Pass Mesh Warping Algorithm

Then, each row of the image is warped individually by linearly interpolating each segment between the x-coordinates defined by the source mesh to the size of the corresponding segment defined by the x-coordinates of the destination Mesh.



The **second pass** performs the exact same procedure on the columns of the image by interpolating the y-coordinates of the meshes.

VidMorph



Fun Morph