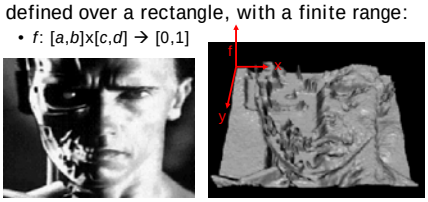
Image warping

Li Zhang
CS559

Slides stolen from Prof Yungyu Chuang
http://www.csie.ntu.edu.tw/~cyy/courses/vfx/07spring/overview/

What is an image

- We can think of an **image** as a function, f: $R^2 \rightarrow R$:
  - $f(x, y)$ gives the **intensity** at position (x, y)
  - defined over a rectangle, with a finite range:
    - f: $[a,b]x[c,d] \rightarrow [0,1]$



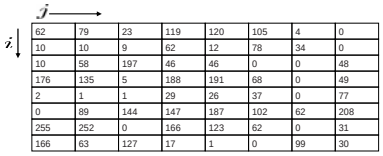
- A color image

$$f(x,y) = \begin{bmatrix} r(x,y) \\ g(x,y) \\ b(x,y) \end{bmatrix}$$

A digital image

- We usually operate on **digital** (**discrete**) images:
  - **Sample** the 2D space on a regular grid
  - **Quantize** each sample (round to nearest integer)
- If our samples are D apart, we can write this as:

  $f[i, j] = \text{Quantize}\{ f(i\ D, j\ D) \}$
- The image can now be represented as a matrix of integer values



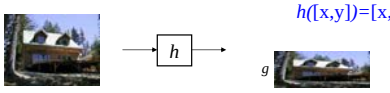
62	79	23	119	120	105	4	0
10	10	9	62	12	78	34	0
10	58	197	46	46	0	0	48
176	135	5	188	191	68	0	49
2	1	1	29	26	37	0	77
0	89	144	147	187	102	62	208
255	252	0	166	123	62	0	31
166	63	127	17	1	0	99	30

Image warping

Change pixels locations to create a new image:

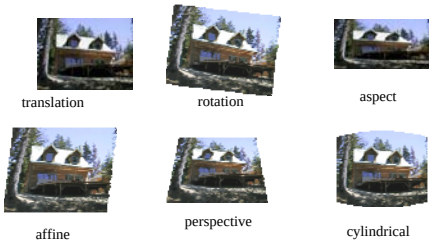
$f(x) = g(h(x))$

$h([x,y])=[x,y/2]$



Parametric (global) warping

Examples of parametric warps:

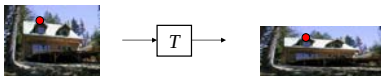


Nonparametric (local) warping

Original

Warped

Parametric (global) warping



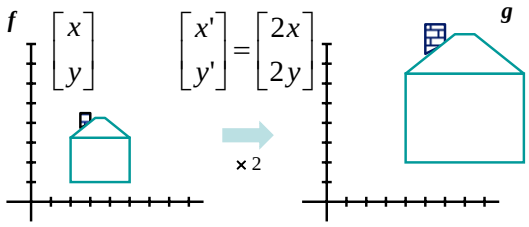
p = (x,y) **p'** = (x',y')

- Transformation T is a coordinate-changing machine: p' = *T*(p)
- What does it mean that *T* is global?
  - can be described by just a few numbers (parameters)
  - the parameters are the same for any point p
- Represent *T* as a matrix: p' = **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:



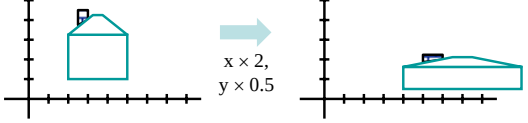
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$

Scaling

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

$$f\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = g\left(\begin{bmatrix} x' \\ y' \end{bmatrix}\right)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 2x \\ 0.5y \end{bmatrix}$$



Scaling

- Scaling operation:

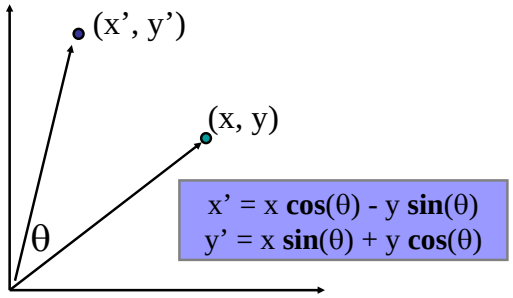
$$x' = ax$$
$$y' = by$$

- Or, in matrix form:

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

What's inverse of S?

2-D Rotation



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 to 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?

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

2D Scale around (0,0)?

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

2x2 Matrices

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

2D Rotate around (0,0)?

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

2D Shear?

$$x' = x + sh_x * y \qquad y' = sh_y * x + y \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & sh_x \\ sh_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2x2 Matrices

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

2D Mirror about Y axis?

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

2D Mirror over (0,0)?

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

All 2D Linear Transformations

- Linear transformations are combinations of …
  - Scale,
  - Rotation,
  - Shear, and
  - Mirror
- Properties of linear transformations:
  - Origin maps to origin
  - Lines map to lines
  - Parallel lines remain parallel
  - Ratios are preserved
  - Closed under composition

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

2x2 Matrices

- What types of transformations can not be represented with a 2x2 matrix?

2D Translation?

$$x' = x + t_x \qquad y' = y + t_y$$

NO!

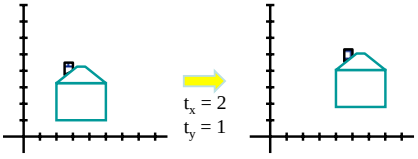
Only linear 2D transformations can be represented with a 2x2 matrix

Translation

- Example of translation

Homogeneous Coordinates

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



Affine Transformations

- Affine transformations are combinations of …
  - Linear transformations, and
  - Translations
- Properties of affine transformations:
  - Origin does not necessarily map to origin
  - Lines map to lines
  - Parallel lines remain parallel
  - Ratios are preserved
  - Closed under composition
  - Models change of basis

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

Projective Transformations

- Projective transformations …
  - Affine transformations, and
  - Projective warps
- Properties of projective transformations:
  - Origin does not necessarily map to origin
  - Lines map to lines
  - Parallel lines do not necessarily remain parallel
  - Ratios are not preserved
  - Closed under composition
  - Models change of basis

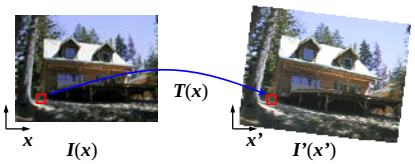
Very Useful In Texture Mapping!

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

$$\begin{bmatrix} x'/w' \\ y'/w' \end{bmatrix}$$

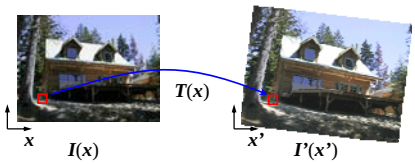
Image warping

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



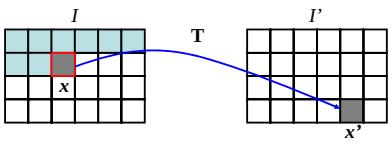
Forward warping

- Send each pixel $I(x)$ to its corresponding location $x' = T(x)$ in $I'(x')$



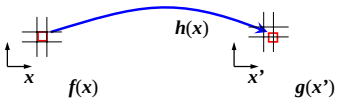
Forward warping

```
fwarp(I, I', T)
{
  for (y=0; y<I.height; y++)
    for (x=0; x<I.width; x++) {
      (x',y')=T(x,y);
      I'(x',y')=I(x,y);
    }
}
```



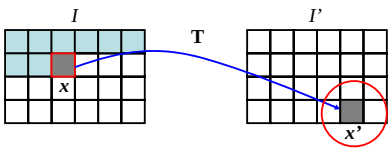
Forward warping

- Send each pixel $I(x)$ to its corresponding location $x' = T(x)$ in $I'(x')$
  - What if pixel lands "between" two pixels?
  - Will be there holes?
  - Answer: add "contribution" to several pixels, normalize later (*splatting*)



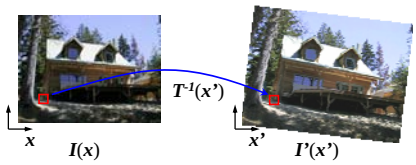
Forward warping

```
fwarp(I, I', T)
{
  for (y=0; y<I.height; y++)
    for (x=0; x<I.width; x++) {
      (x',y')=T(x,y);
      Splatting(I',x',y',I(x,y),kernel);
    }
}
```



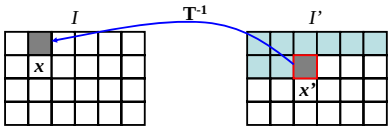
Inverse warping

- Get each pixel $I'(x')$ from its corresponding location $x = T^{-1}(x')$ in $I(x)$



Inverse warping

```
iwarp(I, I', T)
{
  for (y=0; y<I'.height; y++)
    for (x=0; x<I'.width; x++) {
      (x,y)=T-1(x',y');
      I'(x',y')=I(x,y);
    }
}
```

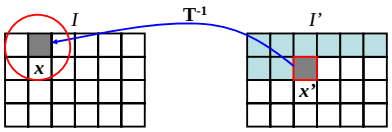


Inverse warping

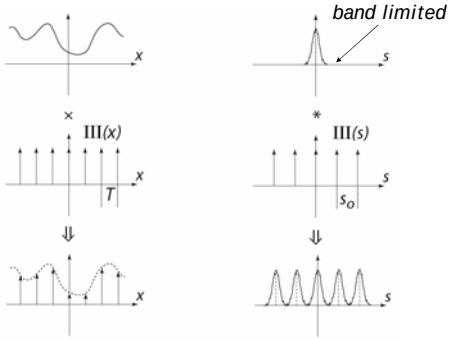
- Get each pixel $I'(x')$ from its corresponding location $x = T^{-1}(x')$ in $I(x)$
  - What if pixel comes from "between" two pixels?
  - Answer: *resample* color value from *interpolated* (*prefiltered*) source image

Inverse warping

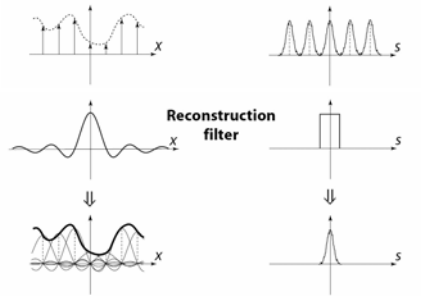
```
iwarp(I, I', T)
{
  for (y=0; y<I'.height; y++)
    for (x=0; x<I'.width; x++) {
      (x,y)=T-1(x',y');
      I'(x',y')=Reconstruct(I,x,y,kernel);
    }
}
```



Sampling



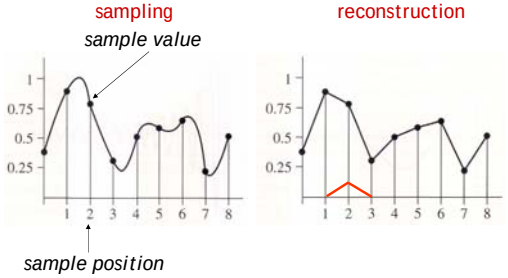
Reconstruction



The reconstructed function is obtained by interpolating among the samples in some manner

Reconstruction

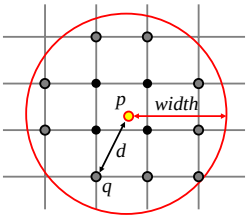
- Reconstruction generates an approximation to the original function. Error is called aliasing.



Reconstruction

- Computed weighted sum of pixel neighborhood; output is weighted average of input, where weights are normalized values of filter kernel k

$$f(p) = \frac{\sum_i k(q_i) f(q_i)}{\sum_i k(q_i)}$$



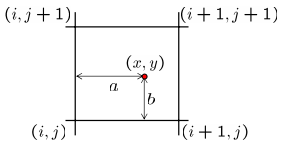

```
color=0;
weights=0;
for all q's dist < width
  d = dist(p, q);
  w = kernel(d);
  color += w*q.color;
  weights += w;
p.Color = color/weights;
```

Reconstruction (interpolation)

- Possible reconstruction filters (kernels):
  - nearest neighbor
  - bilinear
  - bicubic
  - sinc (optimal reconstruction)

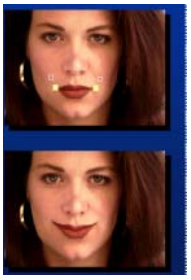
Bilinear interpolation (triangle filter)

- A simple method for resampling images



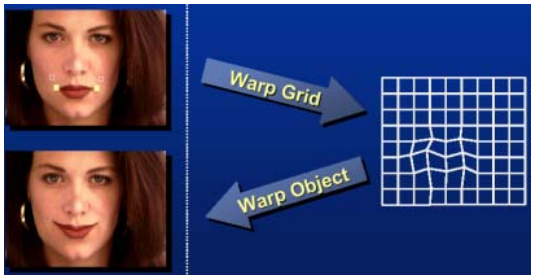
$$\begin{aligned} f(x,y) = \; & (1-a)(1-b) \quad f[i,j] \\ & +a(1-b) \quad f[i+1,j] \\ & +ab \quad f[i+1,j+1] \\ & +(1-a)b \quad f[i,j+1] \end{aligned}$$

Non-parametric image warping



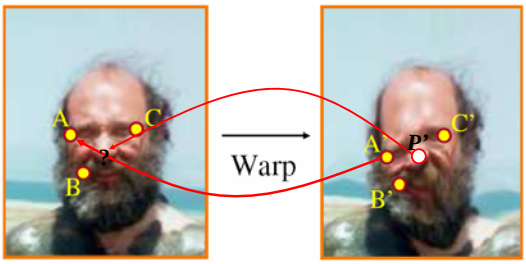
Non-parametric image warping

- Specify a more detailed warp function
- Splines, meshes, optical flow (per-pixel motion)



Non-parametric image warping

- Mappings implied by correspondences
- Inverse warping

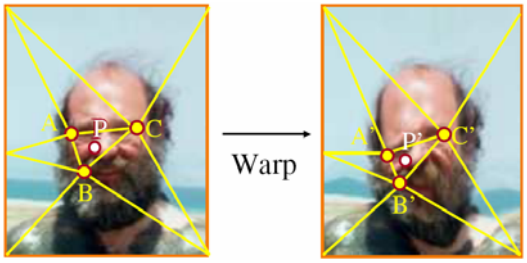


Non-parametric image warping

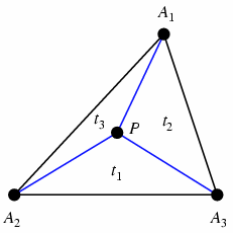
$$P = w_A A + w_B B + w_C C$$

$$P' = w_A A' + w_B B' + w_C C'$$

Barycentric coordinate



Barycentric coordinates



$$P = t_1 A_1 + t_2 A_2 + t_3 A_3$$

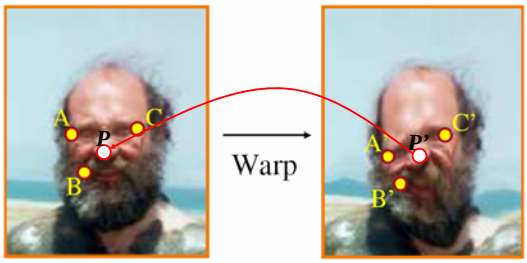
$$t_1 + t_2 + t_3 = 1$$

Non-parametric image warping

$$P = w_A A + w_B B + w_C C$$

$$P' = w_A A' + w_B B' + w_C C'$$

Barycentric coordinate



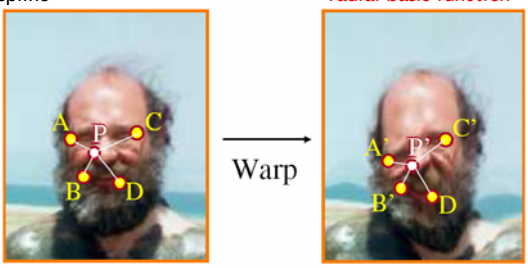
Non-parametric image warping

Gaussian $\rho(r) = e^{-\beta r^2}$

thin plate spline $\rho(r) = r^2 \log(r)$

$$\Delta P = \frac{1}{K} \sum_i k_{X_i}(P') \Delta X_i$$

radial basis function



Demo

- http://www.colonize.com/warp/warp04-2.php
- Warping is a useful operation for mosaics, video matching, view interpolation and so on.

Image morphing

Image morphing

- The goal is to synthesize a fluid transformation from one image to another.
- Cross dissolving is a common transition between cuts, but it is not good for morphing because of the ghosting effects.

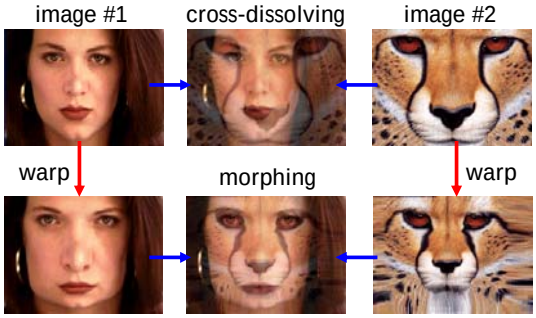
image #1 dissolving image #2

Image morphing

- Why ghosting?
- Morphing = warping + cross-dissolving

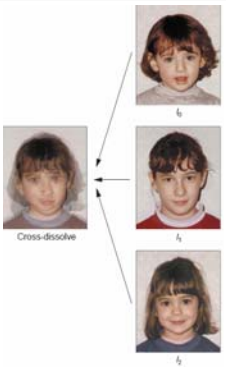
shape (geometric) color (photometric)

Image morphing

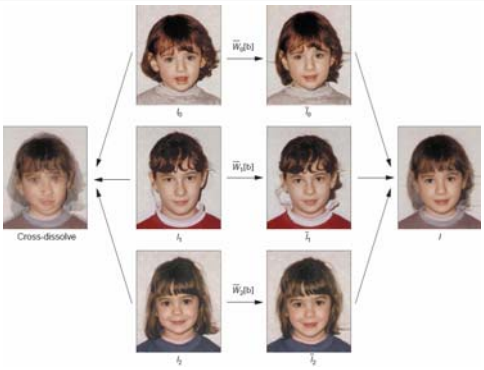


Morphing sequence

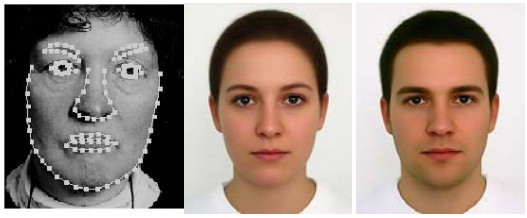
Multi-source morphing



Multi-source morphing



Face averaging by morphing



average faces

The average face

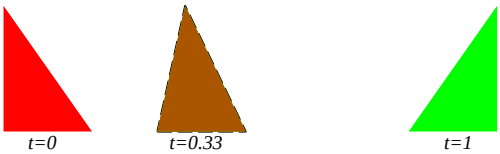
- http://www.uni-regensburg.de/Fakultaeten/phil_Fak_II/Psychologie/Psy_II/beautycheck/english/index.htm

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.

Image morphing

create a morphing sequence: for each time t

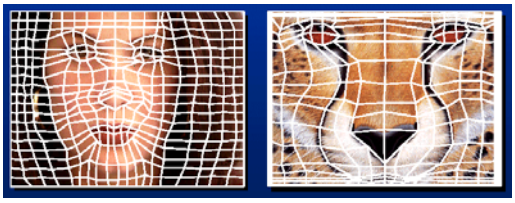
1. Create an intermediate warping field (by interpolation)
2. Warp both images towards it
3. Cross-dissolve the colors in the newly warped images



Warp specification (mesh warping)

- How can we specify the warp?
  1. Specify corresponding *spline control points*

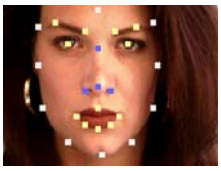
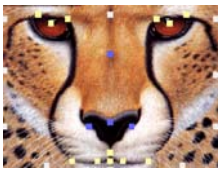
     interpolate to a complete warping function



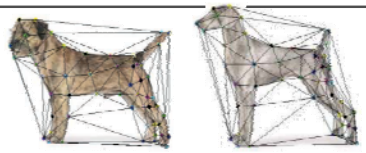
easy to implement, but may not be expressive enough

Warp specification

- How can we specify the warp
  2. Specify corresponding *points*
     - *interpolate* to a complete warping function

Solution: convert to mesh warping



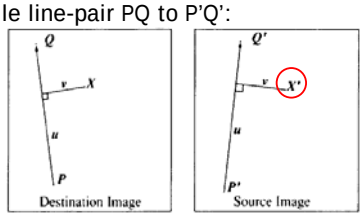
1. Define a triangular mesh over the points
   - Same mesh in both images!
   - Now we have triangle-to-triangle correspondences
2. Warp each triangle separately from source to destination
   - How do we warp a triangle?
   - 3 points = affine warp!
   - Just like texture mapping

Warp specification (field warping)

- How can we specify the warp?
  3. Specify corresponding *vectors*
     - *interpolate* to a complete warping function
     - The Beier & Neely Algorithm


Beier&Neely (SIGGRAPH 1992)

- Single line-pair PQ to P'Q':



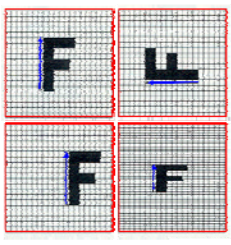
$$u = \frac{(X-P)\cdot(Q-P)}{\|Q-P\|^2} \quad (1)$$

$$v = \frac{(X-P)\cdot Perpendicular(Q-P)}{\|Q-P\|} \quad (2)$$

$$X' = P' + u\cdot(Q'-P') + \frac{v\cdot Perpendicular(Q'-P')}{\|Q'-P'\|} \quad (3)$$

Algorithm (single line-pair)

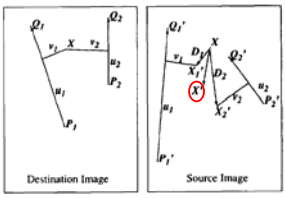
- For each X in the destination image:
  1. Find the corresponding u,v
  2. Find X' in the source image for that u,v
  3. destinationImage(X) = sourceImage(X')
- Examples:

Affine transformation

Multiple Lines

$$D_i = X_i' - X_i$$



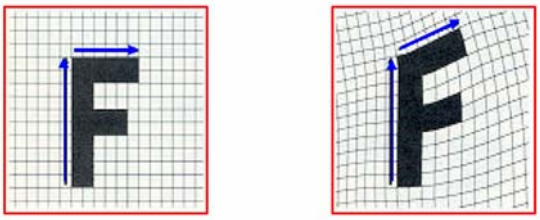
$$weight[i] = \left(\frac{length[i]^p}{a + dist[i]}\right)^b$$

length = length of the line segment,
dist = distance to line segment
The influence of a, p, b. The same as the average of X_i'

Full Algorithm

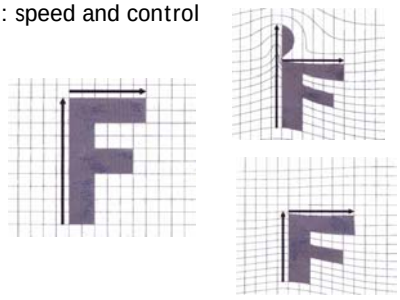
```
WarpImage(SourceImage,  L'[...], L[...])
begin
    foreach destination pixel X do
        XSum = (0,0)
        WeightSum = 0
        foreach line L[i] in destination do
            X'[i]= X transformed by (L[i],L'[i])
            weight[i] = weight assigned to X'[i]
            XSum = Xsum + X'[i] * weight[i]
            WeightSum += weight[i]
        end
        X' = XSum/WeightSum
        DestinationImage(X) = SourceImage(X')
    end
    return Destination
end
```

Resulting warp



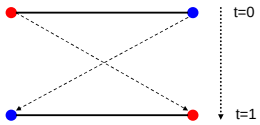
Comparison to mesh morphing

- Pros: more expressive
- Cons: speed and control



Warp interpolation

- How do we create an intermediate warp at time t?
  - linear interpolation for line end-points
  - But, a line rotating 180 degrees will become 0 length in the middle
  - One solution is to interpolate line mid-point and orientation angle



Animation

```
GenerateAnimation(Image0, L0[...],Image1, L1[...])
begin
    foreach intermediate frame time t do
        for i=1 to number of line-pairs do
            L[i] = line t-th of the way from L0[i] to L1[i].
        end
        Warp0 = WarpImage( Image0, L0[...], L[...])
        Warp1 = WarpImage( Image1, L1[...], L[...])
        foreach pixel p in FinalImage do
            FinalImage(p) = (1-t) Warp0(p) + t Warp1(p)
        end
    end
end
```

Animated sequences

- Specify keyframes and interpolate the lines for the inbetween frames
- Require a lot of tweaking

Results

Michael Jackson's MTV "Black or White"
http://www.michaeljackson.com/quicktime_blackorwhite.html

Problem with morphing

- So far, we have performed linear interpolation of feature point positions
- But what happens if we try to morph between two views of the same object?

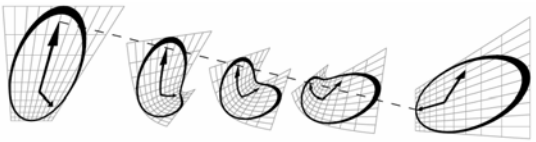


Figure 2: A Shape-Distorting Morph. Linearly interpolating two perspective views of a clock (far left and far right) causes a geometric bending effect in the in-between images. The dashed line shows the linear path of one feature during the course of the transformation. This example is indicative of the types of distortions that can arise with image morphing techniques.

View morphing

- Seitz & Dyer
  http://www.cs.washington.edu/homes/seitz/vmorph/vmorph.htm
- Interpolation consistent with 3D view interpolation

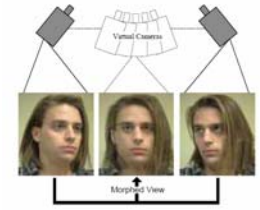


Figure 1: View morphing between two images of an object taken from two different viewpoints produces the illusion of physically moving a virtual camera.

Main trick

- Prewarp with a homography to "pre-align" images
- So that the two views are parallel
  - Because linear interpolation works when views are parallel

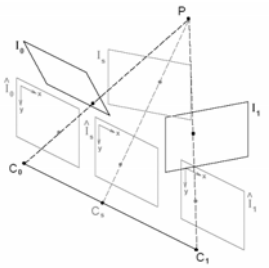


Figure 4: View Morphing in Three Steps. (1) Original images I_0 and I_1 are prewarped to form parallel views $\hat{I}_0$ and $\hat{I}_1$. (2) $\hat{I}_s$ is produced by morphing (interpolating) the prewarped images. (3) $\hat{I}_s$ is postwarped to form I_s.

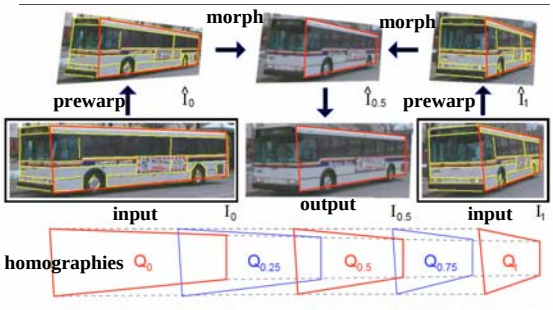


Figure 6: View Morphing Procedure: A set of features (yellow lines) is selected in original images I_0 and I_1. Using these features, the images are automatically prewarped to produce $\hat{I}_0$ and $\hat{I}_1$. The prewarped images are morphed to create a sequence of in-between images, the middle of which, $\hat{I}_{0.5}$, is shown at top-center. $\hat{I}_{0.5}$ is interactively postwarped by selecting a quadrilateral region (marked red) and specifying its desired configuration, $Q_{0.5}$, in $\hat{I}_{0.5}$. The postwarps for other in-between images are determined by interpolating the quadrilaterals (bottom).

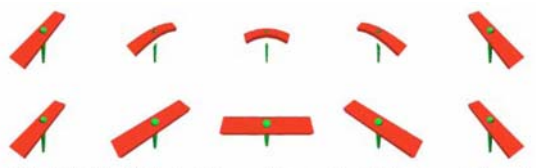


Figure 10: Image Morphing Versus View Morphing. Top: image morph between two views of a helicopter toy causes the in-between images to contract and bend. Bottom: view morph between the same two views results in a physically consistent morph. In this example the image morph also results in an extraneous hole between the blade and the stick. Holes can appear in view morphs as well.

DigiVFX

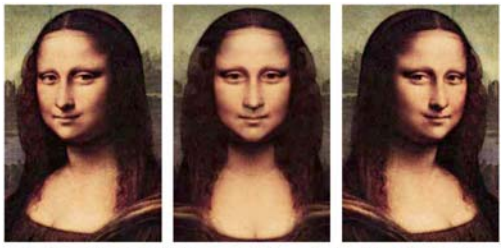


Figure 9: Mona Lisa View Morph. Morphed view (center) is halfway between original image (left) and it's reflection (right).

DigiVFX

Figure 7: Facial View Morphs. Top: morph between two views of the same person. Bottom: morph between views of two different people. In each case, view morphing captures the change in facial pose between original images I_0 and I_1, conveying a natural 3D rotation.