Image warping/morphing

Digital Visual Effects, Spring 2008

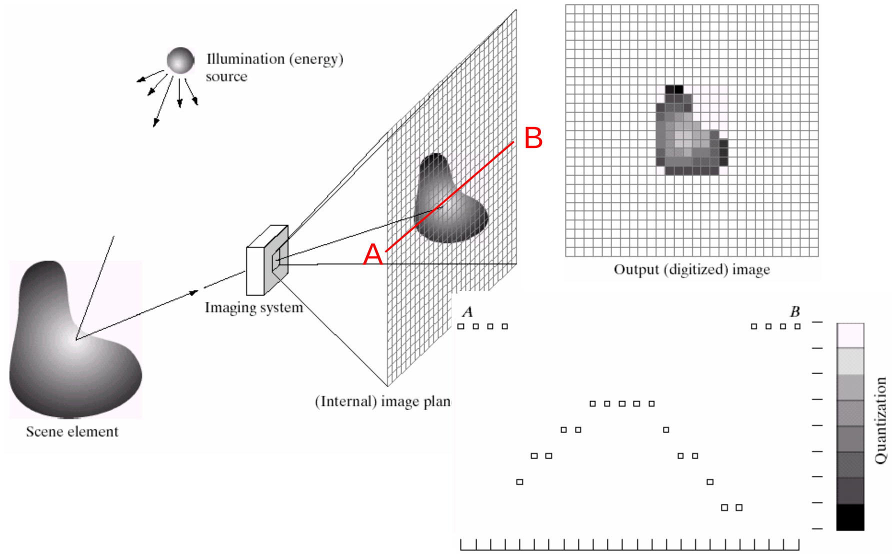
Yung-Yu Chuang

2008/3/11

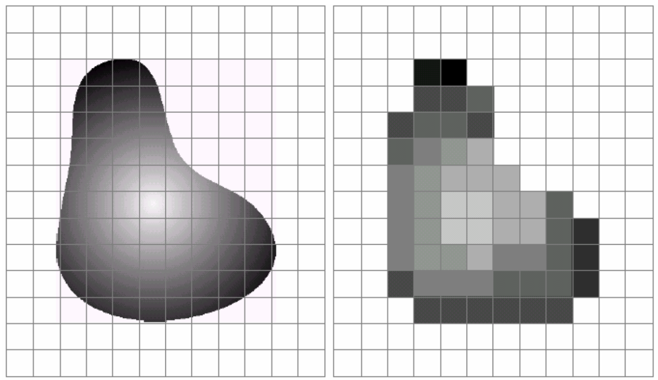
with slides by Richard Szeliski, Steve Seitz, Tom Funkhouser and Alexei Efros

Image warping

Image formation

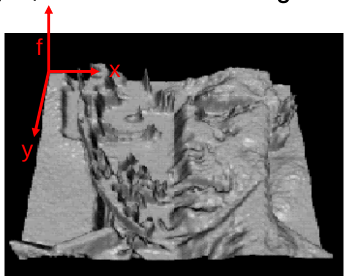


Sampling and quantization



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

$j \rightarrow$, $i \downarrow$

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

image filtering: change *range* of image

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

h(y)=0.5y+0.5

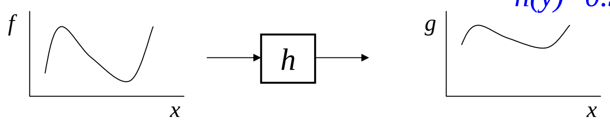


image warping: change *domain* of image

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

h(y)=2y

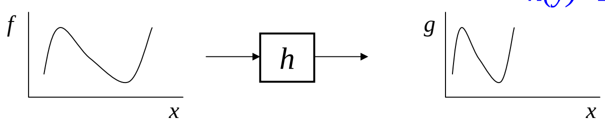


Image warping

image filtering: change *range* of image

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

h(y)=0.5y+0.5

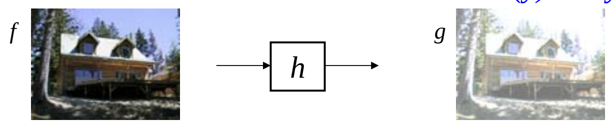
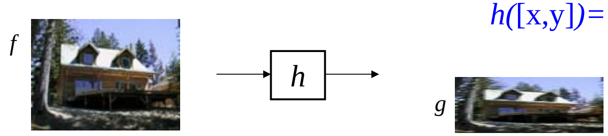


image warping: change *domain* of image

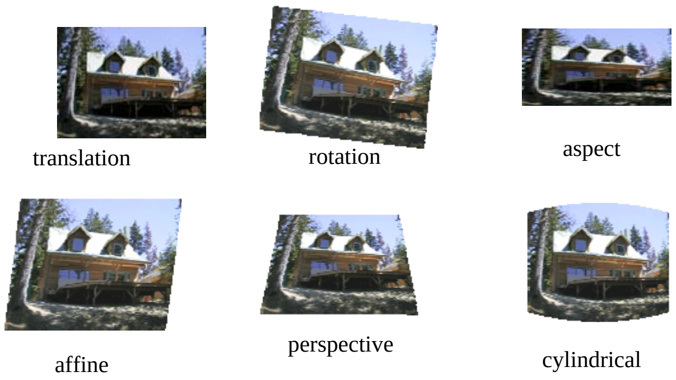
f(x) = g(h(x))

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

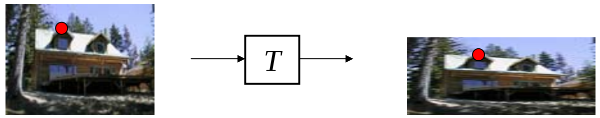


Parametric (global) warping

Examples of parametric warps:



Parametric (global) warping



p = (x,y) **p’** = (x’,y’)

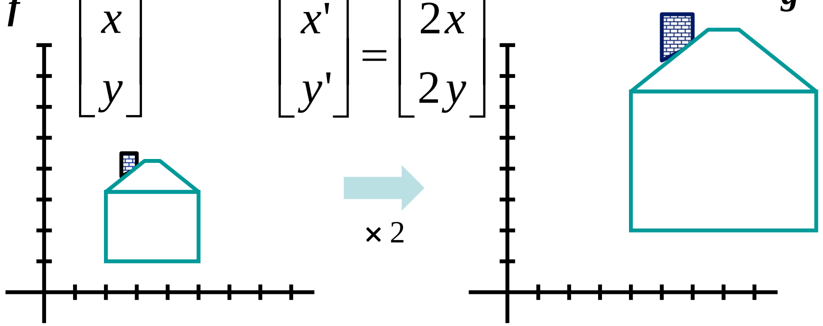
- Transformation T is a coordinate-changing machine: p’ = *T*(p)
- What does it mean that *T* is global?
  - Is the same for any point p
  - can be described by just a few numbers (parameters)
- Represent *T* as a matrix: p’ = **M***p

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

Scaling

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

$$f \quad \begin{bmatrix} x \\ y \end{bmatrix} \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 2x \\ 2y \end{bmatrix} \quad g$$

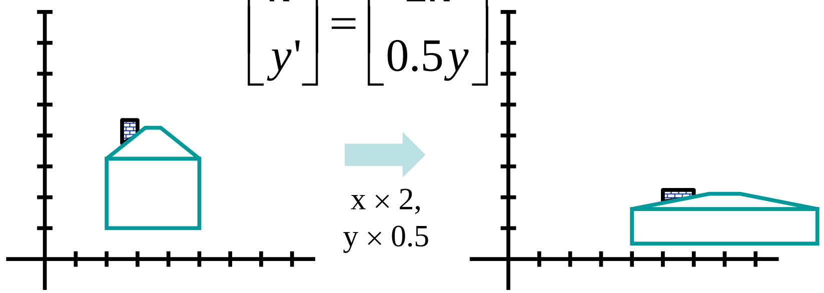


Scaling

- *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:

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

What's inverse of S?

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?

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

2D Scale around (0,0)?

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

2x2 Matrices

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

2D Rotate around (0,0)?

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

2D Shear?

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

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?

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

NO!

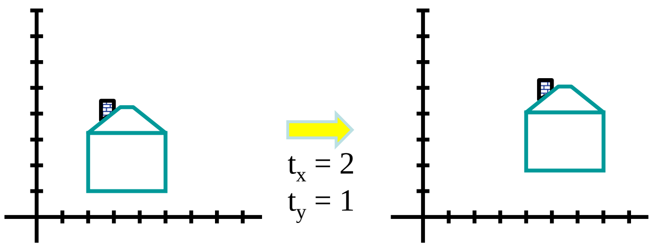
Only linear 2D transformations can be represented with a 2x2 matrix

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

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

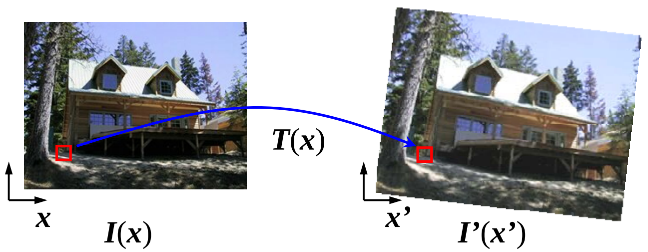
Projective Transformations

- 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

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

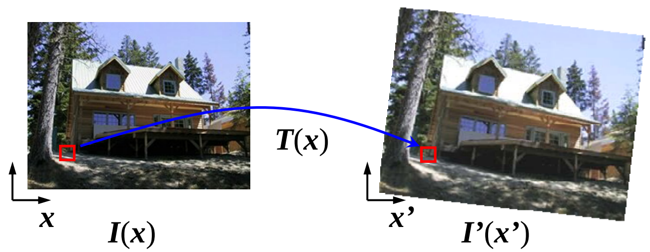
Image warping

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



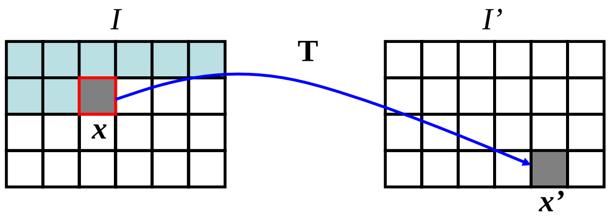
Forward warping

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



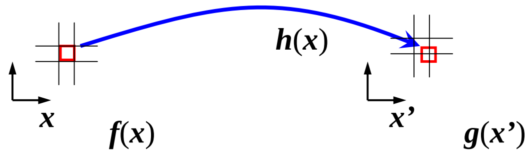
Forward warping

```
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);
    }
}
```



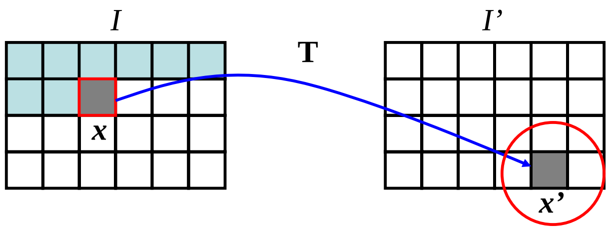
Forward warping

- 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*)



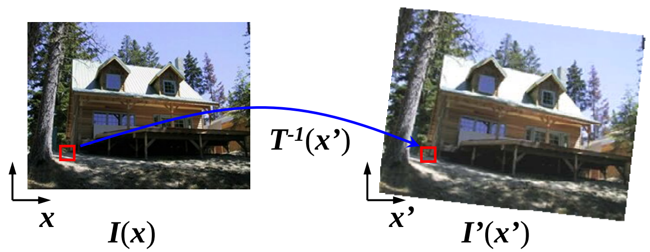
Forward warping

```
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);
    }
}
```



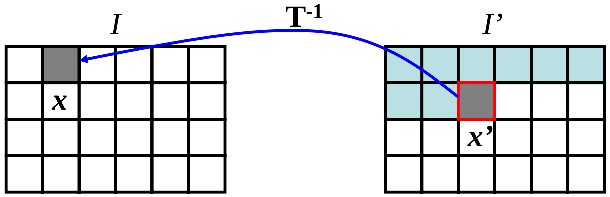
Inverse warping

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



Inverse warping

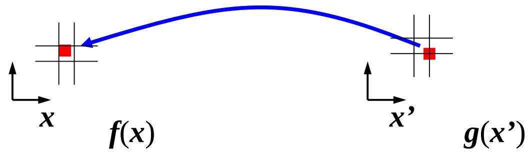
```
iwarp(I, I', T)
{
  for (y=0; y<I'.height; y++)
    for (x=0; x<I'.width; x++) {
      (x,y)=T-1(x',y');
      I'(x',y')=I(x,y);
    }
}
```



Inverse warping

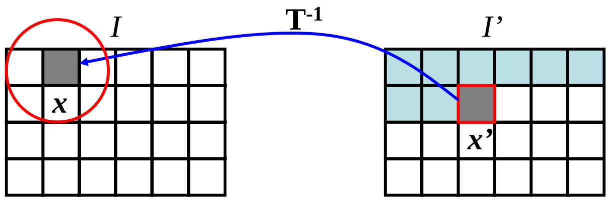
- Get each pixel $I'(x')$ from its corresponding location $x = T^{-1}(x')$ in $I(x)$

  - What if pixel comes from "between" two pixels?
  - Answer: *resample* color value from *interpolated* (*prefiltered*) source image

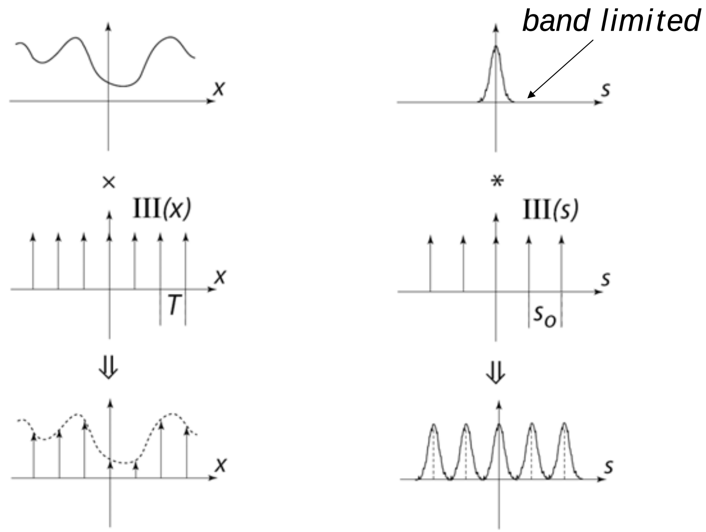


Inverse warping

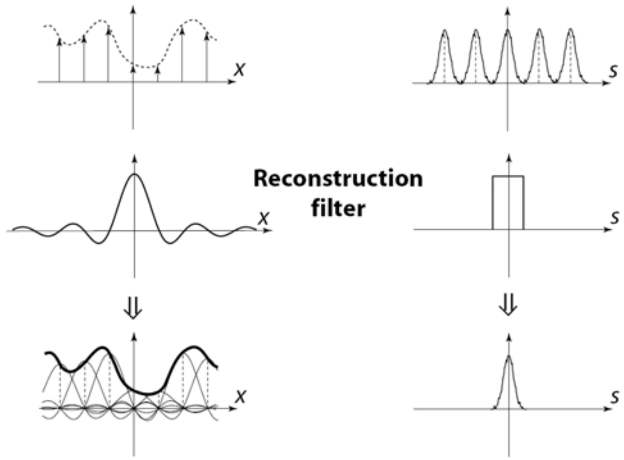
```
iwarp(I, I', T)
{
  for (y=0; y<I'.height; y++)
    for (x=0; x<I'.width; x++) {
      (x,y)=T-1(x',y');
      I'(x',y')=Reconstruct(I,x,y,kernel);
    }
}
```



Sampling



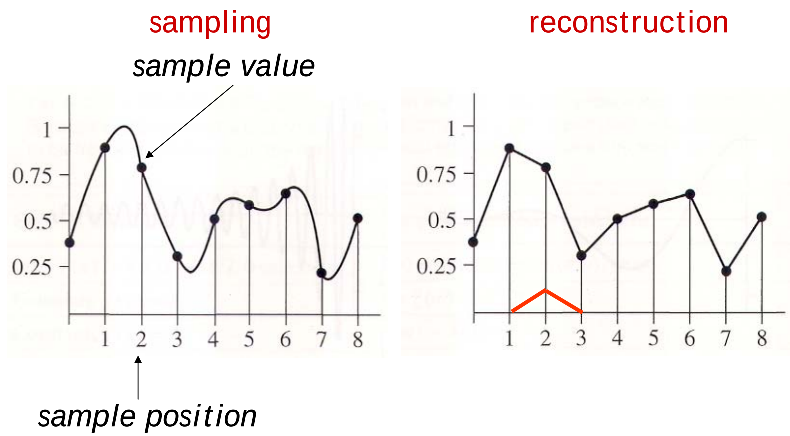
Reconstruction



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

Reconstruction

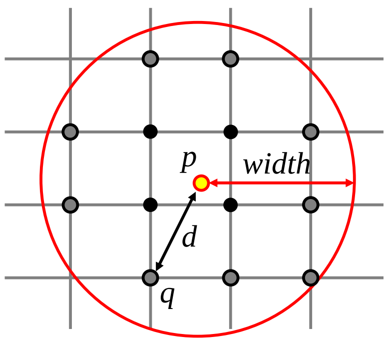
- Reconstruction generates an approximation to the original function. Error is called aliasing.



Reconstruction

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

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



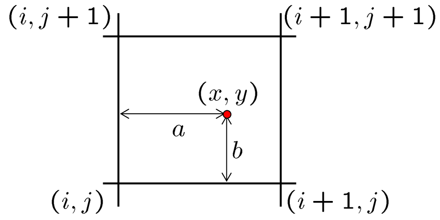

```
color=0;
weights=0;
for all q's dist < width
  d = dist(p, q);
  w = kernel(d);
  color += w*q.color;
  weights += w;
p.Color = color/weights;
```

Reconstruction (interpolation)

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

Bilinear interpolation (triangle filter)

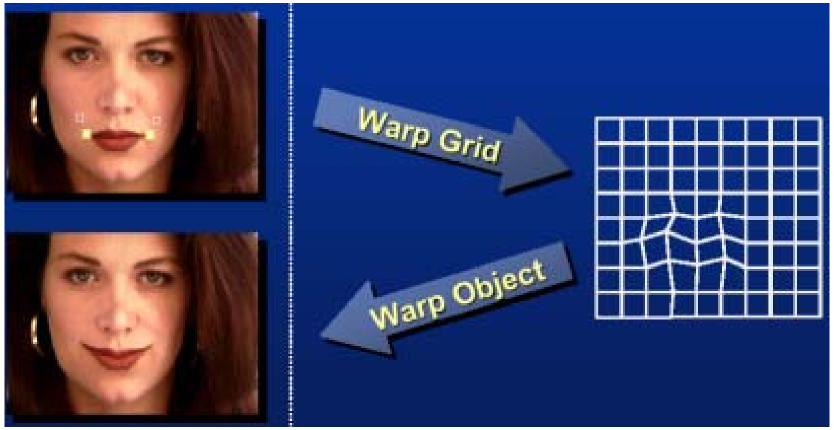
- A simple method for resampling images



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

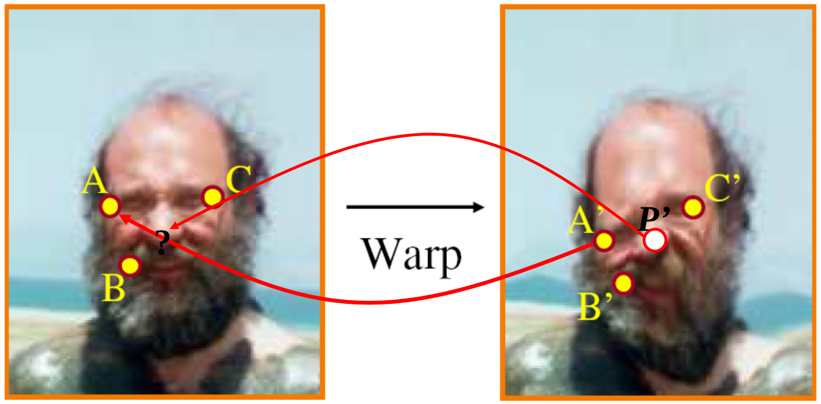
Non-parametric image warping

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



Non-parametric image warping

- Mappings implied by correspondences
- Inverse warping

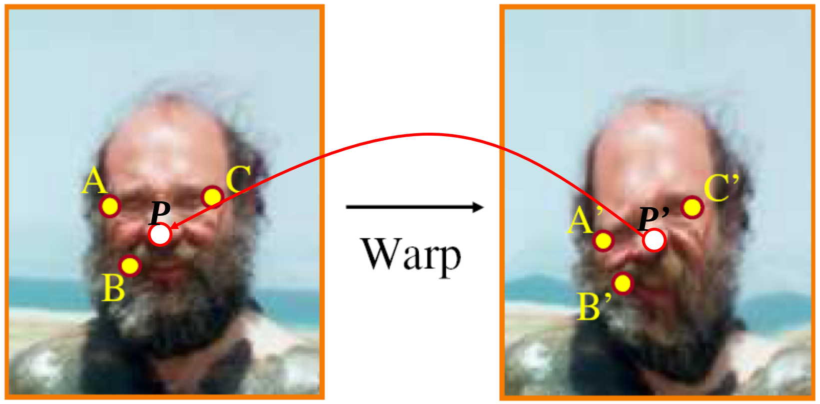


Non-parametric image warping

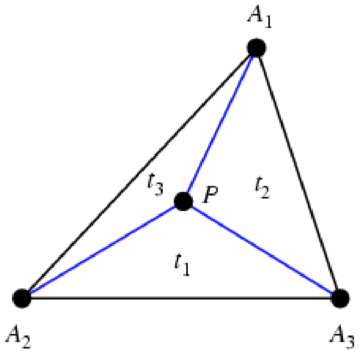
$$P = w_A A + w_B B + w_C C$$

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

Barycentric coordinate



Barycentric coordinates



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

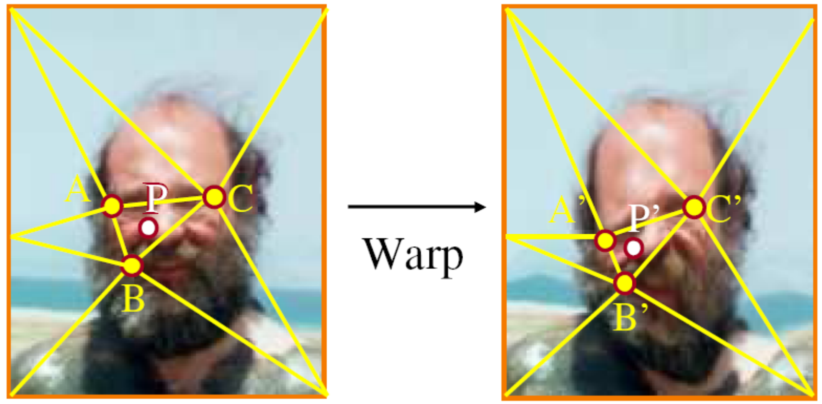
$$t_1 + t_2 + t_3 = 1$$

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



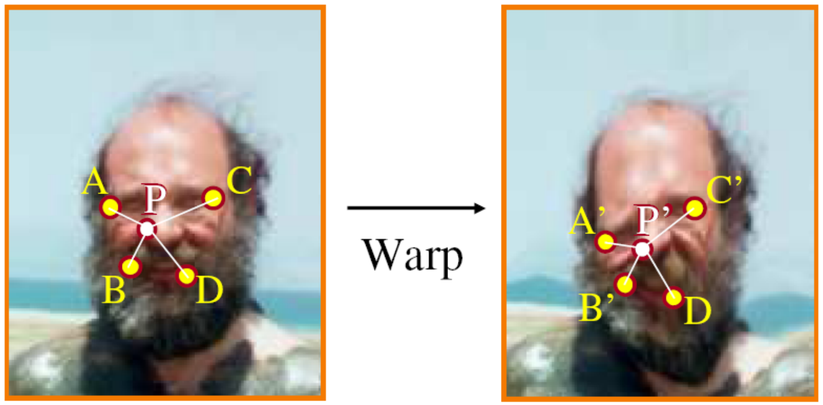
Non-parametric image warping

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

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

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

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

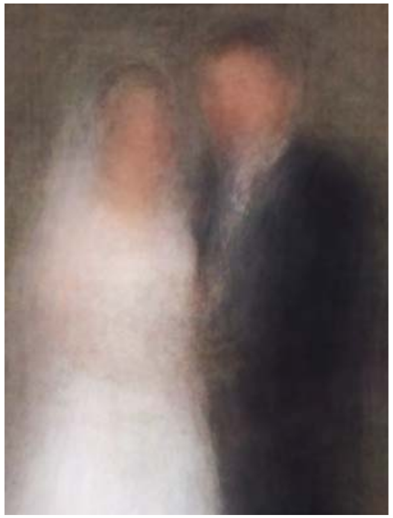
- The goal is to synthesize a fluid transformation from one image to another.
- Cross dissolving is a common transition between cuts, but it is not good for morphing because of the ghosting effects.


image #1


dissolving


image #2

Artifacts of cross-dissolving

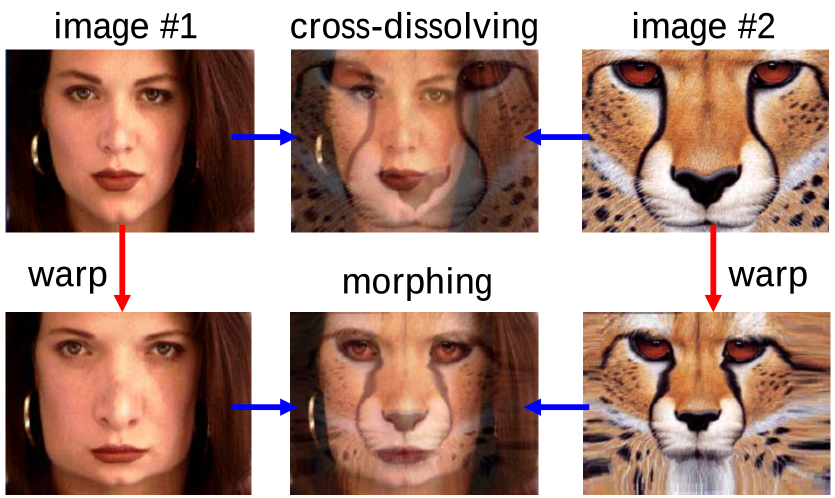
http://www.salavon.com/

Image morphing

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

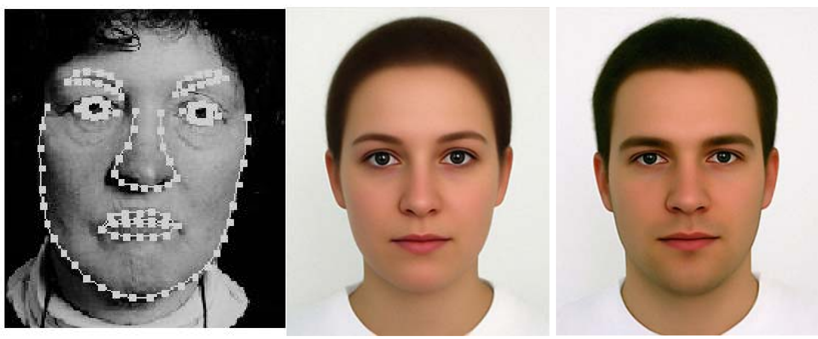
shape (geometric) — color (photometric)

Image morphing



Morphing sequence

Face averaging by morphing

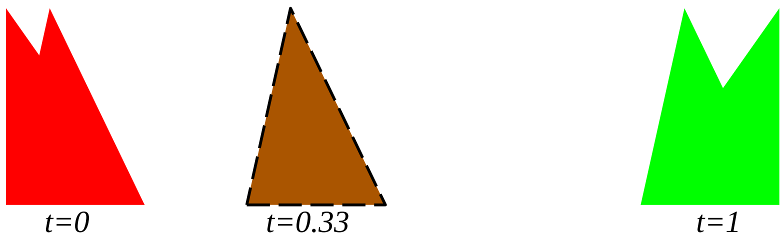


average faces

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



An ideal example (in 2004)

t=0

morphing

t=1

An ideal example

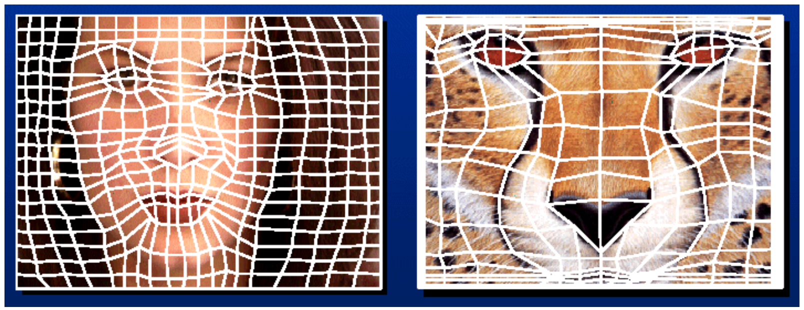
t=0

middle face (t=0.5)

t=1

Warp specification (mesh warping)

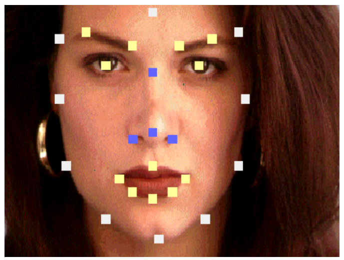
- How can we specify the warp?
  1. Specify corresponding *spline control points*
     interpolate to a complete warping function

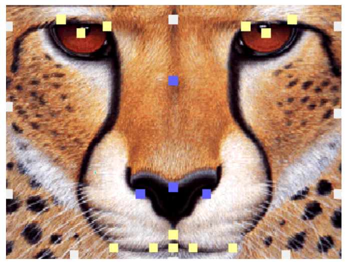


easy to implement, but less expressive

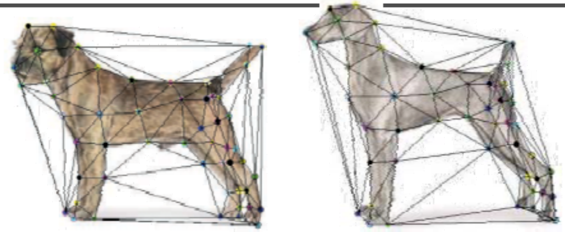
Warp specification

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





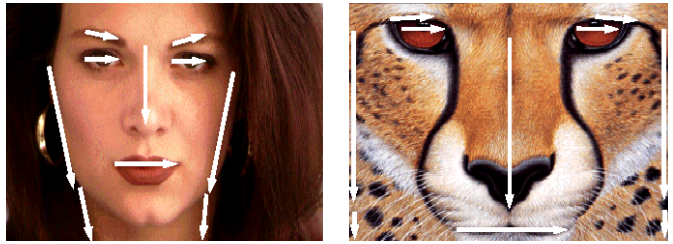
Solution: convert to mesh warping



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

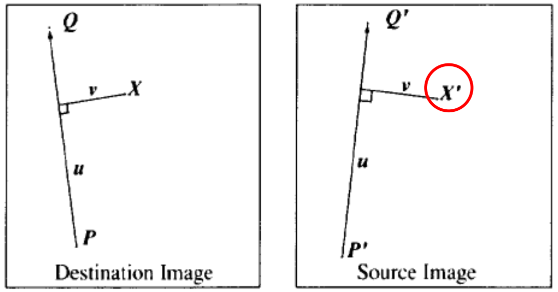
Warp specification (field warping)

- 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':



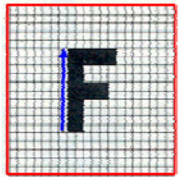
$$u = \frac{(X-P)\cdot(Q-P)}{\|Q-P\|^2} \quad (1)$$

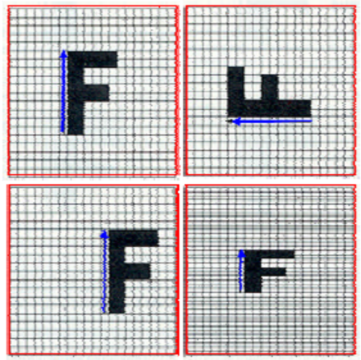
$$v = \frac{(X-P)\cdot Perpendicular(Q-P)}{\|Q-P\|} \quad (2)$$

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

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:

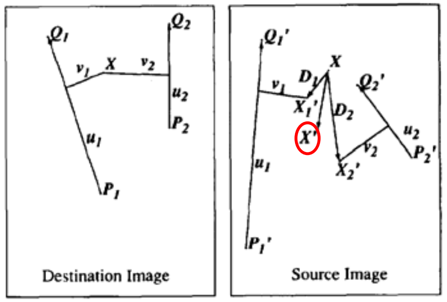




Affine transformation

Multiple Lines

$$D_i = X_i' - X_i$$



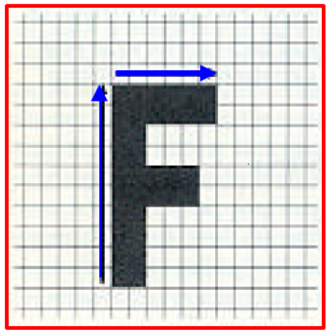
$$weight[i] = \left(\frac{length[i]^p}{a + dist[i]} \right)^b$$

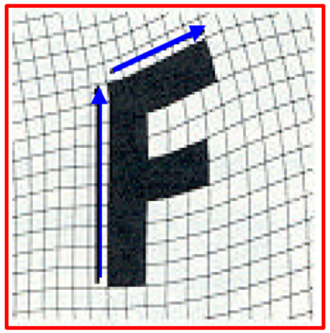
length = length of the line segment,
dist = distance to line segment
The influence of a, p, b. The same as the average of X_i'

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
```

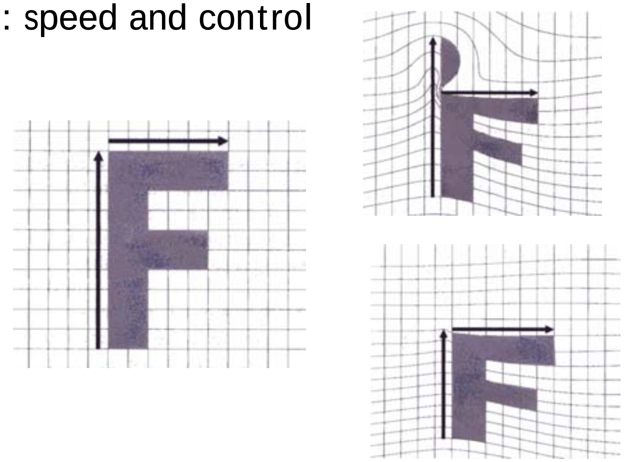
Resulting warp





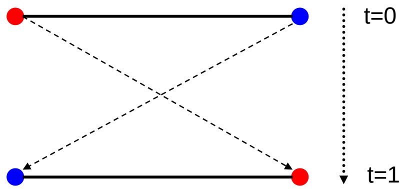
Comparison to mesh morphing

- 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(Image$_0$, L_0[...],Image$_1$, L_1[...])
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 L_0[i] to L_1[i].
        end
        Warp$_0$ = WarpImage(Image$_0$, L_0[...], L[...])
        Warp$_1$ = WarpImage(Image$_1$, L_1[...], L[...])
        foreach pixel p in FinalImage **do**
            FinalImage(p) = (1-t) Warp$_0$(p) + t Warp$_1$(p)
        end
    end
end

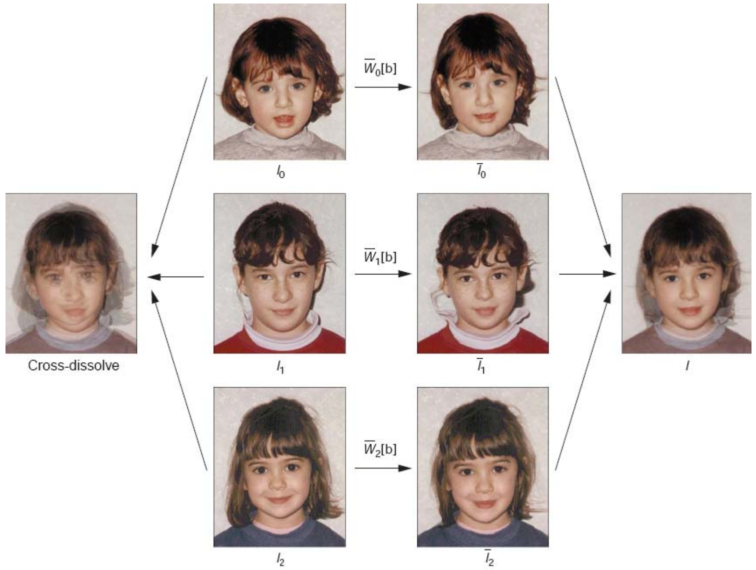
Animated sequences

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

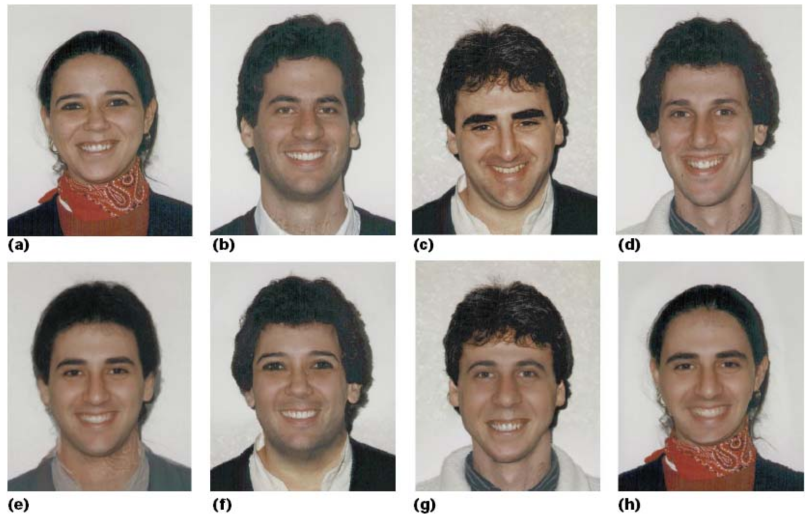
Results

Michael Jackson's MTV "Black or White"

Multi-source morphing



Multi-source morphing



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.