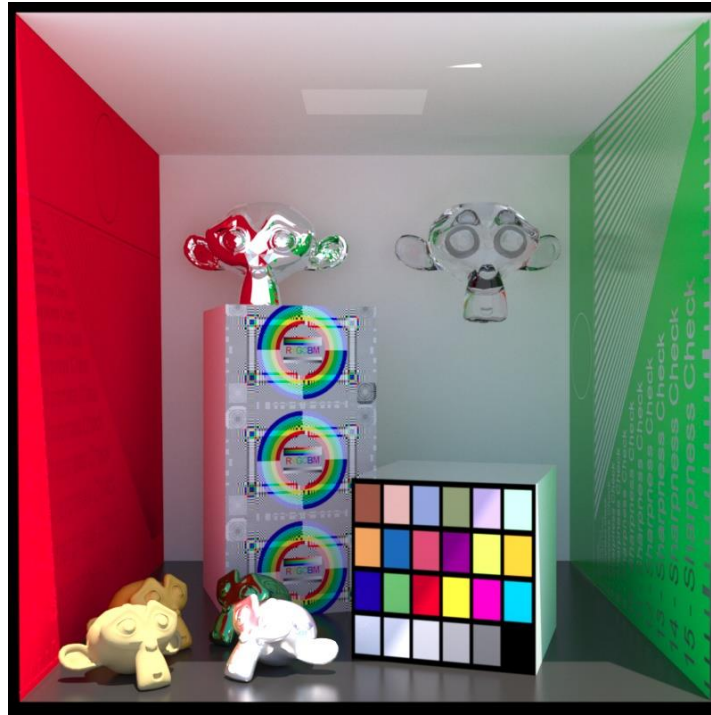


Computer Graphics

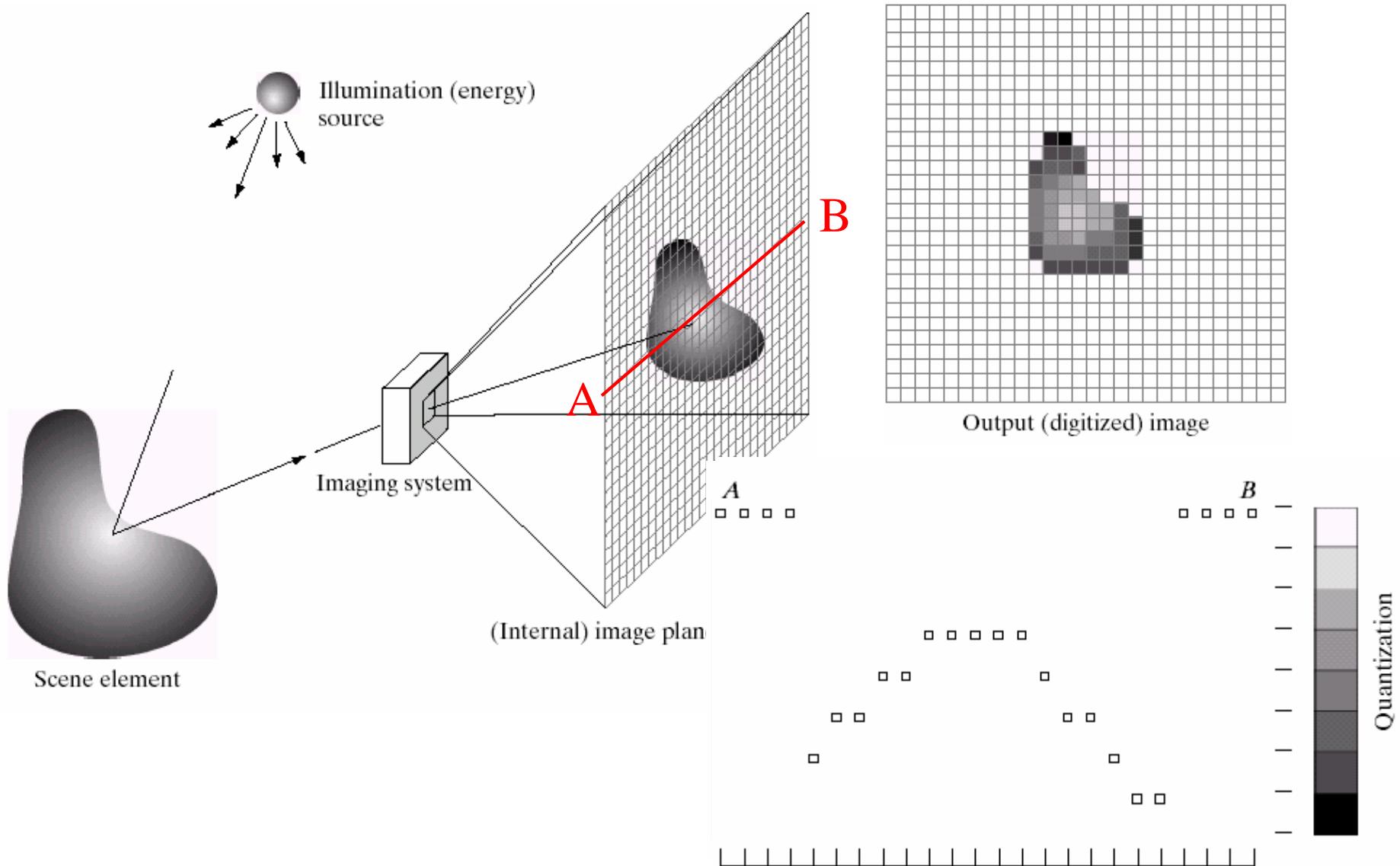
Image warping and morphing



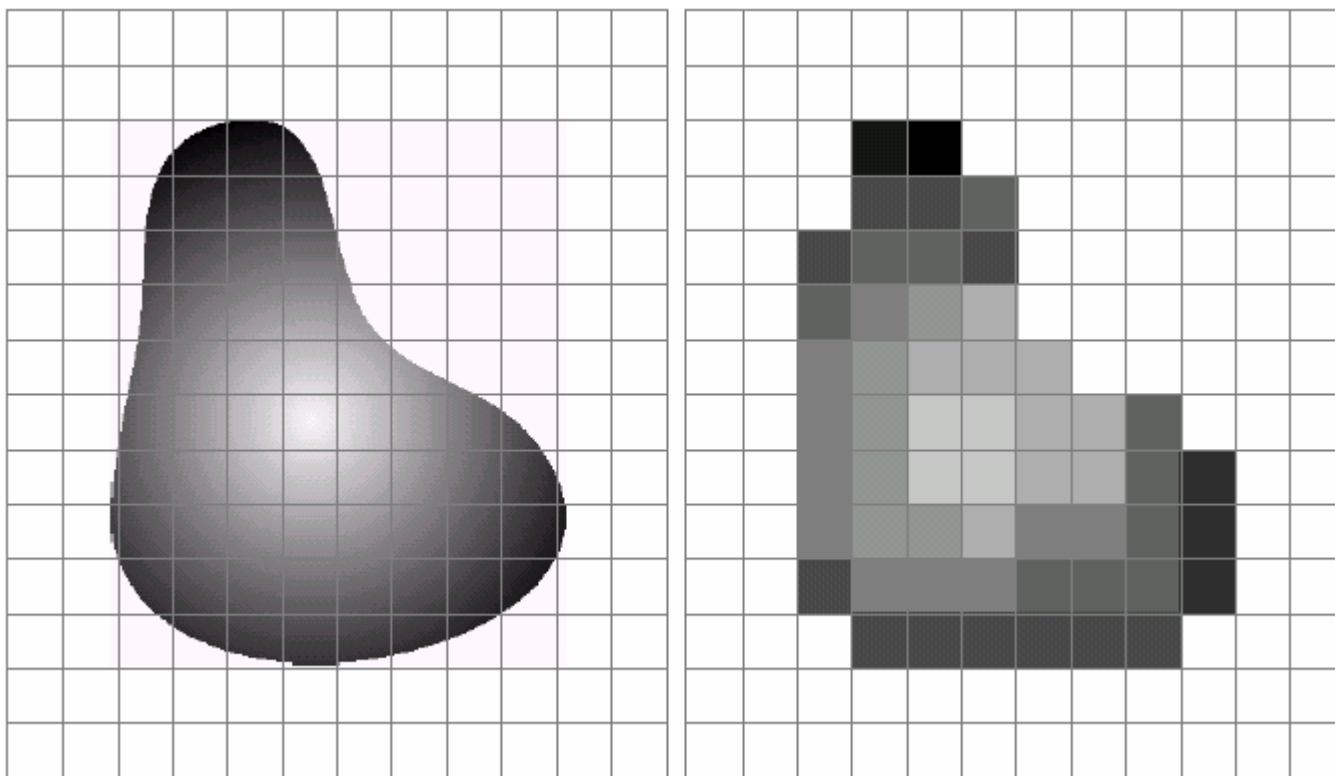
Yu-Shuen Wang, CS, NCTU

Image warping

Image formation

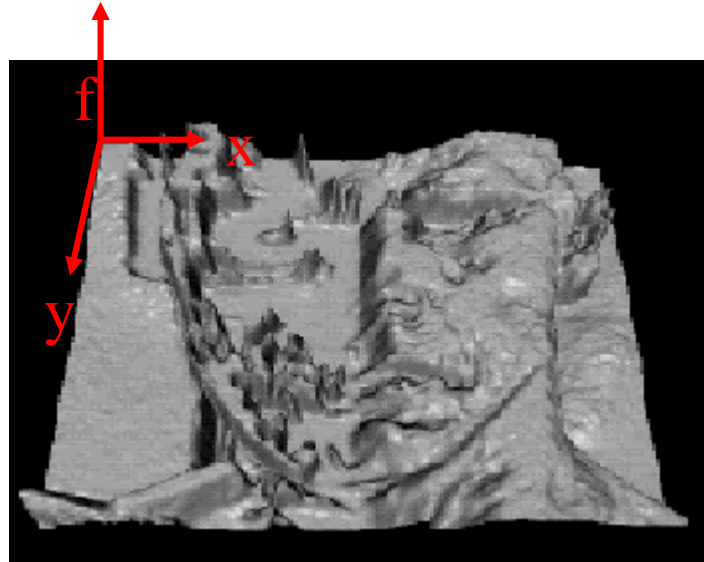


Sampling and quantization



What is an image

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



- 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

$i \downarrow$ $j \longrightarrow$

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

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

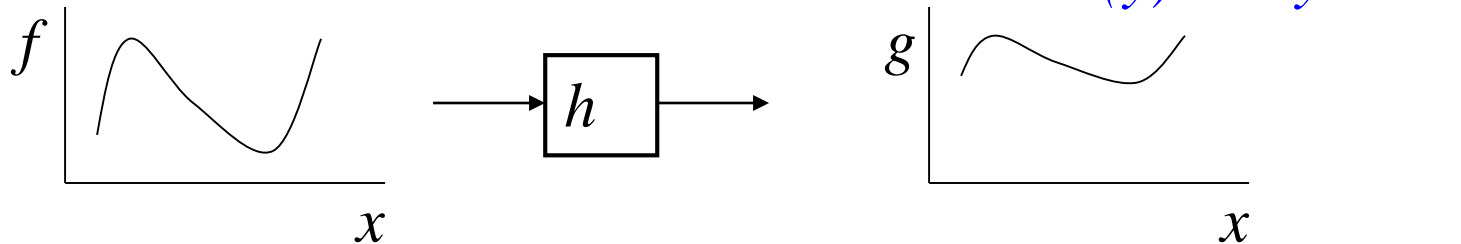


image warping: change **domain** of image

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

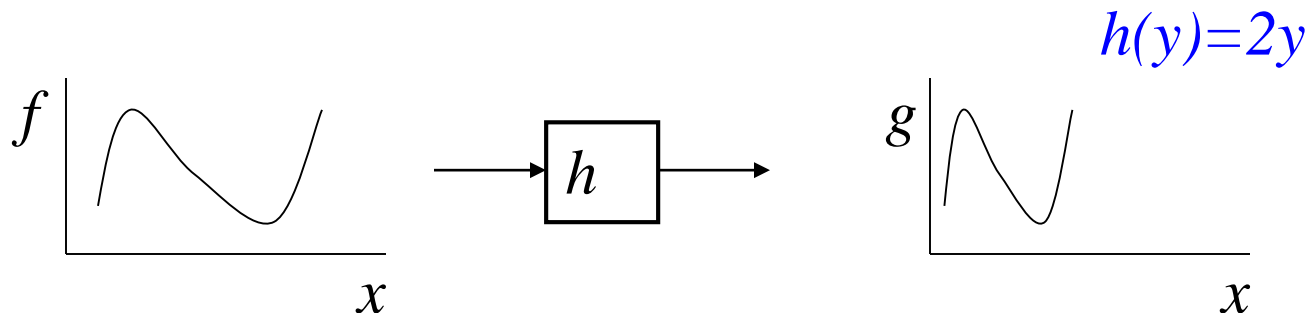


Image warping

image filtering: change **range** of image

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

$$h(y) = 0.5y + 0.5$$

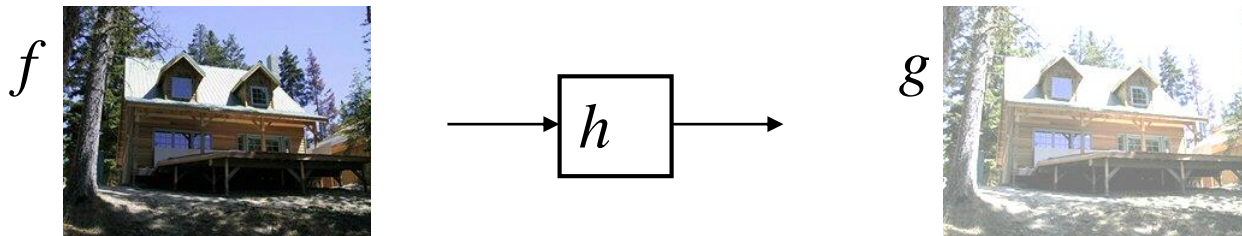
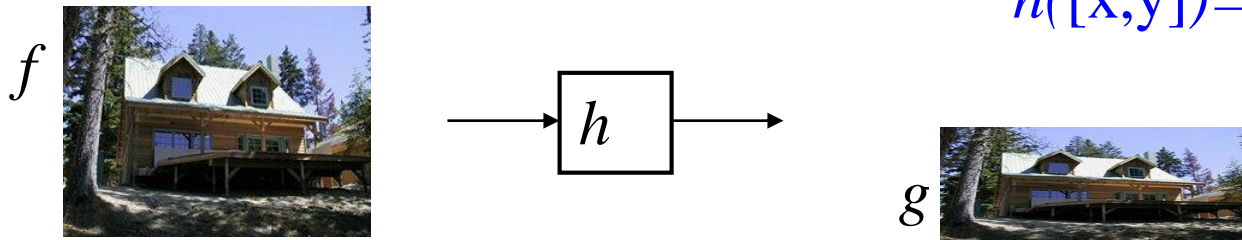


image warping: change **domain** of image

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

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



Parametric (global) warping

Examples of parametric warps:



translation



rotation



aspect



affine



perspective

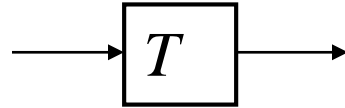


cylindrical

Parametric (global) warping



$$\mathbf{p} = (x, y)$$

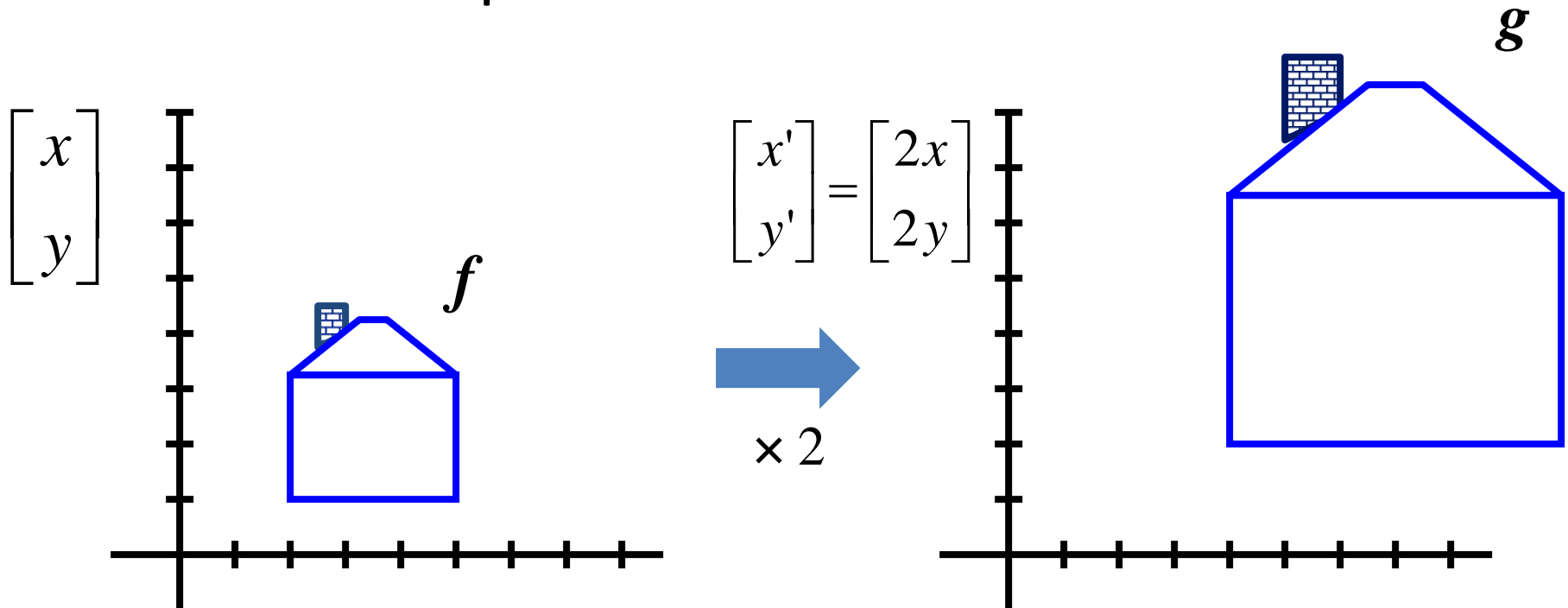


$$\mathbf{p}' = (x', y')$$

- Transformation T is a coordinate-changing machine: $\mathbf{p}' = T(\mathbf{p})$
- What does it mean that T is global?
 - Is the same for any point \mathbf{p}
 - can be described by just a few numbers (parameters)
- Represent T as a matrix: $\mathbf{p}' = \mathbf{M} * \mathbf{p}$
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{M} \begin{bmatrix} x \\ y \end{bmatrix}$$

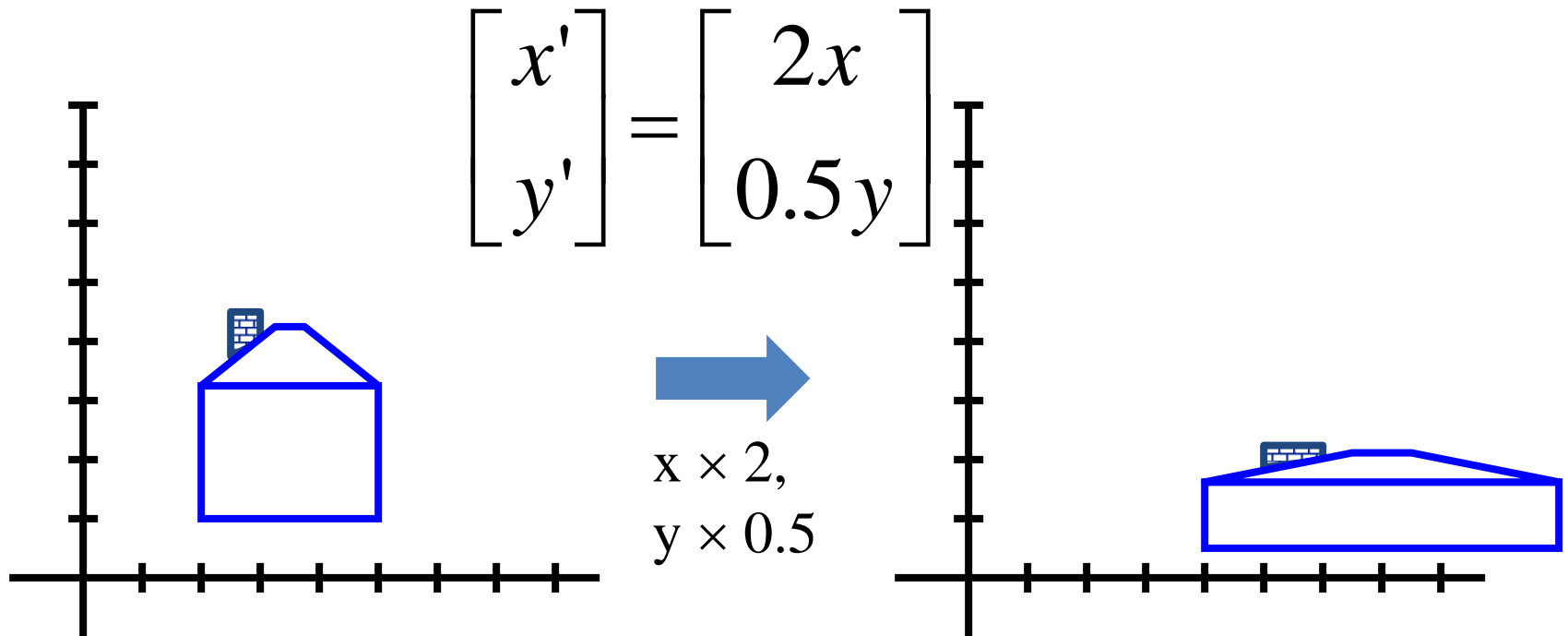
Scaling

- *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:



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?

$$\begin{aligned}x' &= x + t_x \\ y' &= y + t_y\end{aligned}$$

NO!

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

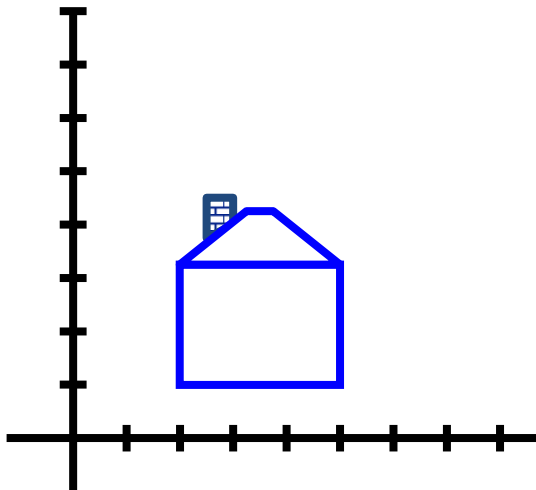
Translation

- Example of translation

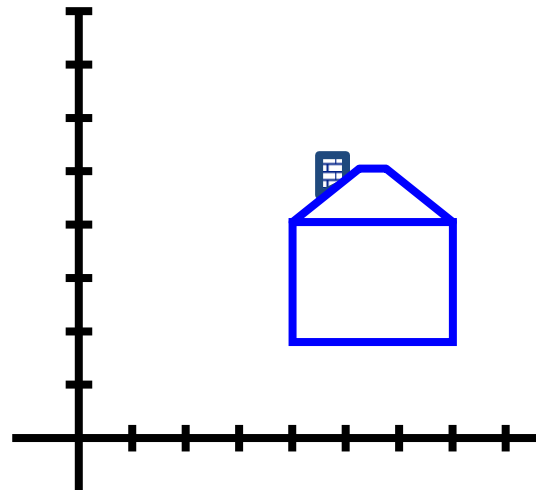
Homogeneous Coordinates



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



$$\begin{aligned} t_x &= 2 \\ t_y &= 1 \end{aligned}$$



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

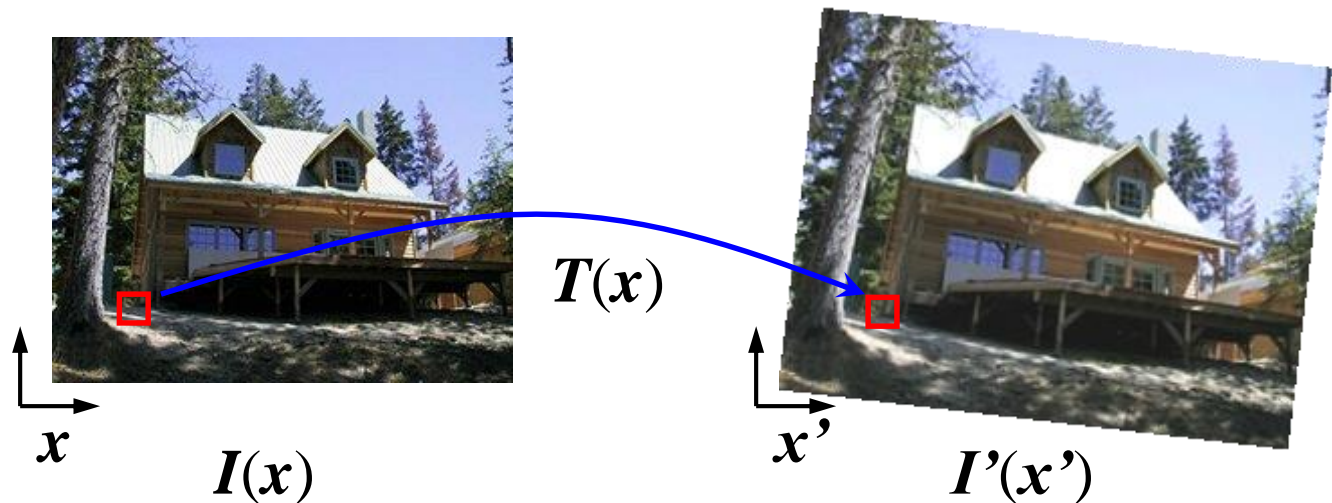
Projective Transformations

- 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

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

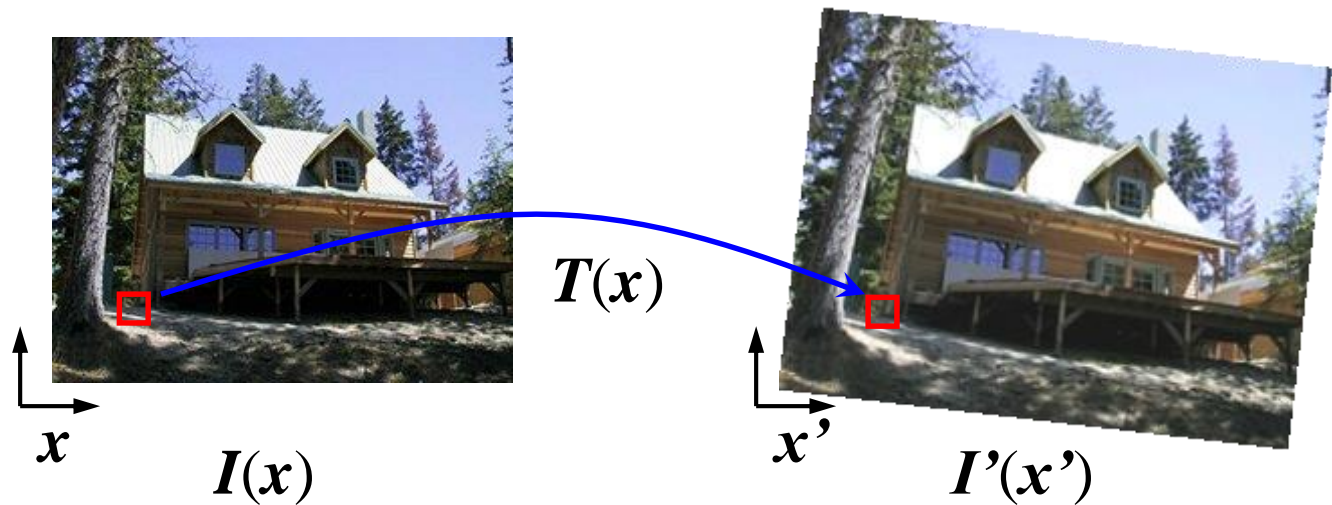
Image warping

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



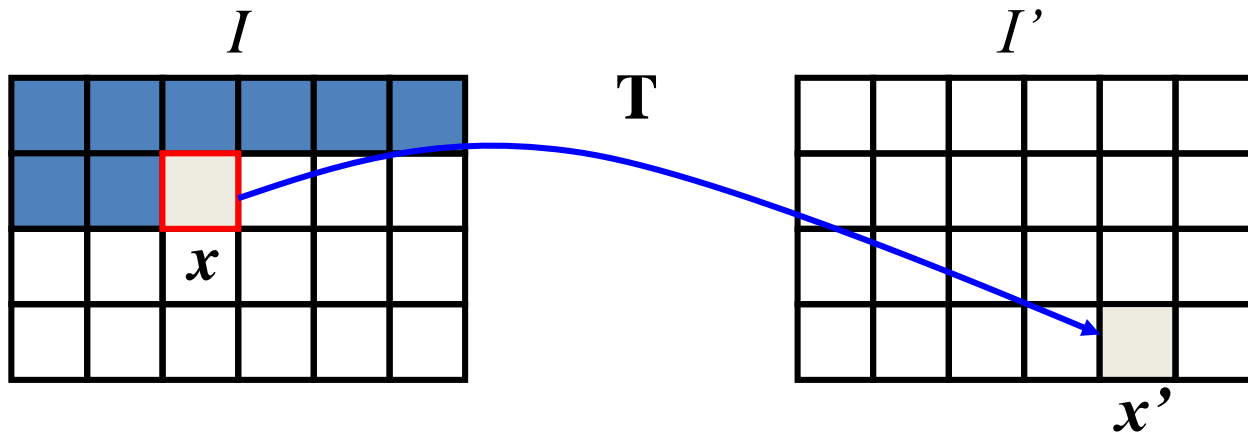
Forward warping

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



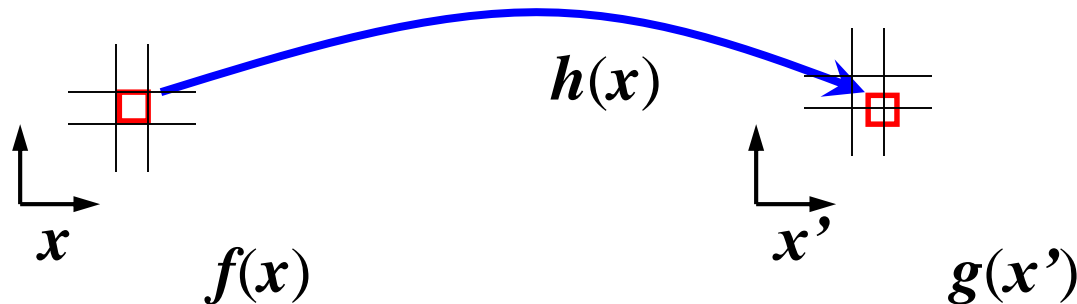
Forward warping

```
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 $I(\mathbf{x})$ to its corresponding location $\mathbf{x}' = \mathbf{T}(\mathbf{x})$ in $I'(\mathbf{x}')$
- 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++)
```

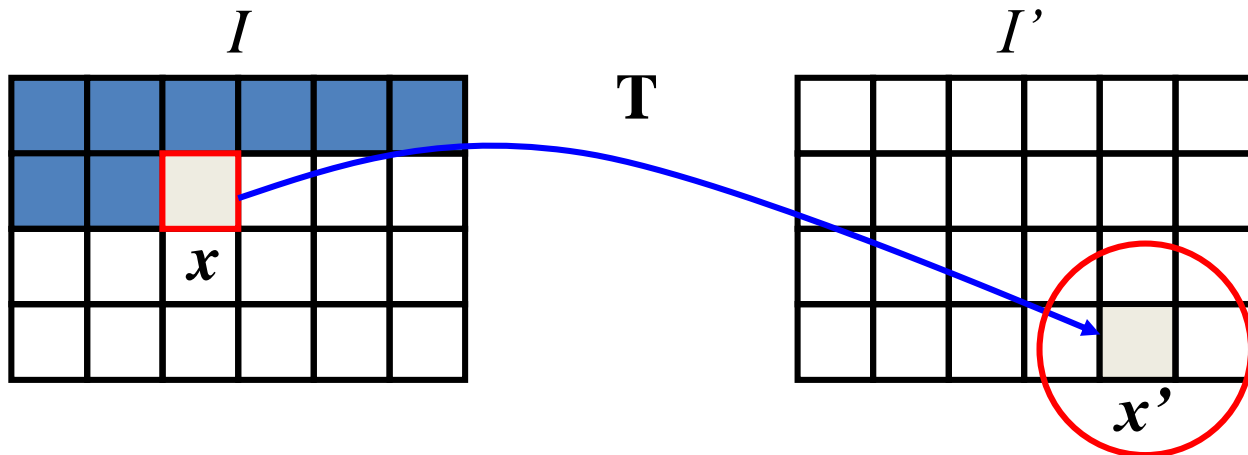
```
    for (x=0; x<I.width; x++) {
```

```
      (x', y') = T(x, y);
```

```
      Splatting(I', x', y', I(x, y), kernel);
```

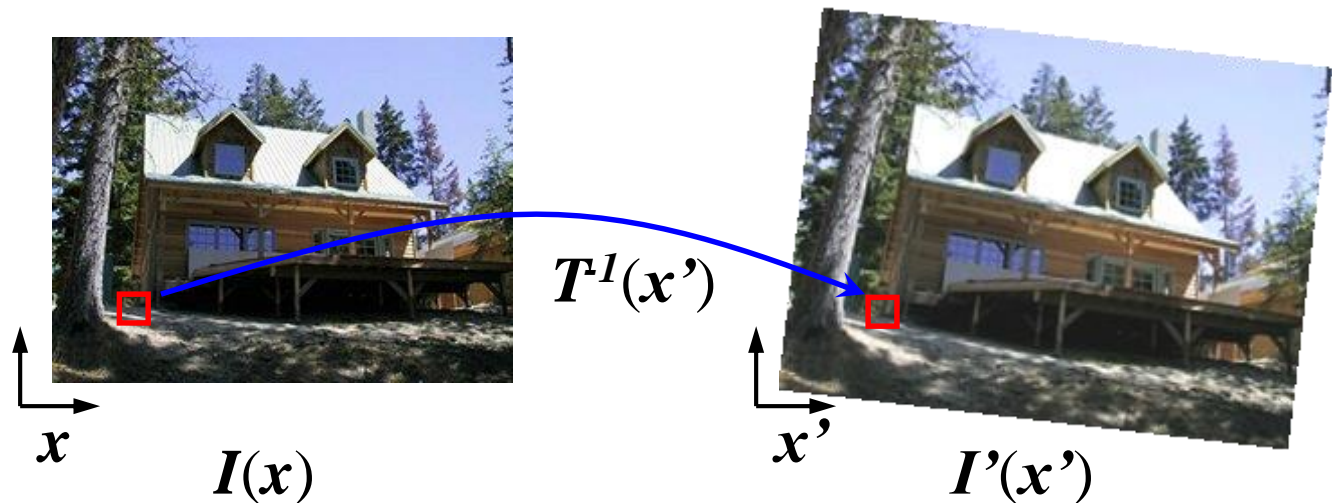
```
    }
```

```
}
```



Inverse warping

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



Inverse warping

```
iwarp(I, I', T)
```

```
{
```

```
  for (y=0; y<I'.height; y++)
```

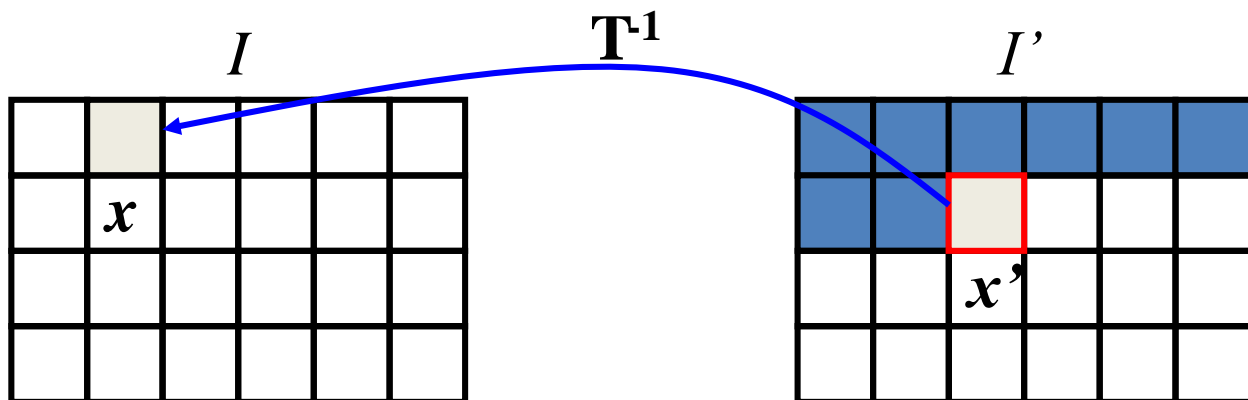
```
    for (x=0; x<I'.width; x++) {
```

```
       $(x, y) = T^{-1}(x', y')$  ;
```

```
       $I'(x', y') = I(x, y)$  ;
```

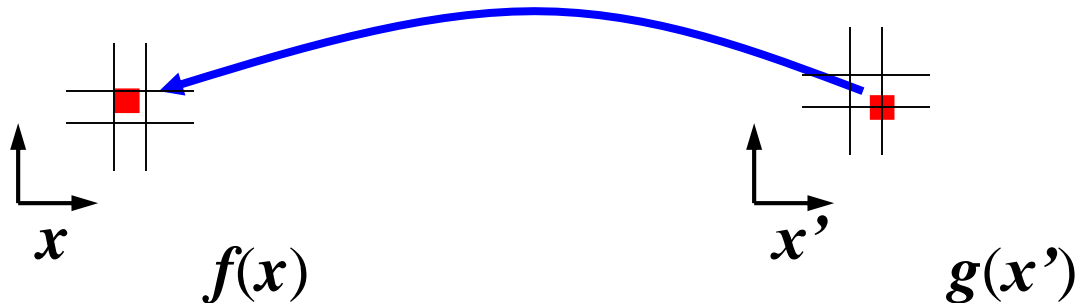
```
    }
```

```
}
```



Inverse warping

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

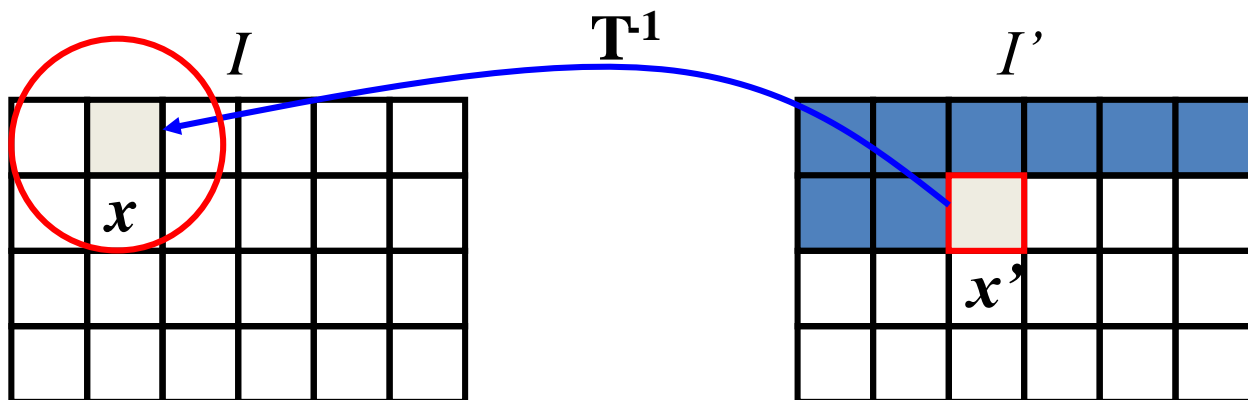
```
    for (x=0; x<I'.width; x++) {
```

```
       $(x, y) = T^{-1}(x', y')$  ;
```

```
       $I'(x', y') = \text{Reconstruct}(I, x, y, \text{kernel})$  ;
```

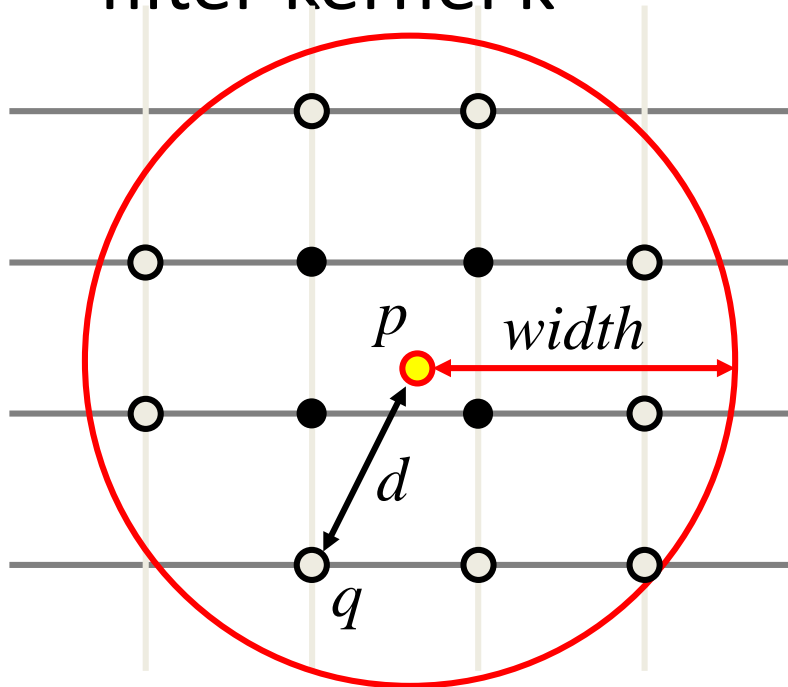
```
    }
```

```
}
```



Reconstruction

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



```
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;
```

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

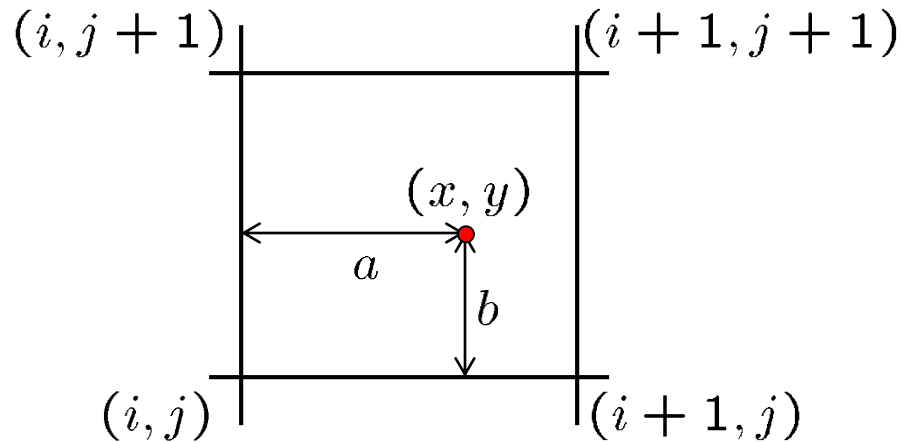
Reconstruction (interpolation)

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



Bilinear interpolation

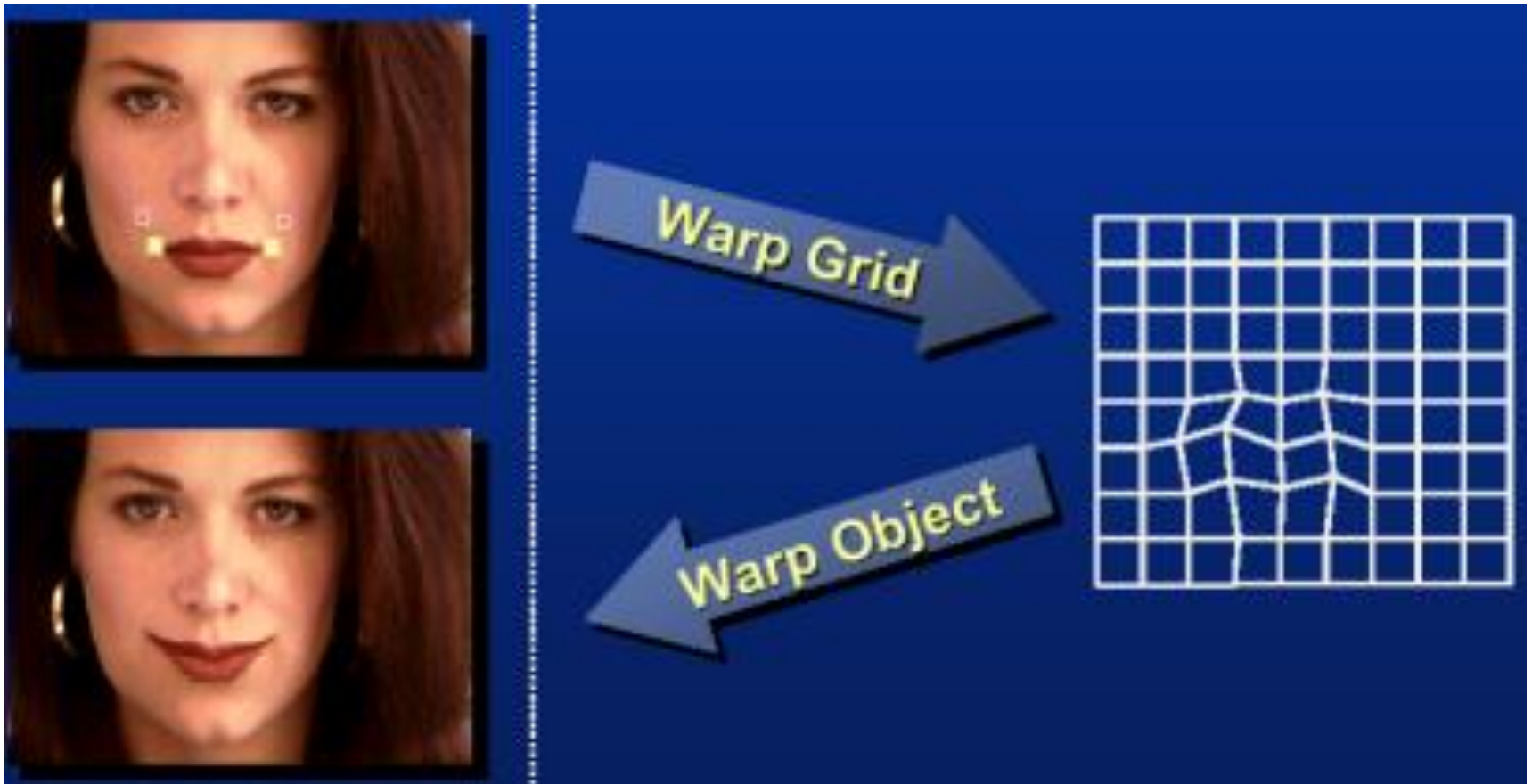
- A simple method for resampling images



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

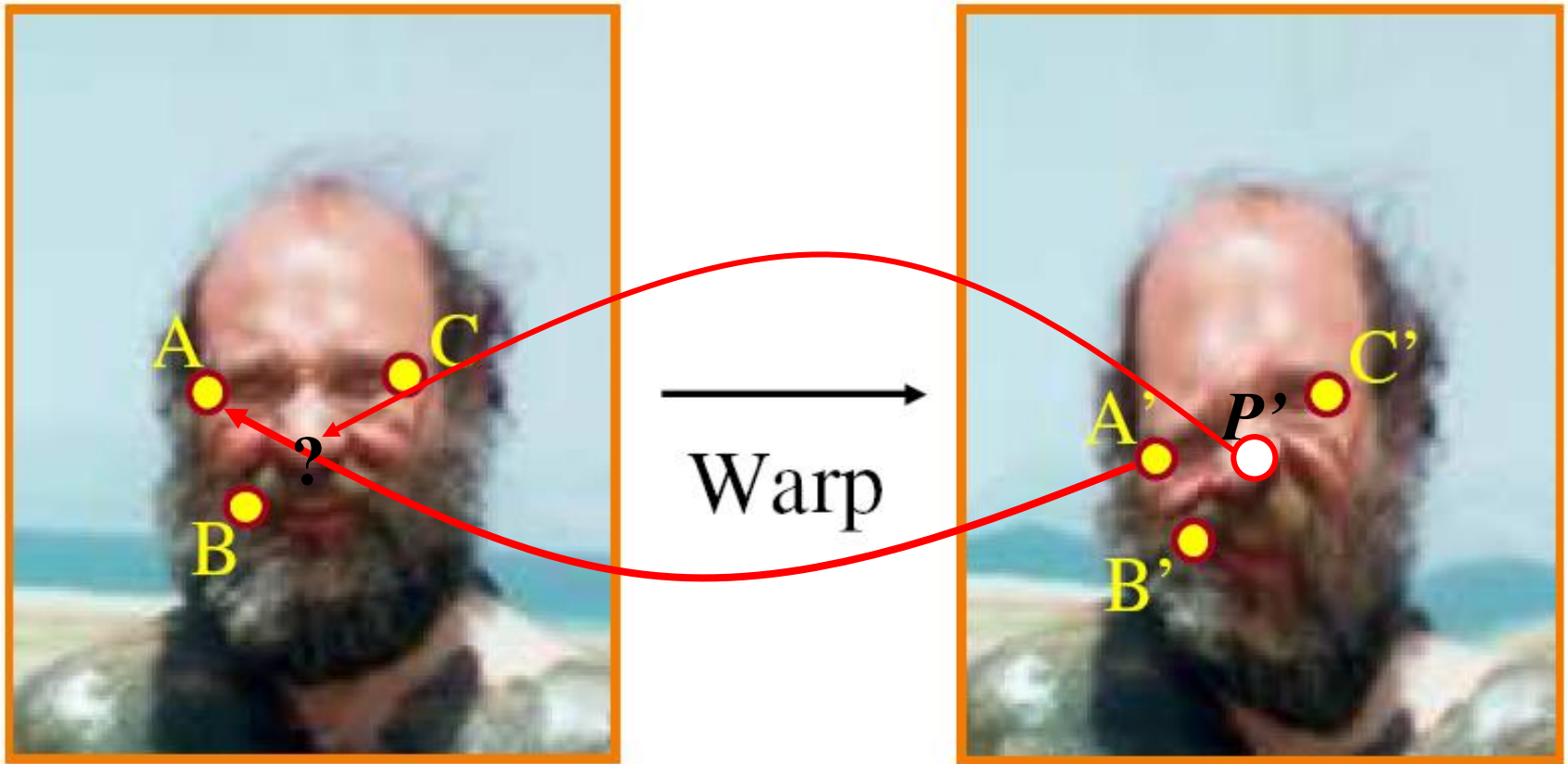
Non-parametric image warping

- Specify a more detailed warp function



Non-parametric image warping

- Mappings implied by correspondences
- Inverse warping

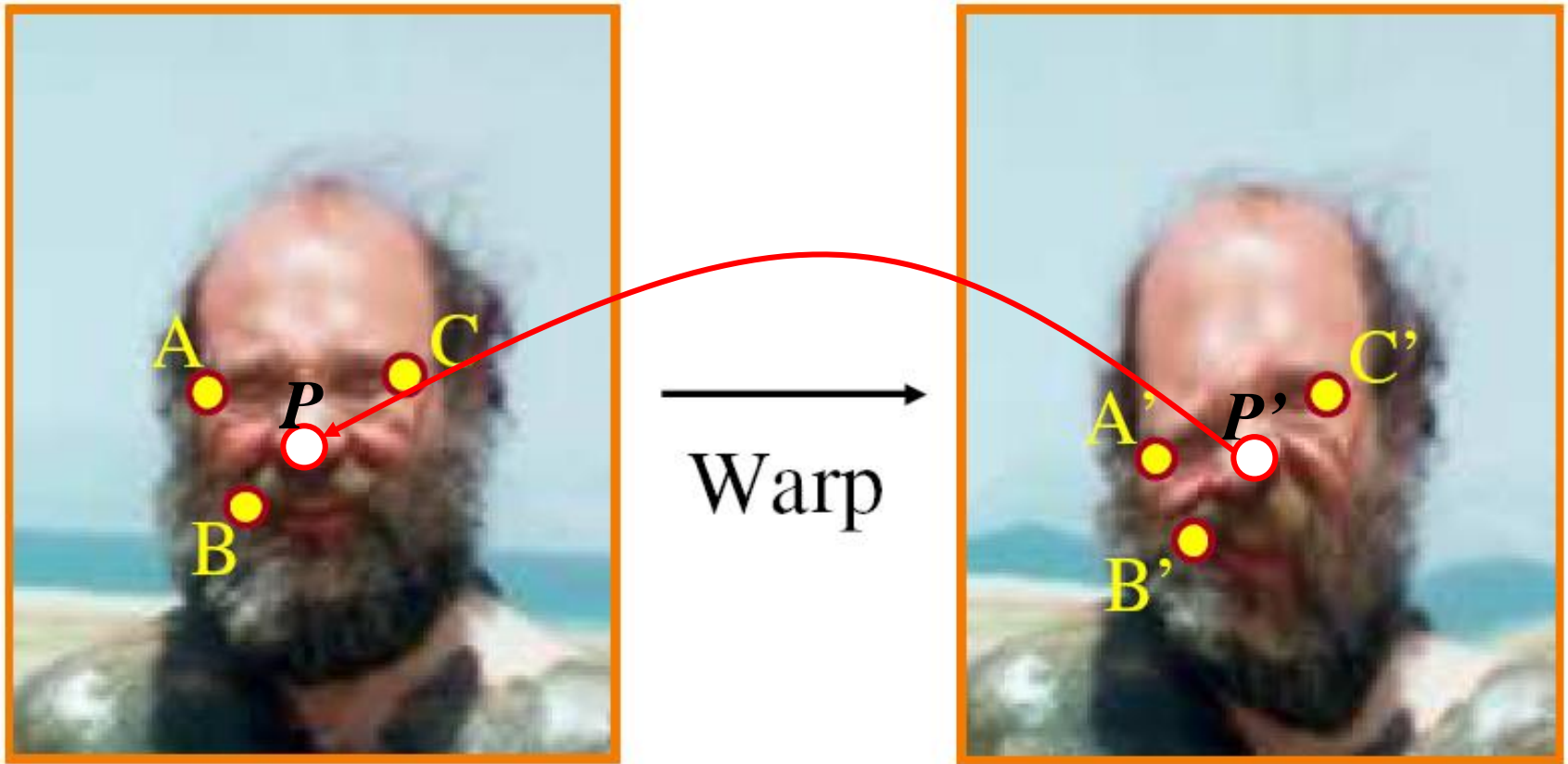


Non-parametric image warping

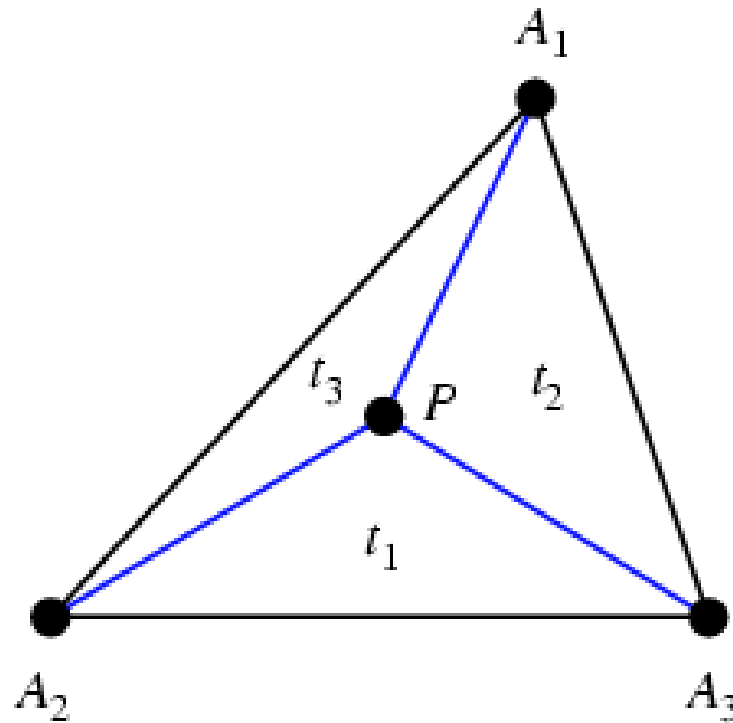
$$P = w_A A + w_B B + w_C C$$

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

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

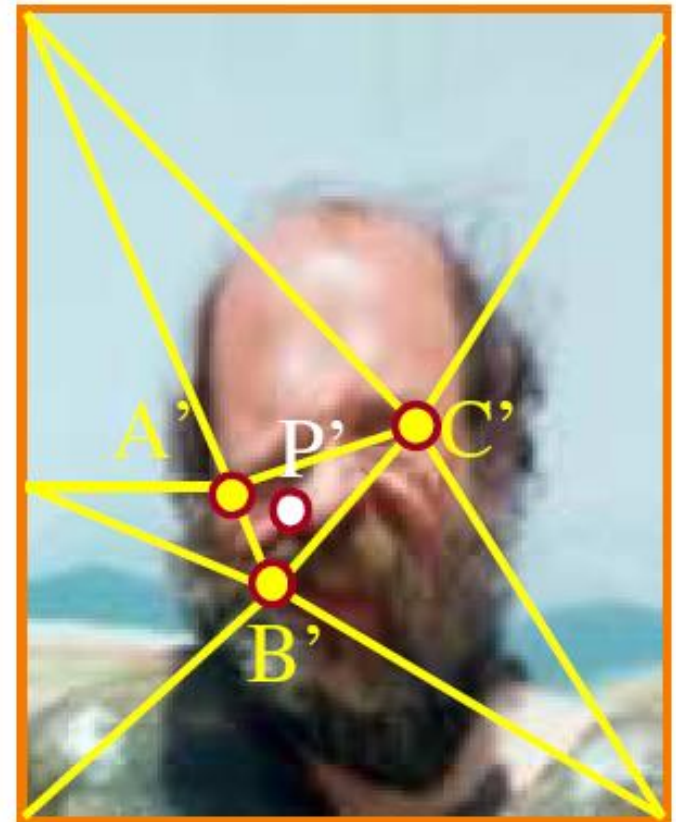
$$P = w_A A + w_B B + w_C C$$

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

Barycentric coordinate



→
Warp



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



→
Warp



Image morphing

Results



Michael Jackson's MTV "Black or White"

<https://www.youtube.com/watch?v=3Iw-vUBo7v4>

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

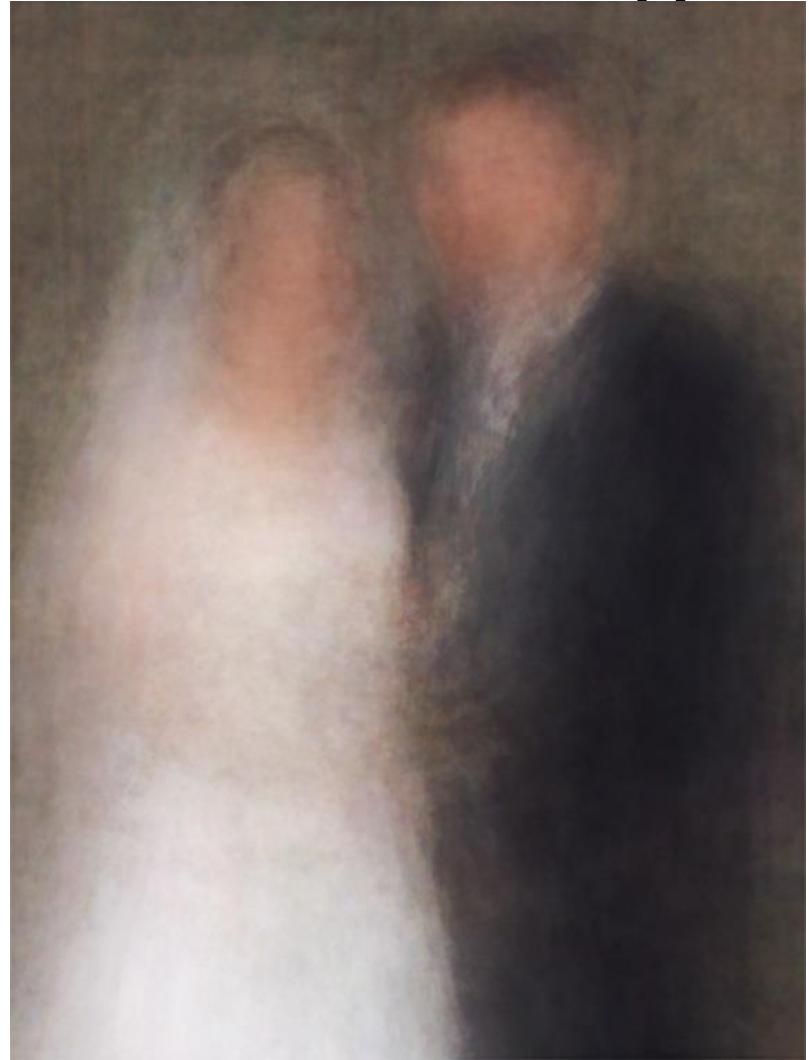


dissolving



image #2

Artifacts of cross-dissolving



<http://www.salavon.com/>

Image morphing

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

shape
(geometric)

color
(photometric)



Image morphing

image #1



cross-dissolving



image #2



warp



morphing



warp



Morphing sequence



<https://www.youtube.com/watch?v=jpOaEc2slaw>

<https://www.youtube.com/watch?v=wZurRt0TidI>

Face averaging by morphing



average faces



Uzbek Welsh West African Vietnamese Chinese Hungarian Japanese Korean Puerto Rican



Thai African American Afghan Central African Burmese Cambodian English Ethiopian Filipino



Finnish French German Greek Indian Iranian Irish Israeli Italian



Mexican Latvian/Lithuanian Mongolian Peruvian Polish Romanian Russian Samoan South African



South Indian Spaniard Swedish Swiss Taiwanese

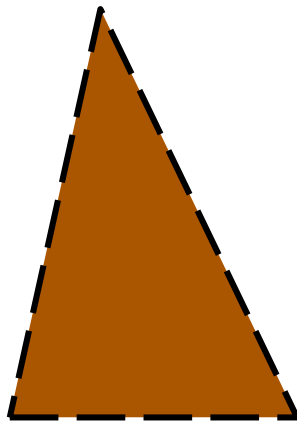
Image morphing

create a morphing sequence: for each time t

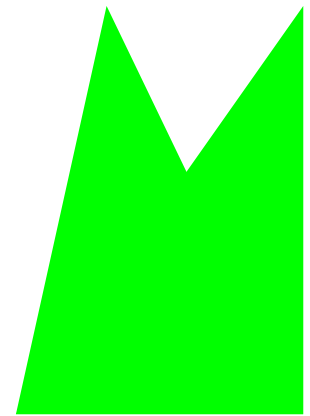
1. Create an intermediate warping field (by interpolation)
2. Warp both images towards it
3. Cross-dissolve the colors in the newly warped images



$t=0$



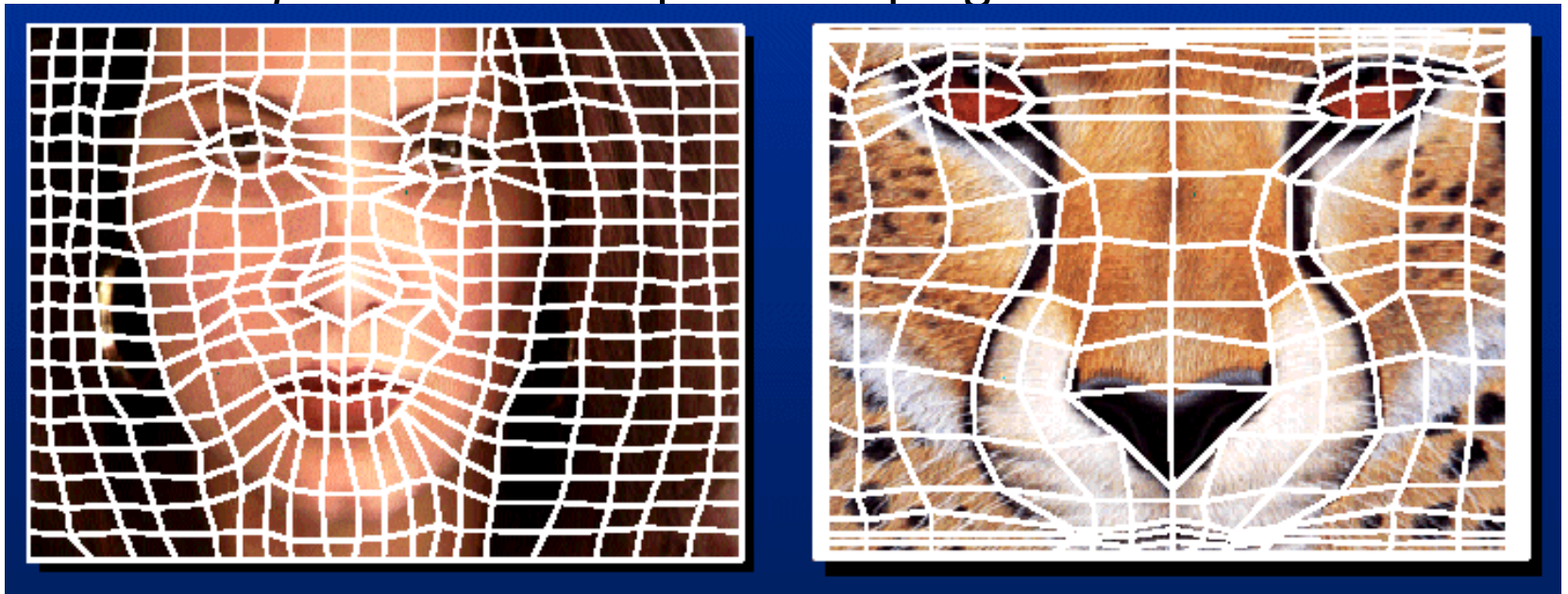
$t=0.33$



$t=1$

Warp specification (mesh warping)

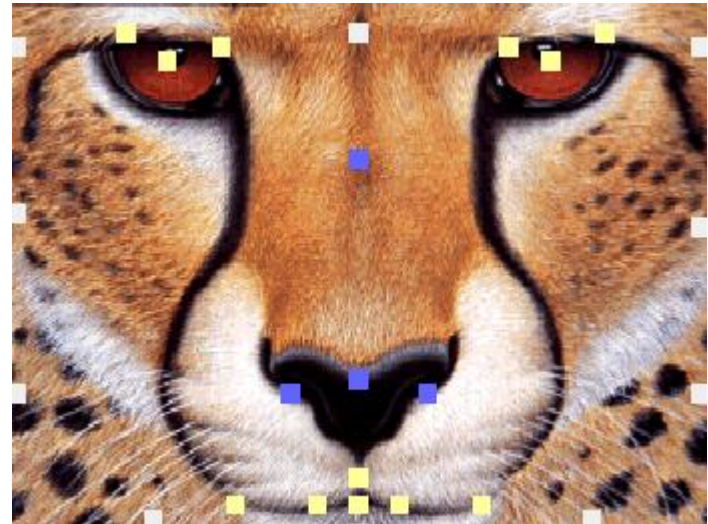
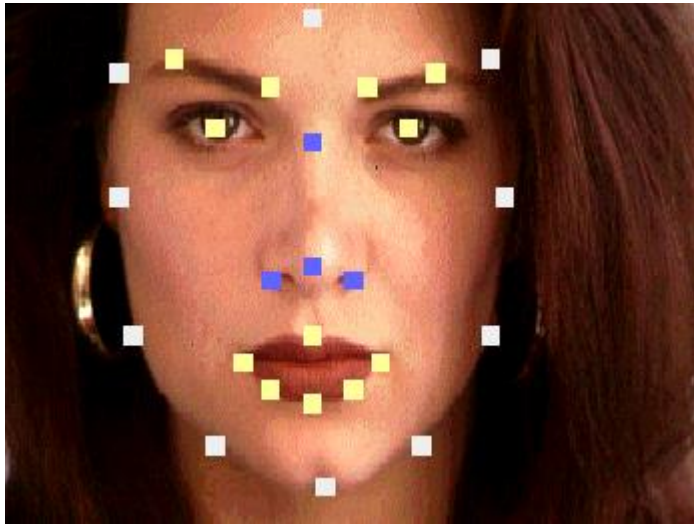
- How can we specify the warp?
 1. Specify corresponding *spline control points*
interpolate to a complete warping function



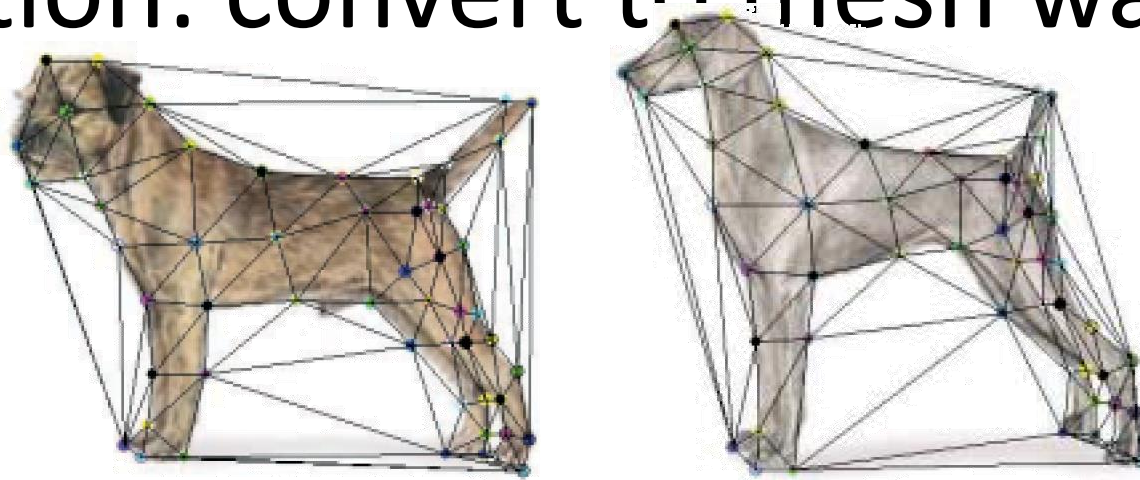
easy to implement, but less expressive

Warp specification

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



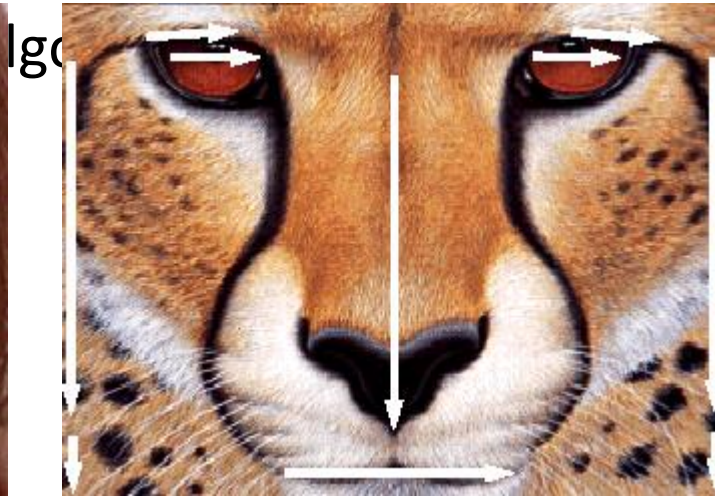
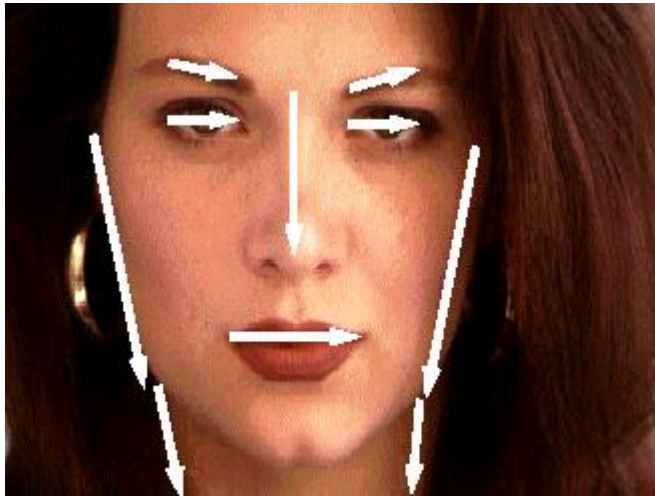
Solution: convert to mesh warping



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

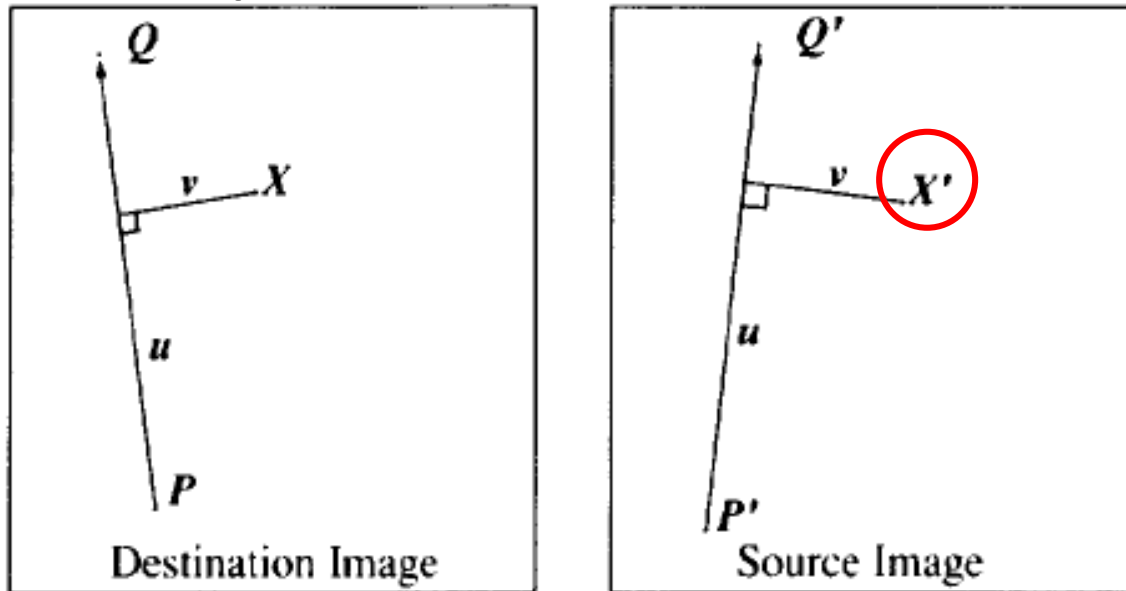
Warp specification (field warping)

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



Beier&Neely (SIGGRAPH 1992)

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



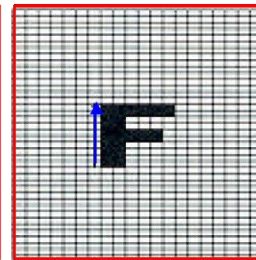
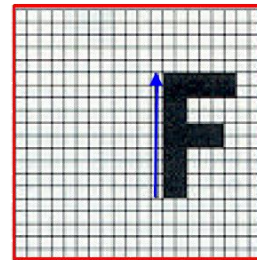
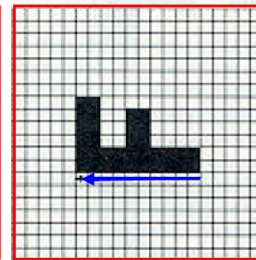
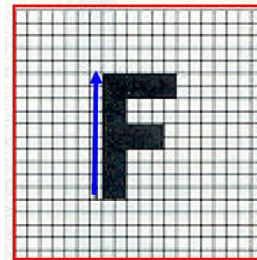
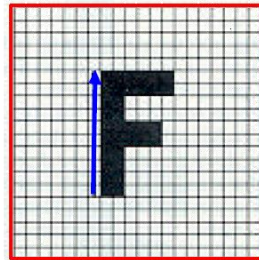
$$u = \frac{(X - P) \cdot (Q - P)}{\|Q - P\|^2} \quad (1)$$

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

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

Algorithm (single line-pair)

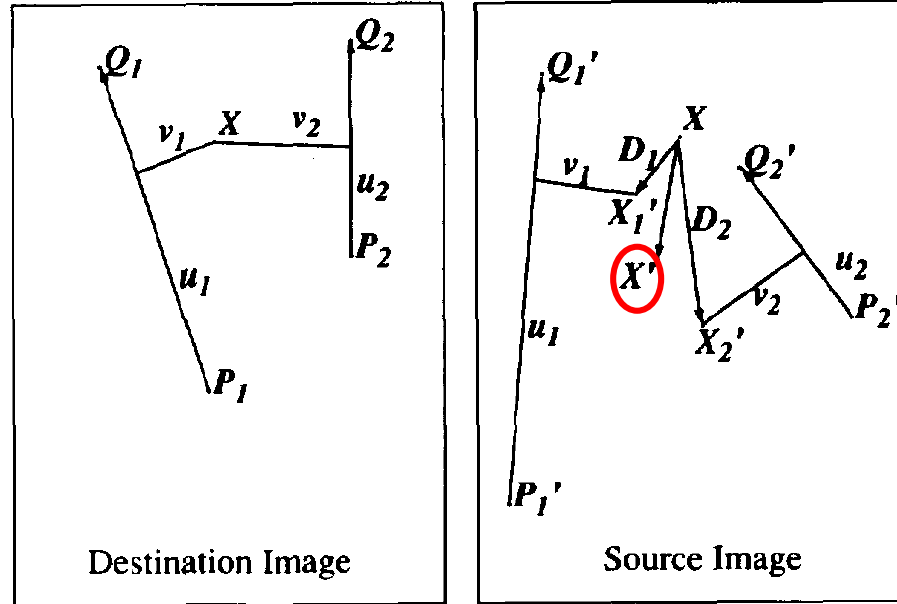
- For each X in the destination image:
 1. Find the corresponding u, v
 2. Find X' in the source image for that u, v
 3. $\text{destinationImage}(X) = \text{sourceImage}(X')$
- Examples:



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,

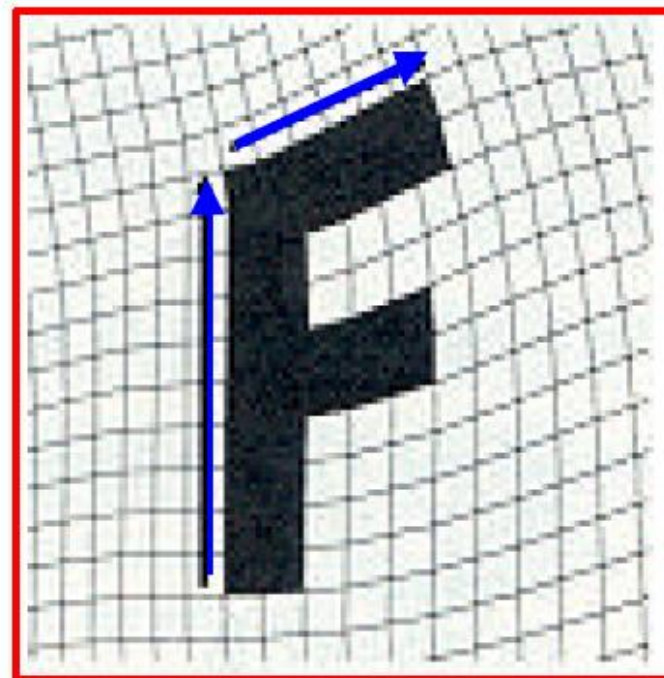
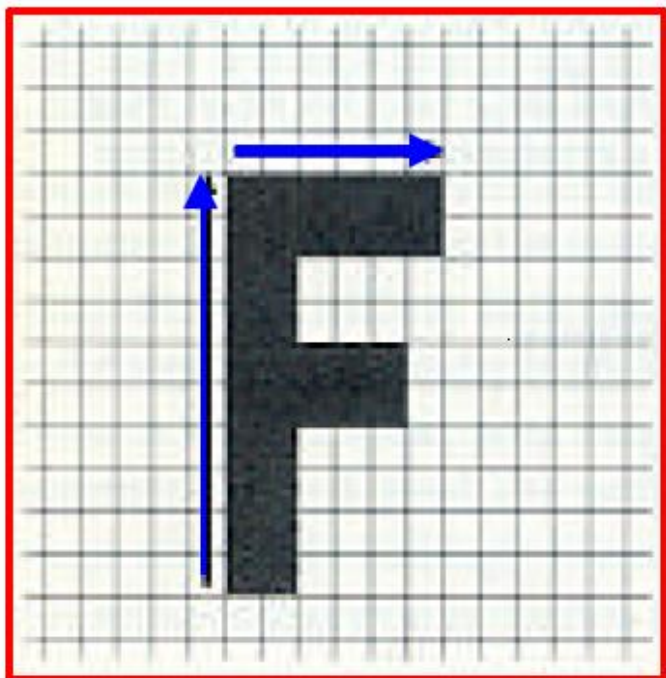
$dist$ = distance to line segment

The influence of a , p , b . The same as the average of X_i'

Full Algorithm

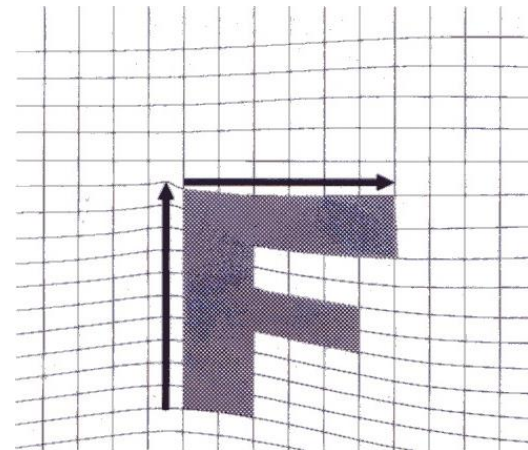
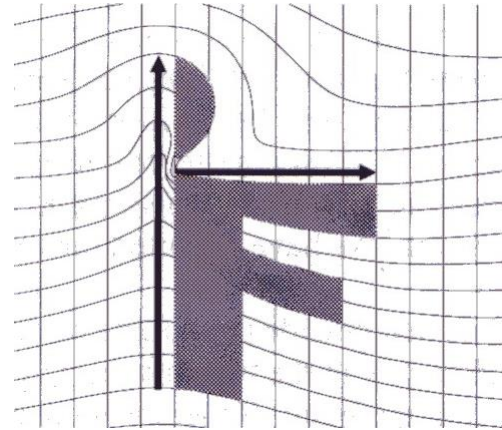
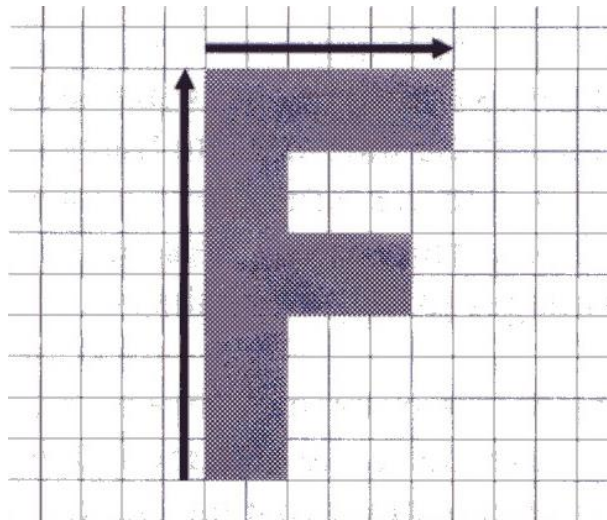
```
WarpImage(SourceImage, L'[...], L[...])
begin
    foreach destination pixel X do
        XSum = (0,0)
        WeightSum = 0
        foreach line L[i] in destination do
            X'[i] = X transformed by (L[i], L'[i])
            weight[i] = weight assigned to X'[i]
            XSum = Xsum + X'[i]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

Resulting warp



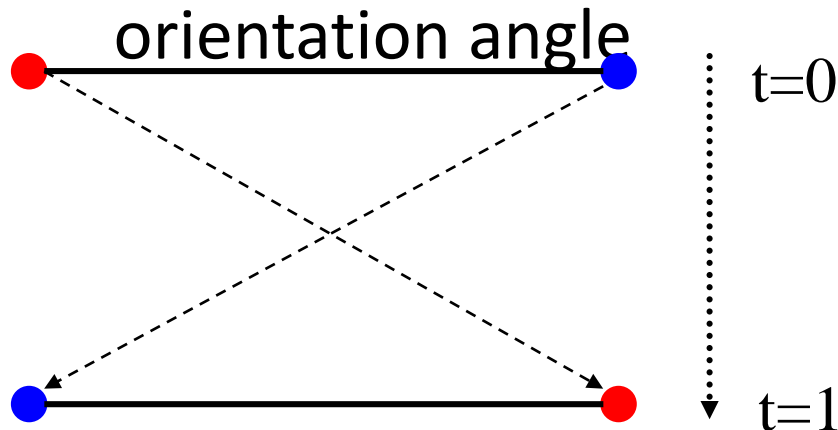
Comparison to mesh morphing

- 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



Animation

GenerateAnimation(Image₀, L₀[...], Image₁, L₁[...])

begin

foreach intermediate frame time t **do**

for $i=1$ to number of line-pairs **do**

$L[i] = \text{line } t\text{-th of the way from } L_0[i] \text{ to } L_1[i].$

end

 Warp₀ = WarpImage(Image₀, L₀[...], L[...])

 Warp₁ = WarpImage(Image₁, L₁[...], L[...])

foreach pixel p in FinalImage **do**

$\text{FinalImage}(p) = (1-t) \text{ Warp}_0(p) + t \text{ Warp}_1(p)$

end

end

end

Animated sequences

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

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.