Image Filtering, Warping, and Morphing

Michael Kazhdan

(601.457/657)

Outline

- Image Filtering
- Image Warping
- Image Morphing

Image Filtering

- What about the case when the modification that we would like to make to a pixel depends on the pixels around it?
  - Blurring
  - Edge Detection
  - Etc.

Multi-Pixel Operations

Stationary/Local Filtering

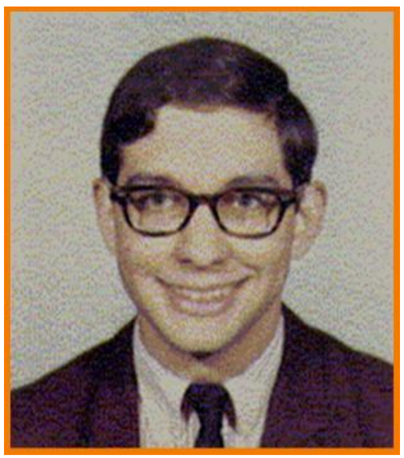
A general approach is to:

1. Define a *mask* of weights telling us how values at adjacent pixels should be combined to generate the new value.
2. Apply the (same) mask at every pixel.*

*Care is needed around at boundary pixels.

Blurring

- To blur across pixels, define a mask:
  - Whose entries sum to one
  - Whose value is larger near the center of the mask
  - Whose values are non-negative



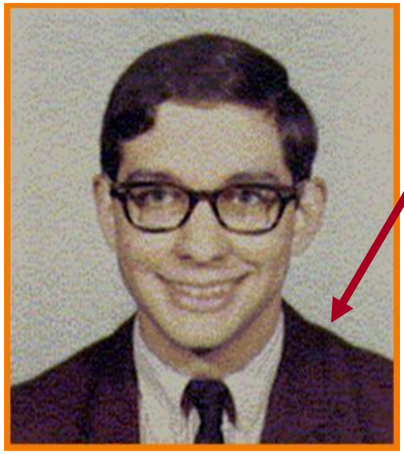
Original

Blur

$$\text{Filter} = \begin{bmatrix} 1/16 & 2/16 & 1/16 \\ 2/16 & 4/16 & 2/16 \\ 1/16 & 2/16 & 1/16 \end{bmatrix}$$

Blurring

Pixel(x,y): red = 36
green = 36
blue = 0

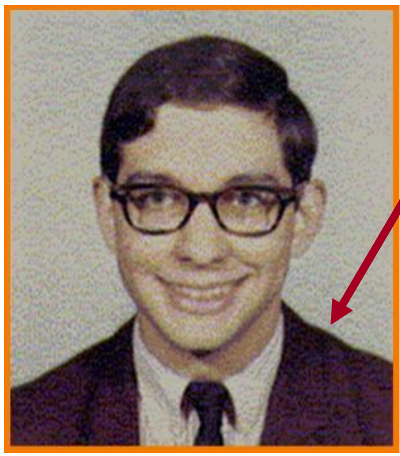


Original

$$\text{Filter} = \begin{bmatrix} \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\ \frac{2}{16} & \frac{4}{16} & \frac{2}{16} \\ \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \end{bmatrix}$$

Blurring

Pixel(x,y): red = 36
green = 36
blue = 0



Original

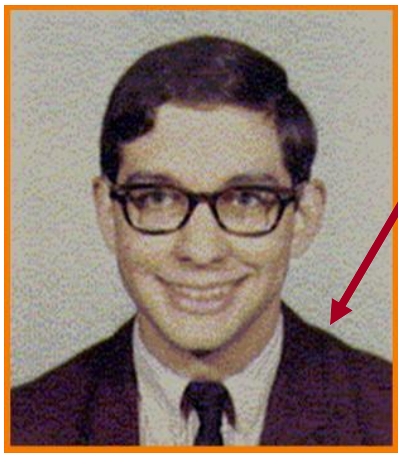
	X - 1	X	X + 1
Y - 1	36	109	146
Y	32	36	109
Y + 1	32	36	73

Pixel(x,y).red and its red neighbors

$$\text{Filter} = \begin{bmatrix} \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\ \frac{2}{16} & \frac{4}{16} & \frac{2}{16} \\ \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \end{bmatrix}$$

Blurring

New value for Pixel(x,y).red =
(36 * 1/16) + (109 * 2/16) + (146 * 1/16)
(32 * 2/16) + (36 * 4/16) + (109 * 2/16)
(32 * 1/16) + (36 * 2/16) + (73 * 1/16)



Original

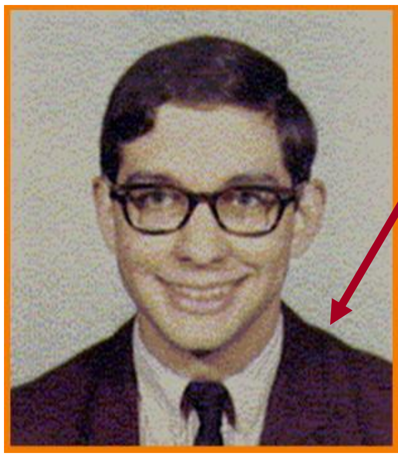
	X - 1	X	X + 1
Y - 1	36	109	146
Y	32	36	109
Y + 1	32	36	73

Pixel(x,y).red and its red neighbors

$$\text{Filter} = \begin{bmatrix} \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\ \frac{2}{16} & \frac{4}{16} & \frac{2}{16} \\ \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \end{bmatrix}$$

Blurring

New value for Pixel(x,y).red = 62.69



Original

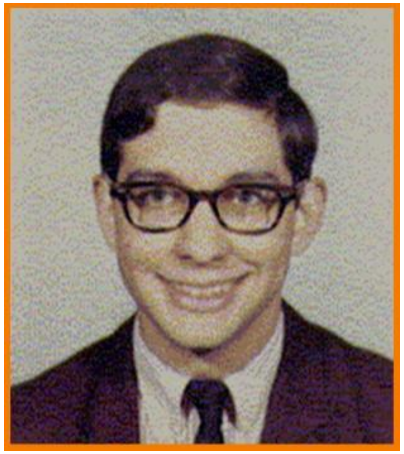
	X - 1	X	X + 1
Y - 1	36	109	146
Y	32	36	109
Y + 1	32	36	73

Pixel(x,y).red and its red neighbors

$$\text{Filter} = \begin{bmatrix} \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\ \frac{2}{16} & \frac{4}{16} & \frac{2}{16} \\ \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \end{bmatrix}$$

Blurring

New value for Pixel(x,y).red = 63



Original

Blur

$$\begin{bmatrix} \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\ \frac{2}{16} & \frac{4}{16} & \frac{2}{16} \\ \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \end{bmatrix}$$

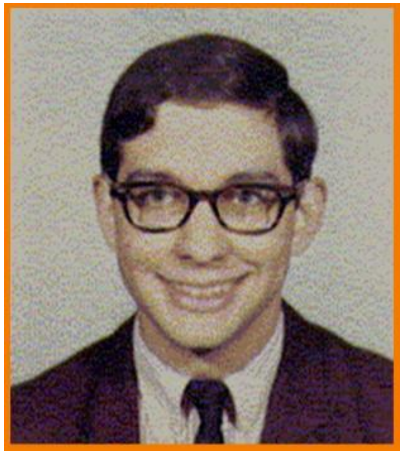
Blurring

- Repeat for each color channel of each pixel.
- Keep source and destination separate to avoid drift.
- For boundary pixels, not all neighbors are used:
  - » Normalize the mask so the values sum to one, or
  - » Assume that the exterior values are black, or
  - » Assume the exterior values can be obtained by reflecting the image across the boundary, or
  - » Assume…

Blurring

- Masks can have arbitrary size:
  - Can expand smaller masks by zero-padding

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 2 & 4 & 2 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} /16 \Leftrightarrow \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} /16$$



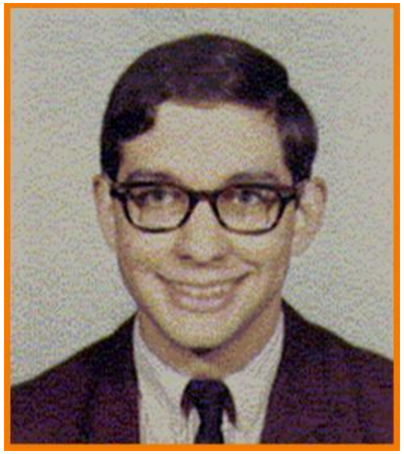
Original

Narrow Blur

Blurring

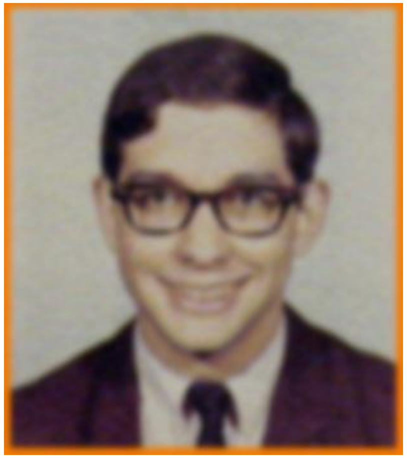
- Masks can have arbitrary size:
  - Can expand smaller masks by zero-padding
  - Can use more non-zero entries to get a wider blur

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 2 & 4 & 2 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} /16 \qquad \begin{bmatrix} 0 & 1 & 2 & 1 & 0 \\ 1 & 2 & 4 & 2 & 1 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \\ 0 & 1 & 2 & 1 & 0 \end{bmatrix} /48$$



Original

Narrow Blur



Wide Blur

Blurring

A general way for defining the entries of an $n \times n$ mask is to use the values of a Gaussian:

$$\text{GaussianMask}[i][j] \sim e^{-\frac{(i-r)^2+(j-r)^2}{4r^2}}$$

$$i, j \in [0, 2r]$$

- r is the (integer) mask radius
- $n = 2r + 1$ is the width
- $0 \leq i \leq 2r$ is the horizontal position in the mask
- $0 \leq j \leq 2r$ is the vertical position in the mask
- **Don't forget to normalize!**

Note:

- The center of the mask is at index (r, r).
- The Gaussian itself has standard deviation $\sigma = \sqrt{2} \cdot r$

Edge Detection

An edge is where the image is far from constant:

⇒ The difference between the value at the pixel and the average value of neighboring pixels is large (in absolute value)

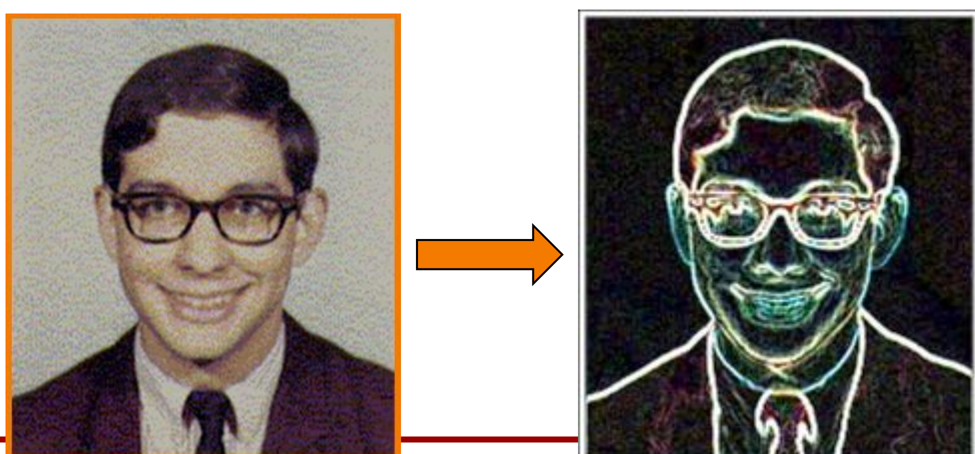
Edge Detection

An edge is where the image is far from constant:

⇒ The difference between the value at the pixel and the average value of neighboring pixels is large (in absolute value)

Define a mask whose:

- Entries sum to zero
- Value is one at the center pixel



Filter $= \frac{1}{8}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$

Edge Detection

An edge is where the image is far from constant:

⇒ The difference between the value at the pixel and the average value of neighboring pixels is large (in absolute value)

Define a mask whose:

- Entries sum to zero
- Value is one at the center pixel

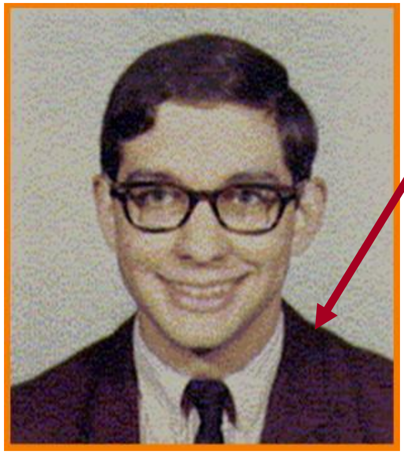
$$\text{Filter} = \frac{1}{\;}\begin{bmatrix} -1 & -1 & -1 \\ & & \\ & & \end{bmatrix}$$

Pixels with large absolute values correspond to edges:

- Positive values: “upper” edges
- Negative values: “lower” edges

Edge Detection

Pixel(x,y): red = 36
green = 36
blue = 0



Original

Filter = $\frac{1}{8}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$

Edge Detection

Pixel(x,y): red = 36
green = 36
blue = 0

Original

	X - 1	X	X + 1
Y - 1	36	109	146
Y	32	$\boxed{36}$	109
Y + 1	32	36	73

Pixel(x,y).red and its red neighbors

$$\text{Filter} = \frac{1}{8}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection

New value for Pixel(x,y).red =
(36 * -1/8) + (109 * -1/8) + (146 * -1/8)
(32 * -1/8) + (36 * 1) + (109 * -1/8)
(32 * -1/8) + (36 * -1/8) + (73 * -1/8)

Original

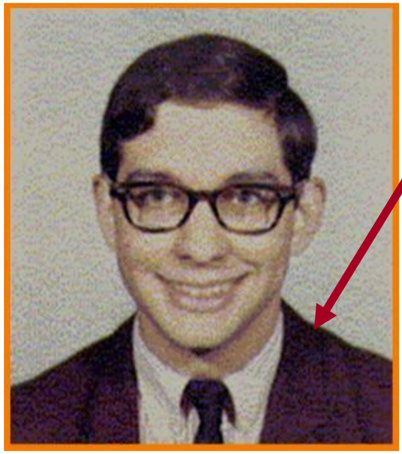
	X - 1	X	X + 1
Y - 1	36	109	146
Y	32	36	109
Y + 1	32	36	73

Pixel(x,y).red and its red neighbors

$$\text{Filter} = \frac{1}{8}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection

New value for Pixel(x,y).red = -285/8



Original

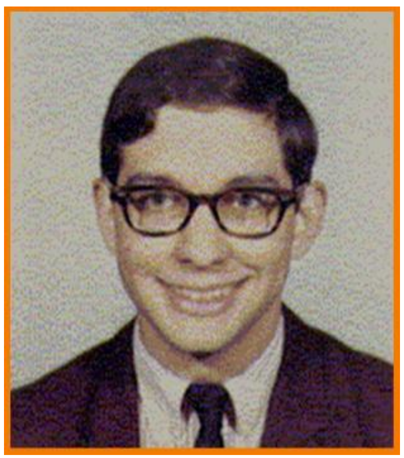
	X - 1	X	X + 1
Y - 1	36	109	146
Y	32	$\boxed{36}$	109
Y + 1	32	36	73

Pixel(x,y).red and its red neighbors

$$\text{Filter} = \frac{1}{8}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection

New value for Pixel(x,y).red = -35.625



Original

Detected Edges

Filter = $\frac{1}{8}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$

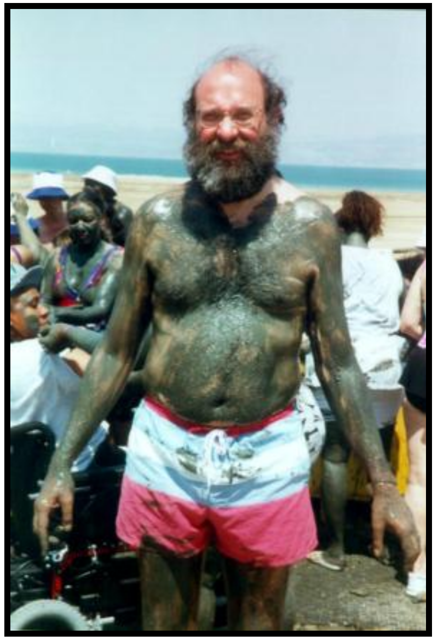
<u>Note</u>:
Output values are not colors, so need to find a way to remap for visualization.

Outline

- Image Filtering
- Image Warping
- Image Morphing

Image Warping

- Move pixels of image
  - Mapping
  - Resampling



Warp

Source image

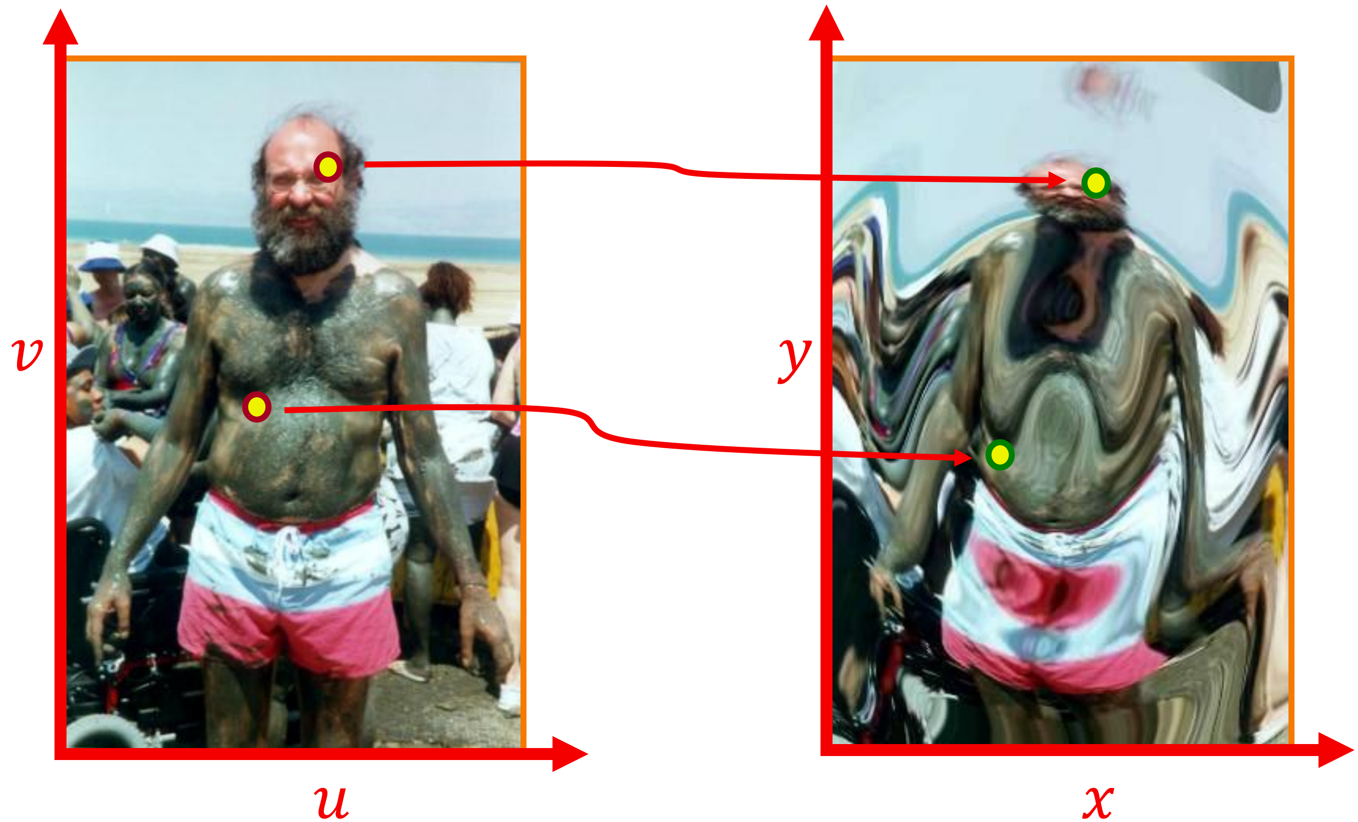
Destination image

Overview

- Mapping
  - Forward
  - Inverse
- Resampling
  - Point sampling
  - Triangle filter
  - Gaussian filter

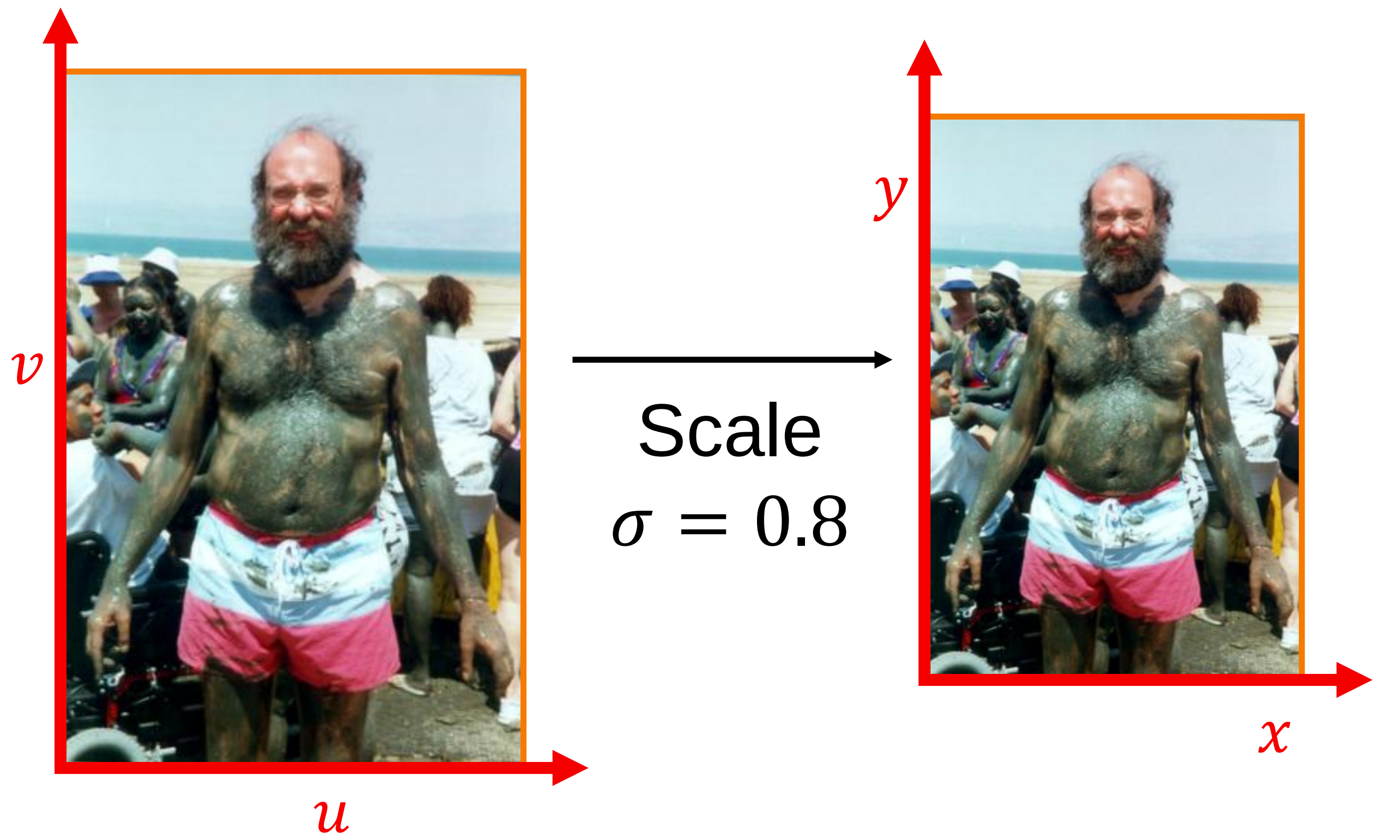
Mapping

- Define transformation
  - Describe the destination $(x, y) = \Phi(u, v)$ for every location (u, v) in the source



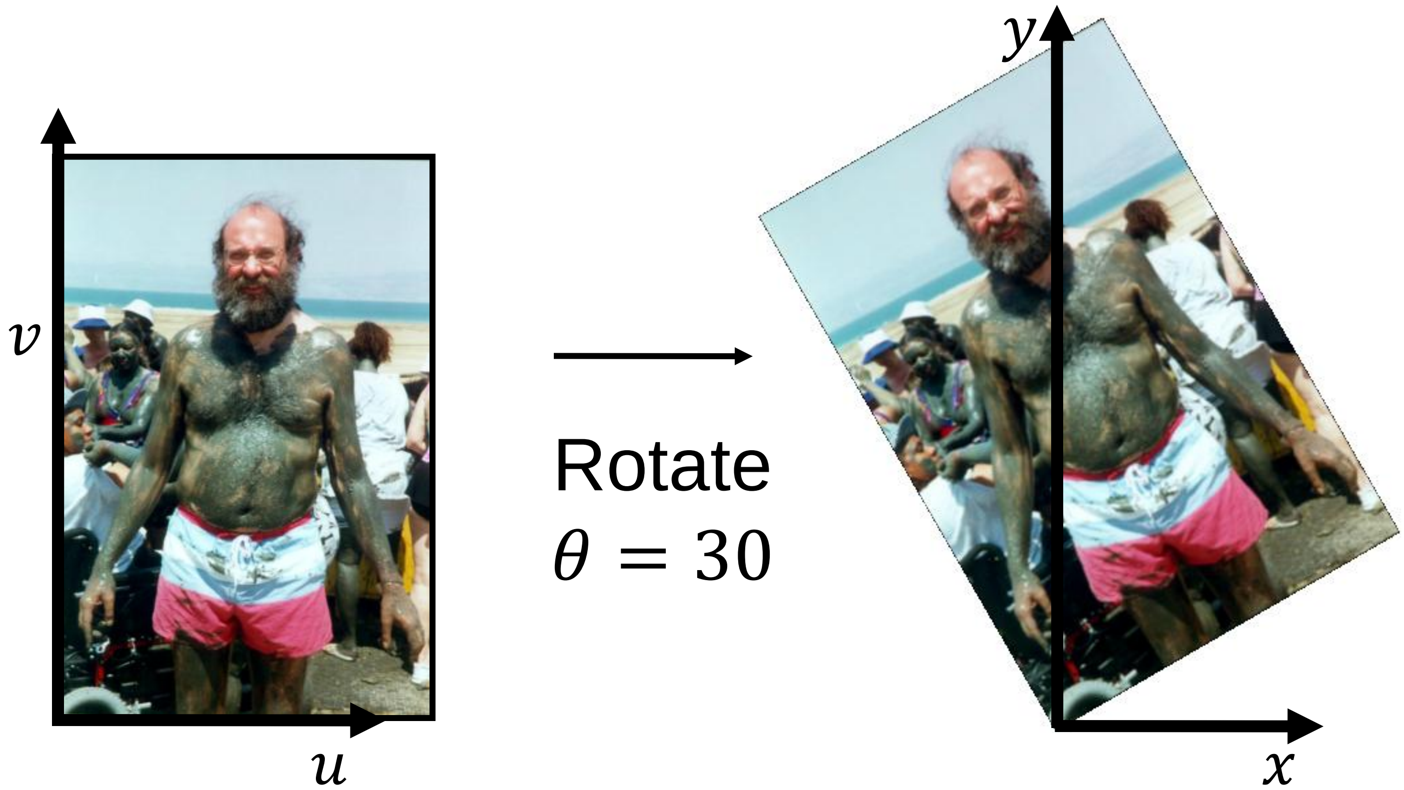
Example Mappings

- Scale by σ:
  - $\Phi(u, v) = (\sigma u, \sigma v)$



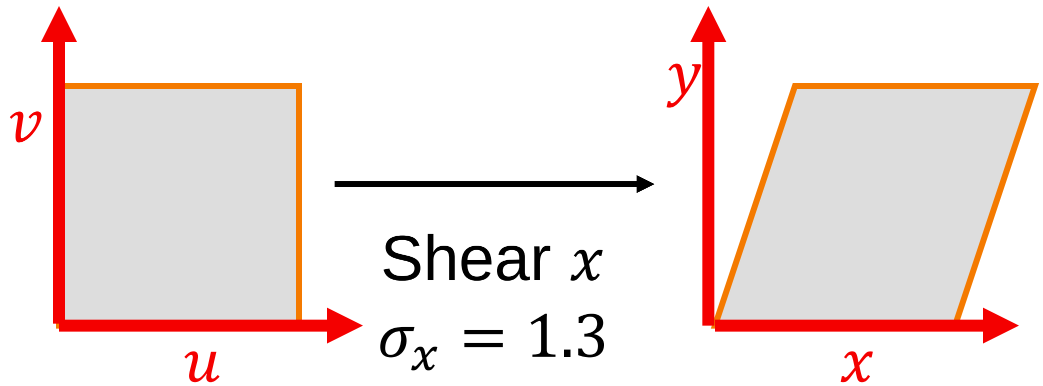
Example Mappings

- Rotate by θ degrees:
  - $\Phi(u, v) = (u\cos\theta - v\sin\theta\,, u\sin\theta + v\cos\theta)$

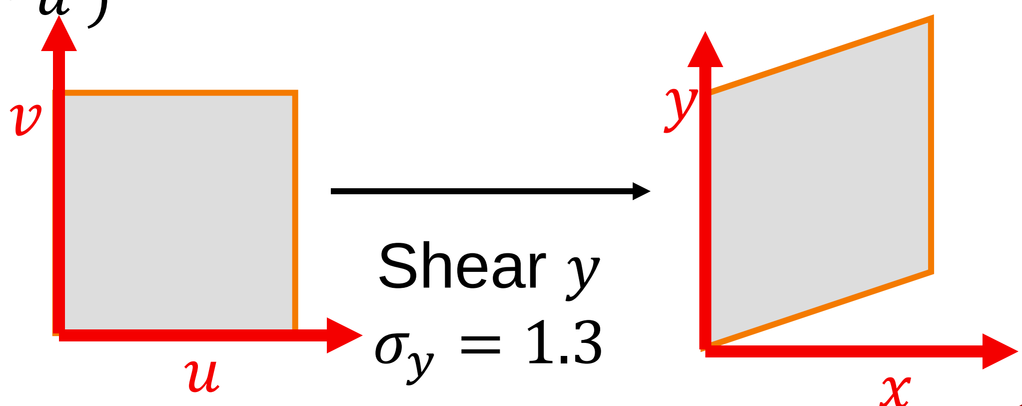


Example Mappings

- Shear in x by σ_x:
  - $\Phi(u, v) = (u + \sigma_x \cdot v, v)$

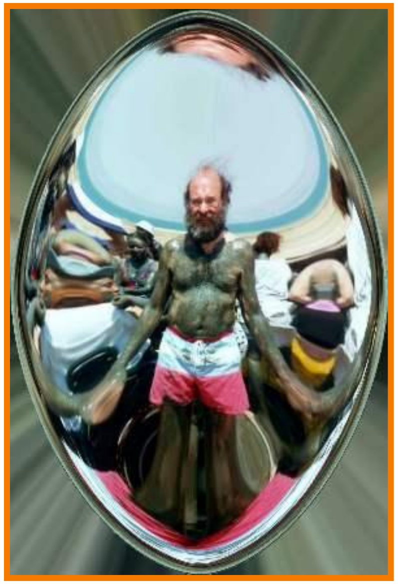


- Shear in y by σ_y:
  - $\Phi(u, v) = (u, v + \sigma_y \cdot u\ )$



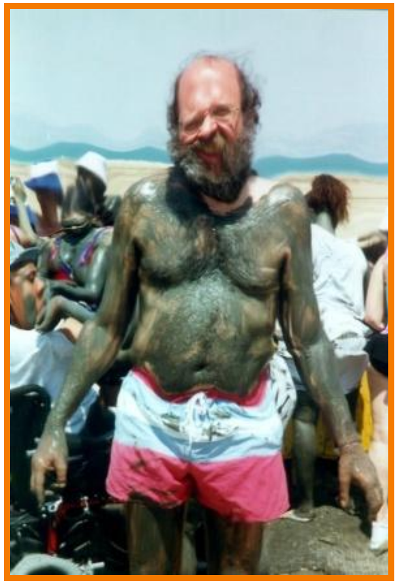
Other Mappings

- Any function of u and v:
  - $\Phi(u, v) = \cdots$



Fish-eye

"Swirl"

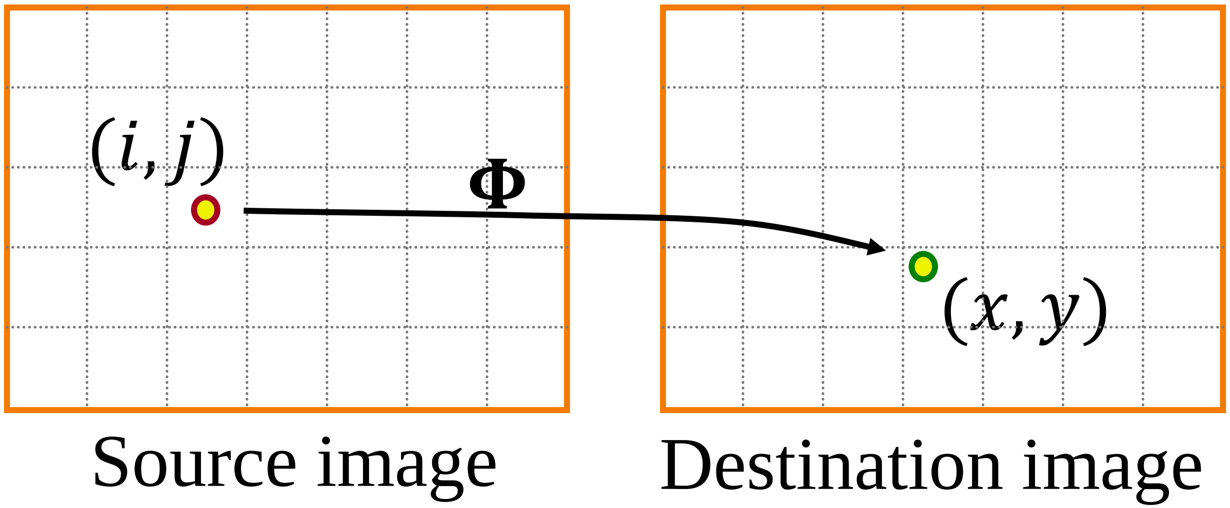


"Rain"

Image Warping Implementation I

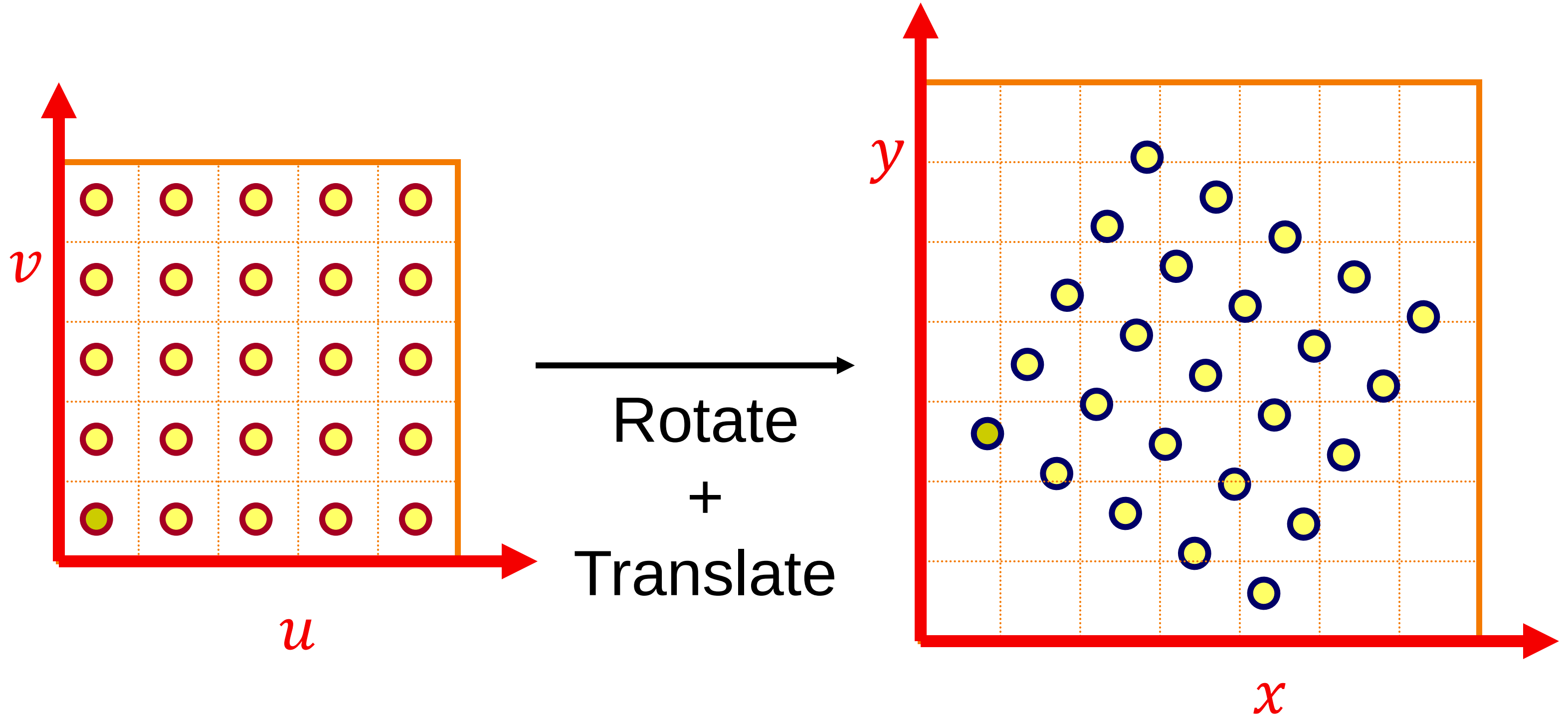
- Forward mapping:

```
for( j=0 ; j<srcHeight ; j++ )
  for( i=0 ; i<srcWidth ; i++ )
    (x,y) = Φ(i,j);
    dst(x,y) = src(i,j);
```



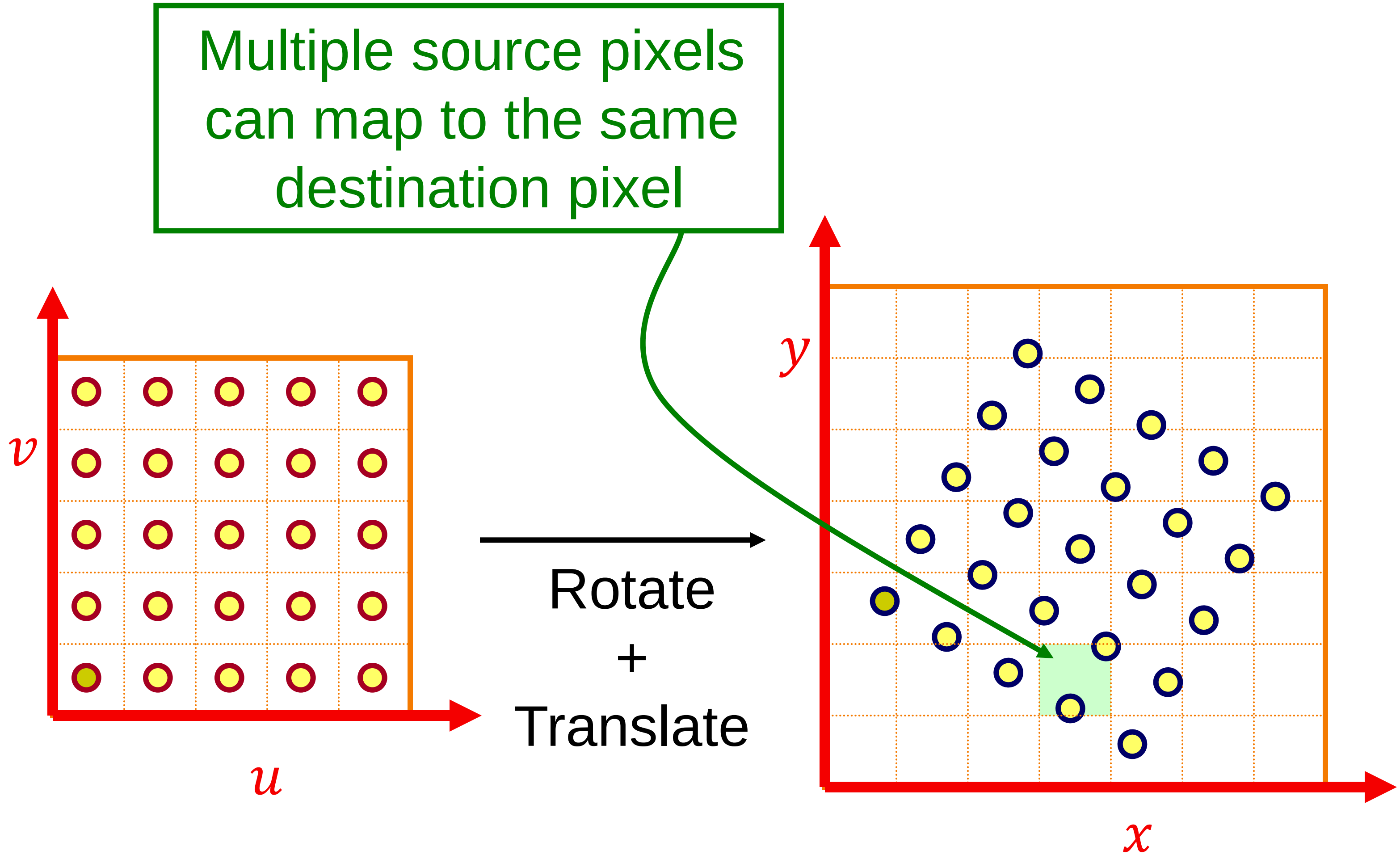
Forward Mapping

- Iterate over source image



Forward Mapping – BAD!

- Iterate over source image



Forward Mapping – BAD!

- Iterate over source image

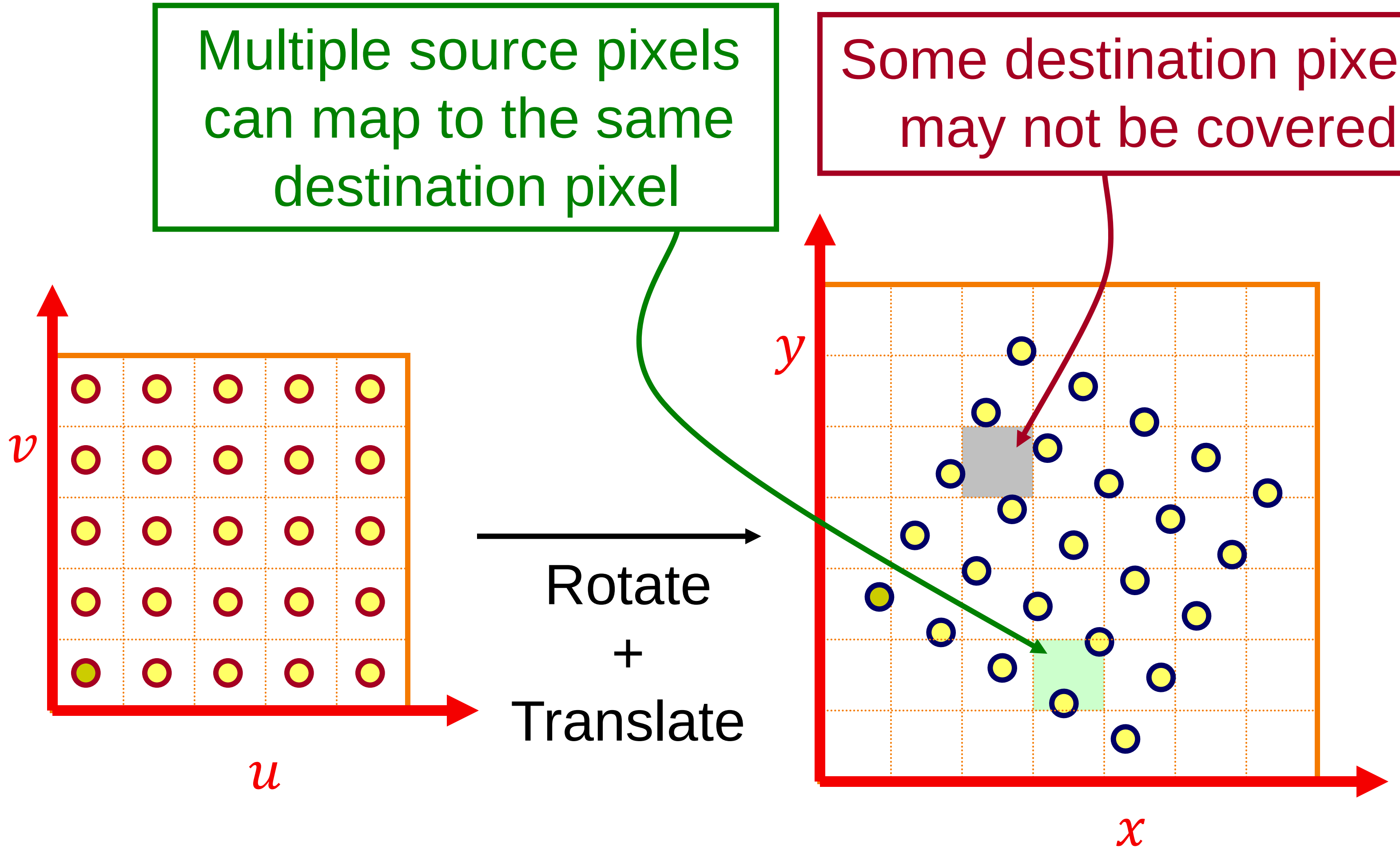
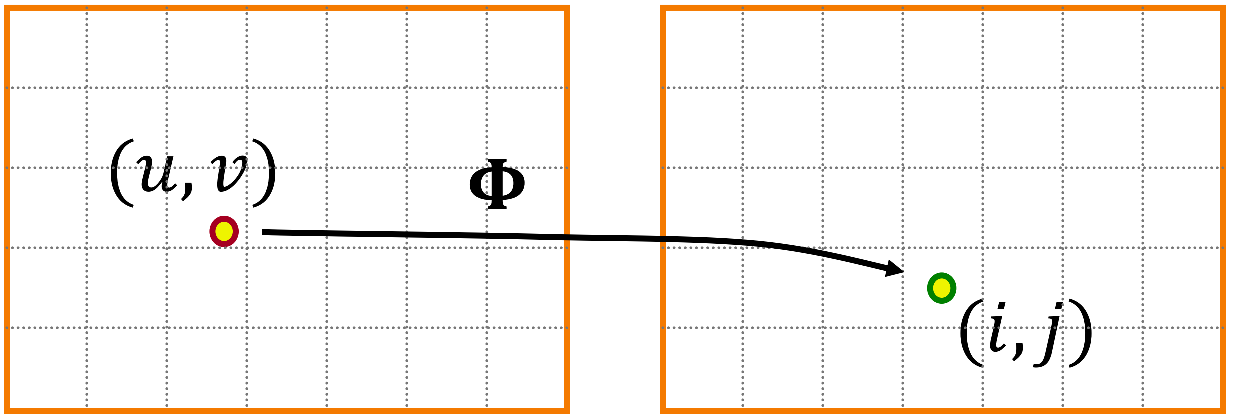


Image Warping Implementation II

- Inverse mapping:

```
for( j=0 ; j<dstHeight ; j++ )
   for( i=0 ; i<dstWidth ; i++ )
      (u,v) = Φ^-1(i,j);
      dst(i,j) = src(u,v);
```

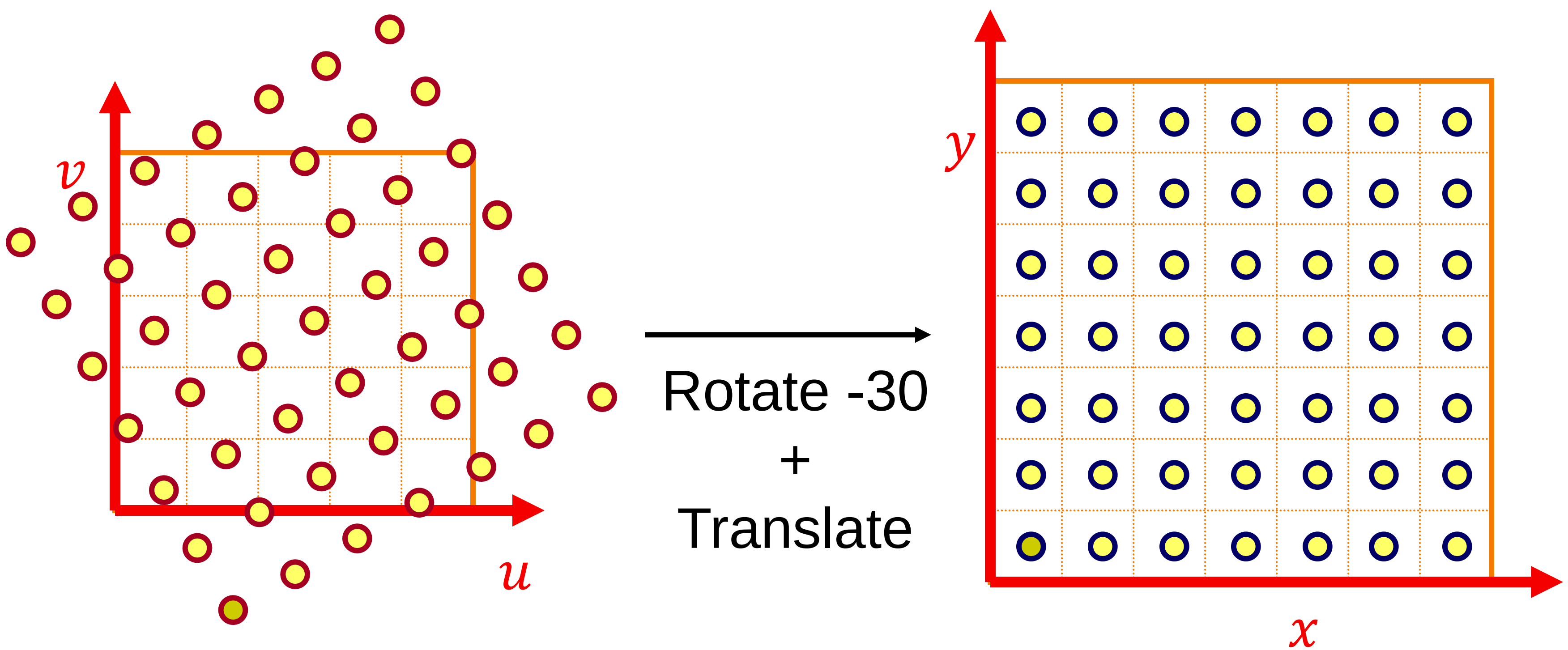


Source image

Destination image

Reverse Mapping – GOOD!

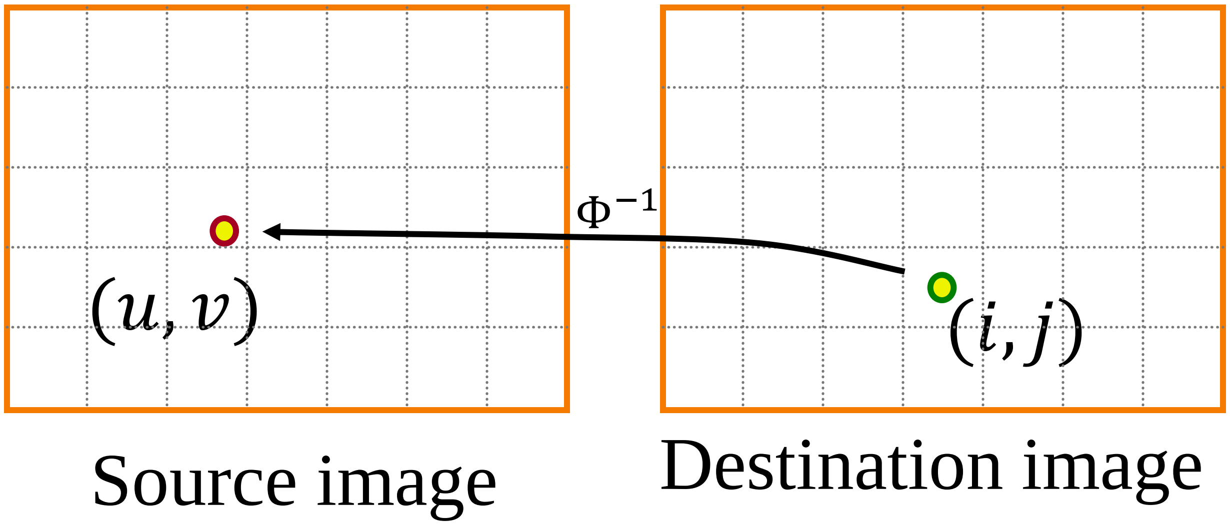
- Iterate over destination image
  - Must resample source



Resampling

- Evaluate source image at $(u, v) = \Phi^{-1}(i, j)$

(u, v) does not usually have integer coordinates



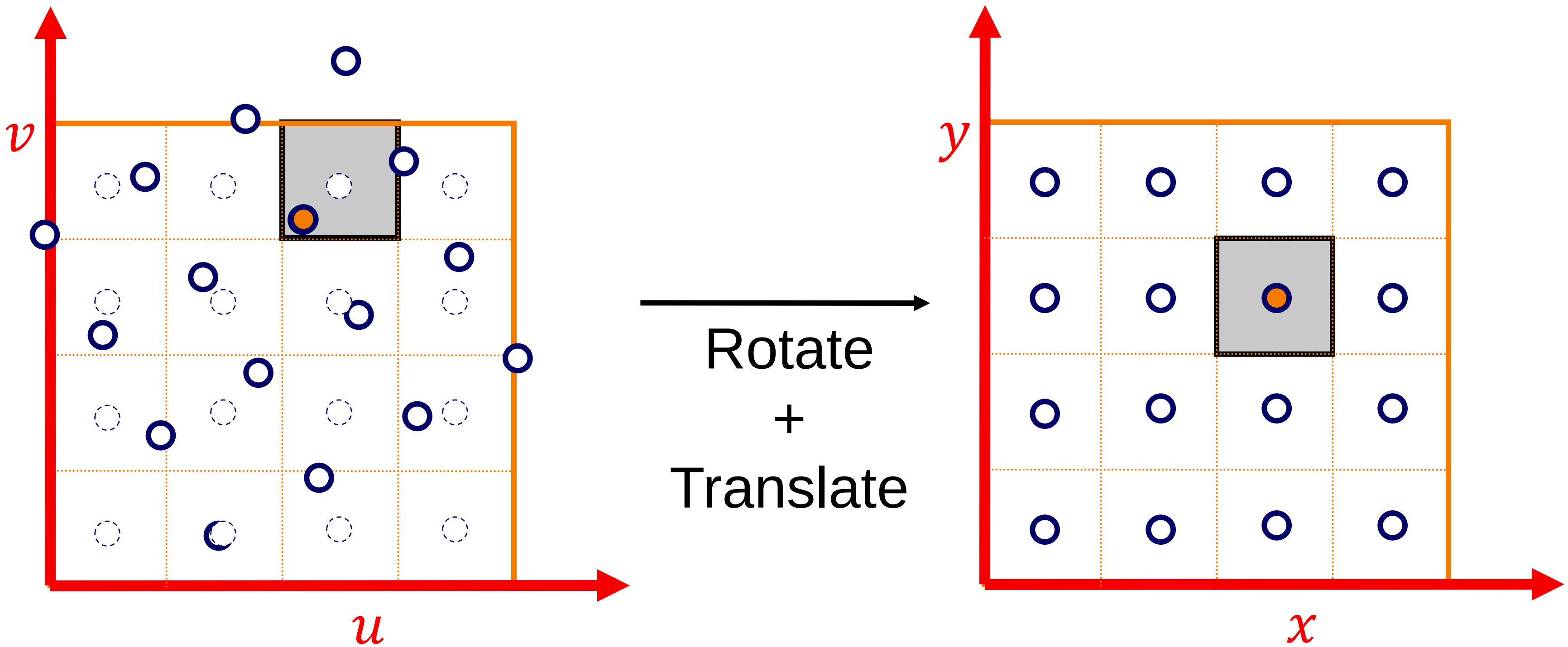
Overview

- Mapping
  - Forward
  - Inverse
- Resampling
  - Nearest Point Sampling
  - Bilinear Sampling
  - Gaussian Sampling

Nearest Point Sampling

- Take value at closest pixel:

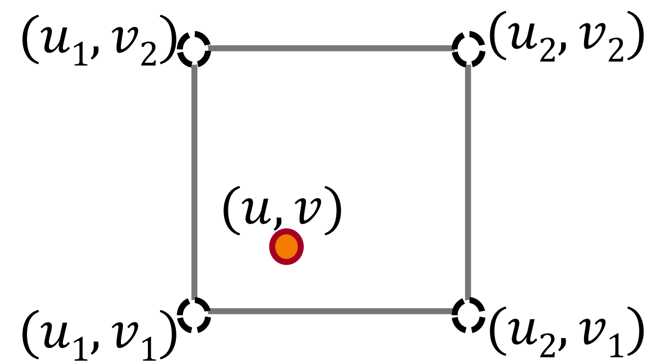
```
int intU = floor(u+0.5);
int intV = floor(v+0.5);
dst(i,j) = src(intU,intV);
```

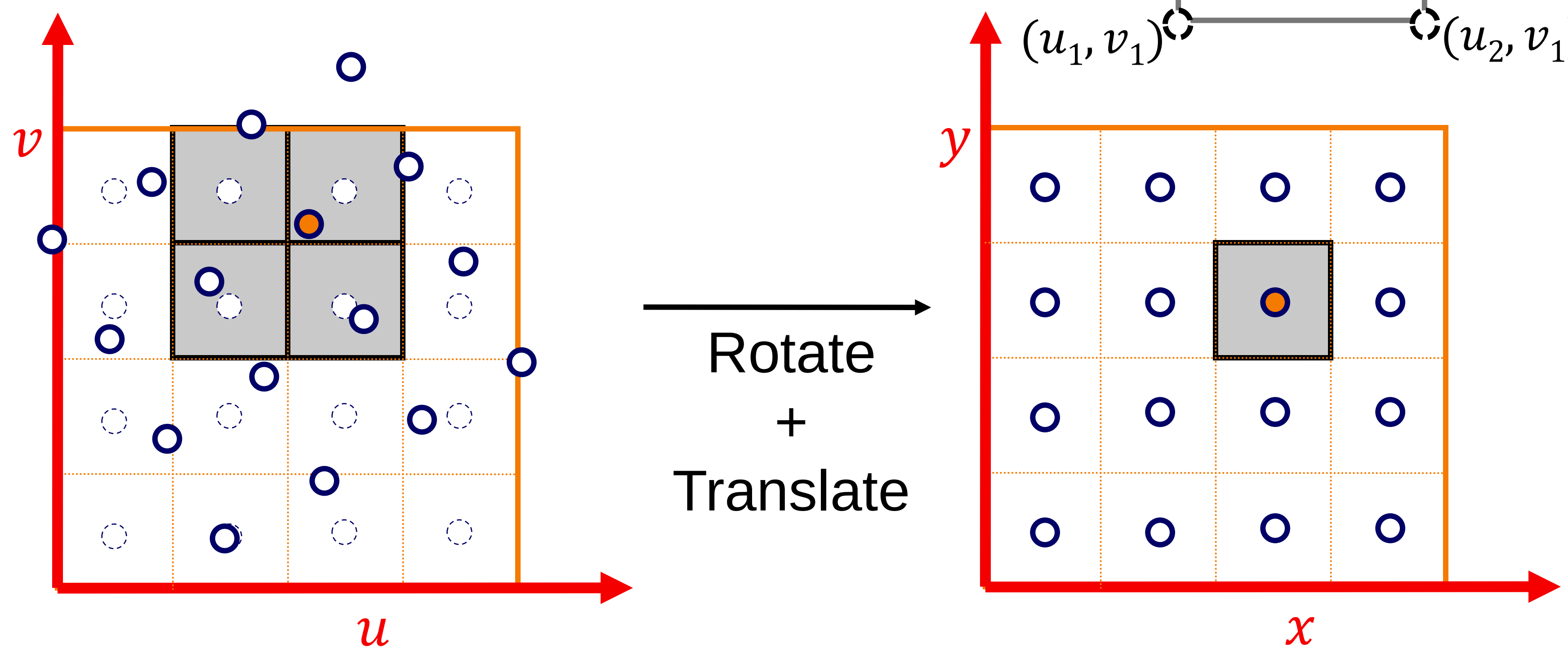


Bilinear Sampling

- Bilinearly interpolate four closest source pixels

$\mathrm{dst}(i, j)$ = Weighted average of source at
(u_1, v_1), (u_2, v_1), (u_1, v_2), and (u_2, v_2)

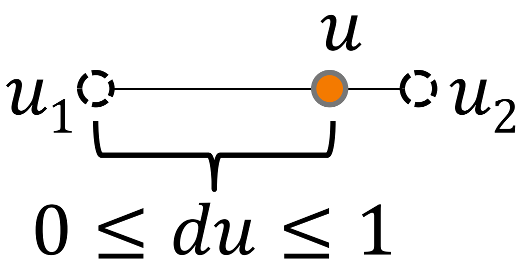




Linear Sampling

- Linearly interpolate two closest source pixels

  dst(i) = linear interpolation of u_1 and u_2




```
u = Φ^{-1}(i)
u1 = floor(u);
u2 = u1 + 1;
du = u - u1;
dst(i) = src(u1)*(1-du) + src(u2)*du;
```

Bilinear Sampling

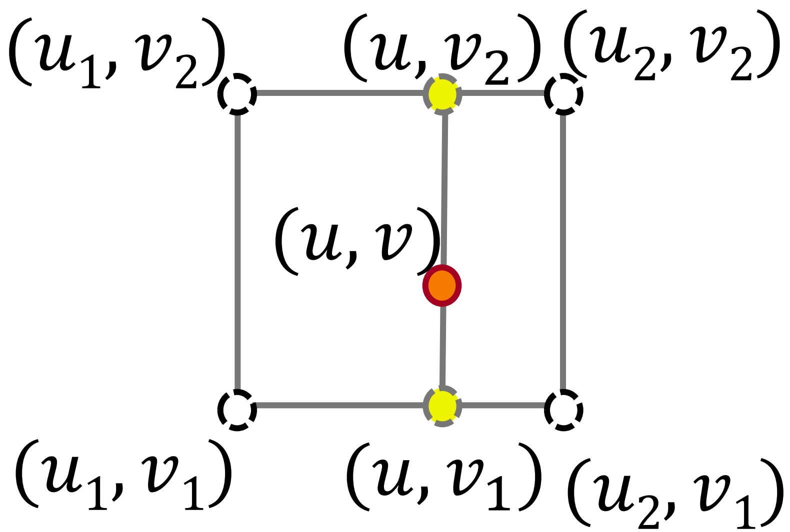
- Bilinearly interpolate four closest source pixels

  a = linear interpolation of $\mathrm{src}(u_1, v_1)$ and $\mathrm{src}(u_2, v_1)$

  b = linear interpolation of $\mathrm{src}(u_1, v_2)$ and $\mathrm{src}(u_2, v_2)$

  $\mathrm{dst}(i, j)$ = linear interpolation of a and b

```
(u,v) = Φ^{-1}(i,j)

u1 = floor( u ) , u2 = u1 + 1;
v1 = floor( v ) , v2 = v1 + 1;

du = u - u1;

a = src(u1,v1)*(1-du)
  + src(u2,v1)*(  du);

b = src(u1,v2)*(1-du)
  + src(u2,v2)*du;

dv = v - v1;
dst(i,j) = a*(1-dv) + b*dv;
```



Bilinear Sampling

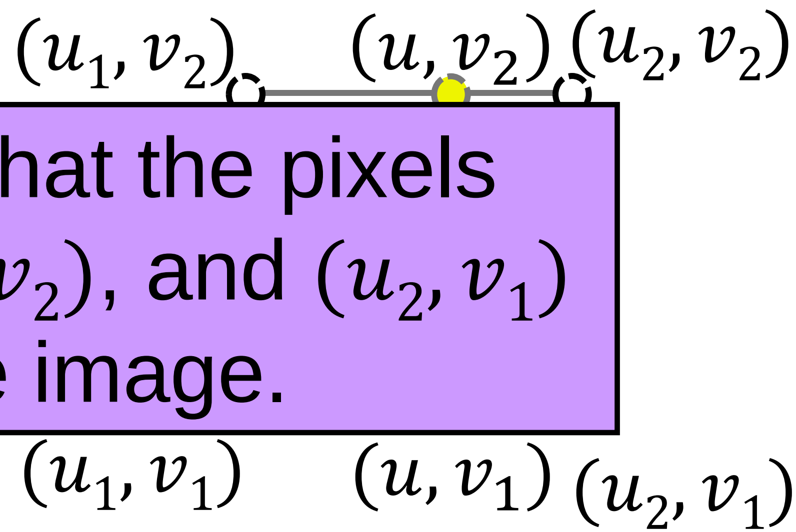
- Bilinearly interpolate four closest source pixels

  a = linear interpolation of $\mathrm{src}(u_1, v_1)$ and $\mathrm{src}(u_2, v_1)$

  b = linear interpolation of $\mathrm{src}(u_1, v_2)$ and $\mathrm{src}(u_2, v_2)$

  $\mathrm{dst}(i, j)$ = linear interpolation of a and b

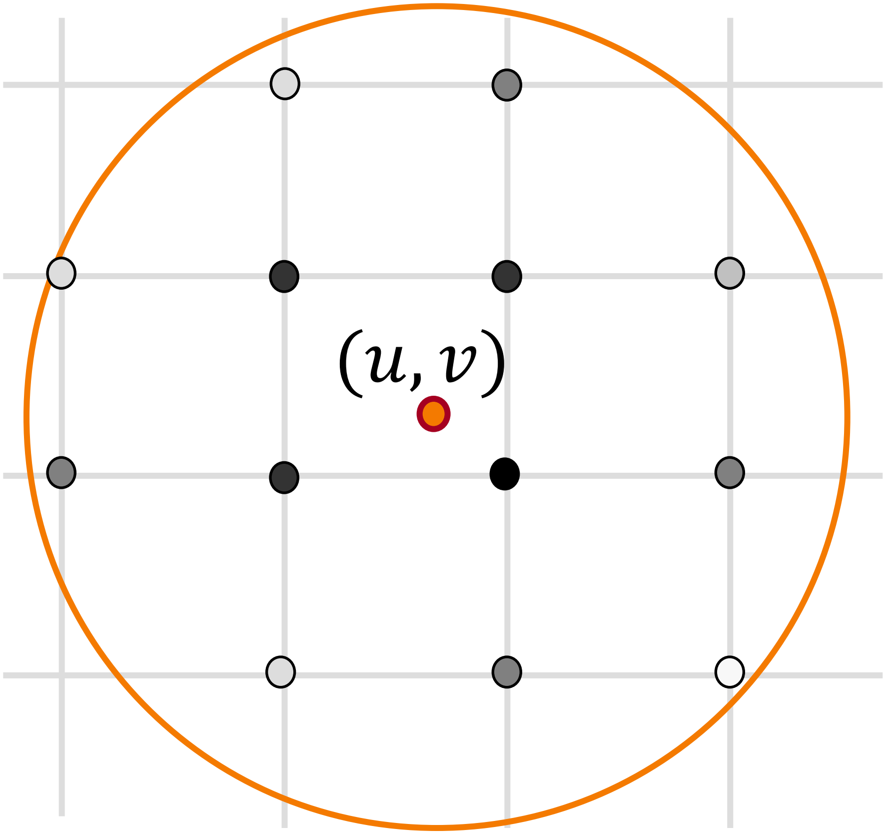
```
(u,v) = Φ^{-1}(i,j)

u1 =
v1 =

du =

a =
   + src(u2,v1)*(  du);

b = src(u1,v2)*(1-du)
   + src(u2,v2)*du;

dv = v - v1;
dst(i,j) = a*(1-dv) + b*dv;
```



Make sure to test that the pixels (u_1, v_1), (u_2, v_2), (u_1, v_2), and (u_2, v_1) are within the image.

Gaussian Sampling

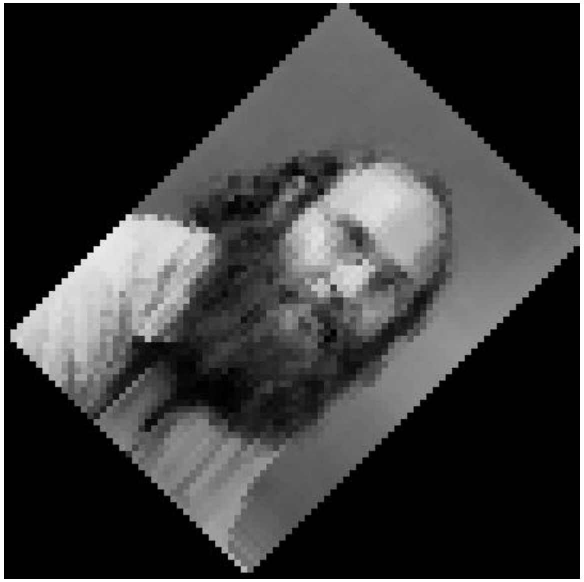
- Compute weighted sum of pixel neighborhood:
  - The blending weights are the normalized values of a Gaussian function.



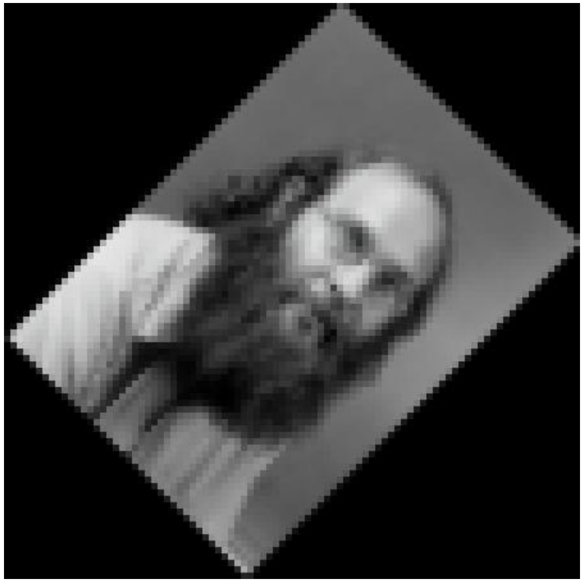
Filtering Methods Comparison

- Trade-offs
  - Jagged edges versus blurring
  - Computation speed

We'll talk more about trade-offs next time.



Nearest



Bilinear

Gaussian

Image Warping Implementation

- Inverse mapping:

```
for( j=0 ; j<dstHeight ; j++ )
  for( i=0 ; i<dstWidth ; i++ )
    (u,v) = Φ^-1(i,j);
    dst(i,j) = resample_src(u,v,r);
```

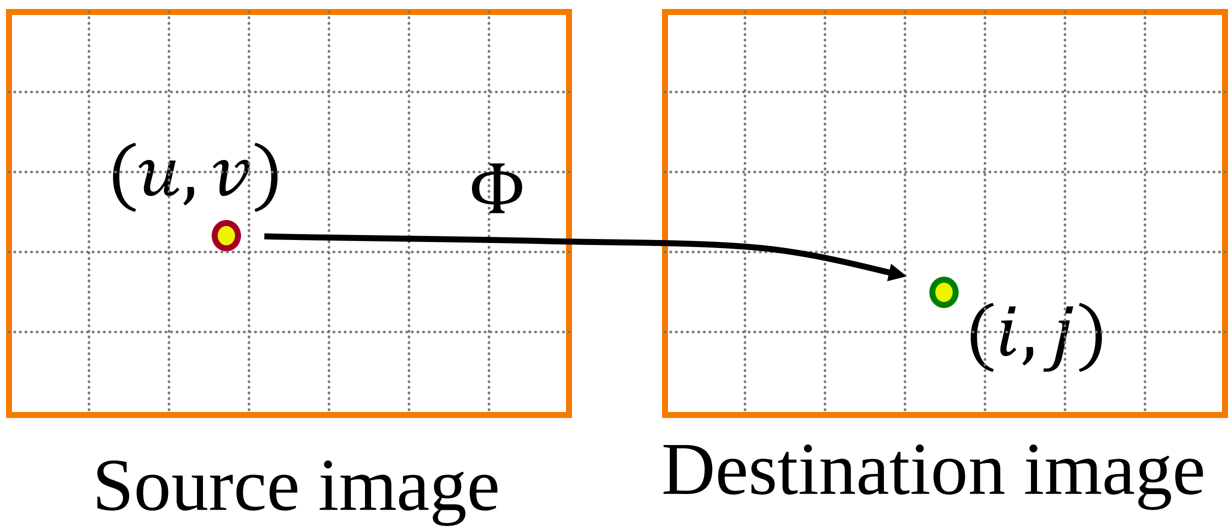
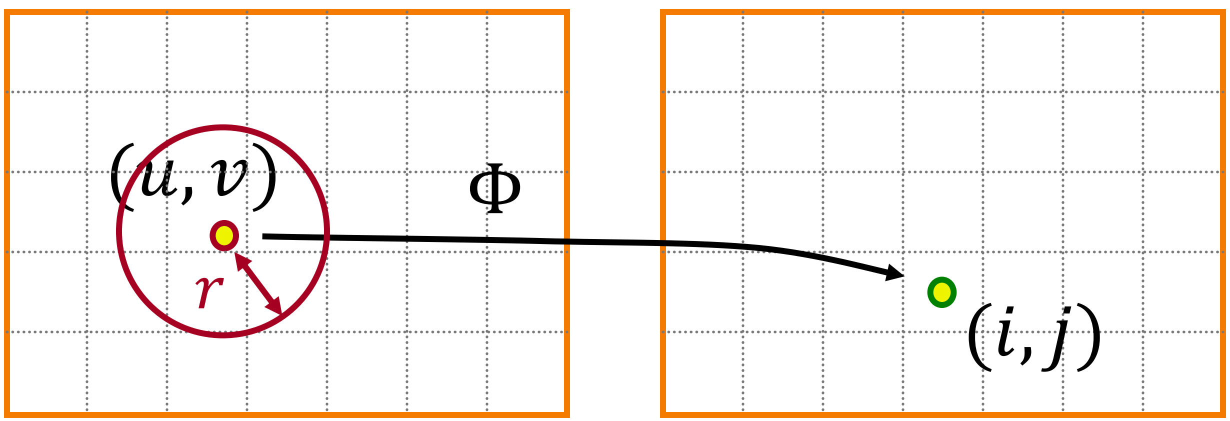


Image Warping Implementation

- Inverse mapping:

```
for( j=0 ; j<dstHeight ; j++ )
  for( i=0 ; i<dstWidth ; i++ )
    (u,v) = Φ^-1(i,j);
    dst(i,j) = resample_src(u,v,r);
```



Source image

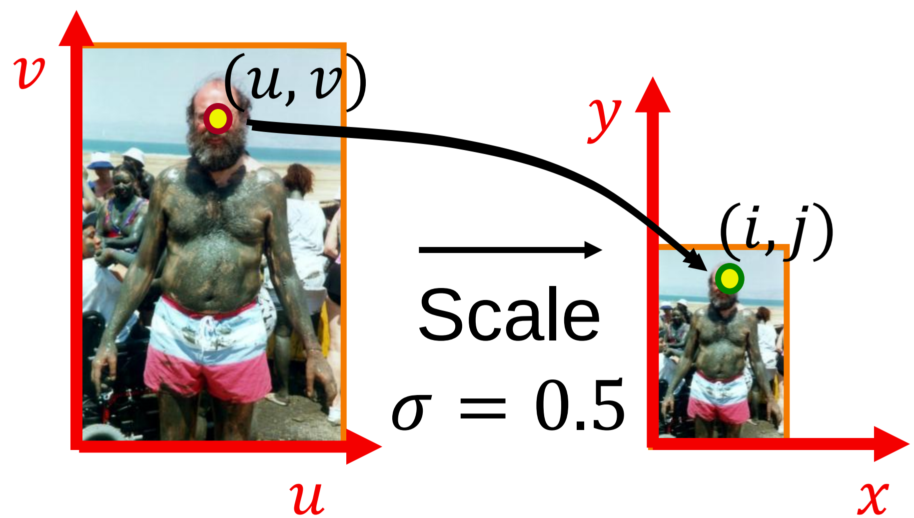
Destination image

Example: Scale

Scale(src , dst , σ):

```
r ≅ ?;
for( j=0 ; j<dstHeight ; j++ )
  for( i=0 ; i<dstWidth ; i++ )
    (u,v) = (i,j) / σ;
    dst(i,j) = resample_src(u,v,r);
```

$$\boxed{r = \frac{1}{\sigma}}$$

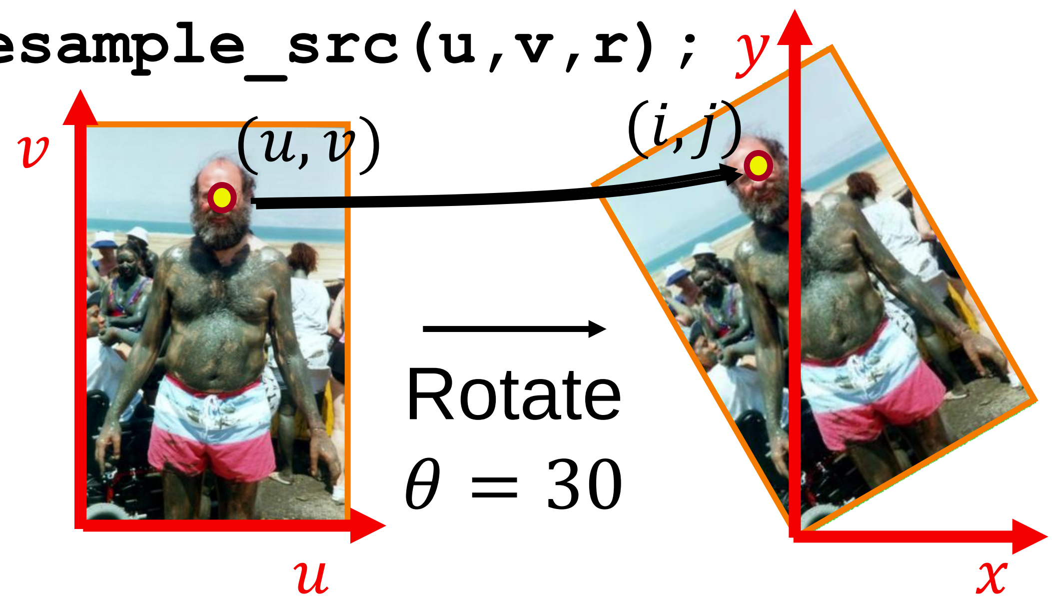


Example: Rotate

Rotate(src , dst , θ):

```
r ≅ ?;
for( j=0 ; j<dstHeight ; j++ )
   for( i=0 ; i<dstWidth ; i++ )
      (u,v) = ( i*cos(-θ) - j*sin(-θ) ,
                i*sin(-θ) + j*cos(-θ) );
      dst(x,y) = resample_src(u,v,r);
```

$\boxed{r = 1}$

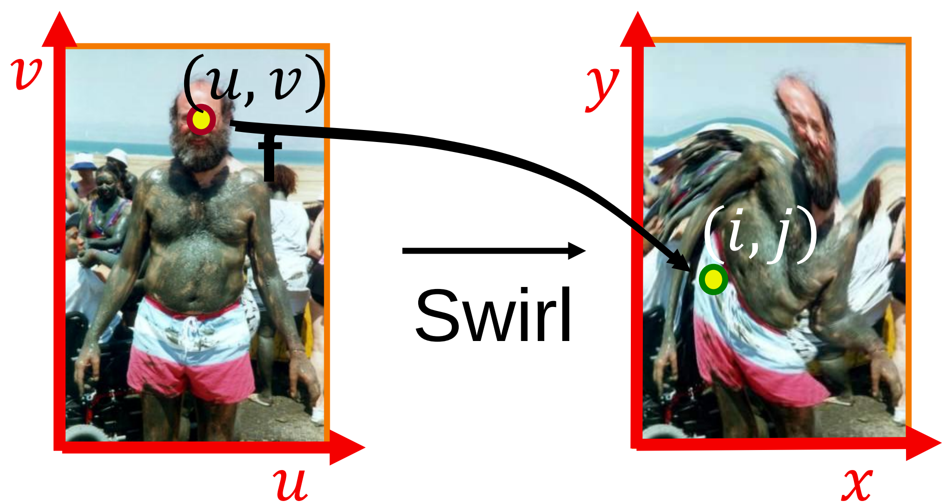


Example:

General(src , dst , Φ):

```
r ≅ ?;
for( j=0 ; j<dstHeight ; j++ )
  for( i=0 ; i<dstWidth ; i++ )
    (u,v) = Φ-1(i,j);
     dst(i,j) = resample_src(u,v,r);
```

$$\boxed{r = ?}$$



Example:

General(src , dst , Φ):

```
r ≅ ?;
for( j=0 ; j<dstHeight ; j++ )
   for( i=0 ; i<dstWidth ; i++ )
      (u,v) = Φ⁻¹(i,j);
       dst(i,j) = resample_src(u,v,r);
```

Instead of using a fixed radius circle to sample the source, we can:

1. Have the radius changes
2. Use an ellipse.

For example, the parameters can be determined by looking at the derivative/Jacobean of Φ.

Outline

- Image Filtering
- Image Warping
- Image Morphing

Image Warping

Recall:

1. For a vector $\vec{v} = (x, y)$, its $90°$ (ccw) rotation is given by:
$$\vec{v}^{\perp} = (-y, x)$$

Image Warping

Recall:

1. For a vector $\vec{v} = (x, y)$, its $90°$ (ccw) rotation is given by:
$$\vec{v}^{\perp} = (-y, x)$$

2. For two vectors $\vec{v}_1 = (x_1, y_1)$ and $\vec{v}_2 = (x_2, y_2)$
$$\langle \vec{v}_1, \vec{v}_2 \rangle \equiv x_1 \cdot x_2 + y_1 \cdot y_2 = \|\vec{v}_1\| \cdot \|\vec{v}_2\| \cdot \cos\theta$$

$\Rightarrow$ The signed length of the projection of v_1 onto the line through v_2 is:
$$\|v_1\| \cdot \cos\theta = \frac{\langle v_1, v_2 \rangle}{\|v_2\|}$$

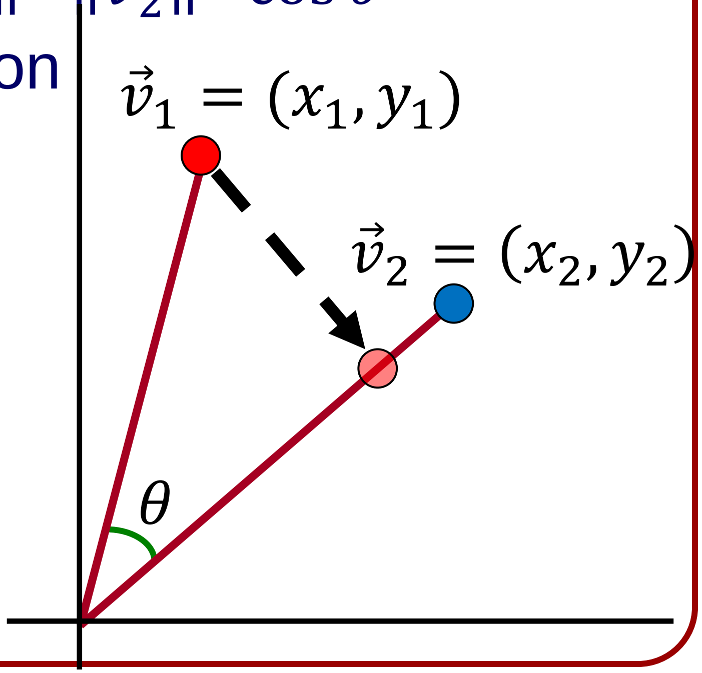


Image Warping

Recall:

1. For a vector $\vec{v} = (x, y)$, its $90°$ (ccw) rotation is given by:
$$\vec{v}^{\perp} = (-y, x)$$

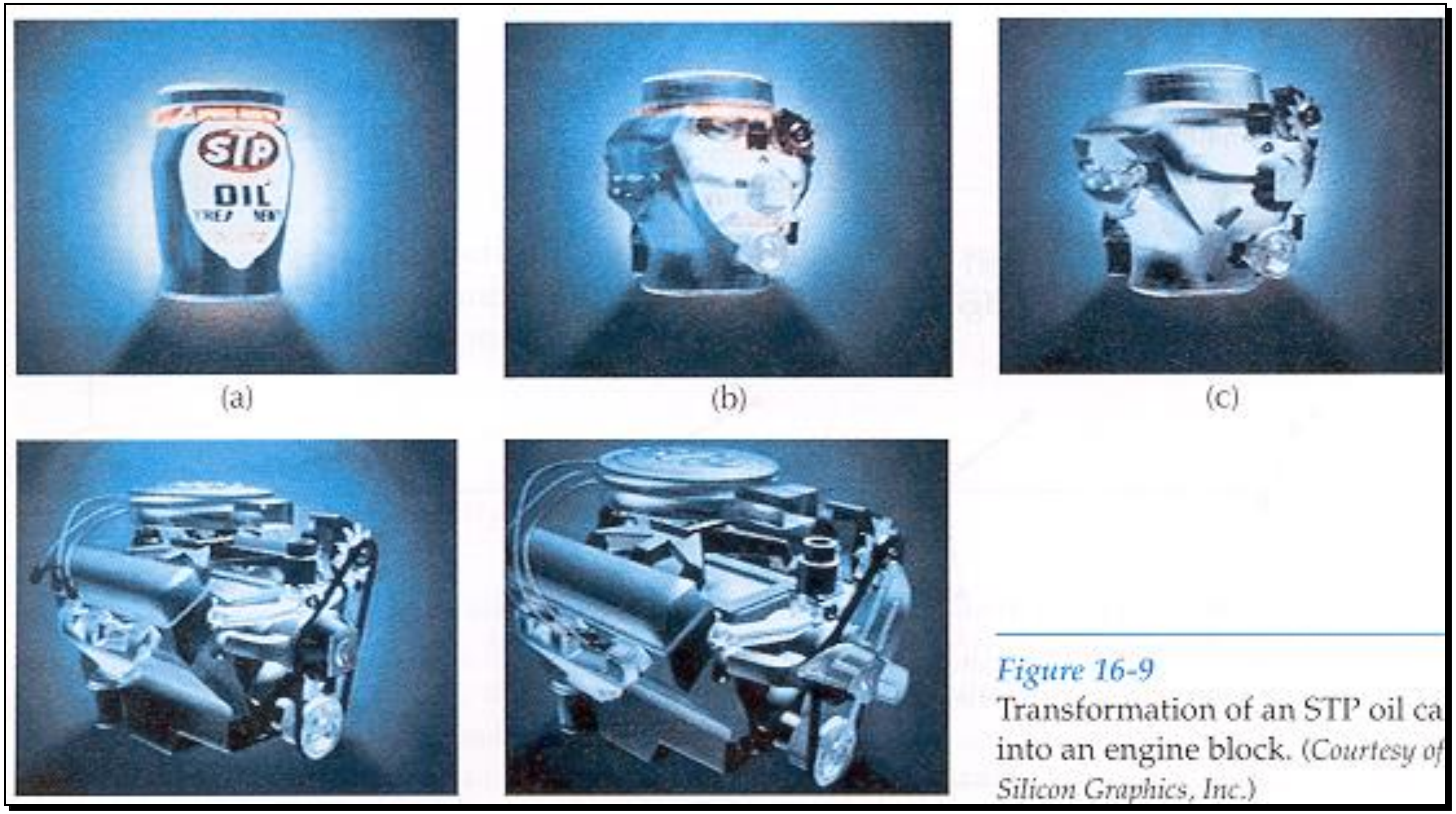
2. For two vectors $\vec{v}_1 = (x_1, y_1)$ and $\vec{v}_2 = (x_2, y_2)$
$$\langle \vec{v}_1, \vec{v}_2 \rangle \equiv x_1 \cdot x_2 + y_1 \cdot y_2 = \|\vec{v}_1\| \cdot \|\vec{v}_2\| \cdot \cos\theta$$

3. The inner-product of two vectors is *isometry-invariant*:
If $\mathbf{R} \in \mathbb{R}^{2\times 2}$ is a rotation, then:
$$\langle \mathbf{R}(\vec{v}_1), \mathbf{R}(\vec{v}_2) \rangle = \langle \vec{v}_1, \vec{v}_2 \rangle$$

$\Rightarrow$ In particular:
$$\left\langle \vec{v}_1^{\perp}, \vec{v}_2^{\perp} \right\rangle = \langle \vec{v}_1, \vec{v}_2 \rangle$$

Image Morphing

- Animate transition between two images



H&B Figure 16.9

Image Morphing

- Animate transition between two images

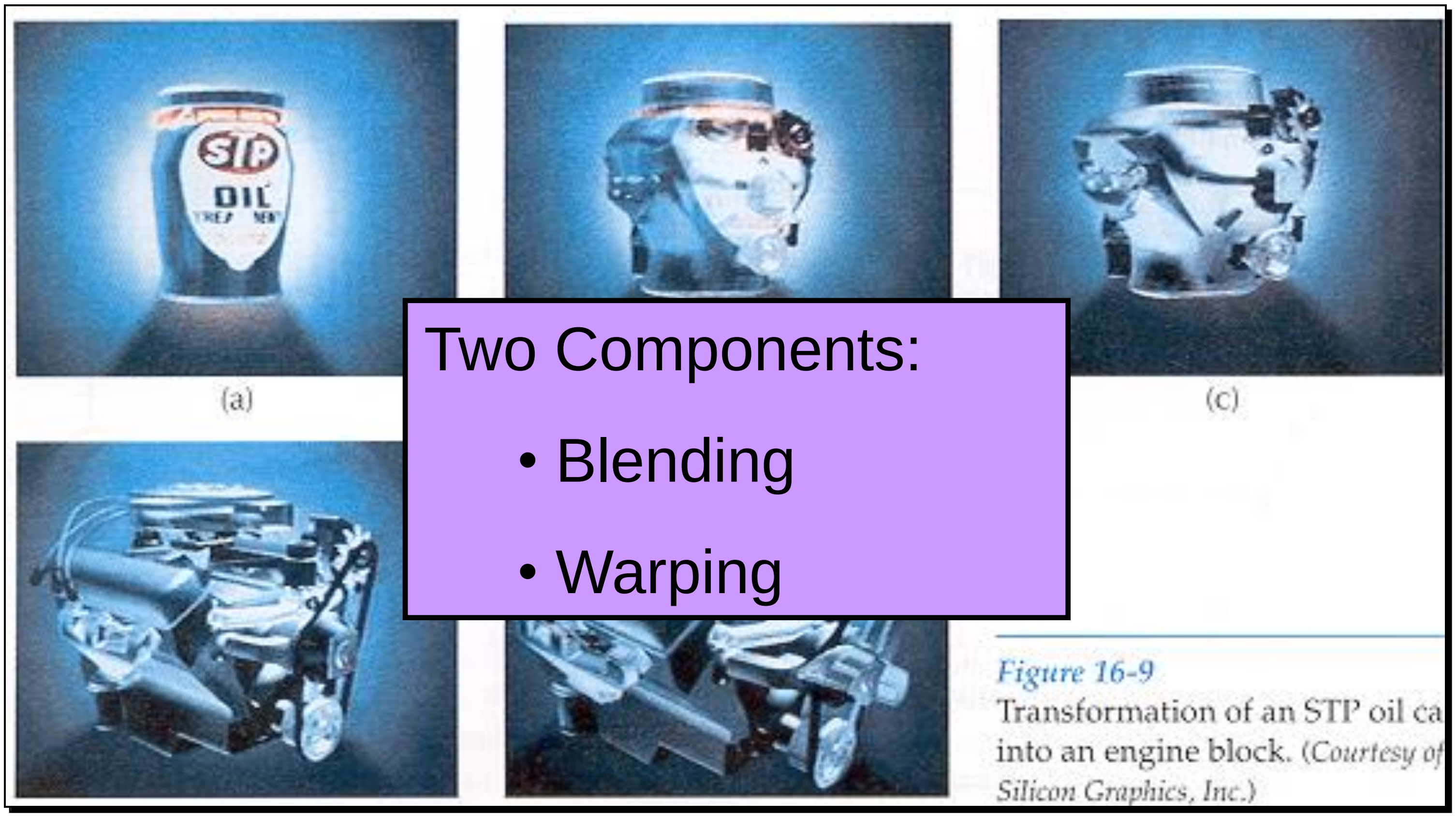


Figure 16-9
Transformation of an STP oil ca
into an engine block. (Courtesy of
Silicon Graphics, Inc.)

Two Components:

- Blending
- Warping

H&B Figure 16.9

Blending

Blend **colors** using an α-blend ($\alpha \in [0,1]$):

$$\text{blend}(i,j,\alpha) = (1-\alpha)\cdot \text{Img}_0(i,j) + \alpha \cdot \text{Img}_1(i,j)$$

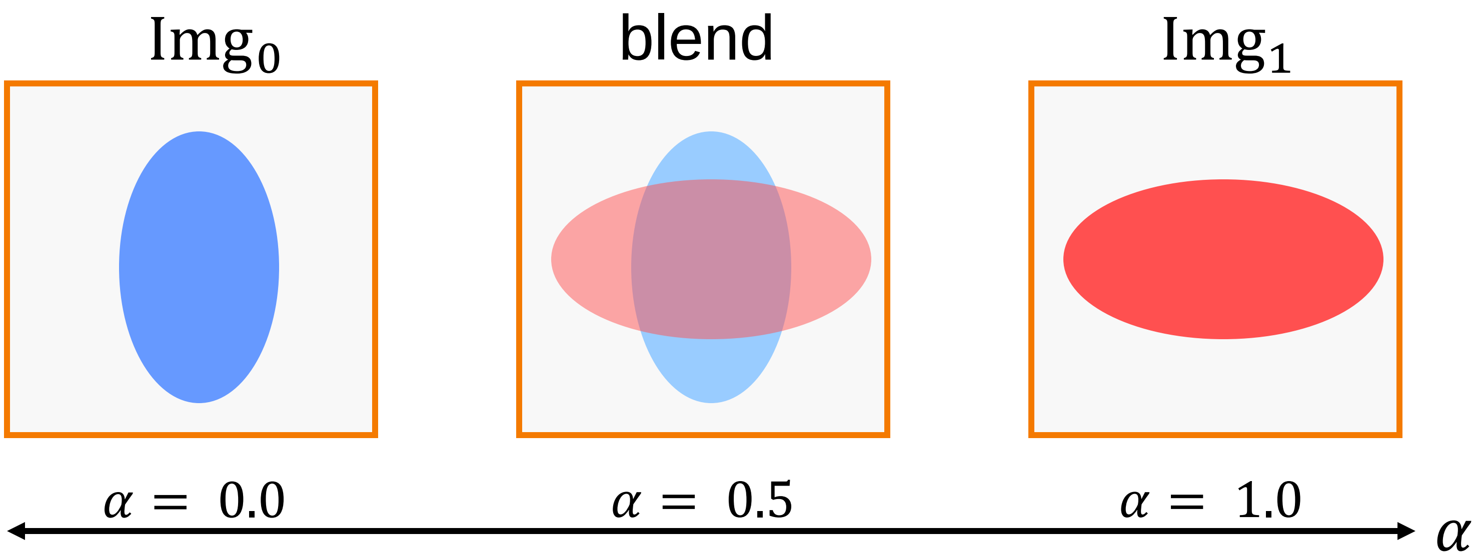


Image Warping

Deform Img_0 so its **shape** matches that of Img_1…

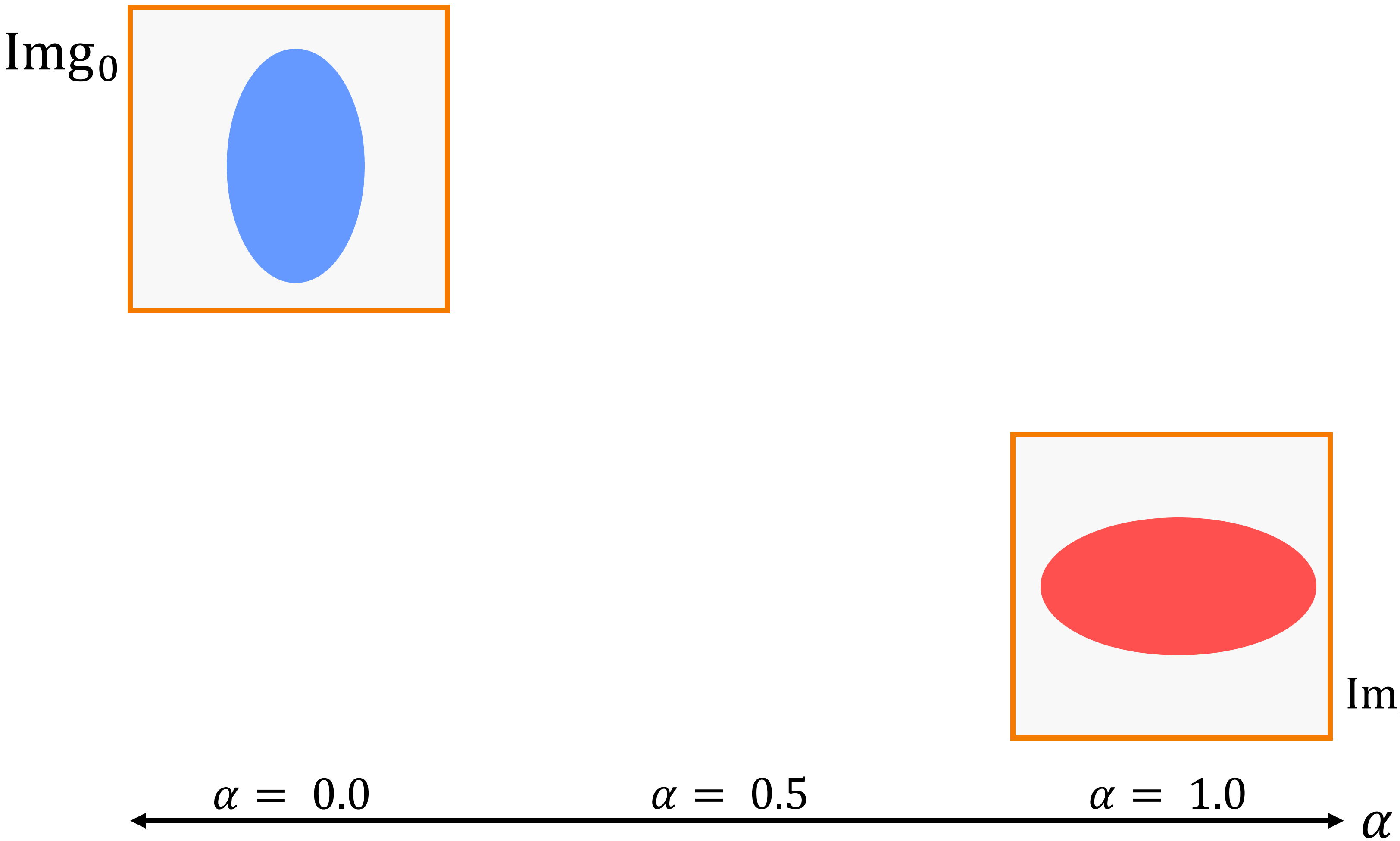


Image Warping

Deform Img_0 so its **shape** matches that of Img_1…

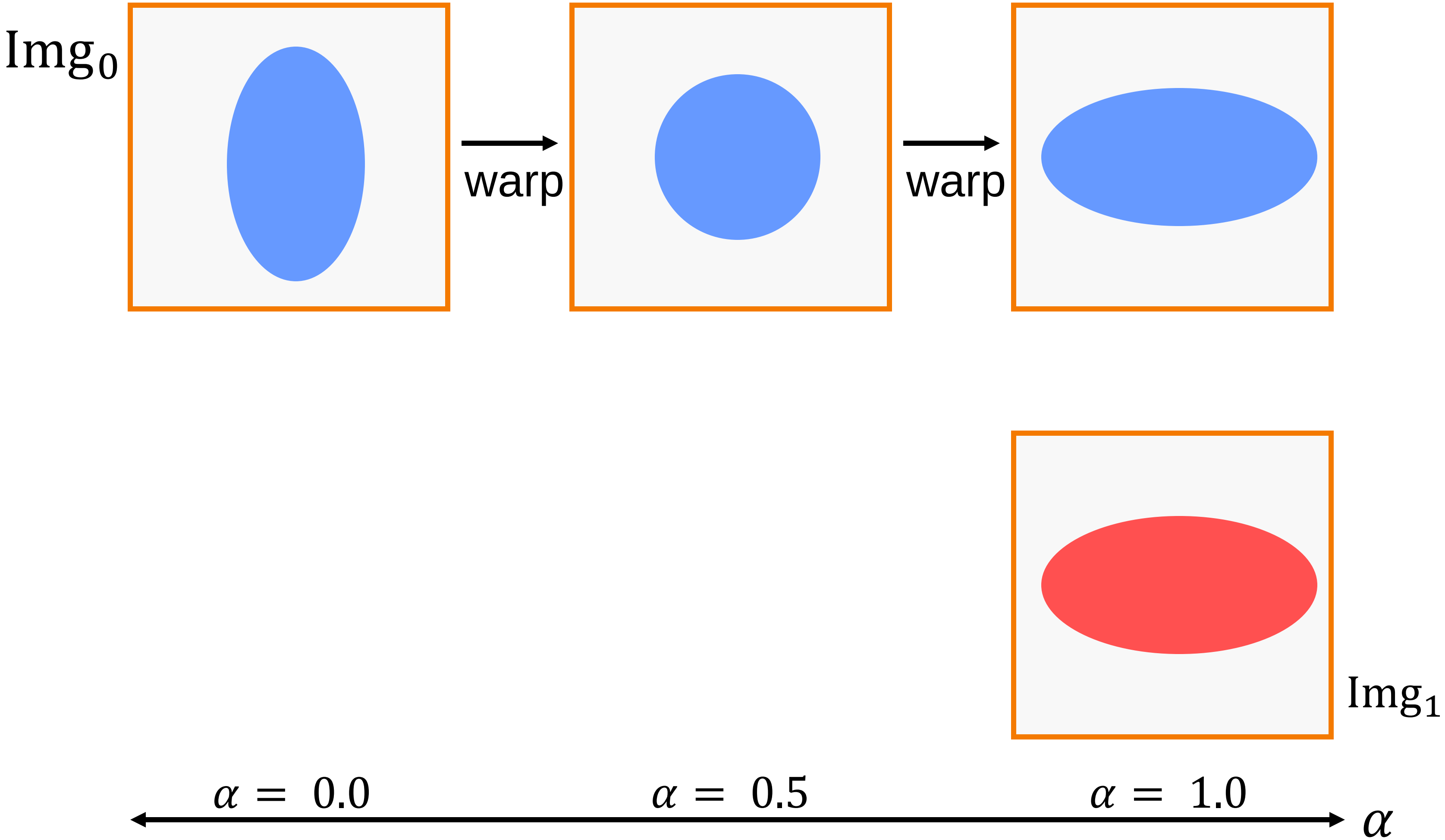


Image Warping

Deform Img_1 so its **shape** matches that of Img_0…

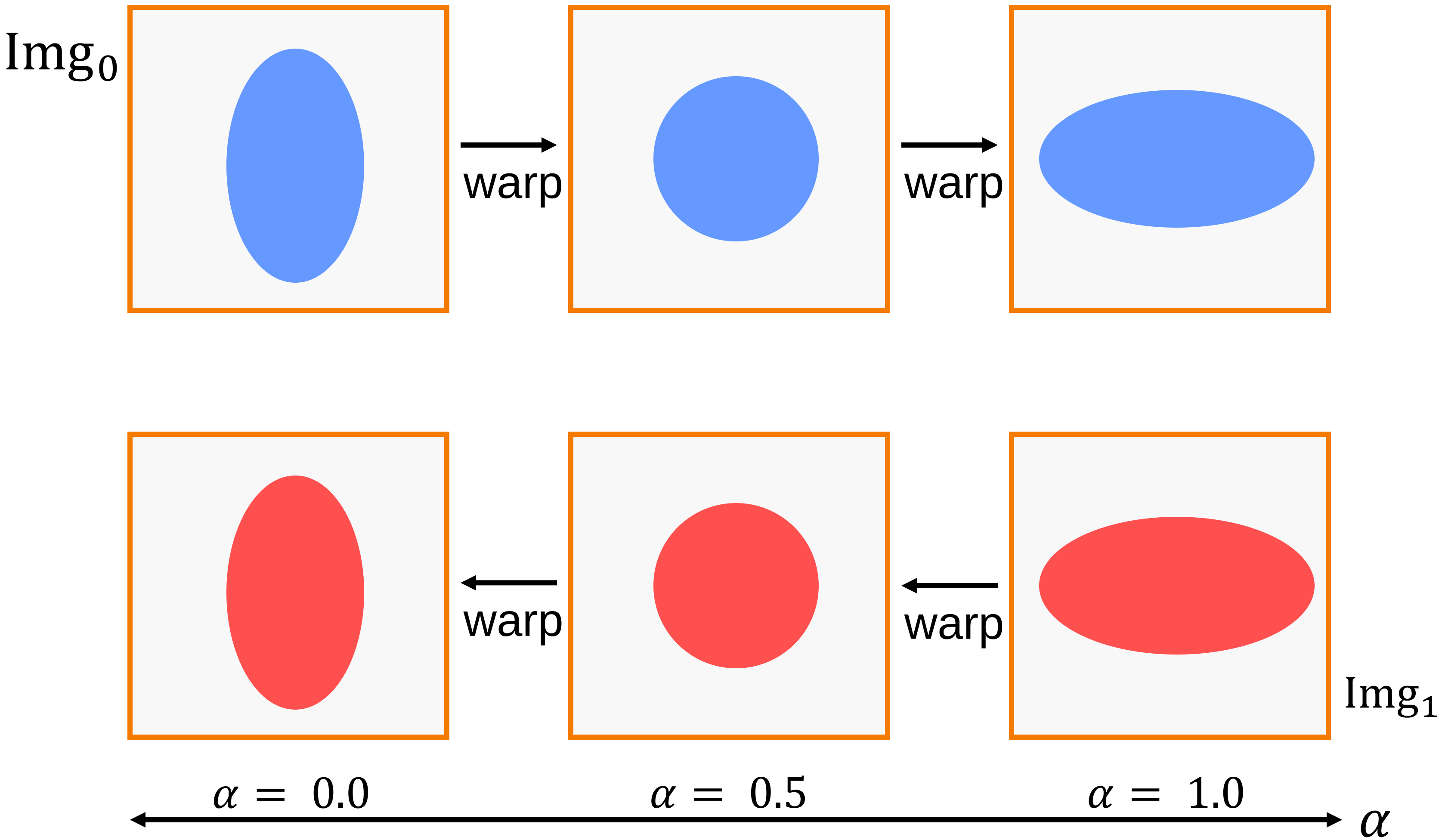


Image Morphing

… then blend **colors**

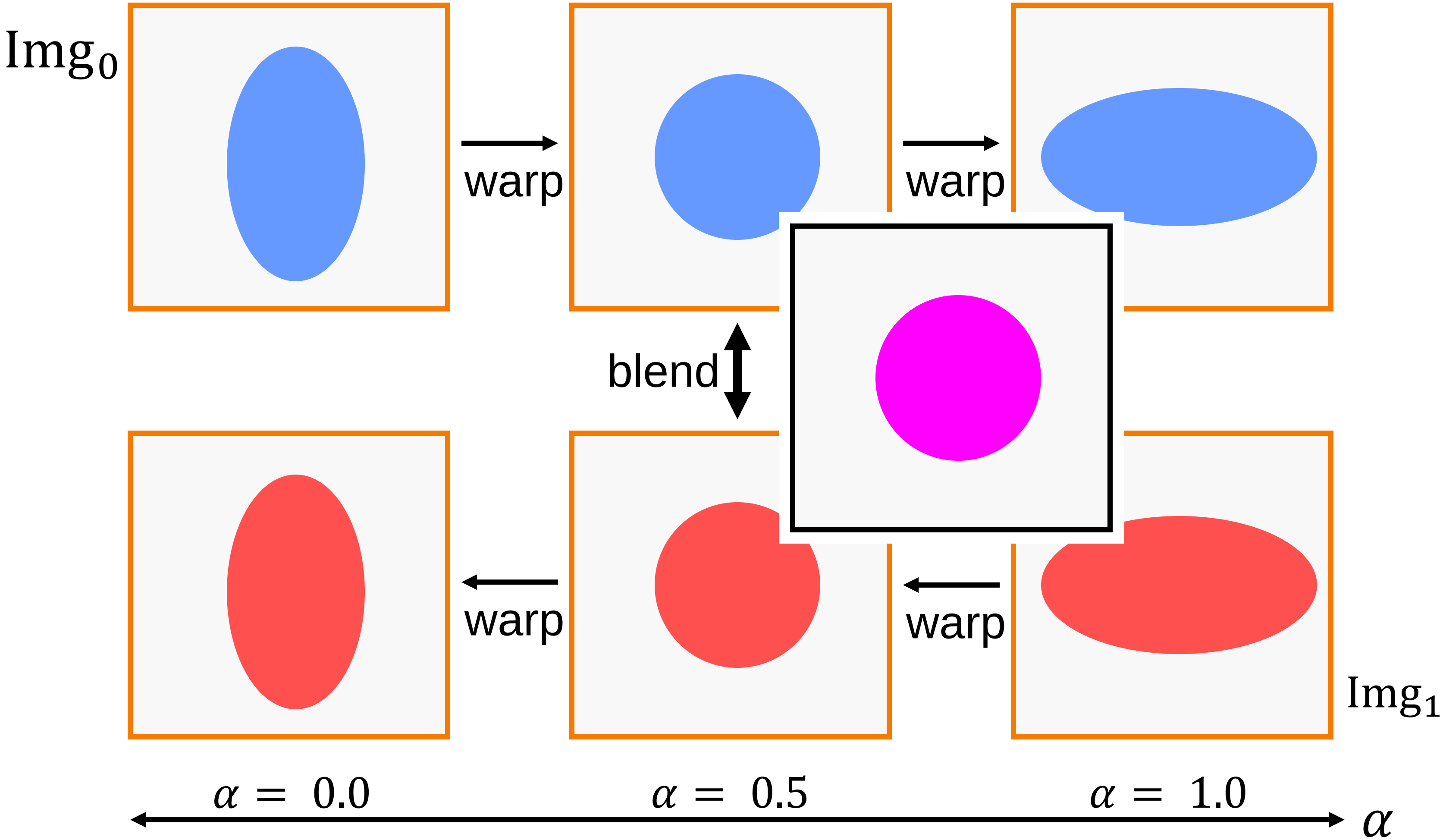
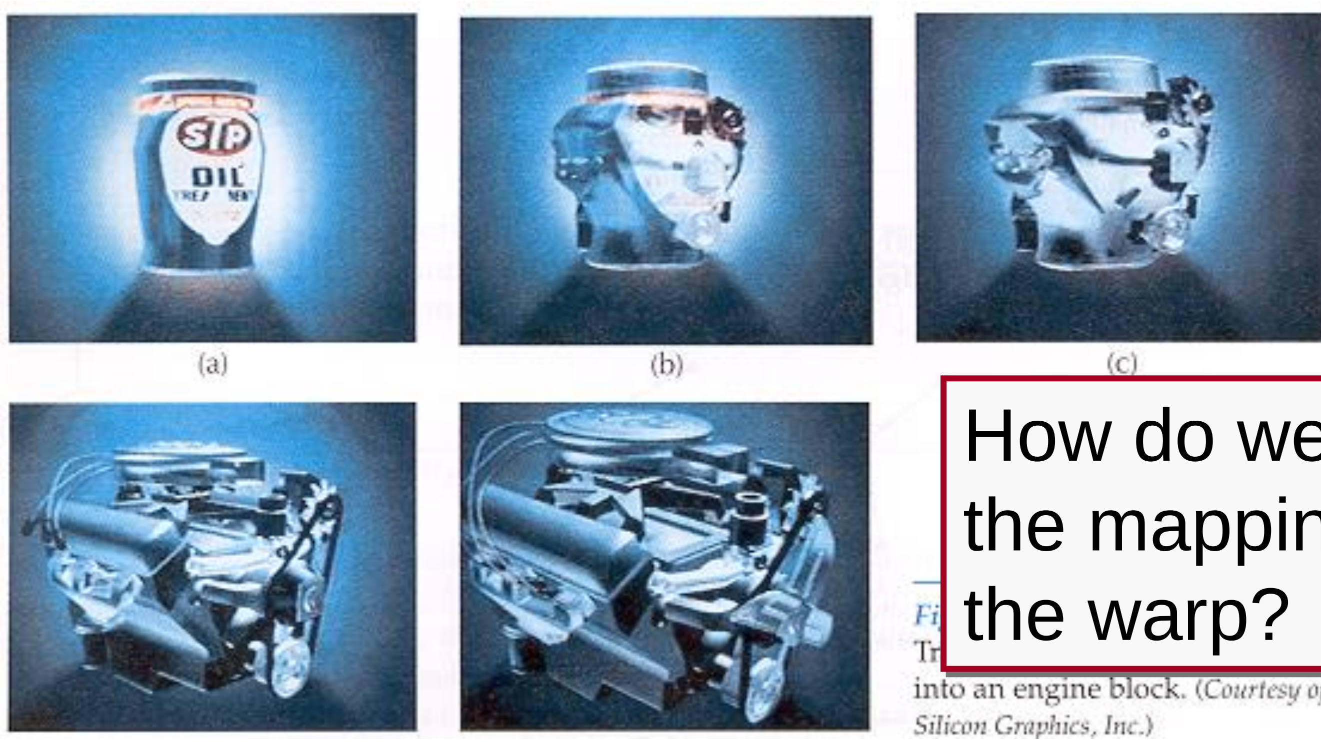


Image Morphing

- The warping step is the hard one
  - Aim to align features in images



Fi
Tr
into an engine block. (*Courtesy of Silicon Graphics, Inc.*)

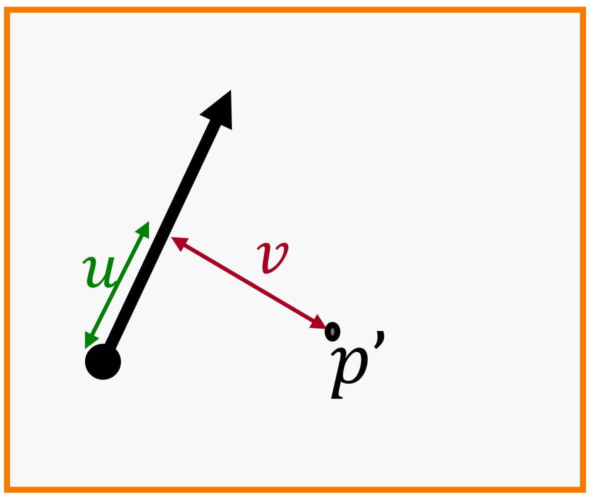
How do we specify the mapping for the warp?

H&B Figure 16.9

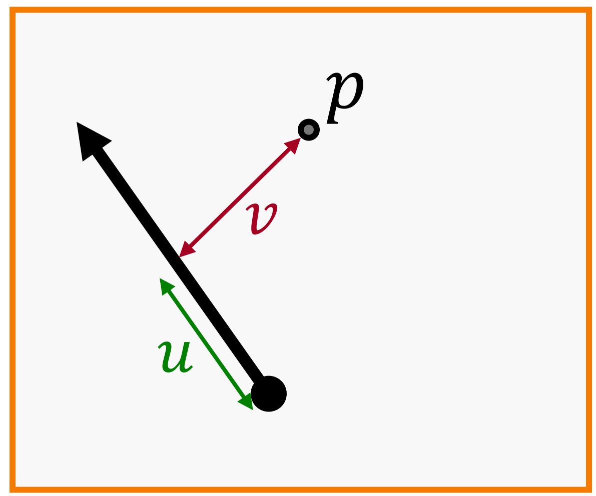
Feature-Based Warping

[Beier & Neeley, 1992] use a pair of lines:

- Given p in the destination image, where is p' in the source?
- Describe p relative to the destination line
- Map the description to the source



Source image



Destination image

u is a signed <u>fraction</u>

v is a signed <u>length</u> (in pixels)

Beier & Neeley
SIGGRAPH 92

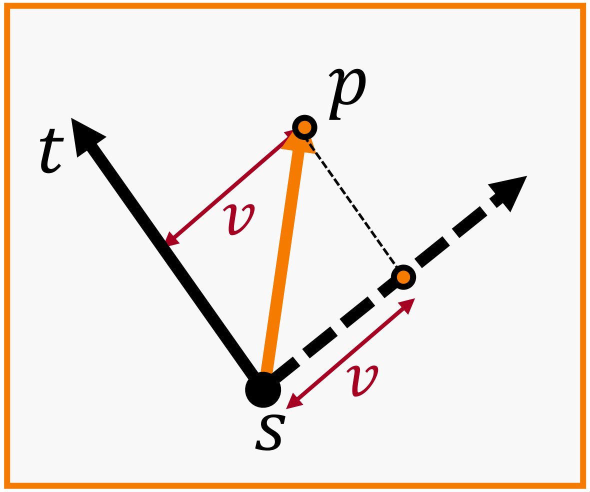
Feature-Based Warping

How do we calculate v (perp. pixel distance)?

Recall:

$\frac{\langle \vec{v}_1, \vec{v}_2 \rangle}{\|\vec{v}_2\|}$ is the signed length of the projection of $\vec{v}_1$ on the line through $\vec{v}_2$.

$$v = \frac{\langle \overbrace{p - s}^{\vec{v}_1}, \overbrace{(t - s)^{\perp}}^{\vec{v}_2} \rangle}{\|(t - s)^{\perp}\|}$$



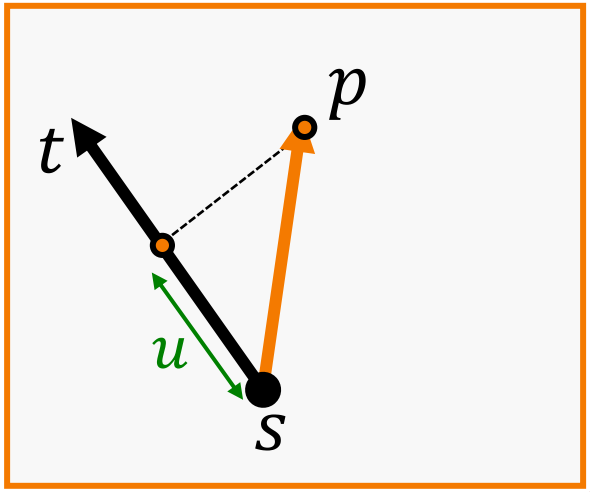
Feature-Based Warping

How do we calculate u (parallel fractional distance)?

Recall:

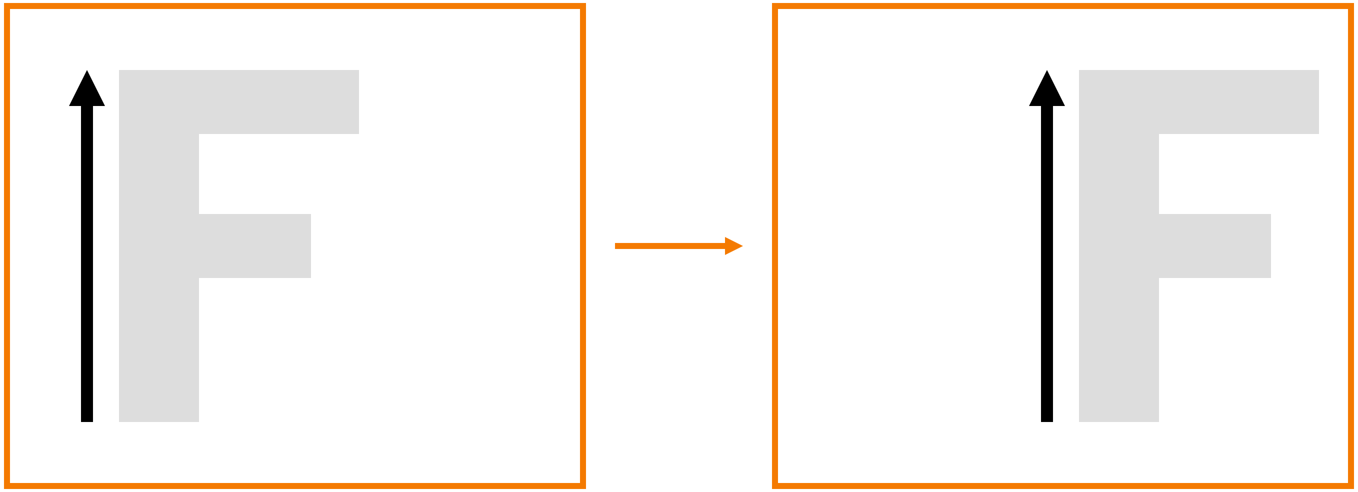
$\frac{\langle \vec{v}_1, \vec{v}_2 \rangle}{\|\vec{v}_2\|}$ is the signed length of the projection of $\vec{v}_1$ on the line through $\vec{v}_2$.

$$u = \frac{\langle \overbrace{p - s}^{\vec{v}_1}, \overbrace{t - s}^{\vec{v}_2} \rangle}{\|t - s\|} \cdot \frac{1}{\|t - s\|}$$



Warping with One Line Pair

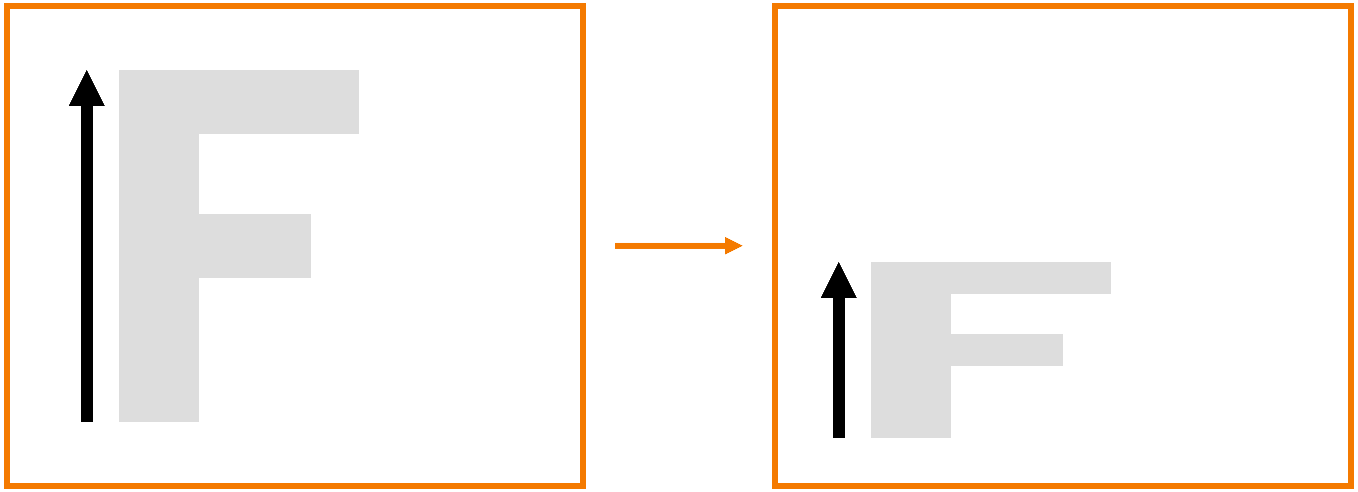
- What happens to the "F"?



Translation!

Warping with One Line Pair

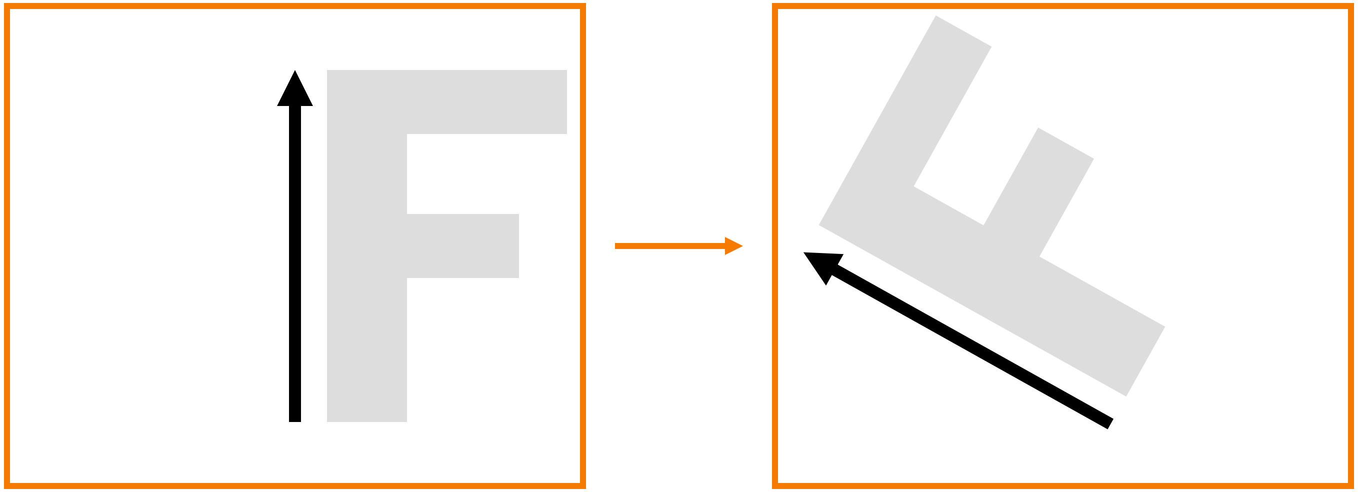
- What happens to the "F"?



<u>Non-uniform</u> scale!

Warping with One Line Pair

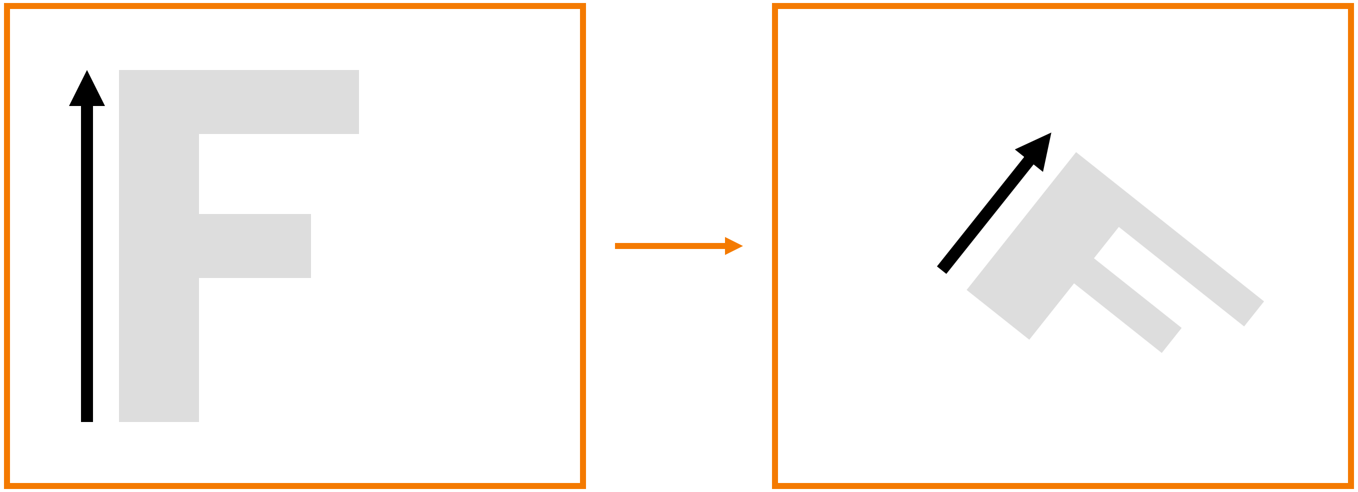
- What happens to the “F”?



Rotation!

Warping with One Line Pair

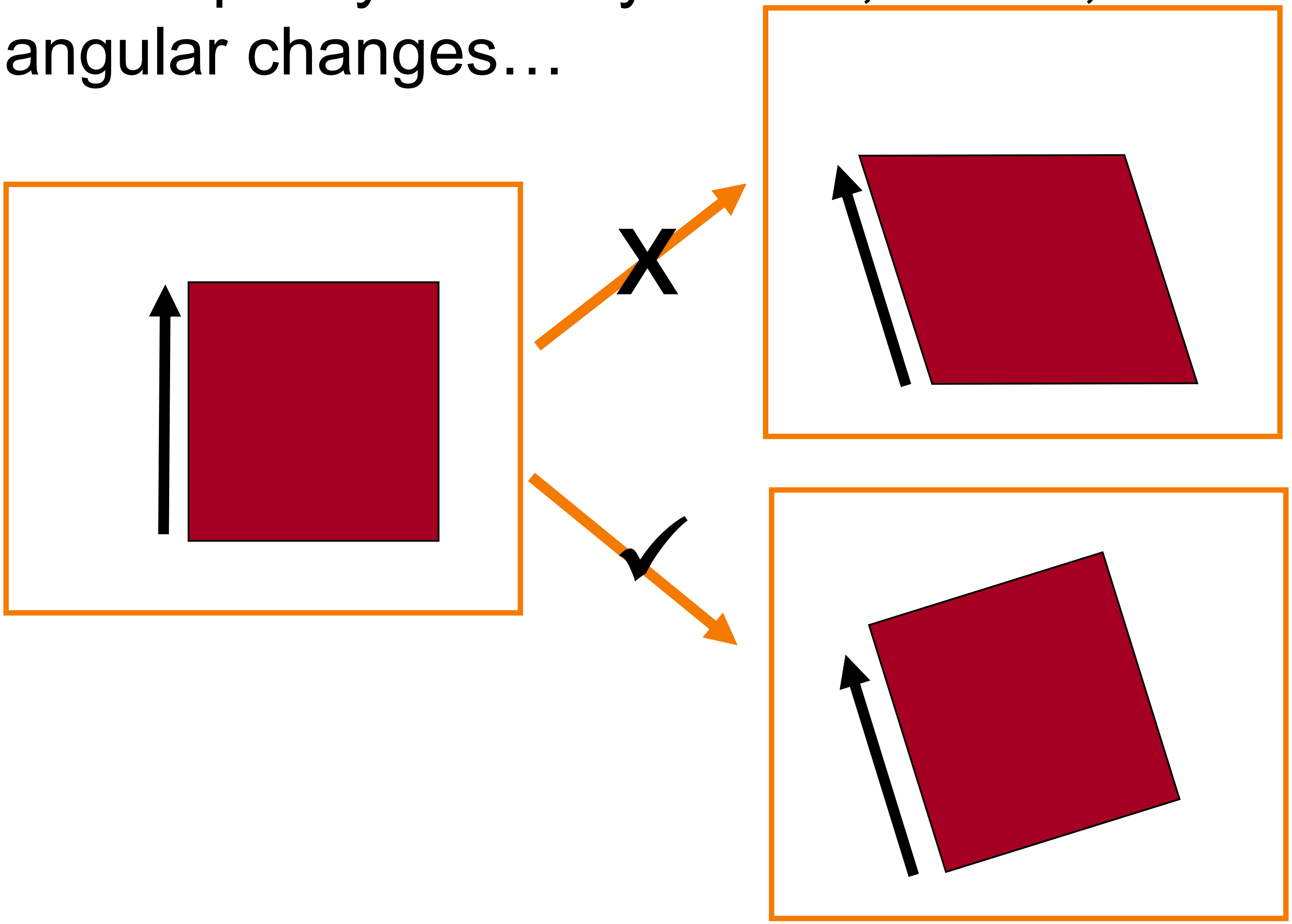
- What happens to the "F"?



What types of <u>affine</u> transformations can't be specified?

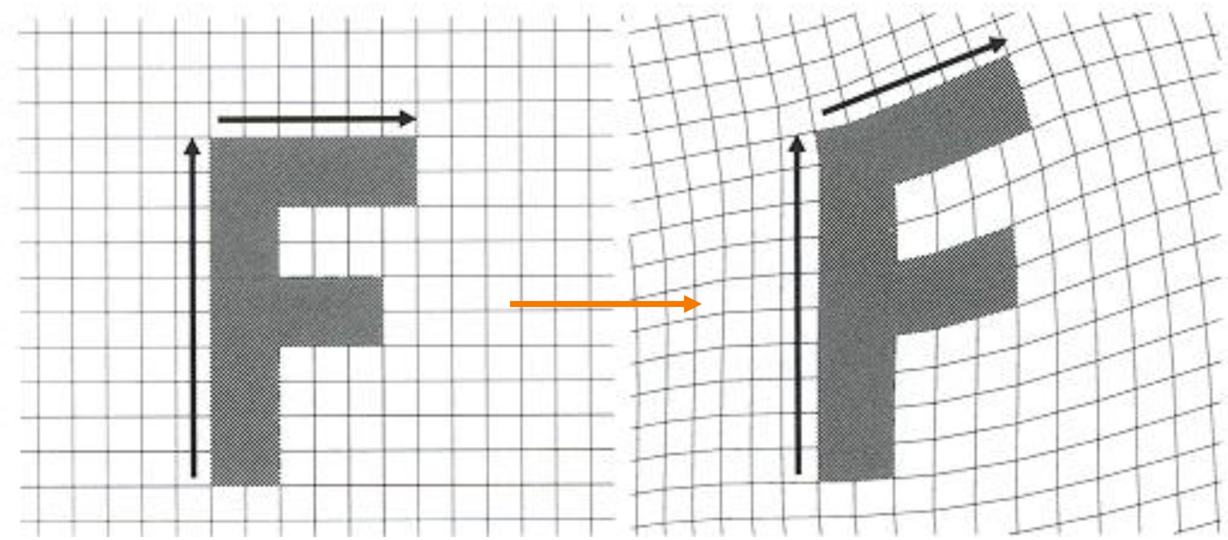
Warping with One Line Pair

- Can’t specify arbitrary scales, skews, mirrors, angular changes…



Warping with Multiple Line Pairs

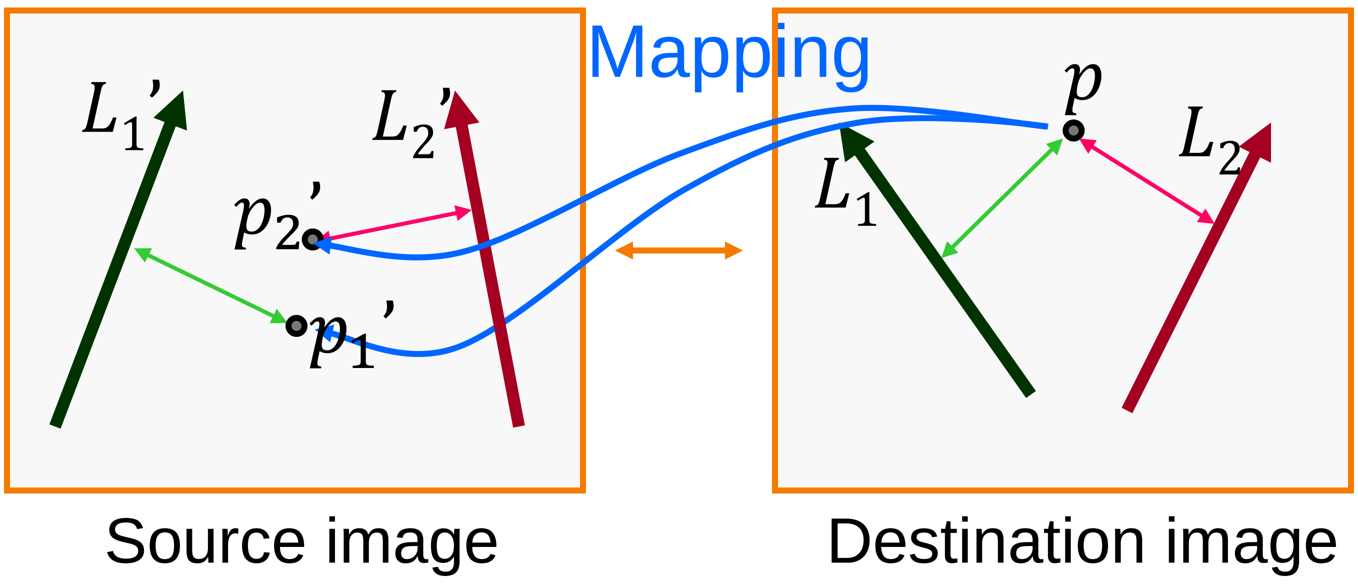
- Use weighted combination of points defined by each pair of corresponding lines



Beier & Neeley, Figure 4

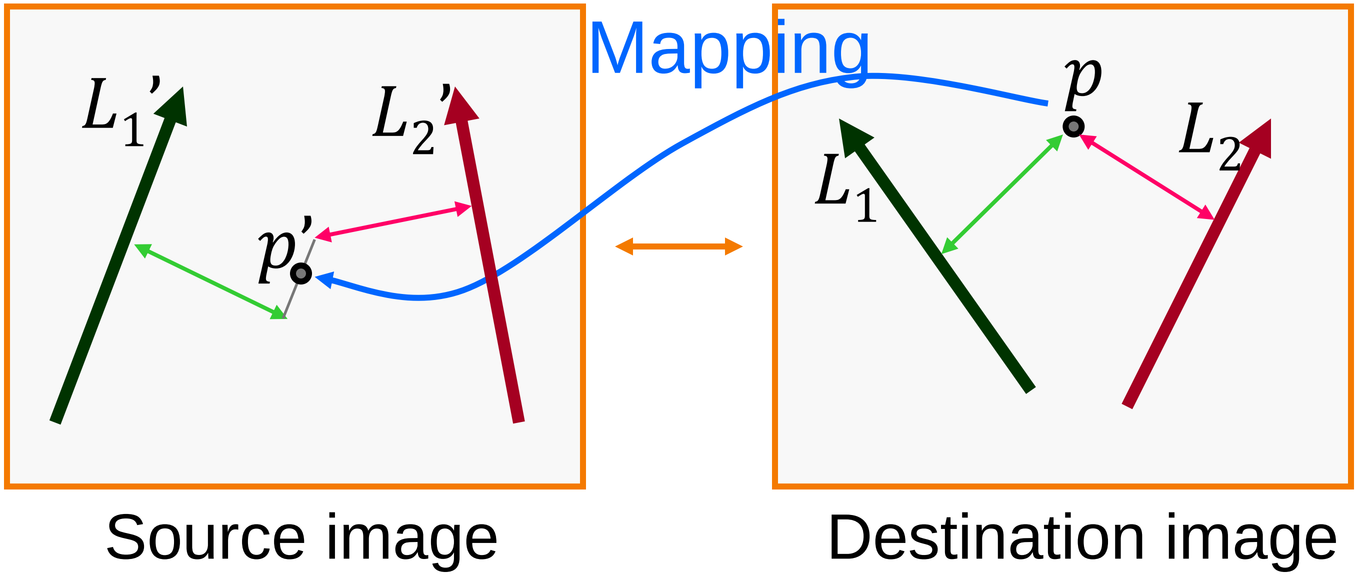
Warping with Multiple Line Pairs

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



Warping with Multiple Line Pairs

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



p' is a weighted average

Weighting Effect of Each Line Pair

- Given a set of line pairs $\{L_{in}[0], \ldots, L_{in}[N]\}$ and $\{L_{out}[0], \ldots, L_{out}[N]\}$, to weight the contribution of each line pair, [Beier & Neely, 1992] use:

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

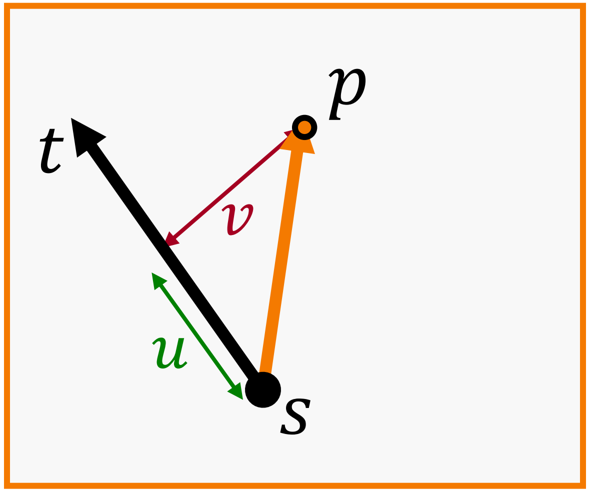
where:

- $\text{length}[i]$ is the length of line $L_{out}[i]$
- $\text{dist}[i](p)$ is the distance from p to $L_{out}[i]$
- a (small), $b \in [0.5,2.0]$, $c \in [0.0,1.0]$ are constants that control the warp

Feature-Based Warping

How do we calculate the unsigned distance from a point p to the line segment from s to t?

$$\text{dist}(p) = \begin{cases} |v| & \text{if } u \in [0,1] \\ \|p - s\| & \text{if } u < 0 \\ \|p - t\| & \text{if } u > 1 \end{cases}$$



Warping

Given a source image and set of corresponding line segment pairs, warp the source to the target by:

- Iterating over each pixel in the target
  - For each pair of line segments
    - » Compute the corresponding position in the source
    - » Compute the weights
  - Average to get the final source position
  - Sample the source (at the source position) to get the color at the target pixel

Warping Pseudocode

Warp(Img_{src} , L_{src}[N] , L_{tgt}[N])
{
    foreach destination pixel p_{trgt}:
        $p_{src} = (0,0)$
        $sum = 0$
        for i = 0 to N:
            $q_{src} = p_{trgt}$ transformed by ($L_{src}[i]$, $L_{trgt}[i]$)
            p_{src} += q_{src} * $weight[i](p_{trgt})$
            sum += $weight[i](p_{trgt})$
        p_{src} /= sum
        $Img_{trgt}(p_{trgt}) = Img_{src}(p_{src})$
    return Img_{trgt}
}

Morphing at $\boldsymbol{\alpha} \in [\mathbf{0}, \mathbf{1}]$

Given two images, given a set of corresponding line segment pairs, and an interpolation time $\alpha \in [0,1]$:

- Compute the α-blend of the of line segments (by blending the end-points)
- Warp the first image using the first set of line segments and the blended line segments
- Warp the second image using the second set of line segments and the blended line segments
- Compute the α-blend of the warped images

Morphing at $\alpha \in [0,1]$ Pseudocode

Morph(Img_0 , $L_0[N]$, Img_1 , $L_1[N]$, α)
{
  foreach $i \in \{1,\dots,N\}$:
    $L_\alpha[i]$ = line α-th of the way from $L_0[i]$ to $L_1[i]$

$$\left.\begin{array}{l} Warp_0 = \text{Warp}(\ Img_0\ ,\ L_0[\,]\ ,\ L_\alpha[\,]\ ) \\ Warp_1 = \text{Warp}(\ Img_1\ ,\ L_1[\,]\ ,\ L_\alpha[\,]\ ) \end{array}\right\}\text{warp}$$

$$\left.\text{return } (1-\alpha) * Warp_0 + \alpha * Warp_1 \right\}\alpha\text{-blend}$$

}

[Beier & Neely, 1992] Example $(\alpha = 0.5)$

Img$_0$

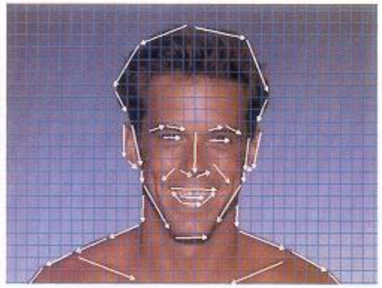
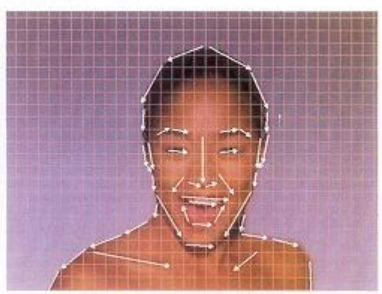


Figure 7

Warp$_0$

Img$_1$

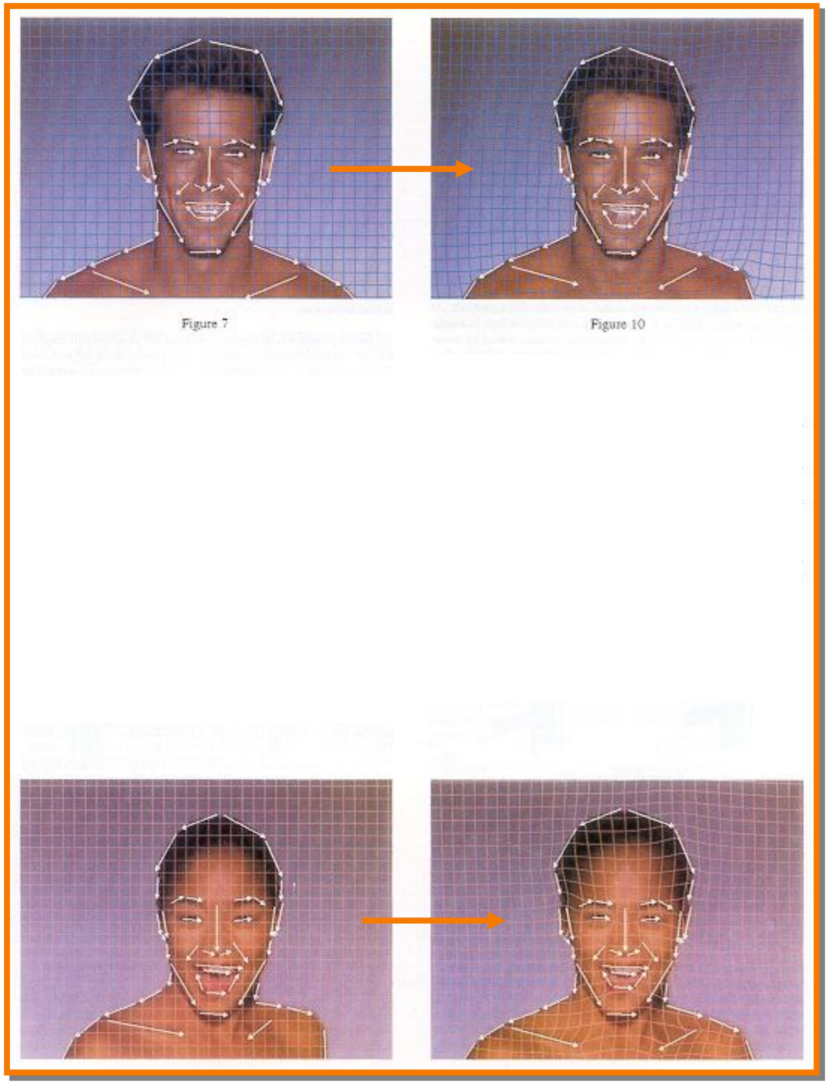


Warp$_1$

[Beier & Neely, 1992] Example ($\alpha = 0.5$)

Img_0

Warp_0



Img_1

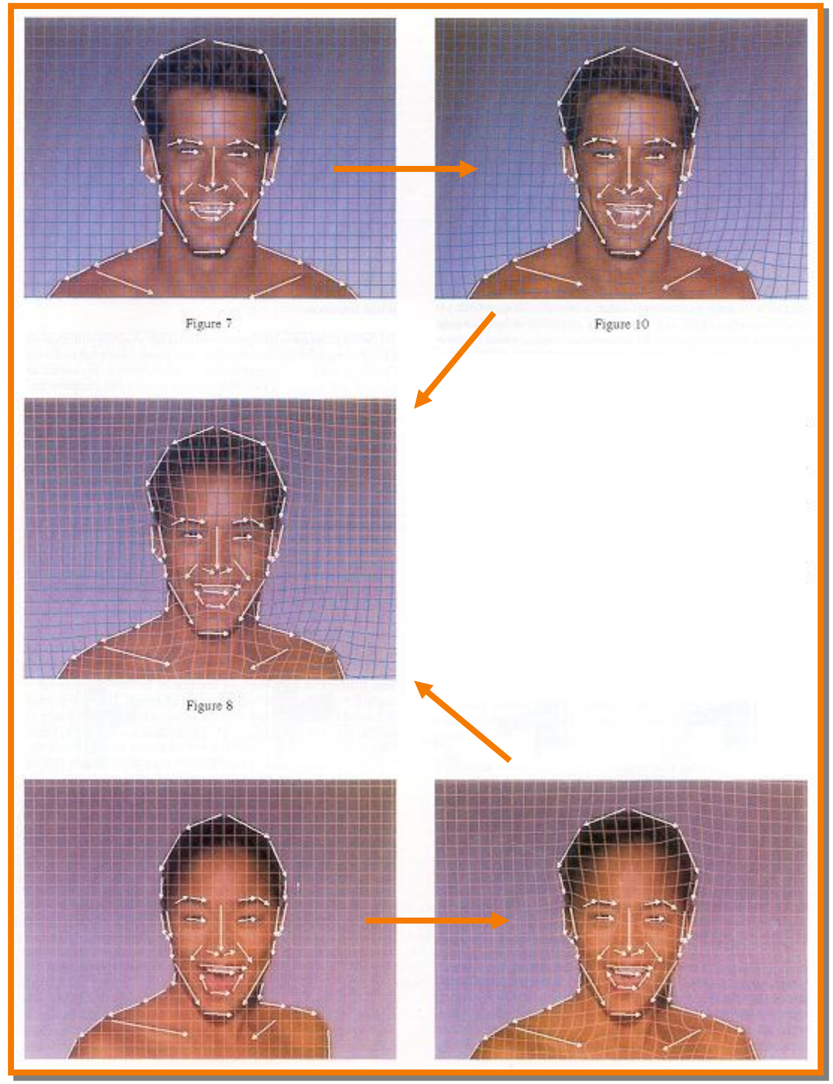
Warp_1

[Beier & Neely, 1992] Example ($\alpha = 0.5$)

Img$_0$

Warp$_0$

Result



Img$_1$

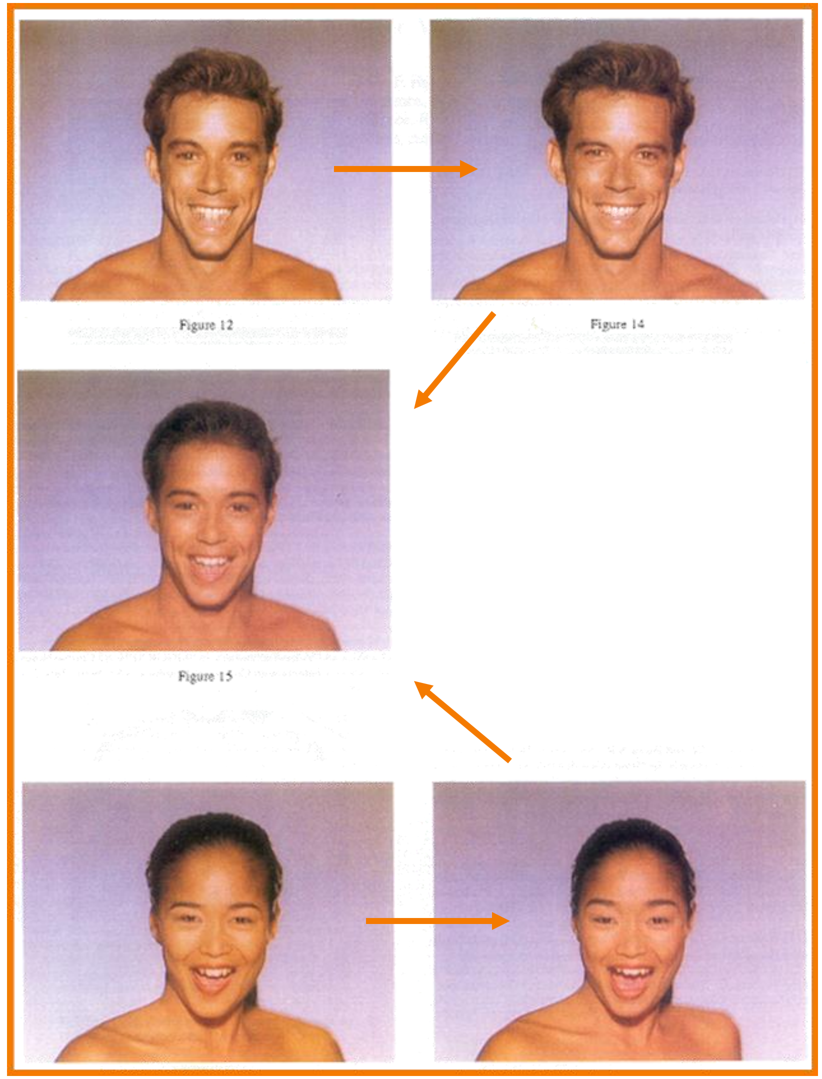
Warp$_1$

[Beier & Neely, 1992] Example ($\alpha = 0.5$)

Img$_0$

Warp$_0$

Result



Img$_1$

Warp$_1$

Animation Pseudocode

Animate(Img_0 , $L_0[N]$, Img_1 , $L_1[N]$, $Imgs_{out}[T+1]$)
{
    foreach $t \in \{0,\ldots,T\}$:
        $Imgs_{out}[t]$ = Morph(Img_0 , $L_0[N]$, Img_1 , $L_1[N]$, t/T)
}

Morphing

Check out Michael Jackson’s “Black or White” video at:

https://www.youtube.com/watch?v=pTFE8cirkdQ

Or the earlier Plymouth Voyager commercial at:

https://www.youtube.com/watch?v=0b939O7dGqQ