Image Filtering, Warping, and Morphing

Michael Kazhdan

(601.457/657)

HB Ch. 4.8
FvDFH Ch. 14.10
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.
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

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

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

**Original**

<table>
<thead>
<tr>
<th></th>
<th>X - 1</th>
<th>X</th>
<th>X + 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y - 1</td>
<td>36</td>
<td>109</td>
<td>146</td>
</tr>
<tr>
<td>Y</td>
<td>32</td>
<td>36</td>
<td>109</td>
</tr>
<tr>
<td>Y + 1</td>
<td>32</td>
<td>36</td>
<td>73</td>
</tr>
</tbody>
</table>

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

**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

Original

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

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)

Filter =
[1/16 2/16 1/16]
[2/16 4/16 2/16]
[1/16 2/16 1/16]
Blurring

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

```
<table>
<thead>
<tr>
<th>X - 1</th>
<th>X</th>
<th>X + 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>109</td>
<td>146</td>
</tr>
<tr>
<td>32</td>
<td>36</td>
<td>109</td>
</tr>
<tr>
<td>32</td>
<td>36</td>
<td>73</td>
</tr>
</tbody>
</table>
```

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

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

Original

Blur

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

\[
\begin{bmatrix}
\frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\
\frac{2}{16} & \frac{4}{16} & \frac{2}{16} \\
\frac{1}{16} & \frac{2}{16} & \frac{1}{16}
\end{bmatrix}
\]
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**

- In general, the mask can have arbitrary size:
  - We can express a smaller mask as a bigger one by padding with zeros.
Blurring

• In general, the mask can have arbitrary size:
  ◦ We can have more non-zero entries to give rise to a wider blur.

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 1 & 2 & 1 & 0 \\
0 & 2 & 4 & 2 & 0 \\
0 & 1 & 2 & 1 & 0 \\
0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 & 1 & 2 & 1 & 0 \\
1 & 2 & 4 & 2 & 1 \\
2 & 4 & 8 & 4 & 2 \\
1 & 2 & 4 & 2 & 1 \\
0 & 1 & 2 & 1 & 0
\end{bmatrix}
/48
\]

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:

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

where $i, j \in [0, 2\sigma + 1]$

- $\sigma$ is the integer mask radius ($n = 2\sigma + 1$ is the width)
- $i$ is the horizontal position in the mask
- $j$ is the vertical position in the mask
- Don’t forget to normalize!
Edge Detection

• An edge is a point in the image where the image is “far” from constant:
  ◦ The difference between the value at the pixel and the value of neighboring pixels is large (in absolute value)
Edge Detection

• To find the edges, define a mask:
  ○ Whose entries sum to zero
  ○ Whose value is one at the center pixel
  ○ Whose values are negative at neighboring pixels

• (Upper) edge pixels are those whose value is larger than the average of its neighbors.

Original  Detected Edges

Filter = \[ \frac{1}{8} \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \]
Edge Detection

- To find the edges, define a mask:
  - Whose entries sum to zero
  - Whose value is one at the center pixel
  - Whose values are negative at neighboring pixels

- (Upper) edge pixels are those whose value is larger than the average of its neighbors.

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

Filter = \[
\begin{bmatrix}
-1 & -1 & -1 \\
-1 & 8 & -1 \\
-1 & -1 & -1 \\
\end{bmatrix}
\]
Edge Detection

Original

<table>
<thead>
<tr>
<th>Y-1</th>
<th>X-1</th>
<th>X</th>
<th>X+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>109</td>
<td>146</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Y</th>
<th>32</th>
<th>36</th>
<th>109</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>36</td>
<td>73</td>
<td></td>
</tr>
</tbody>
</table>

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

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

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**

\[
\text{New value for Pixel}(x,y).\text{red} = \\
(36 \times -1/8) + (109 \times -1/8) + (146 \times -1/8) \\
(32 \times -1/8) + (36 \times 1) + (109 \times -1/8) \\
(32 \times -1/8) + (36 \times -1/8) + (73 \times -1/8)
\]

**Filter**

\[
\frac{1}{8} \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}
\]

<table>
<thead>
<tr>
<th>Y - 1</th>
<th>36</th>
<th>109</th>
<th>146</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>32</td>
<td>36</td>
<td>109</td>
</tr>
<tr>
<td>Y + 1</td>
<td>32</td>
<td>36</td>
<td>73</td>
</tr>
</tbody>
</table>

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

Original
Edge Detection

Original

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

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

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

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

Note: Edge values are not colors, so you will need to find a way to rescale/remap for visualization.
Outline

• Image Filtering
• Image Warping
• Image Morphing
Image Warping

- Move pixels of image
  - Mapping
  - Resampling

Source image → Warp → 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 $\sigma$:
  - $\Phi(u, v) = (\sigma u, \sigma v)$

Scale $\sigma = 0.8$
Example Mappings

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

$\theta = 30$
Example Mappings

• Shear in $x$ by $\sigma_x$:
  \[ \Phi(u, v) = (u + \sigma_x \cdot v, v) \]

• Shear in $y$ by $\sigma_y$:
  \[ \Phi(u, v) = (u, v + \sigma_y \cdot u) \]
Other Mappings

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

![Fish-eye](image1)

![“Swirl”](image2)

![“Rain”](image3)
Image Warping Implementation I

• Forward mapping:

```c
for( v=0 ; v<vmax ; v++ )
    for( u=0 ; u<umax ; u++ )
        (x,y) = \Phi(u,v);
        dst(x,y) = src(u,v);
```

Source image \( (u,v) \) \( \Phi \) Destination image \( (x,y) \)
Forward Mapping

- Iterate over source image
Forward Mapping – BAD!

- Iterate over source image

Multiple source pixels can map to the same destination pixel
Forward Mapping – BAD!

- Iterate over source image

Multiple source pixels can map to the same destination pixel

Some destination pixels may not be covered

Rotate + Translate
Image Warping Implementation II

• Inverse mapping:

\[
\begin{align*}
\text{for( } y=0 & \text{ ; } y<y_{\text{max}} \text{ ; } y++ \text{ )} \\
\text{for( } x=0 & \text{ ; } x<x_{\text{max}} \text{ ; } x++ \text{ )} \\
(u,v) &= \Phi^{-1}(x,y) \\
\text{dst}(x,y) &= \text{src}(u,v)
\end{align*}
\]
Reverse Mapping – GOOD!

- Iterate over destination image
  - Must resample source

Rotate -30 + Translate
Resampling

- Evaluate source image at \((u, v) = \Phi^{-1}(x, y)\)

\((u, v)\) does not usually have integer coordinates

Source image \(\rightarrow\) Destination image

\((u, v)\)

\((x, y)\)
Overview

• Mapping
  ○ Forward
  ○ Inverse

• Resampling
  ○ Nearest Point Sampling
  ○ Bilinear Sampling
  ○ Gaussian Sampling
Nearest Point Sampling

- Take value at closest pixel:
  \[
  \text{int } iu = \text{floor}(u+0.5); \\
  \text{int } iv = \text{floor}(v+0.5); \\
  \text{dst}(x,y) = \text{src}(iu,iv); 
  \]
Bilinear Sampling

- Bilinearly interpolate four closest source pixels

$$\text{dst}(x, y) = \text{Weighted average of source at } (u_1, v_1), (u_2, v_1), (u_1, v_2), \text{ and } (u_2, v_2)$$
Linear Sampling

• Linearly interpolate two closest source pixels

\[ \text{dst}(x) = \text{linear interpolation of } u_1 \text{ and } u_2 \]

\[
\begin{align*}
    u_1 &= \text{floor}(u); \\
    u_2 &= u_1 + 1; \\
    du &= u - u_1; \\
    \text{dst}(u) &= \text{src}(u_1)*(1-du) + \text{src}(u_2)*du;
\end{align*}
\]

\[ 0 \leq du \leq 1 \]
Bilinear Sampling

- Bilinearly interpolate four closest source pixels

\[ a = \text{linear interpolation of src}(u_1, v_1) \text{ and src}(u_2, v_1) \]
\[ b = \text{linear interpolation of src}(u_1, v_2) \text{ and src}(u_2, v_2) \]
\[ \text{dst}(x, y) = \text{linear interpolation of } a \text{ and } b \]

\[ u_1 = \text{floor}(u), \quad u_2 = u_1 + 1; \]
\[ v_1 = \text{floor}(v), \quad v_2 = v_1 + 1; \]
\[ \text{du} = u - u_1; \]
\[ a = \text{src}(u_1,v_1) \ast (1 - \text{du}) + \text{src}(u_2,v_1) \ast \text{du}; \]
\[ b = \text{src}(u_1,v_2) \ast (1 - \text{du}) + \text{src}(u_2,v_2) \ast \text{du}; \]
\[ \text{dv} = v - v_1; \]
\[ \text{dst}(x,y) = a \ast (1 - \text{dv}) + b \ast \text{dv}; \]
Bilinear Sampling

- Bilinearly interpolate four closest source pixels

  \[a = \text{linear interpolation of } \text{src}(u_1, v_1) \text{ and } \text{src}(u_2, v_1)\]

  \[b = \text{linear interpolation of } \text{src}(u_1, v_2) \text{ and } \text{src}(u_2, v_2)\]

  \[\text{dst}(x, y) = \text{linear interpolation of } a \text{ and } b\]

\[u_1 = \text{floor}(u) \quad , \quad u_2 = u_1 + 1;\]
\[v_1 = \text{floor}(v) \quad , \quad v_2 = v_1 + 1;\]
\[du = u - u_1;\]
\[dv = v - v_1;\]
\[a = \text{src}(u_1, v_1) \cdot (1 - du) + \text{src}(u_2, v_1) \cdot du;\]
\[b = \text{src}(u_1, v_2) \cdot (1 - du) + \text{src}(u_2, v_2) \cdot du;\]
\[\text{dst}(x, y) = a \cdot (1 - dv) + b \cdot dv;\]
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:

  \[
  \text{for} \ (y=0 \ ; \ y<\text{ymax} \ ; \ y++) \\
  \quad \text{for} \ (x=0 \ ; \ x<\text{xmax} \ ; \ x++) \\
  \quad (u,v) = \Phi^{-1}(x,y); \\
  \quad \text{dst}(x,y) = \text{resample}_\text{src}(u,v,r);
  \]
Image Warping Implementation

- Inverse mapping:

```c
for( y=0 ; y<yMax ; y++ )
    for( x=0 ; x<xMax ; x++ )
        (u,v) = \Phi^{-1}(x,y);
        dst(x,y) = resample_src(u,v,r);
```
Example: Scale

Scale( src, dst, σ ):

\[ r \approx \sigma \]

\[
\begin{align*}
\text{for} & ( y=0 \ ; \ y<\text{ymax} \ ; \ y++) \\
& \quad \text{for} ( x=0 \ ; \ x<\text{xmax} \ ; \ x++) \\
& \quad (u,v) = (x,y) / \sigma; \\
& \quad \text{dst}(x,y) = \text{resample}_\text{src}(u,v,r);
\end{align*}
\]

\[ r = \frac{1}{\sigma} \]
Example: Rotate

Rotate( src , dst , \( \theta \) ):

\[
\begin{align*}
  r & \approx 1; \\
  \text{for}( y=0 ; y<\text{ymax} ; y++ ) & \\
    \text{for}( x=0 ; x<\text{xmax} ; x++ ) & \\
    (u,v) = ( x\cos(-\theta) - y\sin(-\theta) , \\
              x\sin(-\theta) + y\cos(-\theta) ) ; \\
  \text{dst}(x,y) = \text{resample}_\text{src}(u,v,r) ;
\end{align*}
\]

\[
\begin{align*}
  x &= u \cos \theta - v \sin \theta \\
  y &= u \sin \theta + v \cos \theta
\end{align*}
\]
Example:

**General(src, dst, Φ):**

```plaintext
r ≈ ?;
for( y=0 ; y<ymax ; y++ )
  for(x=0 ; x<xmax ; x++)
    (u,v) = Φ( x,y );
    dst(x,y) = resample_src(u,v,r);
```

Example: General(src, dst, Φ):

```
gr = ?
```

![Swirl](image)
Example:

General( src, dst, \( \Phi \)):

\[
\begin{align*}
\text{for}(\text{y}=0 & \text{ ; y}_{}<\text{ymax} & \text{ ; y}_{}++ ) \\
\text{for}(\text{x}=0 & \text{ ; x}_{}<\text{xmax} & \text{ ; x}_{}++ ) \\
(u\text{,v}) & = \Phi( x\text{,y} ); \\
dst( x\text{,y}) & = \text{resample}_\text{src}(u\text{,v},r);
\end{align*}
\]

Instead of using a fixed radius circle to sample the source, we can make things more “interesting” by:

1. Having the radius changes
2. Using an ellipse.

For example, the parameters can be determined by looking at the derivative/Jacobi of \( \Phi \).
Outline

• Image Filtering
• Image Warping
• Image Morphing
Image Morphing

- Animate transition between two images

H&B Figure 16.9

Transformation of an STP oil can into an engine block. (Courtesy of Silicon Graphics, Inc.)
Image Morphing

- Animate transition between two images

Two Components:
- Blending
- Warping
Blending

Blend **colors** using an \( \alpha \)-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 $Img_1$...

$\alpha = 0.0$ $\alpha = 0.5$ $\alpha = 1.0$
Image Warping

Deform Img₀ so its shape matches that of Img₁...

\[ \alpha = 0.0 \]
\[ \alpha = 0.5 \]
\[ \alpha = 1.0 \]
Image Warping

Deform $\text{Img}_1$ so its **shape** matches that of $\text{Img}_0$…

$\alpha = 0.0$ $\rightarrow$ warp $\rightarrow$ warp $\rightarrow$ $\alpha = 1.0$

$\alpha = 0.5$ $\leftarrow$ warp $\leftarrow$ warp $\leftarrow$ $\alpha = 1.0$
Image Morphing

... then blend colors

$\alpha = 0.0$

$\alpha = 0.5$

$\alpha = 1.0$
Image Morphing

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

H&B Figure 16.9

How do we specify the mapping for the warp?
Feature-Based Warping

• [Beier & Neeley, 1992] use a pair of lines to specify the warp
  ◦ 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

\[
\begin{align*}
\text{Source image} & \quad \quad \text{Destination image} \\
\quad u & \quad \quad \quad v \\
p' & \quad \quad \quad p \\
\end{align*}
\]

\( u \) is a signed \underline{fraction}  
\( v \) is a signed \underline{length} (in pixels)
Feature-Based Warping

How do we calculate $u$ and $v$?

Recall:

$$\langle v_1, v_2 \rangle = x_1 \cdot x_2 + y_1 \cdot y_2$$
$$= ||v_1|| \cdot ||v_2|| \cdot \cos \theta$$

The signed length of the projection of $v_1$ on the line through $v_2$, scaled by the length of $v_2$.

$\Rightarrow$ The signed length of the projection of $v_1$ onto the line through $v_2$ is:

$$\frac{\langle v_1, v_2 \rangle}{||v_2||}$$
How do we calculate $v$ (perp. pixel distance)?

\[ \langle v_1, v_2 \rangle \] is the (signed) length of the projection of $v_1$ on the line through $v_2$, scaled by the length of $v_2$.

\[
v = \frac{\langle p - s, (t - s)\top \rangle}{\| (t - s)\top \|}
\]

The “$\top$” in the superscript denotes “perpendicular”.
Feature-Based Warping

How do we calculate $u$ (parallel fractional distance)?

$\langle v_1, v_2 \rangle$ is the (signed) length of the projection of $v_1$ on the line through $v_2$, scaled by the length of $v_2$.

$$u = \frac{\langle p - s, t - s \rangle}{\|t - s\|} \cdot \frac{1}{\|t - s\|}$$
Warping with One Line Pair

• What happens to the “F”?

Translation!
Warping with One Line Pair

• What happens to the “F”?  

Non-uniform 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 affine 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

Source image

Destination image
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], ..., L_{in}[N]\} and \{L_{out}[0], ..., 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
How do we calculate the 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:

• Iterate 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\textsubscript{in} , L\textsubscript{in}[N] , L\textsubscript{out}[N] )
{
    foreach destination pixel p\textsubscript{out}:
        p\textsubscript{in} = (0,0)
        sum = 0
        for i = 0 to N:
            q\textsubscript{in} = p\textsubscript{out} transformed by ( L\textsubscript{in}[i] , L\textsubscript{out}[i] )
            p\textsubscript{in} += q\textsubscript{in} * weight[i]( p\textsubscript{out} )
            sum += weight[i]( p\textsubscript{out} )
        p\textsubscript{in} /= sum
        Img\textsubscript{out}(p\textsubscript{out}) = Img\textsubscript{in}( p\textsubscript{in} )
    return Img\textsubscript{out}
}
Morphing at $\alpha \in [0, 1]$

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

- Compute the $\alpha$-blend of the 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 $\alpha$-blend of the warped images
Morphing at $\alpha \in [0, 1]$ Pseudocode

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

    $Warp_0 = $ Warp($Img_0, L_0[], L_\alpha[]$)
    $Warp_1 = $ Warp($Img_1, L_1[], L_\alpha[]$)

    return $(1-\alpha) \ast Warp_0 + \alpha \ast Warp_1$
}
[Beier & Neely, 1992] Example \((\alpha = 0.5)\)

\[\text{Img}_0\]

\[\text{Warp}_0\]

\[\text{Img}_1\]

\[\text{Warp}_1\]
[Beier & Neely, 1992] Example $\ (\alpha = 0.5)$

$\text{Img}_0$  

$\text{Warp}_0$

$\text{Img}_1$  

$\text{Warp}_1$
[Beier & Neely, 1992] Example ($\alpha = 0.5$)

**Img$_0$**

**Warp$_0$**

**Result**

**Img$_1$**

**Warp$_1$**
[Beier & Neely, 1992] Example \((\alpha = 0.5)\)
Animation Pseudocode

Animate( Img₀, L₀[N], Img₁, L₁[N], Imgₜₙₜ[N+1] )
{
    foreach t∈{0,...,T}:
        Imgₜₙₜ[t] = Morph( Img₀, L₀[N], Img₁, L₁[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