Image warping/morphing

Digital Visual Effects
Yung-Yu Chuang

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

<table>
<thead>
<tr>
<th>i</th>
<th>62</th>
<th>79</th>
<th>23</th>
<th>119</th>
<th>120</th>
<th>105</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>10</td>
<td>11</td>
<td>16</td>
<td>46</td>
<td>46</td>
<td>0</td>
<td>0</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>176</td>
<td>135</td>
<td>5</td>
<td>198</td>
<td>191</td>
<td>68</td>
<td>0</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>29</td>
<td>35</td>
<td>37</td>
<td>0</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>99</td>
<td>144</td>
<td>147</td>
<td>197</td>
<td>102</td>
<td>62</td>
<td>208</td>
</tr>
<tr>
<td></td>
<td>255</td>
<td>252</td>
<td>0</td>
<td>166</td>
<td>123</td>
<td>62</td>
<td>0</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>196</td>
<td>83</td>
<td>127</td>
<td>17</td>
<td>1</td>
<td>0</td>
<td>99</td>
<td>50</td>
</tr>
</tbody>
</table>

Image warping

image filtering: change range of image
$$g(x) = h(f(x))$$

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

Image warping

image filtering: change range of image
$$g(x) = h(f(x))$$

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

image warping: change domain of image
$$h(x) = 0.5y + 0.5$$

$$h(x) = 2y$$
Parametric (global) warping

Examples of parametric warps:
- Translation
- Rotation
- Aspect
- Affine
- Perspective
- Cylindrical

**Parametric (global) warping**

- 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: $p' = M \cdot p$

**Scaling**

- **Scaling** a coordinate means multiplying each of its components by a scalar
- **Uniform scaling** means this scalar is the same for all components:
  \[
  f\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} 2x \\ 2y \end{bmatrix}
  \]
  \[
  g\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} 2x \\ 0.5y \end{bmatrix}
  \]

- **Non-uniform scaling**: different scalars per component:
  \[
  f\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} x' \\ y' \end{bmatrix}
  \]
  \[
  g\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} 2x \\ 0.5y \end{bmatrix}
  \]

- Translation $p = (x, y)$
- Transformation $T$:
  \[
  p' = (x', y')
  \]

- Scaling
- Uniform scaling
- Non-uniform scaling

- $f\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} x' \\ y' \end{bmatrix}$
- $g\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} 2x \\ 0.5y \end{bmatrix}$
Scaling

• Scaling operation:
  \[ x' = ax \]
  \[ y' = by \]

• Or, in matrix form:
  \[
  \begin{bmatrix}
  x' \\
  y'
  \end{bmatrix} =
  \begin{bmatrix}
  a & 0 \\
  0 & b
  \end{bmatrix}
  \begin{bmatrix}
  x \\
  y
  \end{bmatrix}
  \]

Scaling matrix \( S \)

What's inverse of \( S \)?

2-D Rotation

• This is easy to capture in matrix form:
  \[
  \begin{bmatrix}
  x' \\
  y'
  \end{bmatrix} =
  \begin{bmatrix}
  \cos(\theta) & -\sin(\theta) \\
  \sin(\theta) & \cos(\theta)
  \end{bmatrix}
  \begin{bmatrix}
  x \\
  y
  \end{bmatrix}
  \]

  \[
  R
  \]

• Even though \( \sin(\theta) \) and \( \cos(\theta) \) are nonlinear to \( \theta \),
  - \( x' \) is a linear combination of \( x \) and \( y \)
  - \( y' \) is a linear combination of \( x \) and \( y \)

• What is the inverse transformation?
  - Rotation by \(-\theta\)
  - For rotation matrices, \( \det(R) = 1 \) so \( R^{-1} = R^T \)

2x2 Matrices

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

  2D Identity?
  \[
  \begin{bmatrix}
  x' \\
  y'
  \end{bmatrix} =
  \begin{bmatrix}
  1 & 0 \\
  0 & 1
  \end{bmatrix}
  \begin{bmatrix}
  x \\
  y
  \end{bmatrix}
  \]

  2D Scale around (0,0)?
  \[
  \begin{bmatrix}
  x' \\
  y'
  \end{bmatrix} =
  \begin{bmatrix}
  s_x & 0 \\
  0 & s_y
  \end{bmatrix}
  \begin{bmatrix}
  x \\
  y
  \end{bmatrix}
  \]

  2D Shear?
  \[
  \begin{bmatrix}
  x' \\
  y'
  \end{bmatrix} =
  \begin{bmatrix}
  1 & sh_x \\
  sh_y & 1
  \end{bmatrix}
  \begin{bmatrix}
  x \\
  y
  \end{bmatrix}
  \]

  2D Rotate around (0,0)?
  \[
  \begin{bmatrix}
  x' \\
  y'
  \end{bmatrix} =
  \begin{bmatrix}
  \cos(\theta) & -\sin(\theta) \\
  \sin(\theta) & \cos(\theta)
  \end{bmatrix}
  \begin{bmatrix}
  x \\
  y
  \end{bmatrix}
  \]

  \[
  R
  \]
2x2 Matrices

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

2D Mirror about Y axis?
\[
\begin{align*}
x' &= -x \\
y' &= y
\end{align*}
\]
\[
\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}
\]

2D Mirror over (0,0)?
\[
\begin{align*}
x' &= -x \\
y' &= -y
\end{align*}
\]
\[
\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}
\]

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?
\[
\begin{align*}
x' &= x + t_x \\
y' &= y + t_y
\end{align*}
\]
\[
\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}
\]

Translation

- Example of translation

Homogeneous Coordinates

\[
\begin{align*}
x' &= x + t_x \\
y' &= y + t_y
\end{align*}
\]

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

\[ \text{fwarp}(I, I', T) \]
\[
\begin{align*}
    &\{ \\
    &\quad \text{for } (y=0; y<I.\text{height}; y++) \\
    &\quad \quad \text{for } (x=0; x<I.\text{width}; x++) \\
    &\quad \quad \quad (x',y')=T(x,y); \\
    &\quad \quad \quad I'(x',y')=I(x,y); \\
    &\quad \} \\
\end{align*}
\]

- 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)

\[ h(x) \]
\[ f(x) \]
\[ g(x') \]

Some destination may not be covered

Many source pixels could map to the same destination

\[ \text{fwarp}(I, I', T) \]
\[
\begin{align*}
    &\{ \\
    &\quad \text{for } (y=0; y<I.\text{height}; y++) \\
    &\quad \quad \text{for } (x=0; x<I.\text{width}; x++) \\
    &\quad \quad \quad (x',y')=T(x,y); \\
    &\quad \quad \quad \text{Splatting}(I',x',y',I(x,y),\text{kernel}); \\
    &\quad \} \\
\end{align*}
\]

Answer: add “contribution” to several pixels, normalize later (splatting)
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

\[
iwarp(I, I', T)
\{
\text{for } (y=0; y<I'.height; y++)
\text{for } (x=0; x<I'.width; x++)
\{
  (x, y) = T^{-1}(x', y');
  I'(x', y') = \text{Reconstruct}(I(x, y, kernel));
\}
\}\]
Inverse warping

- No hole, but must resample
- What value should you take for non-integer coordinate? Closest one?

Reconstruction

- Reconstruction generates an approximation to the original function. Error is called aliasing.
- Computed weighted sum of pixel neighborhood; output is weighted average of input, where weights are normalized values of filter kernel $k$

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

```python
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;
```
Triangle filter

![Triangle filter diagram](image)

Gaussian filter

![Gaussian filter diagram](image)

Sampling

![Sampling diagram](image)

Reconstruction

![Reconstruction diagram](image)

The reconstructed function is obtained by interpolating among the samples in some manner.
**Reconstruction (interpolation)**

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

**Bilinear interpolation (triangle filter)**

- A simple method for resampling images

\[
f(x, y) = (1 - a)(1 - b) f[i, j] + a(1 - b) f[i + 1, j] + ab f[i + 1, j + 1] + (1 - a)b f[i, j + 1]
\]

**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

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

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

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

radial basis function

Non-parametric image warping

\[ \Delta P = \frac{1}{K} \sum_i k_{X_i}(P') \Delta X_i \]
Image warping

- Warping is a useful operation for mosaics, video matching, view interpolation and so on.

An application of image warping: face beautification

Data-driven facial beautification

Facial beautification
Facial beautification

Training set
- Face images
  - 92 young Caucasian female
  - 33 young Caucasian male

Feature extraction
Feature extraction

- Extract 84 feature points by BTSM
- Delaunay triangulation -> 234D distance vector (normalized by the square root of face area)

Support vector regression (SVR)

- Similar concept to SVM, but for regression
- RBF kernels

Beautification engine

- Given the normalized distance vector $v$, generate a nearby vector $v'$ so that $f_b(v') > f_b(v)$
- Two options
  - KNN-based
  - SVR-based
KNN-based beautification

\[ w_i = \frac{b_i}{\|v - v_i\|} \]

\[ v' = \frac{\sum_{i=1}^{K} w_i v_i}{\sum_{i=1}^{K} w_i} \]

SVR-based beautification

- Directly use \( f_b \) to seek \( v' \)

\[ v' = \arg \min_u E(u), \text{ where } E(u) = -f_b(u) \]

- Use standard no-derivative direction set method for minimization
- Features were reduced to 35D by PCA

SVR-based beautification

- Problems: it sometimes yields distance vectors corresponding to invalid human face
- Solution: add log-likelihood term (LP)

\[ E(u) = (\alpha - 1)f_b(u) - \alpha LP(u) \]

- LP is approximated by modeling face space as a multivariate Gaussian distribution

\[ P(\hat{u}) = \frac{1}{(2\pi)^{N/2} \sqrt{\prod \lambda_i}} \prod \exp \left( -\frac{\beta_i^2}{2\lambda_i} \right) \]

- \( \hat{u} \)'s i-th component

\[ LP(\hat{u}) = \sum \frac{-\beta_i^2}{2\lambda_i} + \text{const} \]

u's projection in PCA space

PCA

\[ \lambda_1 \]

\[ \lambda_2 \]
Embedding and warping

- Convert modified distance vector \( v' \) to a new face landmark

\[
E(q_1, \ldots, q_N) = \sum_{e_{ij}} \alpha_{ij} \left( \|q_i - q_j\|^2 - d_{ij}^2 \right)^2
\]

1 if \( i \) and \( j \) belong to different facial features
10 otherwise

- A graph drawing problem referred to as a stress minimization problem, solved by LM algorithm for non-linear minimization

Distance embedding

Distance embedding

- Post processing to enforce similarity transform for features on eyes by minimizing

\[
\sum \|Sp_i - q_i\|^2
\]

\[
S = \begin{pmatrix}
  a & b & t_x \\
  -b & a & t_y \\
  0 & 0 & 1
\end{pmatrix}
\]
**User study**

<table>
<thead>
<tr>
<th>Method</th>
<th>Score (std)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original portrait</td>
<td>3.37 (0.49)</td>
</tr>
<tr>
<td>Warped to mean</td>
<td>3.75 (0.49)</td>
</tr>
<tr>
<td>KNN-beautified (best)</td>
<td>4.14 (0.51)</td>
</tr>
<tr>
<td>SVR-beautified</td>
<td>4.51 (0.49)</td>
</tr>
</tbody>
</table>

**Results (in training set)**

**Results (not in training set)**

**By parts**

- (a) eyes
- (b) mouth
- (c) full
- (d)
- (e) full
- (f)
Different degrees

50%  100%

Facial collage

Results

- video

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](image #1) ![dissolving](image) ![Image 2](image #2)

Artifacts of cross-dissolving

http://www.salavon.com/

Image morphing

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

  - shape (geometric)
  - color (photometric)

![image #1](warp) ![cross-dissolving](morphing) ![image #2](warp)
**Morphing sequence**

**Face averaging by morphing**

**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)**
An ideal example

middle face (t=0.5)

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':

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:

Multiple Lines

\[ D_i = X'_i - X_i \]

\[ \text{weight}[i] = \frac{\text{length}[i]^a}{\text{a + dist}[i]^b} \]

\( \text{length} \) = length of the line segment,
\( \text{dist} \) = 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 (I[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

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

Results

Michael Jackson's MTV "Black or White"

Multi-source morphing

• Specify keyframes and interpolate the lines for the inbetween frames
• Require a lot of tweaking
**Multi-source morphing**

![Multi-source morphing images](image)

**References**

- *Data-Driven Enhancement of Facial Attractiveness*, SIGGRAPH 2008

**Reference software**

- **Morphing software review**
- I used [FantaMorph](https://www.fantamorph.com) 30-day evaluation version. You can use any one you like.
Morphing is not only for faces