Image warping/morphing

Digital Visual Effects

Yung-Yu Chuang

with slides by Richard Szeliski, Steve Seitz, Tom Funkhouser and Alexei Efros
Image warping
Image formation
Sampling and quantization
What is an image

- We can think of an image as a function, $f : \mathbb{R}^2 \rightarrow \mathbb{R}$:
  - $f(x, y)$ 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

\[
\begin{array}{cccccccc}
62 & 79 & 23 & 119 & 120 & 105 & 4 & 0 \\
10 & 10 & 9 & 62 & 12 & 78 & 34 & 0 \\
10 & 58 & 197 & 46 & 46 & 0 & 0 & 48 \\
176 & 135 & 5 & 188 & 191 & 68 & 0 & 49 \\
2 & 1 & 1 & 29 & 26 & 37 & 0 & 77 \\
0 & 89 & 144 & 147 & 187 & 102 & 62 & 208 \\
255 & 252 & 0 & 166 & 123 & 62 & 0 & 31 \\
166 & 63 & 127 & 17 & 1 & 0 & 99 & 30
\end{array}
\]
Image warping

image filtering: change range of image
\[ g(x) = h(f(x)) \]
\[ h(y) = 0.5y + 0.5 \]

image warping: change domain of image
\[ g(x) = f(h(x)) \]
\[ h(y) = 2y \]
Image warping

image filtering: change range of image
\[ g(x) = h(f(x)) \]

\[ h(y) = 0.5y + 0.5 \]

image warping: change domain of image
\[ g(x) = f(h(x)) \]

\[ h([x,y]) = [x, y/2] \]
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: $p' = M*p$
  
  \[
  \begin{bmatrix}
  x' \\
  y'
  \end{bmatrix}
  =
  M
  \begin{bmatrix}
  x \\
  y
  \end{bmatrix}
  \]
Scaling

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

\[
\begin{align*}
\begin{bmatrix} x \\ y \end{bmatrix} & \rightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 2x \\ 2y \end{bmatrix} \\
\end{align*}
\]
Scaling

• Non-uniform scaling: different scalars per component:

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} = \begin{bmatrix}
x' \\
y'
\end{bmatrix}
\]

\[
f\left(\begin{bmatrix}
x \\
y
\end{bmatrix}\right) = g\left(\begin{bmatrix}
x' \\
y'
\end{bmatrix}\right)
\]

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} = \begin{bmatrix} 2x \\ 0.5y \end{bmatrix}
\]

x \times 2, 
y \times 0.5
Scaling

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

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

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?

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

\[
\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}
\]

2D Scale around (0,0)?

\[
\begin{align*}
x' &= s_x \times x \\
y' &= s_y \times y
\end{align*}
\]

\[
\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}
\]
2x2 Matrices

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

2D Rotate around (0,0)?

\[ x' = \cos \theta \times x - \sin \theta \times y \]
\[ y' = \sin \theta \times x + \cos \theta \times y \]

\[
\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}
\]

2D Shear?

\[ x' = x + sh_x \times y \]
\[ y' = sh_y \times x + y \]

\[
\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & sh_x \\ sh_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}
\]
**2x2 Matrices**

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

**2D Mirror about Y axis?**

\[
x' = -x \\
y' = y
\]

\[
\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}
\]

**2D Mirror over (0,0)?**

\[
x' = -x \\
y' = -y
\]

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

\[ x' = x + t_x \]
\[ y' = y + t_y \]

**NO!**

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

- Example of translation

Homogeneous Coordinates

\[
\begin{pmatrix}
x' \\
y' \\
1
\end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\
0 & 1 & t_y \\
0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\
y \\
1 \end{pmatrix} = \begin{pmatrix} x + t_x \\
y + t_y \\
1 \end{pmatrix}
\]

\(t_x = 2\)
\(t_y = 1\)
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

\[
\text{fwarp}(I, I', T) \\
\{ \\
\text{for (y=0; y<} I.\text{height; y++)} \\
\text{for (x=0; x<} I.\text{width; x++)} \{ \\
(x', y') = T(x, y); \\
I'(x', y') = I(x, y); \\
\}
\}
\]
Forward warping

Some destination may not be covered

Many source pixels could map to the same destination
Forward warping

- Send each pixel \( l(x) \) to its corresponding location \( x' = T(x) \) in \( l'(x') \)
- What if pixel lands “between” two pixels?
- Will be there holes?
- Answer: add “contribution” to several pixels, normalize later (splatting)
Forward warping

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

\[
\text{Splatting}(I', x', y', I(x, y), \text{kernel}); 
\]

\[
\begin{array}{c}
I \\
\hline
\text{I} \\
\hline
\end{array}
\]

\[
\begin{array}{c}
I' \\
\hline
\text{I'} \\
\hline
\end{array}
\]

\[
T \\
\hline
\text{T} \\
\hline
\end{array}
\]

\[
\begin{array}{c}
\text{Splatting}(I', x', y', I(x, y), \text{kernel}); \\
\end{array}
\]
Inverse warping

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

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

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

- What if pixel comes from “between” two pixels?
- Answer: resample color value from interpolated (prefiltered) source image
Inverse warping

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

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

- It could cause aliasing
Reconstruction

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

**sample value**

**sample position**

**sampling**

**reconstruction**
Reconstruction

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

\[ p = \frac{\sum_i k(q_i)q_i}{\sum_i k(q_i)} \]

color=0;
weights=0;
for all $q$’s dist < width
\[ d = \text{dist}(p, q); \]
\[ w = \text{kernel}(d); \]
\[ \text{color} += w*q.\text{color}; \]
\[ \text{weights} += w; \]
\[ p.\text{Color} = \text{color/weights}; \]
Triangle filter

Input

Output

\[(u_1, v_2)\]

\[(u, v)\]

\[(u_1, v_1)\]

\[(u_2, v_2)\]

a

b
Gaussian filter
Sampling

\textit{band limited}

\[ \text{III}(x) \]

\[ \downarrow \]

\[ \text{III}(s) \]

\[ \downarrow \]
Reconstruction

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

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

Warp
Barycentric coordinates

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

$$t_1 + t_2 + t_3 = 1$$
Non-parametric image warping

\[ P = w_A A + w_B B + w_C C \]

\[ P' = w_A' A' + w_B' B' + w_C' C' \]

Barycentric coordinate
Non-parametric image warping

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

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

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

radial basis function
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
Facial beautification

- **Original Facial Data**
  - feature points
  - distances vector

- **Training Set**
- **Beautification engine**
  - Modified distances vector
  - distance embedding

- **Input image**
- **Image warp**
- **Result image**
Training set

• Face images
  - 92 young Caucasian female
  - 33 young Caucasian male
Feature extraction

Original Facial Data
- feature points
- distances vector

Training Set

Beautification engine

Modified distances vector
- distance embedding

Input image
- image warp
- result image
Feature extraction

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

Original Facial Data
- feature points
- distances vector

Training Set

Beautification engine

Modified distances vector
- distance embedding

input image → image warp → result image
Support vector regression (SVR)

- Similar concept to SVM, but for regression
- RBF kernels
- $f_b(v)$

SVM: $f(x) = \text{sign}(w^T x + b)$

SVR: $f(x) = w^T x + b$

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} \|w\|^2 \\
\text{subject to} & \quad y_i(w^T \cdot x_i + b) \geq 1
\end{align*}
\]

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} \|w\|^2 \\
\text{subject to} & \quad \|y_i - (w^T \cdot x_i + b)\| \leq \varepsilon
\end{align*}
\]
Beautification process

• 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

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

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

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

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

\( \hat{u} \)’s i-th component
\( \beta_i \)’s projection in PCA space
\( \lambda_i \) i-th eigenvalue
PCA
Embedding and warping

Original Facial Data
- feature points
- distances vector

Training Set

Beautification engine

Modified distances vector

input image → image warp → result image
Distance embedding

- Convert modified distance vector $v'$ to a new face landmark

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

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

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

\[ \sum \left\| S p_i - q_i \right\|^2 \]

\[ S = \begin{pmatrix} a & b & t_x \\ -b & a & t_y \\ 0 & 0 & 1 \end{pmatrix} \]
Results (in training set)
## User study

<table>
<thead>
<tr>
<th>Method</th>
<th>Rating (Std Dev)</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 (not in training set)
By parts

eyes

(a) (b) (c)

mouth

(d) (e) full (f)
Different degrees

50%

100%
Facial collage
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.
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
Face averaging by morphing

average faces
Image morphing

create a morphing sequence: for each time $t$

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

$t=0$  
$t=0.33$  
$t=1$
An ideal example (in 2004)
An ideal example

t=0  middle face (t=0.5)  t=1
Warp specification (mesh warping)

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

  easy to implement, but less expressive
Warp specification

• How can we specify the warp
  2. Specify corresponding *points*
    • *interpolate* to a complete warping function
Solution: convert to mesh warping

1. Define a triangular mesh over the points
   - Same mesh in both images!
   - Now we have triangle-to-triangle correspondences

2. Warp each triangle separately from source to destination
   - How do we warp a triangle?
   - 3 points = affine warp!
   - Just like texture mapping
Warp specification (field warping)

- How can we specify the warp?
  3. Specify corresponding vectors
     - *interpolate* to a complete warping function
     - The Beier & Neely Algorithm
• Single line-pair PQ to P’Q’:

\[
\begin{align*}
u &= \frac{(X-P) \cdot (Q-P)}{\| Q-P \|^2} \\
v &= \frac{(X-P) \cdot \text{Perpendicular} (Q-P)}{\| Q-P \|} \\
X' &= P' + u \cdot (Q' - P') + \frac{v \cdot \text{Perpendicular} (Q' - P')}{\| Q' - P' \|}
\end{align*}
\]
Algorithm (single line-pair)

• For each X in the destination image:
  1. Find the corresponding u,v
  2. Find X’ in the source image for that u,v
  3. destinationImage(X) = sourceImage(X’)

• Examples:

Affine transformation
Multiple Lines

\[ D_i = X_i' - X_i \]

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

\text{length} = \text{length of the line segment,}
\text{dist} = \text{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]
        endforeach
        X' = XSum/WeightSum
        DestinationImage(X) = SourceImage(X')
    endforeach
end
return Destination
end
Resulting warp
Comparison to mesh morphing

- Pros: more expressive
- Cons: speed and control
Warp interpolation

• How do we create an intermediate warp at time t?
  - linear interpolation for line end-points
  - But, a line rotating 180 degrees will become 0 length in the middle
  - One solution is to interpolate line mid-point and orientation angle
Animation

GenerateAnimation(Image₀, L₀[...], Image₁, L₁[...])
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 L₀[i] to L₁[i].
      end
      Warp₀ = WarpImage(Image₀, L₀[...], L[...])
      Warp₁ = WarpImage(Image₁, L₁[...], L[...])
      foreach pixel p in FinalImage do
         FinalImage(p) = (1-t) Warp₀(p) + t Warp₁(p)
      end
   end
end
Animated sequences

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

Michael Jackson’s MTV “Black or White”
Multi-source morphing
Multi-source morphing
Woman in arts
References

- Data-Driven Enhancement of Facial Attractiveness, SIGGRAPH 2008