Image Morphing with the Beier-Neely Method

Feng Zhu

University of Victoria

fengzhu@uvic.ca

October 29, 2015
Overview

1. Introduction to Image Morphing
2. Beier-Neely Image Morphing Algorithm
3. Software Implementation
4. Result
5. Analysis
6. Conclusion
Image Morphing

- **Image Morphing** is an image processing technique to turn one image into another through a smooth transition.
- Source image is where the morphing starts.
- Target Image is where the morphing ends.
- Intermediate frames are the morphed images.

![Image morphing example](image.jpg)

**Figure 1.1**: Image morphing
First movies with morphing
- *Willow*, 1988
- *Indiana Jones and the Last Crusade*, 1989

First music video with morphing
- *Black or White*, Michael Jackson, 1991

Disney animations with speeding production
- Mickey Mouse
- *SpongeBob SquarePants*
- Gopher Broke
Morphing Techniques

  Relates image features with meshes; Interpolate between mesh nodes to generate frames in the transformation.

  Relates image features with directed line segments; Interpolate between line segments to generate frames.

  Apply surface interpolation over scattered data; Find a “minimally bended” smooth surface passing through all given points.
Image Blending

- Pixel-by-pixel color interpolation
- Produce cross-dissolving visual effect

Figure 2.1: Image cross-dissolving

- artificial, non-physical, with “double image” effect
- apply image warping to align object features in both images
Image Warping

- Warping performs coordinate transformations to distort spacial configuration of images.
- Warping maps each pixel from one position to another.

Figure 2.2: Image warping
Image Morphing in General

- Image Morphing = Image warping + Image blending

![Image Morphing Steps]

Source → Warp → Blend → Morphed Image

Target → Warp
Algorithm 1 General image morphing algorithm

Input: source image $S$, target image $D$
Output: a sequence of morphed images $\{I_t\}_{t=0}^1$

for each intermediate frame at stage $t \in [0, 1]$ do
    Warp image $S$: $W_S = \text{warp}(S, t)$
    Warp image $D$: $W_D = \text{warp}(D, t)$
    Blend $W_S$ and $W_D$: $I_t = \text{blend}(W_S, W_D, t)$
endfor

Since image blending is the same for all morphing algorithms, the difference lies in the image warping process.
Feature-based image morphing technique:

- Performs warping by using object features
- Features are **user-specified** directed **line segments**
- One-to-one correspondence between features

**Figure 2.4**: Feature line segments
Line-Segment Interpolation

- Morphing result consists of a sequence of intermediate frames
- Each frame is computed with its corresponding feature line segments
- Interpolate between feature line segments in source and target images
Warp with One Line-Segment Pair

1. \( \nu \): the perpendicular distance from \( X \) to line \( PQ \)
2. \( \lambda \): the distance from \( P \) to the projection of \( X \)
3. \( \lambda/||PQ|| = \lambda'/||P'Q'|| \)
Figure 2.5: One line-segment pair example
Each feature line segment is associated with a **weight** determining the influence

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

- \(a\), \(b\), and \(p\) control the influence of distance, weight, and length
Figure 2.7: Multiple line-segment pair example
The software consists of three programs:

- `select_features`
- `morph_images`
- `frames_to_video`

**Figure 3.1**: Software structure
Tools and Libraries

- Linux system with C++ a compiler that supports C++ 11
- Libraries such as SPL, CGAL, OpenGL, GLUT, and STL
- Free software FFmpeg
- Versions of tools verified to work:
  - GCC 4.8.2
  - SPL 1.1.15
  - CGAL 4.5.2
  - OpenGL/GLUT 3.0
  - FFmpeg 2.5.3
Graphical User Interface (GUI): Manually select feature line segments

**Input**: image files, names of corresponding feature data files

**Output**: data files with feature line segments
Feature Data File

Figure 3.2: Feature data file

- Entry indicates the number of features in the file
- Each line contains the endpoints of a feature line segment
**morph_images & frames_to_video Programs**

**morph_images:**
- **Input**: image files, corresponding feature data files
- **Output**: a sequence of intermediate frames (e.g., morphed images)
- **Options**: number of frames, basename, warping parameters, ...

**frames_to_video:**
- **Input**: intermediate frames
- **Output**: a video displaying the morphing result
Analysis

- Achieve satisfactory morphing visual effect
- Performance with $720 \times 486$ size, 100 features
  - 2 min/frame on SGI 4D25 (CPU 20MHz, Memory 64 MB)
  - 2 secs/frame on ASUS X455L (CPU 3.1GHz, Memory 8GB)
- Advantages and Disadvantages:
  - Expressive: Only the user-specified features affect the morphing, and others are blended smoothly
  - Efficient: Drawing line segments VS placing dozens of mesh points
  - Speed: Global computation, all the line segments need to be referenced for every pixel, slows down the speed
Conclusion

**Summary**
- Beier-Neely morphing algorithm produces reasonable results
- Our software has implemented the Beier-Neely method effectively

**Future Work**
- Automatic feature detection to reduce the amount of work
- Combine points, curves, and line segments
References

T. Beier and S. Neely (1992)
Feature-Based Image Metamorphosis
*ACM SIGGRAPH Computer Graphics* 26(2), 35-42

G. Wolberg (1998)
Image morphing: a survey
*The visual computer* 14(8), 360-372

A. V. Feciorescu (2020)
Image morphing techniques
*Journal of Industrial Design and Engineering Graphics* 6(1), 25-28
Linear Line-Segment Interpolation

**Figure 6.1**: Linear interpolation

- Calculate feature line segments for each intermediate frame
- Given \( PQ \) and \( P'Q' \), generate \( \{P_iQ_i\}_{i=1}^N \) by interpolation
- Incremental step \( \Delta P \): \( \Delta P = (P - P')/N \), \( \Delta Q = (Q - Q')/N \)
- For \( P_iQ_i \): \( P_i = P + \Delta P_i \), and \( Q_i = Q + \Delta Q_i \)
Calculations

\[ X' = P' + u \cdot (Q' - P') + \frac{v \cdot \text{perpendicular}(Q' - P')}{\|Q' - P'\|}, \] (1)

where

\[ u = \frac{\lambda}{\|Q - P\|}, \] (2)

\[ v = \frac{(X - P) \cdot \text{perpendicular}(Q - P)}{\|Q - P\|}, \] (3)

\[ \lambda = \frac{(X - P) \cdot (Q - P)}{\|Q - P\|}, \] (4)
Parameters $a$, $b$, $p$

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

- $a$ determines the smoothness and precision of the user’s control over the warping. A lower value of $a$ implies a tighter control but less smooth warping effect. The bigger the $a$ is, the less effect of distance is. ($a > 0$)

- $b$ determines how the influence of different feature line segments decays with distance. A large $b$ means a pixel will only be affected by the closest feature line segment, and a zero value implies every feature line segment has the same relative influence. ($b \in [0.5, 2]$)

- $p$ determines how the length of a feature line segment influences the weight. A zero value means length has no influence and a higher value means weight is affected more by length. ($p \in [0, 1]$)
Algorithm 2 Algorithm for transformation with multiple feature line-segment pairs

Input: source image $S$, feature line-segment set $P_1Q_1, P_2Q_2, \ldots, P_nQ_n$

Output: destination image $D$

1. **for** each pixel with position $X$ **do**
2.  \[ D_{\text{sum}} = (0, 0), \ W_{\text{sum}} = 0 \]
3.  **for** each $P_iQ_i$ **do**
4.    calculate $u$ and $v$ for $X$ based on $P_iQ_i$
5.    find $X'$ with the $u$ and $v$
6.    calculate displacement $d_i = X' - X$
7.    calculate the weight $\text{weight} = (\text{length}^p / (a + \text{distance}))^b$
8.    \[ D_{\text{sum}} = d_i \text{weight} + D_{\text{sum}} \]
9.    \[ W_{\text{sum}} = \text{weight} + W_{\text{sum}} \]
10. **endfor**
11. \[ X' = X + D_{\text{sum}} / W_{\text{sum}} \]
12. copy the value of the pixel at $X'$ to that of the pixel at $X$: \[ D(X) \equiv D(X') \]
Algorithm 3 Special cases

1: if $X'$ falls outside the image domain then
2: find the pixel coordinate $X'_C$ closest to $X'$ on the boundary of the source image
3: update $X'$: $X' = X'_C$
4: endif
5: if $X'$ contains non-integer coordinate then
6: find the pixel coordinate $X'_I$ by interpolating the neighbours of $X'$ and rounding the interpolation result
7: update $X'$: $X' = X'_I$
8: endif