Lecture 18: Tuesday, 4 November 1997

Today:
  Image Morphing
  Volume Morphing
Next Tuesday: HKN evaluations
Today, rest of semester: Advanced Topics
  Image/Shape Morphing
  Color Perception
  Rendering Acceleration (Ray Casting, Radiance)
  Rendering Acceleration (Visibility Computations)
  Rendering Hardware
  Graphics Careers

Last year's 6.837 project: web morpher
`http://web.mit.edu/sylee /www/morph/code/morp`

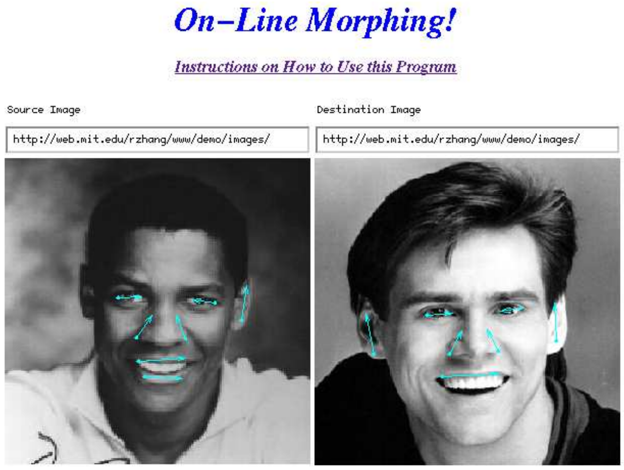
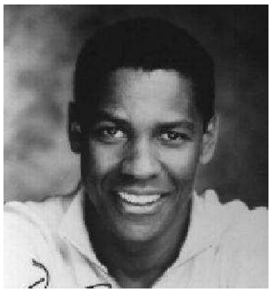
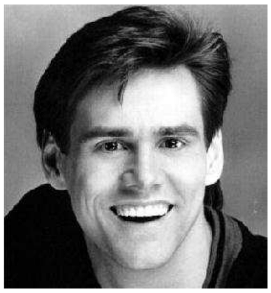


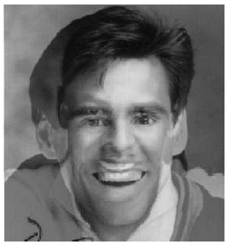
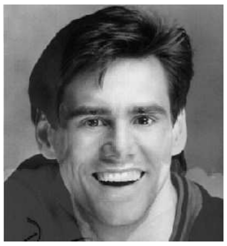
Image Morphing

From "metamorphosis"
What is morphing algorithm supposed to do?
  Continuously transform image A to image B

Cross-Dissolve

First attempt: *cross-dissolve* images:

  (Cross-dissolve at 25, 50, and 75%)
This is clearly unsatisfactory
True morphing algorithms combine
  geometric deformation
  color cross-dissolve
Today: Beier & Neely (Siggraph '92)
  (MJ "Black or White" video)

Forward/Backward Image Mapping

Forward mapping:
- For each source pixel p' in I_A
  - Find destination pixel p, color C
  - Copy source p' over to destination p
- Problem?

Most image algorithms use *backward mapping*:
- For each destination pixel p
  - Find source pixel(s) p'
  - Filter, copy source pixels to p

Beier/Neely Morphing

Define image $I(t)$, $0 \leq t \leq 1$
Specify *segment pairs* $P'_iQ'_i, P_iQ_i$ in I_A, I_B
Deform $P'_iQ'_i$ to P_iQ_i
Continuously cross-dissolve between I_A, I_B
  as function of parameter t
What are $I(0)$, $I(1)$?

Single Segment Pair

Simple: *coordinatize* images based on segments
- Define u as position *along* segment $0 \leq u \leq 1$
- Define v as *perpendicular distance* from segment
  - Note: segment is *oriented* from P to Q

For each destination pixel x, y in I_B
- Compute u, v for that pixel
- Find pixel x', y' at u, v in I_A
- Lerp from (x', y') to (x, y) and $I_A[x', y']$ to $I_B[x, y]$

Expressiveness

With only two line segments, can you get:
- Pure rotation?
- Pure translation?
- Anisotropic scaling?
- Isotropic scaling?
- General affine transformation (e.g., shear)?

Multiple Line Segments

Form *weighted sum* of segment effects
depending on distance from the line:

$$w = \left(\frac{L^p}{a+d}\right)^b$$

Where

a, b, p are adjustable global parameters
L is length of segment
d is distance from pixel to segment
(three cases, depending on u!)

Adjustable Parameters

a controls "strength" of segments:
Small a: segment dominates in its vicinity
Large a: smoother morphing but less control
Typically $a > 0$ (else undefined at crossings)

b controls "falloff" speed from all segments
Small b: equal effect from all segments
Large b: pixels affected only by "nearest" lines
Typically, $0.5 \leq b \leq 2$

p controls "length weight" of each segment
Typically, $0 \leq p \leq 1$

Multiple Line Segments

Once every point has a weight w
w.r.t. each line segment, we write

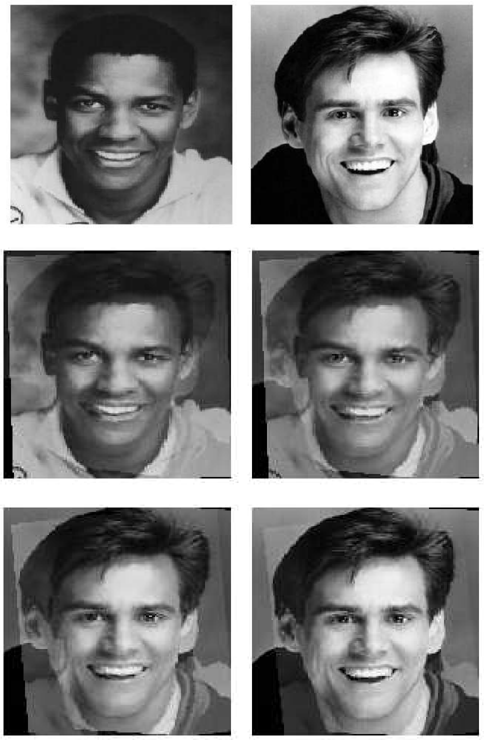
```
for each destination pixel x, y
  Dsum[2] = (0, 0);         // total displacement
  wsum = 0;                 // total weight
  for each segment (P_i, Q_i)
    compute (u, v) from (x, y, P_i, Q_i)
    compute (x', y') from (P_i', Q_i', u, v)
    D_i = (x' - x, y' - y)  // src/dest displacement
    L = length of segment (P_i, Q_i)
    d_i = distance from (x, y) to segment (P_i, Q_i)
          p              b
    w = (L  / (a + d_i))
    Dsum += D_i * w  // vector-valued
    wsum += w
  (x', y') = (x, y) + Dsum / wsum
  dst[x, y] = src[x', y']
```


Expressiveness

With two or more lines
Are uniform scales, shears possible?
Is transformation affine?

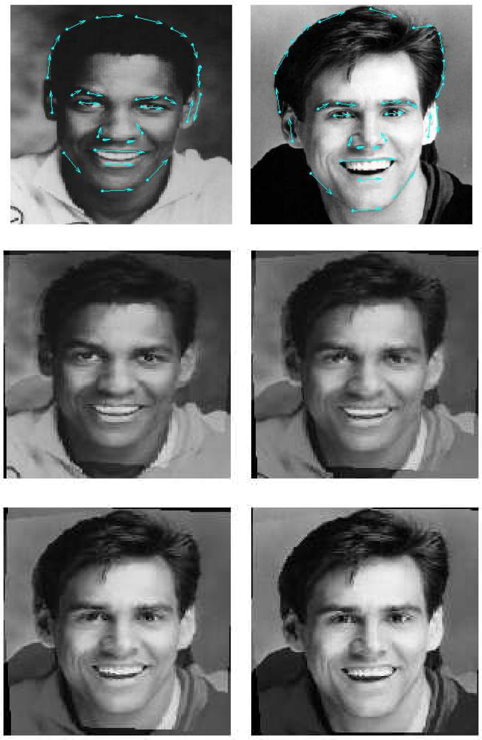
Examples

Source, Destination, segment pairs



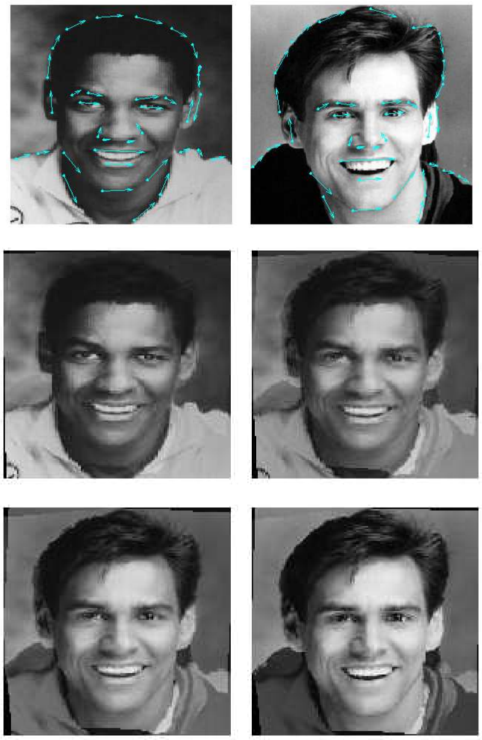
Examples

Source, Destination, segment pairs



Examples

Source, Destination, segment pairs



Problems with Image Morphing

Viewpoint:
  Must redo morph for each viewpoint change
Occlusion:
  Features invisible in source, destination
    cannot be made to appear in sequence
Ghosting
  Must add extra guide segments

Generalizations

Auto-adjustment of parameters
Automatic feature matching
In 2D, extend to spline curves
  Distance metric much more complex
Algorithm doesn't run at interactive speed
  Accelerate, in software or hardware

Volume Morphing (Lerios, Garfinkle, Levoy)

Extends image morphing to **volumetric** models
Requirements (source S, target T):
  Morphs should be plausible, realistic objects
  Renderings must depict smooth transition
LGL propose an interactive, two-pass algorithm:
  Warping: S, T warped to volumes S', T'
  Blending: S', T' combined into morph volume

Feature-Based 3D Warping

Human operator specifies feature pairs which
  are to be counterparts in the morph
Features are like magnets, shaped as
  points, segments, rectangles, and boxes
Each feature has a local origin, orthobasis,
  and extents along each basis vector
Associating two features f_s, f_t as a pair means
  that f_s will be moved, turned, and stretched
  to match its counterpart feature f_t

Warping Steps (Single Pair)

Interpolation: interpolate local coordinate
  systems, scaling factors to produce S'

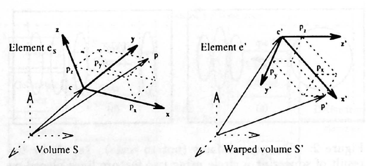


Figure 3: Single element warp. In order to find the point **p** in volume $\mathcal{S}$ that corresponds to $\mathbf{p}'$ in $\mathcal{S}'$, we first find the coordinates (p_x, p_y, p_z) of $\mathbf{p}'$ in the scaled local system of element e'; **p** is then the point with coordinates (p_x, p_y, p_z) in the scaled local system of element e_s. To simplify the figure, we have assumed unity scaling factors for all elements.

Inverse mapping: for each point p' in S',
  find corresponding point p in S by:
    expressing p' in basis of S' as p'_x, p'_y, p'_z
    forming linear combination of $\mathbf{x}, \mathbf{y}, \mathbf{z}$
      (Namely, $\mathbf{c} + p_x s_x \mathbf{x} + p_y s_y \mathbf{y} + p_z s_z \mathbf{z}$)

Warping All Pairs

Analogous to image morphing algorithm
Each pair defines a field acting throughout volume
Weighted averaging scheme gives point correspondences
inverse square law used, $w = \frac{1}{(d+\epsilon)^2}$
Points: d is distance from $\mathbf{p}'$ to $\mathbf{c}$
Segments: d is distance from support line
Rectangles: d is distance from plane
Boxes: d is distance from box's surface

Blending Stage

Smooths any misalignment of S', T'
Fades features with different color/opacity
Must carefully (non-linearly) interpolate opacity

Summary

So, to produce morph at t, $0 \leq t \leq 1$:
Produce S', lerping S to T (using t)
Produce T', lerping T to S (using $1 - t$)

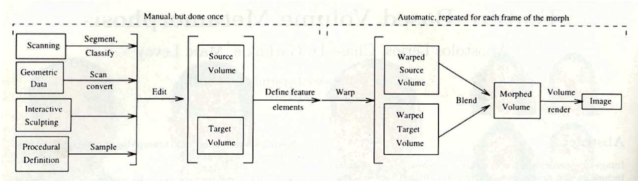


Figure 1: Data flow in a morphing system. Editing comprises retouching and aligning the volumes for cosmetic reasons.

Blend S' and T' into morphed volume
(Scan-convert, blending voxel values)

Performance

Each point influenced by all elements!
Warping time proportional to number of pairs
Quoted runtime of 160 minutes for a
$300 \times 300 \times 300$ volume with 30 pairs
Optimizations (yield factor of 50 speedup):
Approximate warp with piecewise linear function
Use octree subdivision of source, target

Single segment pair

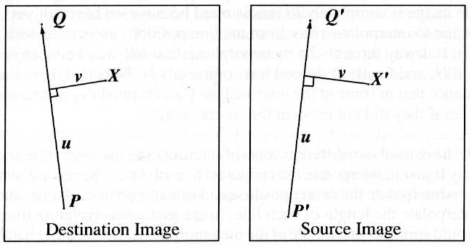


Figure 1: Single line pair

Action of single pair

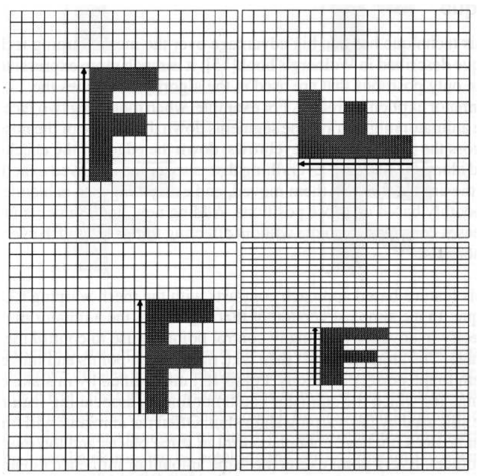


Figure 2: Single line pair examples

Multiple segment pairs

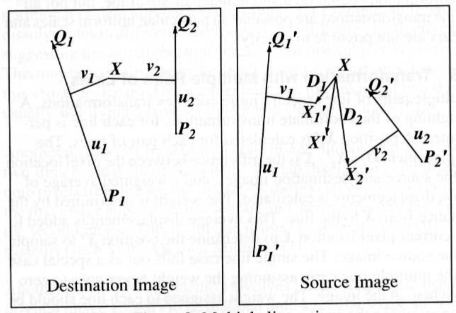


Figure 3: Multiple line pairs

Action of multiple pairs

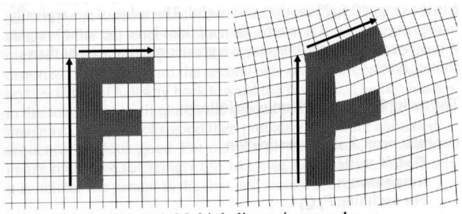


Figure 4: Multiple line pair example

Handling discontinuities

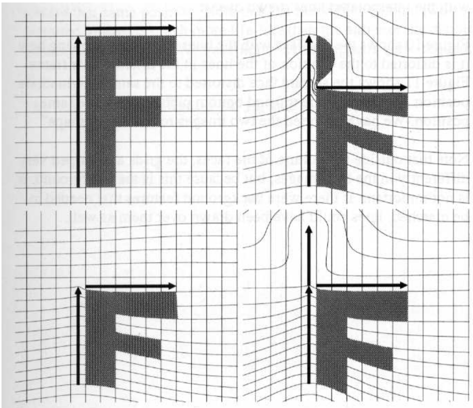


Figure 6: Ghostbusting