Pset question?
Pset submission system

• is experimental
  – belt & suspenders: also use stellar
  – send comments to Yichang

• tests are just for your convenience
  – grading will have more exhaustive tests
Pset 2

• is out
• is hard
• some answers are in these slides
Photo session

• After this lecture, at 2:30
  – 32−D424
  – alternative:
    Thursday 4pm
    Friday 3:30pm
  – let me know if you can’t make it
Domain operations
Domain transform

- Apply a function $f$ from $\mathbb{R}^2$ to $\mathbb{R}^2$ to the image domain
- If $(y, x)$ had color $c$ in the input, $f(y, x)$ has color $c$ in the output
Step Three:
Get the Push Left tool from the Toolbar (as shown here). It was called the Shift Pixels tool in Photoshop 6 and 7, but Adobe realized that you were getting used to the name, so they changed it, just to keep you off balance.

Step Four:
Choose a relatively small brush size (like the one shown here) using the Brush Size field near the top-right of the Liquify dialog. With it, paint a downward stroke starting just above and outside the love handle and continuing downward. The pixels shifts back in toward the body, removing the love handle as you paint. (Note: If you need to remove love handles on the left side of the body, paint upward rather than downward. Why? That’s just the way it works.) When you click OK, the love handle repair is complete.
Domain transform issues

• Apply a function $f$ from $\mathbb{R}^2$ to $\mathbb{R}^2$ to the image domain
• looks easy enough

• But two big issues:
  – which direction do we transform
  – how do we deal with non-integer coordinates?
  – And for warping: how do we specify $f$?
Take home messages

• Main loop over OUTPUT pixels
  – Makes sure you cover all of them
• Use inverse transform
• Reconstruction makes a difference
  – Linear much better than nearest neighbor
Basic programming tools
Loops & iterators

• Bad news: we’ll need some for loop for resampling
• iterators

```python
def imIter(im):
    for y in xrange(im.shape[0]):
        for x in xrange(im.shape[1]):
            yield y, x
```
Debug?
Padding problems

- Sometimes, we try to read outside the image
  - e.g. $y$, $x$ are negative
  - For example, we try to rotate an image
def getBlackPadded(im, y, x):
    if (x<0) or (x>=im.shape[1]) or (y<0) or (y>=im.shape[0]):
        return numpy.array([0, 0, 0])
    else: return im[y, x]
def clipX(im, x): return min(width(im) - 1, max(x, 0))
def clipY(im, y): return min(height(im) - 1, max(y, 0))
def getSafePix(im, y, x):
    return im[clipY(im, y), clipX(im, x)]
Questions?
Warping & Morphing
Important scientific question

• How to turn Dr. Jekyll into Mr. Hyde?
• How to turn a man into a werewolf?

• Powerpoint cross-fading?
Important scientific question

• How to turn Dr. Jekyll into Mr. Hyde?
• How to turn a man into a werewolf?

• Powerpoint cross-fading?

• or

• Image Warping and Morphing?

From An American Werewolf in London
Digression: old metamorphoses

- Unless I’m mistaken, both employ the trick of making already-applied makeup turn visible via changes in the color of the lighting, something that works only in black-and-white cinematography. It’s an interesting alternative to the more familiar Wolf Man time-lapse dissolves. This technique was used to great effect on Fredric March in Rouben Mamoulian’s 1932 film of *Dr. Jekyll and Mr. Hyde*, although Spencer Tracy eschewed extreme makeup for his 1941 portrayal.
Dr. Jekyll and Mr. Hyde, 1932
Dr. Jekyll and Mr. Hyde, 1932
Dr. Jekyll and Mr. Hyde, 1941
Challenge

• “Smoothly” transform a face into another
• Related: slow motion interpolation
  interpolate between key frames
Averaging images

• Cross-fading

\[
\text{output}[y, x] = t \cdot \text{im1}[y, x] + (1-t) \cdot \text{im2}[y, x]
\]
Problem with cross fading

• Features (eyes, mouth, etc) are not aligned
• It is probably not possible to get a global alignment
• We need to interpolate the LOCATION of features – domain transform!
Averaging points (location)

• P & Q are two 2D points (in the “domain”)
• V = t P + (1-t) Q
Warping

• Move pixel spatially: \( C'(x,y) = C(f(x,y)) \)
• Leave colors unchanged
Warping

• Deform the domain of images (not range)
• Central to morphing
• Also useful for
  – Optical aberration correction
  – Video stabilization
  – Slimming people down
  – ...
Recap & questions

- Color (range) interpolation:
  \[ \text{output}[y, x] = t \times \text{im1}[y, x] + (1-t) \times \text{im2}[y, x] \]

- Location (domain) interpolation: \( V = t \times P + (1-t) \times Q \)

- Warping: domain transform \( \text{out}(x, y) = \text{im}(f^{-1}(x, y)) \)
Morphing: combine both

• For each pixel
  – Transform its location like a vector (domain)
  – Then linearly interpolate colors (range)
Morphing

- Input: two images $I_0$ and $I_{N+1}$

- Expected output: image sequence $I_i$, with $i \in 1..N$

- User specifies sparse correspondences on the images
Morphing

• For each intermediate frame $I_t$
  – Interpolate feature locations $P^t_i = (1 - t) P^0_i + t P^1_i$
  – Perform two warps: one for $I_0$, one for $I_1$
    • Deduce a dense warp field from the pairs of features
    • Warp the pixels
  – Linearly interpolate the two warped images
Warping
Warping

- Imagine your image is made of rubber
- warp the rubber

No prairie dogs were armed when creating this image
Warping

Assume we are given the spatial mapping

- How do we compute the warped image?
Careful: warp vs. inverse warp

How do you perform a given warp:

- **Forward warp**
  - For each input pixel, compute output location and copy color there
Forward warp and gaps
Careful: warp vs. inverse warp

How do you perform a given warp:

• Forward warp
  – Potential gap problems

• Inverse lookup the most useful
  – For each output pixel
    • Lookup color at inverse-warped location in input
Questions?
How do we specify the warp?

- Before, we saw simple transformations
  - linear, affine, perspective

- But we want more flexibility
Image Warping – parametric

- Move control points to specify a spline warp
- Spline produces a smooth vector field
Warp specification - dense

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

But we want to specify only a few points, not a grid
Warp specification - sparse

- How can we specify the warp?
  - Specify corresponding *points*
  - *interpolate* to a complete warping function
  - How do we do it?

How do we go from feature points to pixels?
Beier and Neely

- Specify warp based on pairs of segments
  - [http://dl.acm.org/citation.cfm?id=134003](http://dl.acm.org/citation.cfm?id=134003)
  - Feature-Based Metamorphosis, SIGGRAPH 1992
  - Used in Michael Jackson’s Black and White Music Video
  - Pset 2!!
Questions?
Segment-based warping
Problem statement

- **Inputs:** One image, two lists of segments before and after, in the image domain
- **Goal:** warp the image “following” the displacement of the segments
Idea

- Each before/after pair of segment implies a planar transformation
  - simple and linear

Single line transforms
**Idea**

- Each before/after pair of segment implies a planar transformation
  - simple and linear

*Single line transforms*
Consider a pair of segments, corresponding to a before and after configuration. You need to implement the computation of the \(u\) and \(v\) coordinates of a 2D point with respect to a segment as described in the slides and in the paper. Given these coordinates, you can then compute the new \(x, y\) position of this point given the location of the other segment. Use simple examples to test your method.

Warping

Once you are convinced that you can transform 2D points according to a pair of before/after segments, implement a resampling function that warps an entire image according to such a pair of segments. Again, use simple examples to test this function. Once you are done with this, you have completed the hardest part of the assignment.

The function should be callable according to

\[
\text{warpBy1}(\text{im, segment}(0,0, 10,0), \text{segment}(10, 10, 30, 15))
\]

You can use the javascript UI to specify the segments, using the same image on both side for reference.

4.3 Warping according to multiple pairs of segments

Extend the above code to perform transformations according to multiple pairs of segments. For each pixel, transform its 2D coordinates according to each pair of segments and take a weighted average according to the length of each segment and the distance of the pixel to the segments. More specifically, the
Idea

- Each before/after pair of segment implies a planar transformation
- Then take weighted average of transformations

Single line transforms

Transform wrt 2 lines
Transform wrt 1 segment

- Define a coordinate system with respect to segment
  - 1 dimension, \( u \), along segment
  - 1 dimension, \( v \), orthogonal to segment
Transform wrt 1 segment

- Define a coordinate system with respect to segment
  - 1 dimension, $u$, along segment
  - 1 dimension, $v$, orthogonal to segment
- Compute $u$, $v$ in one image
  - The after one, because we use the inverse transform
- Compute point corresponding to $u$, $v$ in second image

![Destination Image](image1.jpg)

![Source Image](image2.jpg)
Computing $u, v$

- $u = \frac{\mathbf{P}X \cdot \mathbf{PQ}}{||\mathbf{PQ}||^2}$
  - this way $u$ is 0 at $P$ and 1 at $Q$
- $v = \frac{\mathbf{P}X \cdot \perp\mathbf{PQ}}{||\mathbf{PQ}||}$
  - where $\perp\mathbf{PQ}$ is $\mathbf{PQ}$ rotated by 90 degrees, and has length $||\mathbf{PQ}||$
  - unlike $u$ which is normalized, $v$ is in distance units
Transforming a point given u, v

- $X' = P' + uP'Q + v \frac{P'Q'}{||P'Q'||}$
- The u component is scaled according to segment scaling
- But v is absolute (see output3)
  - They say they tried to scale v as well but it didn’t work as well

![Diagram of input and output images with line transforms](image_url)
Questions?
Multiple segments

• For each point X
  – For each segment pair s_before[i], s_after[i]
    • Transform X into X’i according to this pair
  – Compute weighted average of all transformed X’i
    • weight according to distance to segments

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

where a, b, p control the influence

Transform wrt 2 lines
Distance to a segment

- Multiple cases...
Debugging: example

• Debugging my distance function
Questions?

• More debugging
Morphing
Input images
Segments
Interpolate segments

$t=0.5$
Warp images to segments\([t]\)

The red segments are at the same location in both images. Image features such as eyes are aligned.
Warp images to segments $[t]$

The red segments are at the same location in both images. Image features such as eyes are aligned.
Interpolate color
Interpolate color
Result
Recap

• For each intermediate frame $I_t$
  
  – Interpolate segment locations $y^t_i = (1 - t) x^{0}_i + t x^{1}_i$
  
  – Perform two warps: one for $I_0$, one for $I_1$
    – Deduce a dense warp field from the pairs of features
    – Warp the pixels
  
  – Linearly interpolate the two warped images
Michael Jackson’ BW

- Uses the very technique we just studied
Pset 2
Requirements

- Scaling using nearest-neighbor reconstruction
- Scaling with bi-linear reconstruction
- Rotation using linear reconstruction (6.865)
- Image warping according to one pair of segments
- Image warping according to two lists of segments, using weighted warping
- Image morphing
- Morph sequence between you and a peer (due on Wednesday).
The whacky UI

segmentsBefore=numpy.array([segment(87, 131, 109, 129),
segment(142, 126, 165, 129)])

segmentsAfter=numpy.array([segment(81, 112, 107, 107),
segment(140, 102, 163, 101)])
NB

• This is a hard Pset
• Code is going to be slow
• Start early
• Debug as you go
Photo session

• After this lecture, at 2:30
  – 32–D424
  – alternative:
    Thursday 4pm
    Friday 3:30pm
  – let me know if you can’t make it
Willow

- 1988, special effects by ILM
- First use of morphing
Women in Art video

http://youtube.com/watch?v=nUDIoN-_Hxs
Bells and whistles
Morphing & matting

• Extract foreground first to avoid artifacts in the background

(c) $\alpha = 0.0$
(d) $\alpha = 0.2$
(e) $\alpha = 0.4$

(f) $\alpha = 0.6$
(g) $\alpha = 0.8$
(h) $\alpha = 1.0$
Uniform morphing

Figure 4. Uniform metamorphosis
Non-uniform morphing

Figure 5. Nonuniform metamorphosis

http://www-cs.ccny.cuny.edu/~wolberg/pub/cgi96.pdf
Video

- Lots of manual work
Recap & Significance
Recap

• Idea that linear interpolation introduces blur
• Separation of shape and color
• Idea of non-rigid alignment of different images
  – Applications to medical data
• Applications, related to
  – Special effects
  – Face recognition
  – Video frame interpolation
  – MPEG
More morphing madness

• Gondry’s Rolling Stones Video
Application: Motion magnification

- with Liu, Torralba, Freeman & Adelson [Siggraph 2005]
- Analyze motion in video (robust to occlusion)
- Warp according to magnified motion
Motion magnification

✦ with Liu, Torralba, Freeman & Adelson [Siggraph 2005]
✦ Analyze motion in video (robust to occlusion)
✦ Magnify motion that is hard to see
Motion Magnification

Ce Liu
Antonio Torralba
William T. Freeman
Fredo Durand
Edward H. Adelson

Massachusetts Institute of Technology
Computer Science and Artificial Intelligence Laboratory
References
Refs

- http://www.visgraf.impa.br/cgi-bin/morphQuery.cgi?output=html
- http://www.cg.tuwien.ac.at/research/ca/mrm/
- http://w3.impa.br/~morph/sites.html
- http://w3.impa.br/~morph/sig-course/slides.html
- http://www.owlnet.rice.edu/~elec539/Projects97/morphjrks/morph.html
- http://www.fmrib.ox.ac.uk/~yongyue/thinplate.html
- http://www.uoguelph.ca/~mwirth/PHD_Chapter4.pdf
Software

- http://www.morpheussoftware.net/
- http://www.debugmode.com/winmorphe/  
- http://www.stoik.com/products/morphman/mm1_main.htm  
- http://www.creativecow.net/articles/zwar_chris/morph/index.html  
- http://meesoft.logicnet.dk/SmartMorph/  
- http://www.asahi-net.or.jp/~FX6M-FJMY/mop00e.html  
- http://morphing-software-review.toptenreviews.com/  
Advanced use of warping

Debugging
Debugging

• Doubt everything
• Debug pieces in isolation
  – Binary search/divide and conquer
• Display/print everything
  – In particular intermediate results
• Create simple inputs
  – where you can easily manually compute the result
  – e.g. constant image, edge image, etc.
  – use small images (e.g. 3x3)
  – including (especially) inputs to intermediate stages
  – use input that isolate different failure modes
• Change one thing at a time
  – e.g. to verify that a given command has the effect you want, modify it to break it