The small green squares are the same color. Hold your hand to the screen and block out the adjacent blocks, showing only the two green square, and you'll see how dramatic the chromatic induction effect is.

These two brown shades are different colors—yet the square blocks appear to be the same shade of brown. It is only when your eye follows the line down that you see how different the shades are. Cover the bottom part of the illustration with your hand and look only at the brown boxes to see the way the chromatic induction effect impacts the way you see color.
Domain Transforms, Warping & Morphing

Fredo Durand
MIT EECS
6.815/6.865
Piazza 101

- Do not email us technical question
  - But do email us about stuff such as extensions
- Ask one question per post
- Do not use the response field to say you have the same problem
- Write answers in the response field
Pset question?
Pset 2

• is out
• is hard
• some answers are in these slides
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
Transformation

- Simple parametric transformations
  - linear, affine, perspective, etc

illustration by Rick Szeliski
Warping

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

No prairie dogs were armed when creating this image
Application of warping: weight loss

- Liquify in photoshop
**figure 9.37**
Selecting the entire left side of the image avoids potential artifacts.

**figure 9.38**
Dragging a Free Transform handle to narrow the selected area.

**figure 9.39**
The Liquify filter’s Warp tool pushes pixels forward as you drag.
**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 $f$?
Questions?
Basic
resampling
Naive scaling

- Take all the pixels in input and transform them to their output location

\[ \text{im}[y, x] \Rightarrow \text{out}[k*y, k*x] \]

def scaleBAD(im, k):
    out = constantIm(im.shape[0]*k, im.shape[1]*k, 0)
    for y, x in imIter(im):
        out[k*y, k*x]= im[y, x]
    return out
Use the inverse transform!!!!!

- Main loop on output pixels
  - out[y, x] <= im[y/k, x/k]

```python
def scaleWithInverse(im, k):
    out = constantIm(im.shape[0]*k, im.shape[1]*k, 0.0)
    for y, x in imIter(out):
        out[y,x]=im[y/k, x/k]
    return out
```
Questions?
Remaining problem

- A little too “blocky”
- Because we round to the nearest integer pixel coordinates
  - called nearest neighbor reconstruction
Linear reconstruction

- im is now a 1D array along x
- reconstruct im[1.3]
- \( =0.7*im[1]+0.3*im[2] \)
Bilinear reconstruction

• Take 4 nearest neighbors
• Weight according to x & y fractional coordinates
• Can be done using two 1D linear reconstructions along x then y (or y then x)
Recall nearest neighbor
Bilinear
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
Questions?
Better reconstruction

• Consider more than 4 pixels:
  - bicubic, Lanczos, etc.

• Try to sharpen edges

• Use training database of low-res/high-res pairs
Bicubic (Photoshop)

Ignore small color issues
Bill Freeman’s super-resolution
Questions?
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]
Edge Padding

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, 1932
• Jekyll & Hide 1932:
  – 35:13
  – ch18 1:06:45
  – ch19 1:17:50

• Jekyll & Hide 1941:
  – ch20 1:25:13
Challenge

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

• Cross-fading

\[
\text{output}[y, x] = t \times \text{im1}[y, x] + (1-t) \times \text{im2}[y, x]
\]
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:
  \[ V = t \ P + (1-t) \ Q \]

• 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

Tuesday, February 14, 12
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?

Slide Alyosha Efros
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

![Diagram showing input and output images before and after warping.](image_url)
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
Test

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

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

\[
\ldots
\]

→ warpBy1(im, segment(0,0, 10,0), segment(10, 10, 30, 15)) 
→

\[
\ldots
\]
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
- Compute $u$, $v$ in one image
  - The after one, because we use the inverse transform
- Compute point corresponding to $u$, $v$ in second image
Computing \( u, v \)

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

- \( X' = P' + uP'Q + v \text{perpendicular}(P'Q')/||P'Q'|| \)
- 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 showing the transformation process with input and output images.](image)
Questions?
Multiple segments

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

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

where a, b, p control the influence

Transform wrt 2 lines
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
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([(87, 131, 109, 129), (142, 126, 165, 129)])

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

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

- Wednesday afternoon at 1pm or 2pm or 3pm or 4pm.
- In 32-D424
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
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
• Gondry’s Beck “Cell Phones are Dead”
Apps!

• Most polished app add ever: http://www.demoncam.com/
Making of Demoncam ad

• http://www.youtube.com/watch?v=YSBqXtOco5g&feature=related
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/winnmorph/
- 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/
• http://www.collegehumor.com/video/6705603/north-korean-photoshop-tutorial