Notes

- Assignment 3 on the web
  - Due November 17
  - Read a motion capture paper, write a report
- Final project: on the web
  - Dates:
    - November 12 (next Friday): Last day to finalize who is on your team and what you are doing (talk to me BEFORE then!)
    - December 3 (last lecture): Code due. See PDF file on the website for what else you need to do or submit

Notes (2)

- Assignment 2 questions?
- Assignment 2 movies
  - But don’t take them off the web yet (TA needs to see them too :-))

Morphing

- Before going further, let’s take an aside into a 2D animation technique: morphing
  - This will tie into 3D techniques later
- Basic idea:
  - We have two images (or video sequences) that we want to blend together
  - But simply cross-dissolving looks bad unless the image features line up just right
  - So as well as cross-dissolving, deform (“warp”) images to match features

Cross Dissolving

- Say we have two images, \( c_1(x,y) \) and \( c_2(x,y) \)
- Letting time \( t \) go from 0 to 1 (rescale to whatever you want) we define an intermediate image as
  \[
  c(x,y,t) = (1-t)c_1(x,y) + t c_2(x,y)
  \]
- More generally: let \( c_1 \) and \( c_2 \) depend on time too (videos)
- More generally: replace \((1-t)\) and \( t \) with \((1-f(x,y,t))\) and \( f(x,y,t) \) for any appropriate function \( f \)
  - E.g. dissolve part of the frame faster than another part
### Warping

- If a feature appears in both images (e.g., eyes) but at different locations, then both are partially visible in intermediate frames of cross-dissolve
  - We instead want features of first image to be replaced by features of second
- So figure out deformation that matches up the features
  - Deform image 1 to \(d_1(x,y)=c_1(warp_1(x,y))\) so that \(d_1(x,y)\) and \(c_2(x,y)\) have their features at the same \((x,y)\) locations
  - Deform image 2 to \(d_2(x,y)=c_2(warp_2(x,y))\) similarly
  - Note: maybe \(warp_1\) and \(warp_2\) are inverses

### Defining a warp

- Don’t want to have to specify where every pixel moves!

- Typical scenario:
  - Artist identifies features in image 1 and corresponding features in image 2
  - Computer automatically fills in rest of warp in a smooth way
  - Artist looks at result, adds more feature correspondences if unhappy
- Also of interest: have computer figure out corresponding features automatically

### Matching point features

- Simplest case (for computer): features are simply points
- Then it’s just a data interpolation problem: find smooth \(warp(x,y)\) so that \(warp(x_i,y_i)=(u_i,v_i)\) for each \(I\)
  - \((x_i,y_i)\) is the location of feature in one image
  - \((u_i,v_i)\) is the location in the other image
- Moving Least Squares (MLS)
- Radial Basis Functions (RBF)

### Better features

- If all you have is points, artist needs to do a lot of work
  - Silhouette must transform fairly exactly
  - Other high contrast curves too (e.g., eye lids, lips, ...)
  - So lots of points everywhere
  - Worse: need to get correspondence just right (don’t mix up points, and get arclength parameters right)
  - Editing correspondences is then painful
- Preferable to allow user to specify curves or at least line segments
Warping with line segments

- Textbook 3.8.2
- Features are line segments
- Define a rigid body + scale transformation to follow segment
- Each point \((x,y)\) gets mapped to some weighted average of where the transformations for each segment would take it
  - Weight according to distance from segment

Problems

- What about crossing lines?
- Weird things may happen: uninvertible maps, folding over, ...
- Lines not the ideal primitive for dealing with curvy image features