Computational Photography

Week 7

Instructor: Lou Kratz

Image Warping

All 2D Linear Transformations

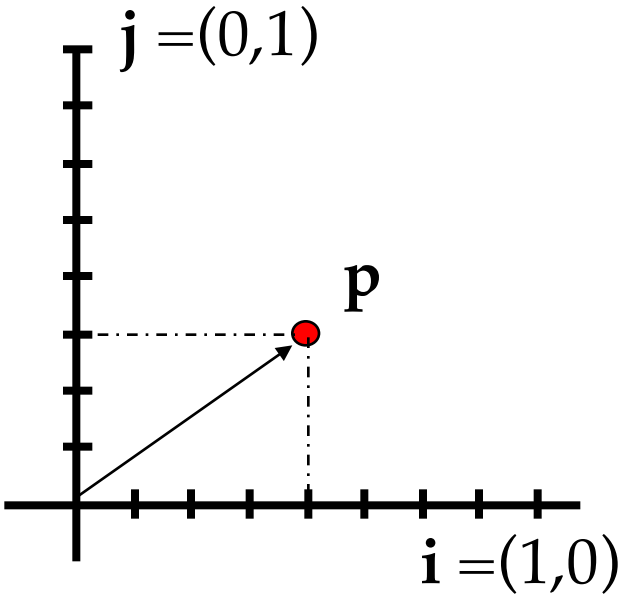
- Linear transformations are combinations of …
  - Scale,
  - Rotation,
  - Shear, and
  - Mirror

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

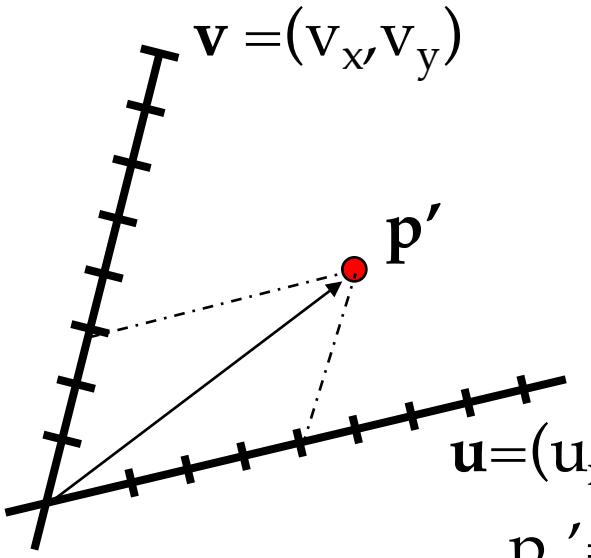
- Properties of linear transformations:
  - Origin maps to origin
  - Lines map to lines
  - Parallel lines remain parallel
  - Ratios are preserved
  - Closed under composition

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} \begin{bmatrix} i & j \\ k & l \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Linear Transformations as Change of Basis



$\mathbf{p}=4\mathbf{i}+3\mathbf{j} = (4,3)$



$\mathbf{p'}=4\mathbf{u}+3\mathbf{v}$

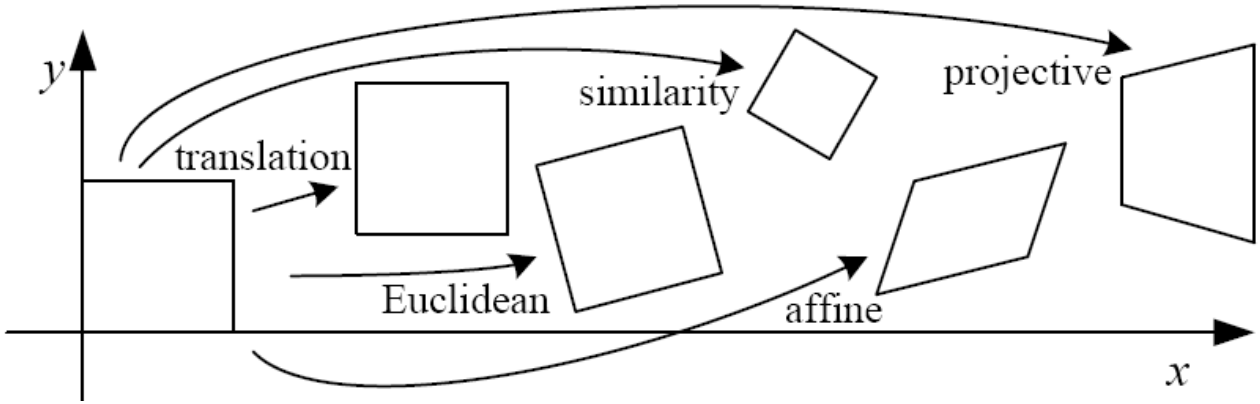
$p_x'=4u_x+3v_x$

$p_y'=4u_y+3v_y$

$$\mathbf{p}' = \begin{bmatrix} u_x & v_x \\ u_y & v_y \end{bmatrix} \begin{bmatrix} 4 \\ 3 \end{bmatrix} = \begin{bmatrix} u_x & v_x \\ u_y & v_y \end{bmatrix} \mathbf{p}$$

- Any linear transformation is a basis!!!

2D Image Transformations

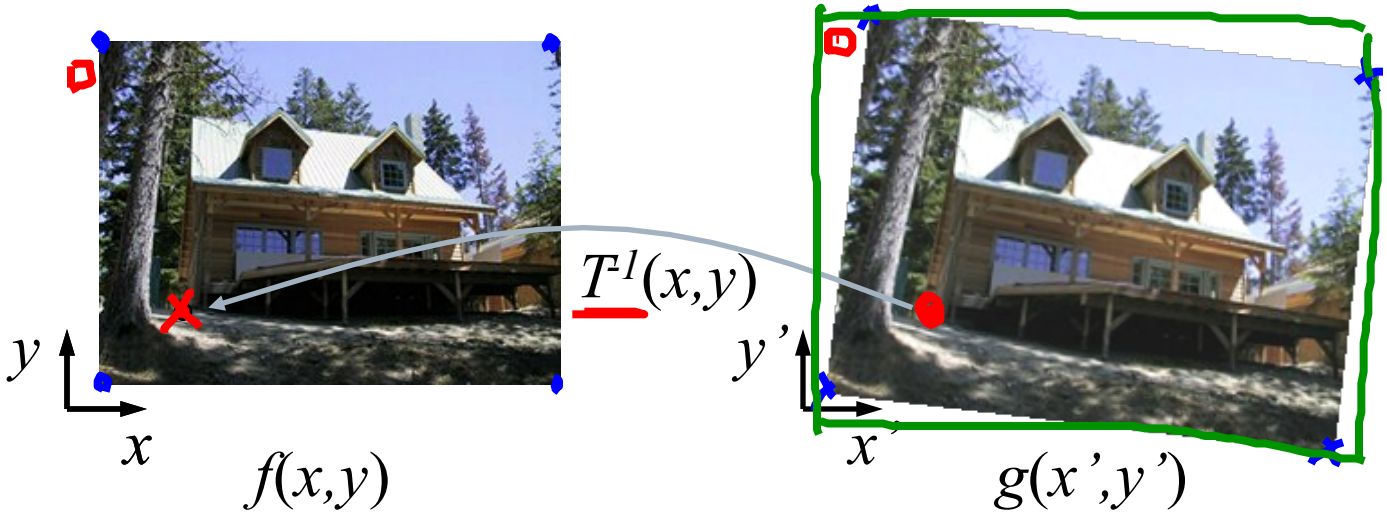


Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	2	orientation $+\cdots$	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	3	lengths $+\cdots$	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	4	angles $+\cdots$	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times3}$	6	parallelism $+\cdots$	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times3}$	8	straight lines	

These transformations are a nested set of groups

- Closed under composition and inverse is a member

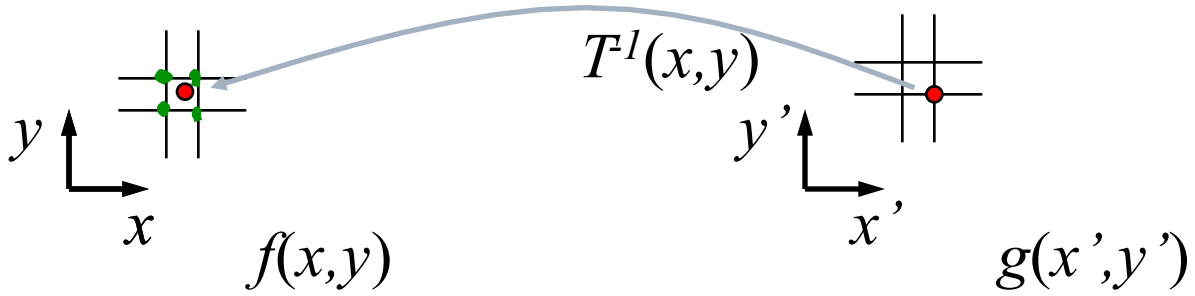
Inverse Warping



- Get each pixel $g(x',y')$ from its corresponding location
- $(x,y) = T^{-1}(x',y')$ in the first image

Q: what if pixel comes from "between" two pixels?

Inverse Warping



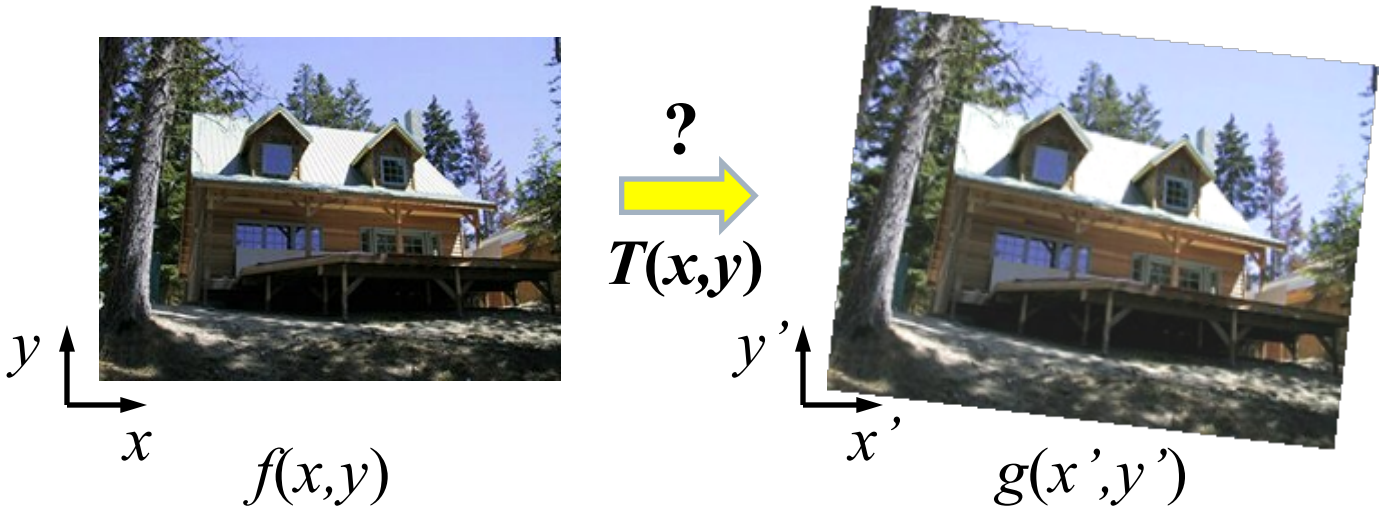
- Get each pixel $g(x',y')$ from its corresponding location
- $(x,y) = T^{-1}(x',y')$ in the first image

Q: what if pixel comes from "between" two pixels?
A: *Interpolate* color value from neighbors

- nearest neighbor, bilinear, Gaussian, bicubic

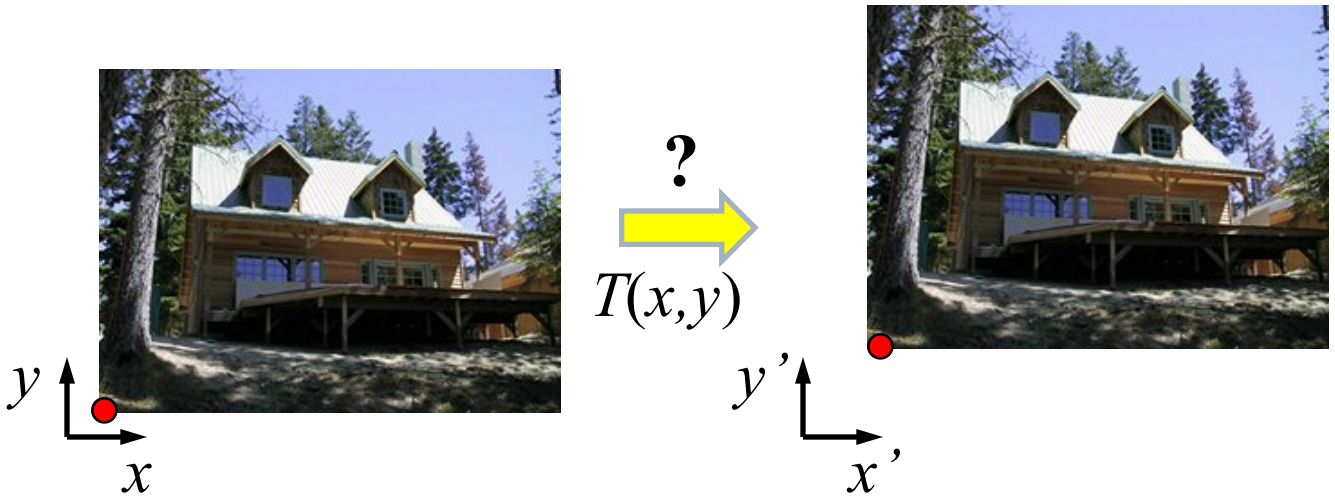
Image Morphing

Recovering Transformations



- What if we know *f* and *g* and want to recover the transform T?
  - willing to let user provide correspondences
    - How many do we need?

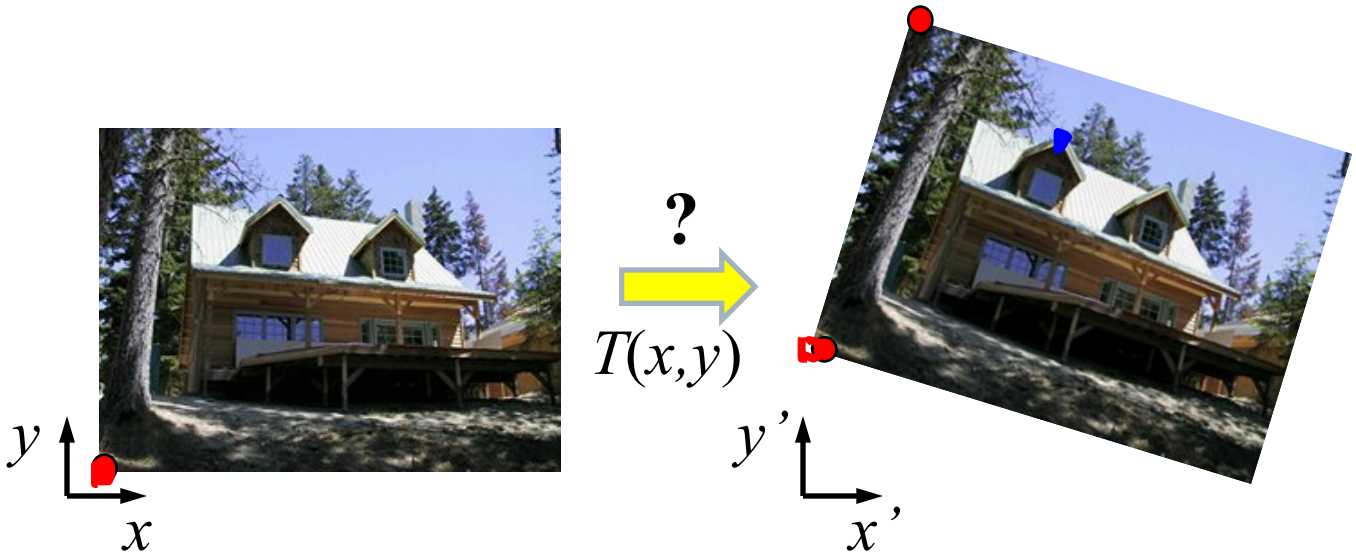
Translation: # correspondences?



- How many correspondences needed for translation?
- How many Degrees of Freedom?
- What is the transformation matrix?

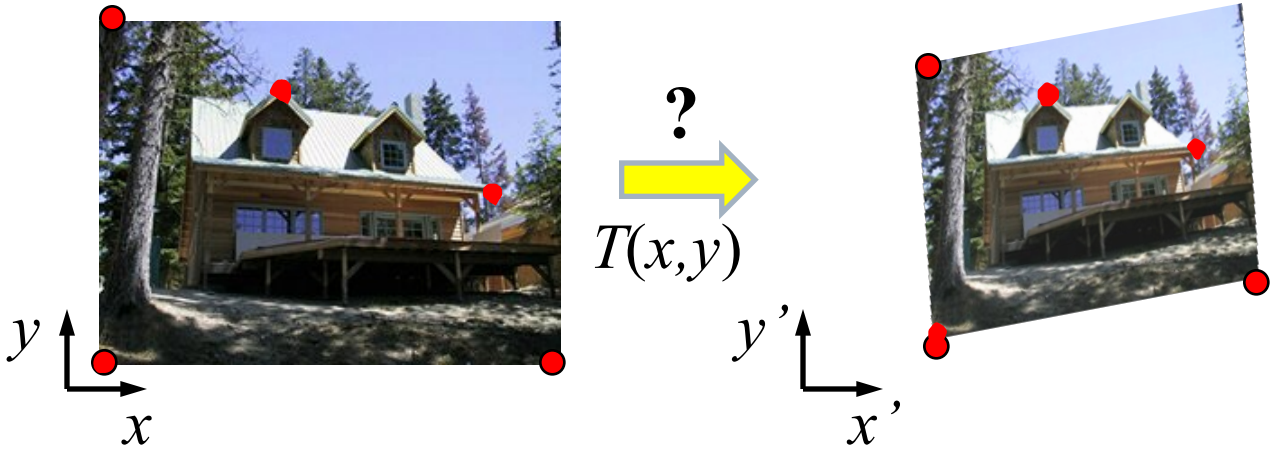
$$\mathbf{M} = \begin{bmatrix} 1 & 0 & p'_x - p_x \\ 0 & 1 & p'_y - p_y \\ 0 & 0 & 1 \end{bmatrix}$$

Euclidian: # correspondences?



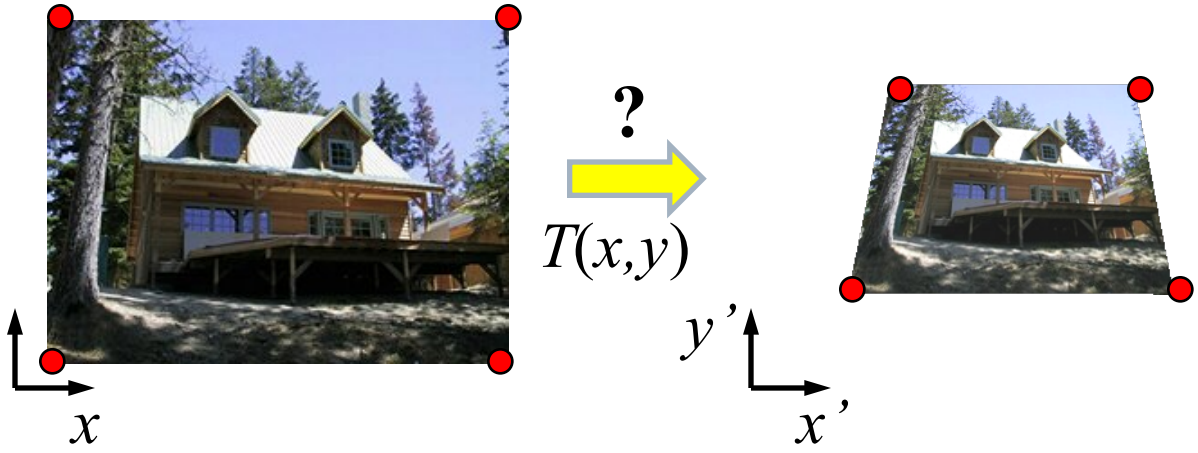
- How many correspondences needed for translation+rotation?
- How many DOF?

Affine: # correspondences?



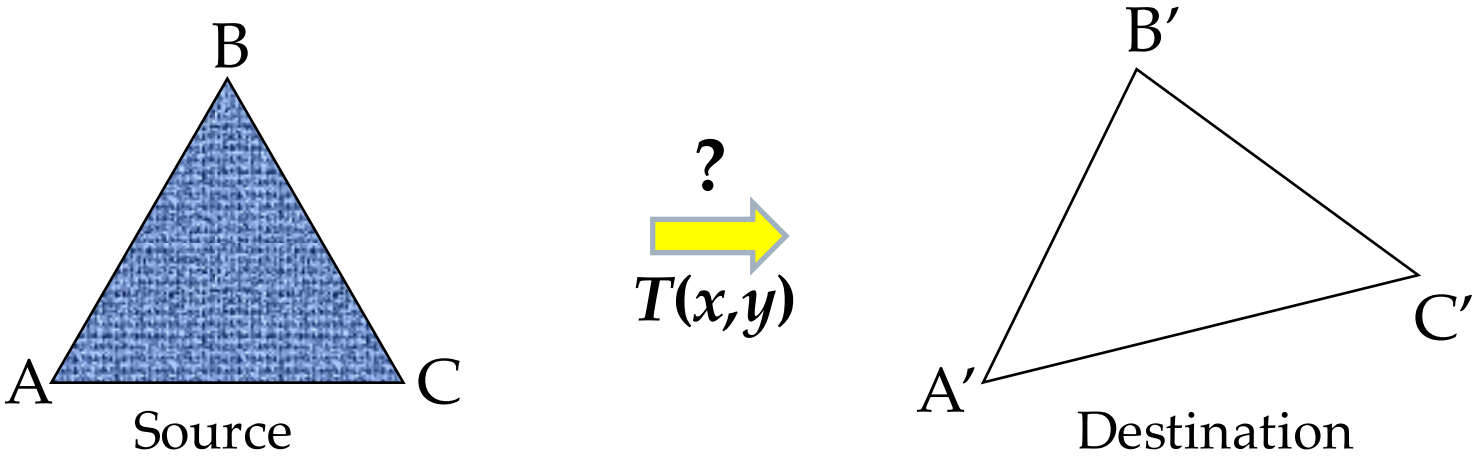
- How many correspondences needed for affine?
- How many DOF?

Projective: # correspondences?



- How many correspondences needed for projective?
- How many DOF?

Example: Warping Triangles

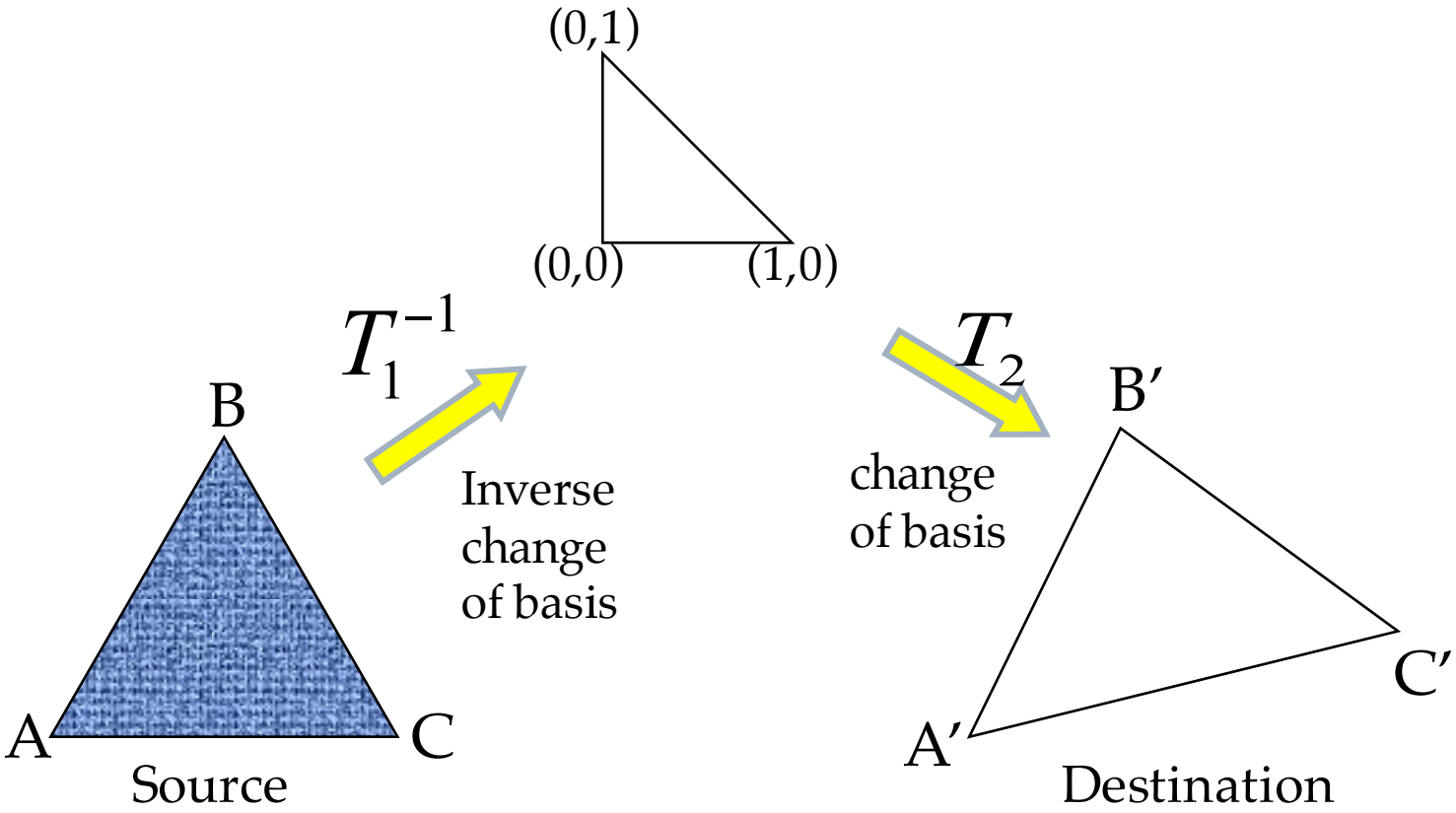


- Given two triangles: ABC and A′B′C′ in 2D (12 numbers)
- Need to find transform T to transfer all pixels from one to the other.
- What kind of transformation is T?
- How can we compute the transformation matrix:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Least Squares!

HINT: Warping Triangles

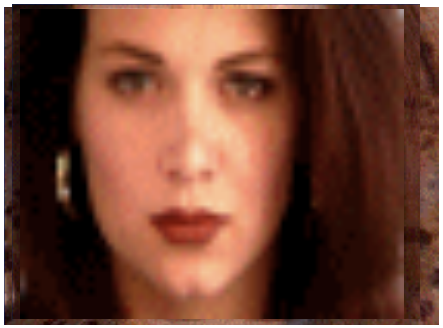


Don't forget to move the origin too!

- Very useful for Project 2…

Image Morphing

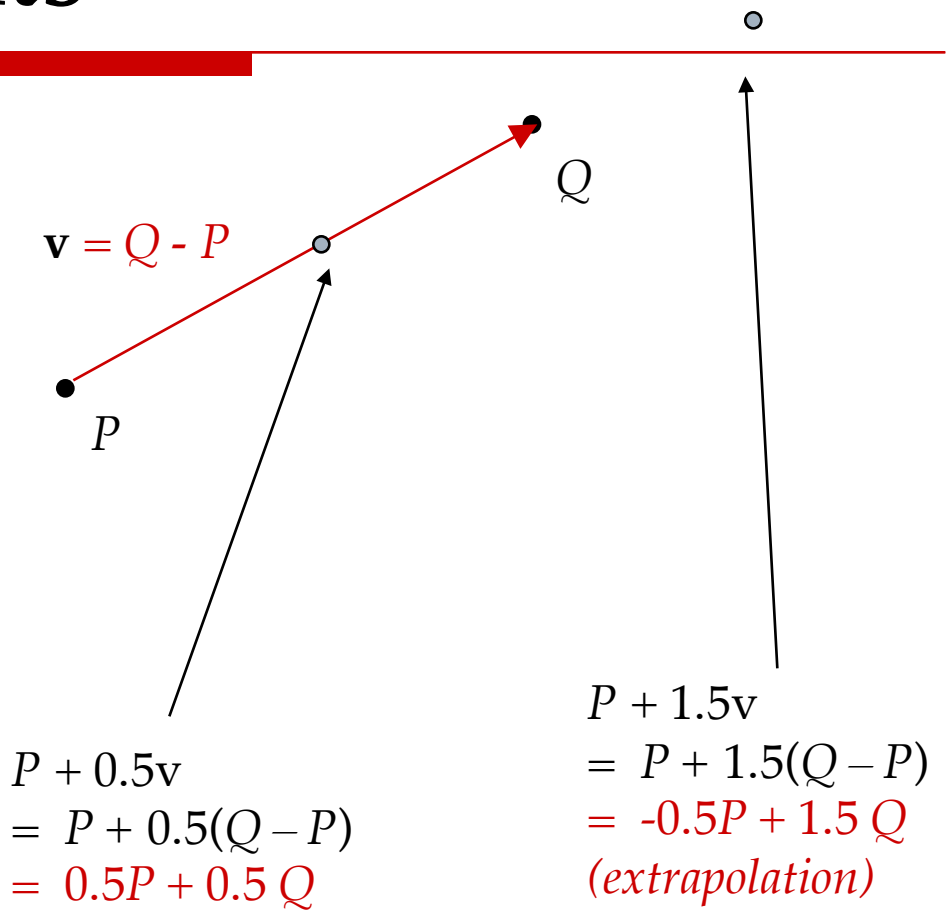
Morphing = Object Averaging


- The aim is to find "an average" between two objects
  - Not an average of two images of objects…
  - …but an image of the average object!
  - How can we make a smooth transition in time?
    - Do a "weighted average" over time t
- How do we know what the average object looks like?
  - We don't have a clue!
  - But we can often fake something reasonable
    - Usually required user/artist input

Averaging Points

What's the average of P and Q?



Linear Interpolation (Affine Combination):
New point $aP + bQ$,
defined only when $a+b = 1$
So $aP+bQ = aP+(1-a)Q$

- P and Q can be anything:
  - points on a plane (2D) or in space (3D)
  - Colors in RGB or HSV (3D)
  - Whole images (m-by-n D)… etc.

Idea #1: Cross-Dissolve

- Interpolate whole images:

  $Image_{halfway} = (1-t)*Image_1 + t*image_2$

- This is called **cross-dissolve** in film industry

- But what is the images are not aligned?

Idea #2: Align, then cross-disolve

- Align first, then cross-dissolve
  - Alignment using global warp – picture still valid

Dog Averaging

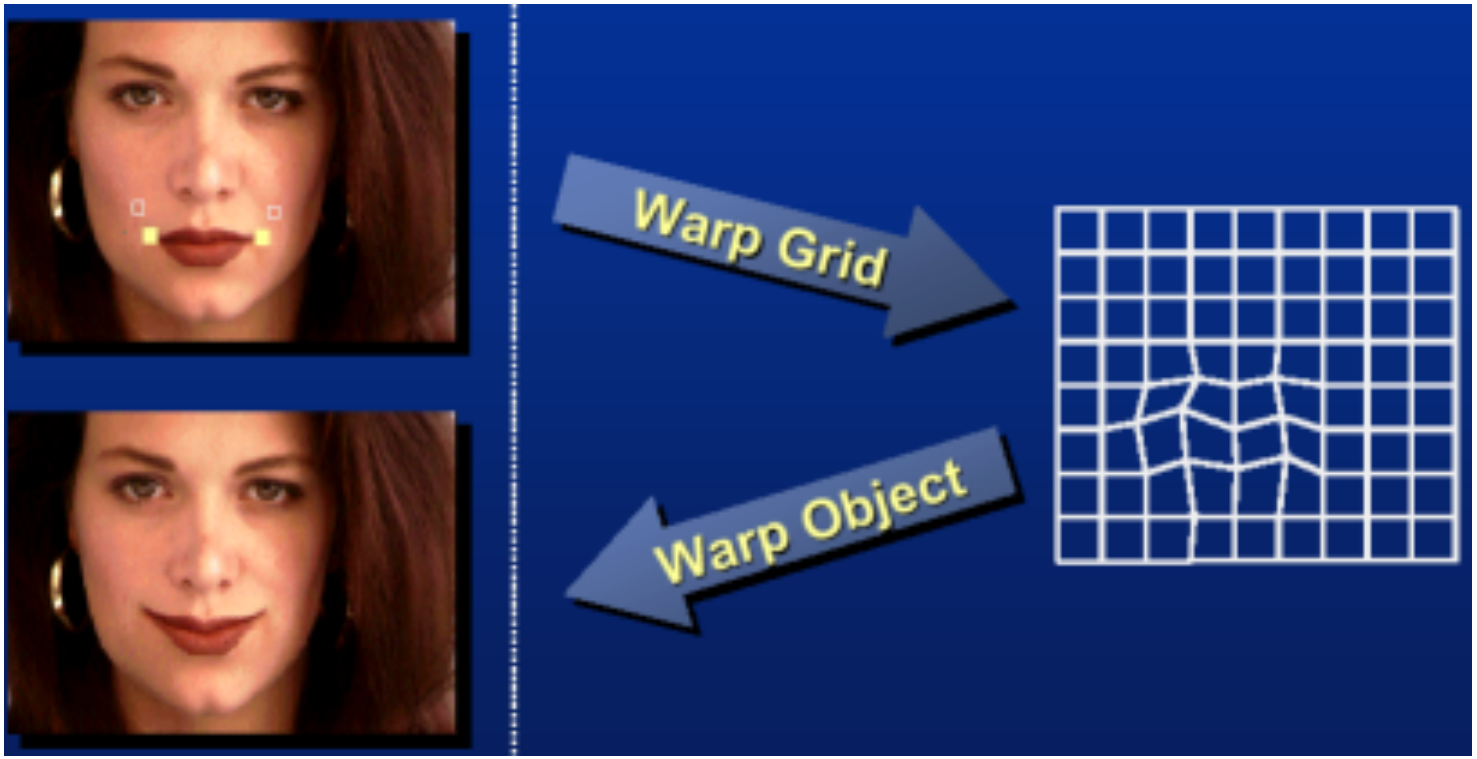
- What to do?
  - Cross-dissolve doesn't work
  - Global alignment doesn't work
    - Cannot be done with a global transformation (e.g. affine)
  - Any ideas?
- Feature matching!
  - Nose to nose, tail to tail, etc.
  - This is a local (non-parametric) warp

Local (non-parametric) Image Warping

- Need to specify a more detailed warp function
  - Global warps were functions of a few (2,4,8) parameters
  - Non-parametric warps u(x,y) and v(x,y) can be defined independently for every single location x,y!
  - Once we know vector field u,v we can easily warp each pixel (use backward warping with interpolation)

Image Warping – Non-parametric

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

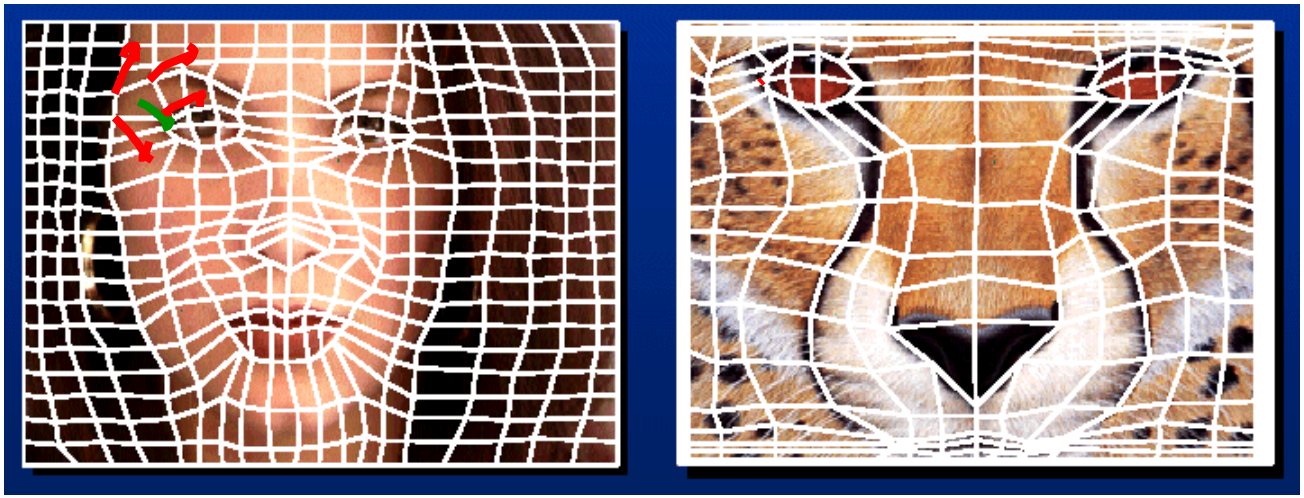


Warp Specification - Dense

- How can we specify the warp?

  Specify corresponding *spline control points*

  - *interpolate* to a complete warping function



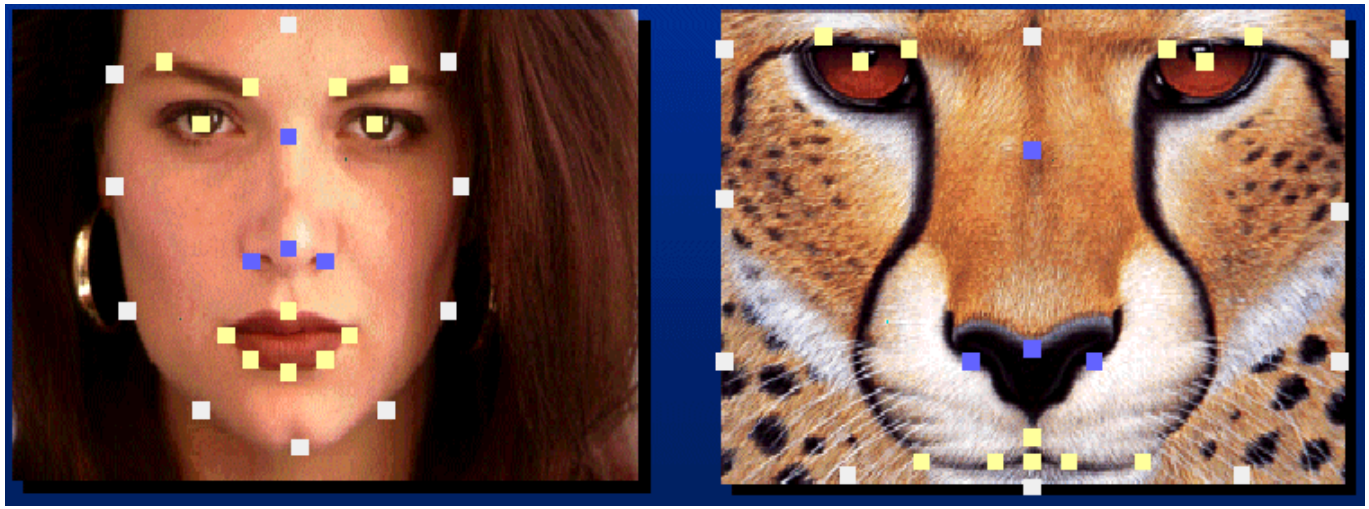
But we want to specify only a few points, not a grid

Warp Specification - Sparse

- How can we specify the warp?

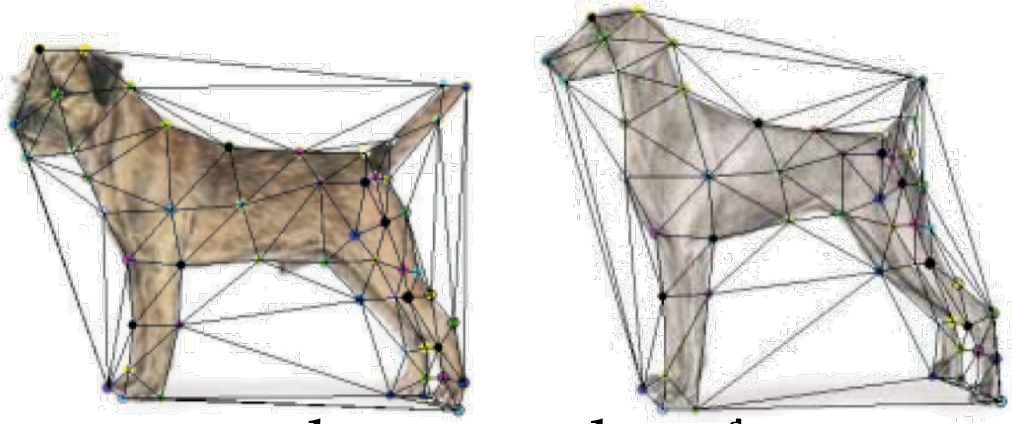
  Specify corresponding *points*

  - *interpolate* to a complete warping function
  - How do we do it?



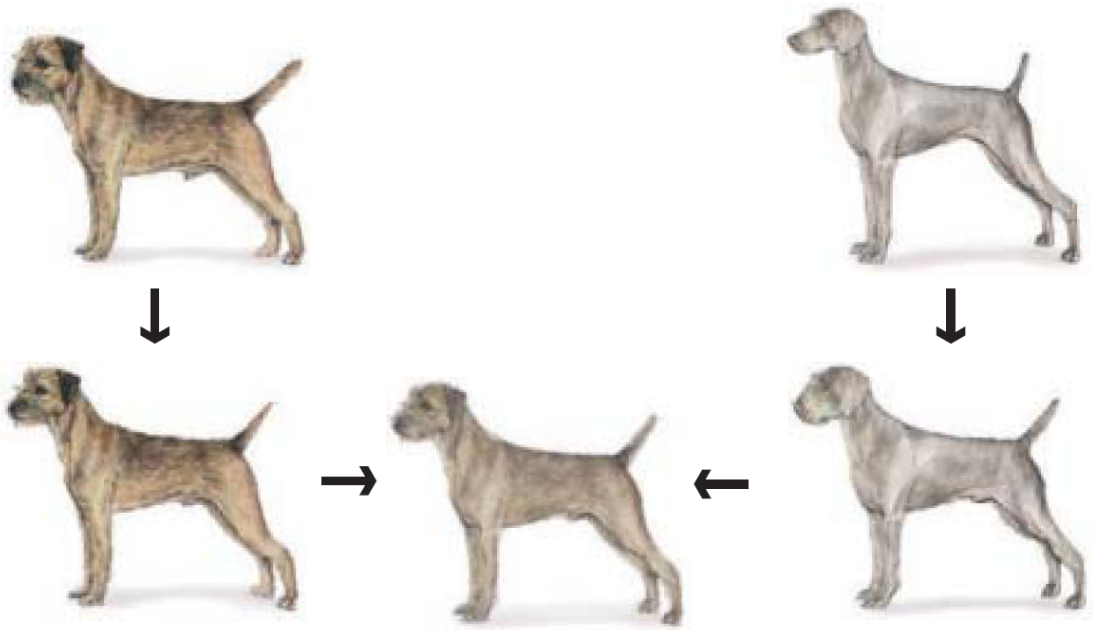
How do we go from feature points to pixels?

Triangular Mesh



1. Input correspondences at key feature points
2. Define a triangular mesh over the points
   - **Same mesh in both images!**
     - Now we have triangle-to-triangle correspondences
3. Warp each triangle separately from source to destination
   - How do we warp a triangle?
   - 3 points = affine warp!

Idea #3: Local warp, then cross-dissolve



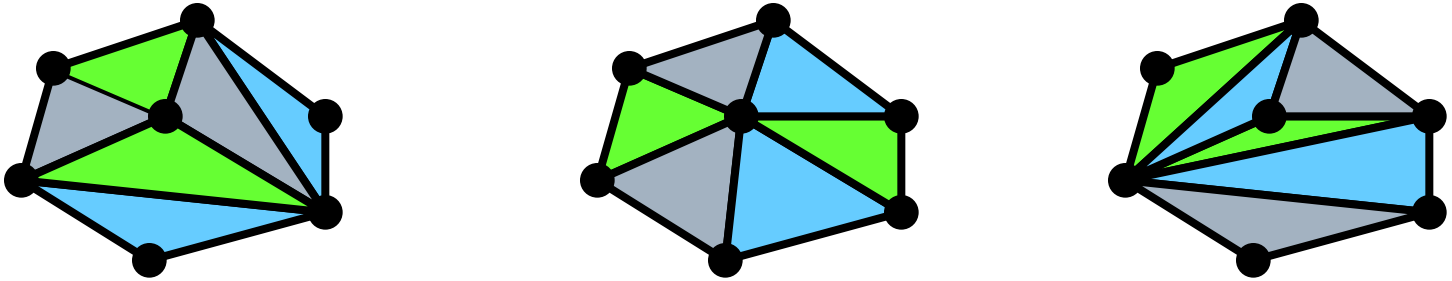
Morphing procedure:

for every triangle,

1. Find the average shape (the "mean dog"☺)
   - local warping
2. Find the average color
   - Cross-dissolve the warped images

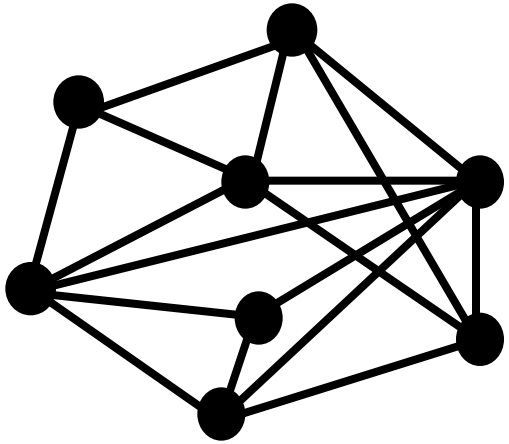
Triangulations

- A *triangulation* of set of points in the plane is a *partition* of the convex hull to triangles whose vertices are the points, and do not contain other points.
- There are an exponential number of triangulations of a point set.



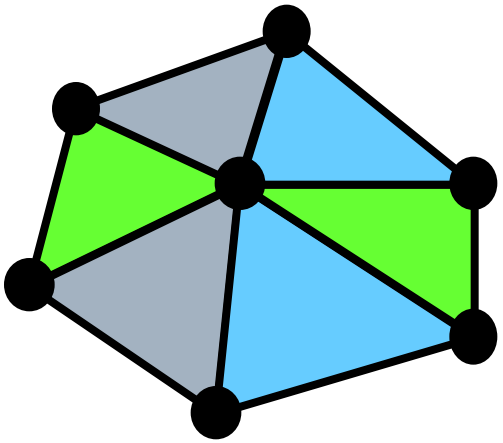
An O(n^3) Triangulation Algorithm

- Repeat until impossible:
  - Select two sites.
  - If the edge connecting them does not intersect previous edges, keep it.

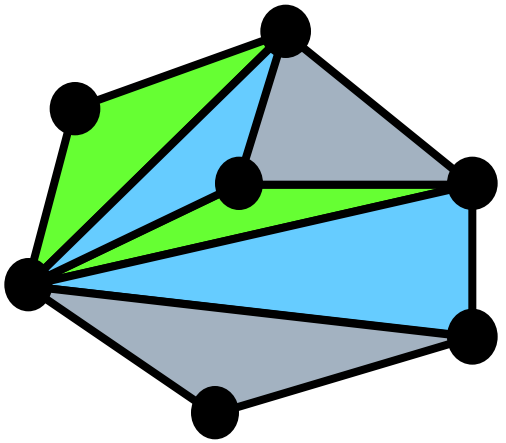


"Quality" Triangulations

- Let $a(T) = (a_1, a_2, .., a_{3t})$ be the vector of angles in the triangulation T in increasing order.
- A triangulation T_1 will be "better" than T_2 if $a(T_1) > a(T_2)$ lexicographically.
- The Delaunay triangulation is the "best"
  - Maximizes smallest angles



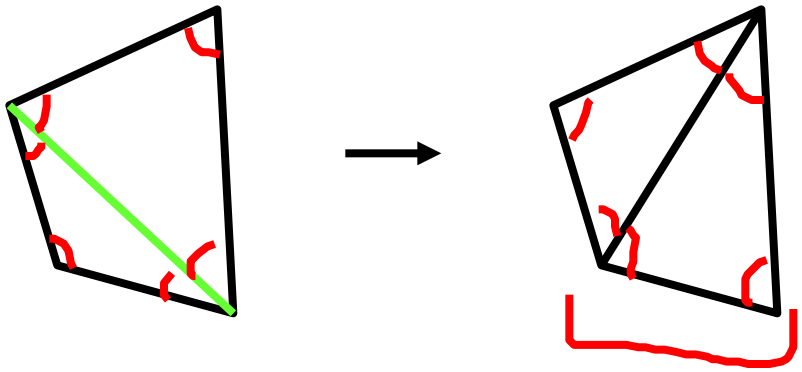
good



bad

Improving a Triangulation

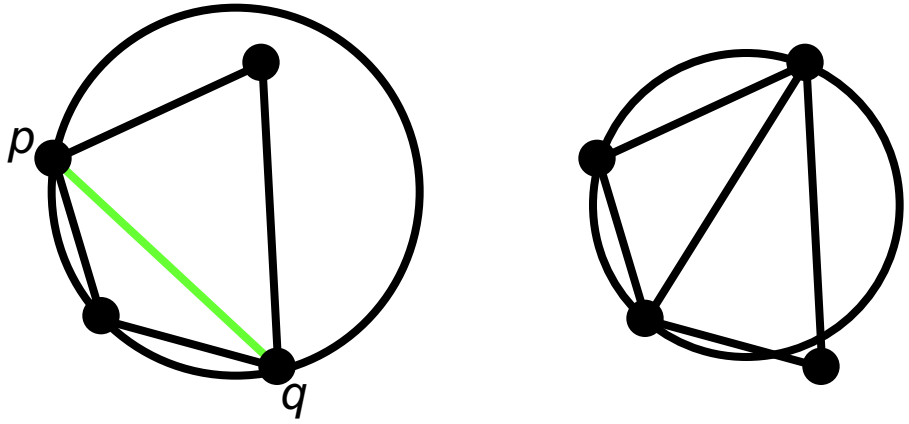
- In any convex quadrangle, an *edge flip* is possible. If this flip *improves* the triangulation locally, it also improves the global triangulation.



- If an edge flip improves the triangulation, the first edge is called *illegal*.

Illegal Edges

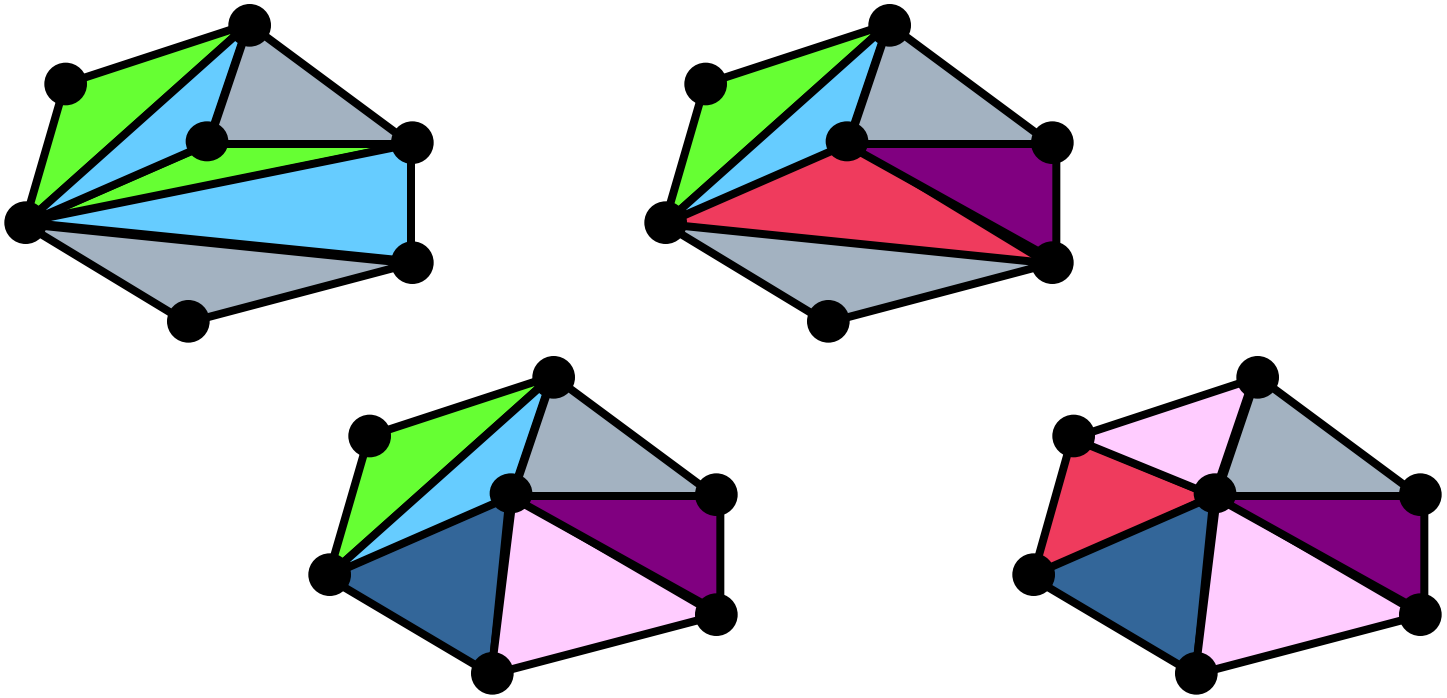
- **Lemma:** An edge *pq* is illegal iff one of its opposite vertices is inside the circle defined by the other three vertices.
- **Proof:** By Thales' theorem.



- **Theorem:** A Delaunay triangulation does not contain illegal edges.
- **Corollary:** A triangle is Delaunay iff the circle through its vertices is empty of other sites.
- **Corollary:** The Delaunay triangulation is not unique if more than three sites are co-circular.

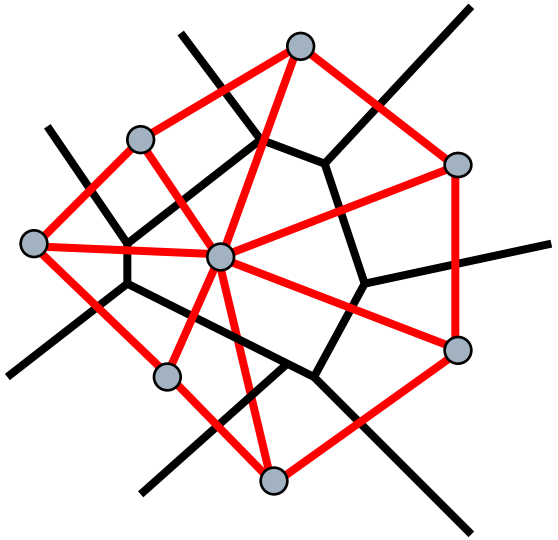
Naïve Delaunay Algorithm

- Start with an arbitrary triangulation. Flip any illegal edge until no more exist.
- Could take a long time to terminate.

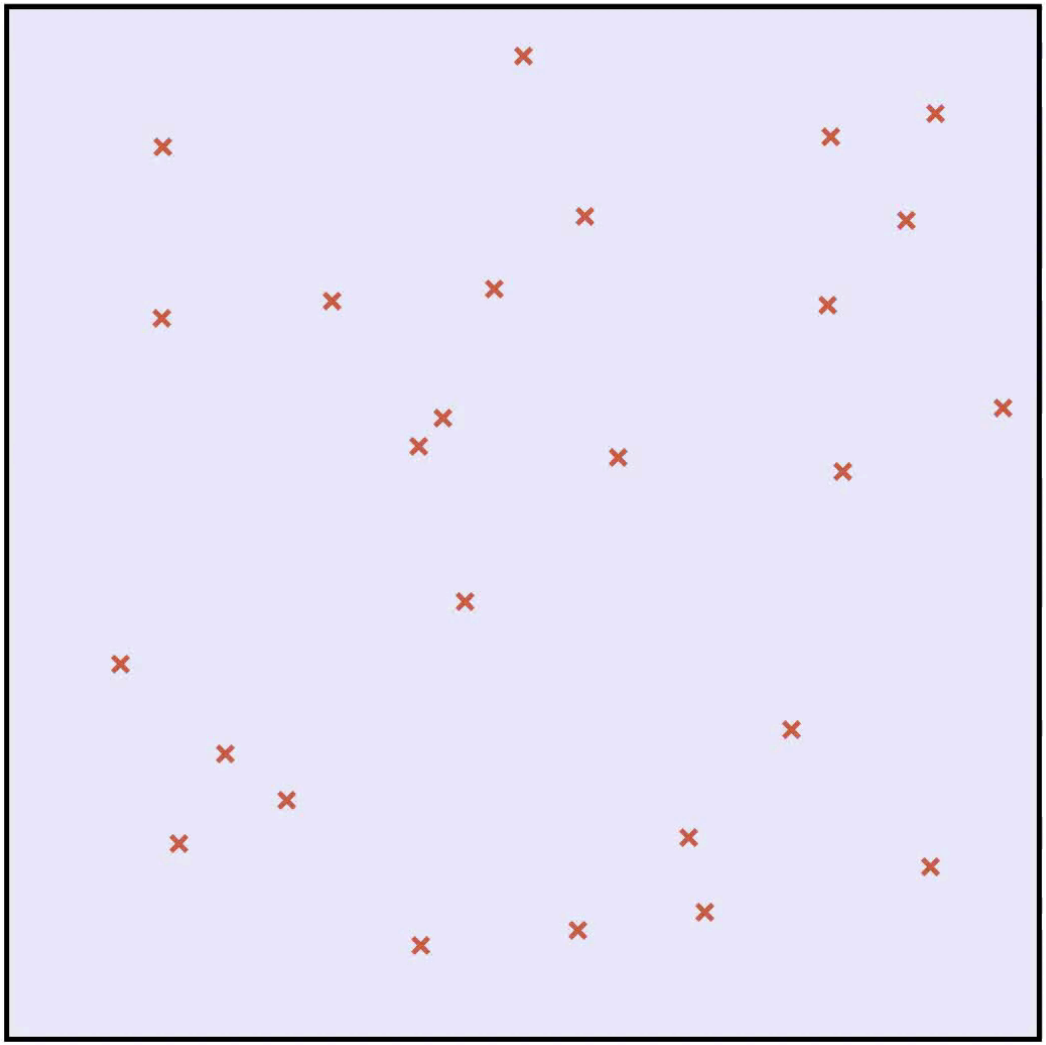


Delaunay Triangulation by Duality

- General position assumption: There are no four co-circular points.
- Draw the dual to the Voronoi diagram by connecting each two neighboring sites in the Voronoi diagram.
- **Corollary:** The DT may be constructed in O($n$log$n$) time.
- This is what scipy's `delaunay` function uses.



Voronoi Planar Sweep



https://www.youtube.com/watch?v=k2P9yWSMaXE

Image Morphing

- We know how to warp one image into the other, but how do we create a morphing sequence?
  1. Create an intermediate shape (by interpolation)
  2. Warp both images towards it
  3. Cross-dissolve the colors in the newly warped images

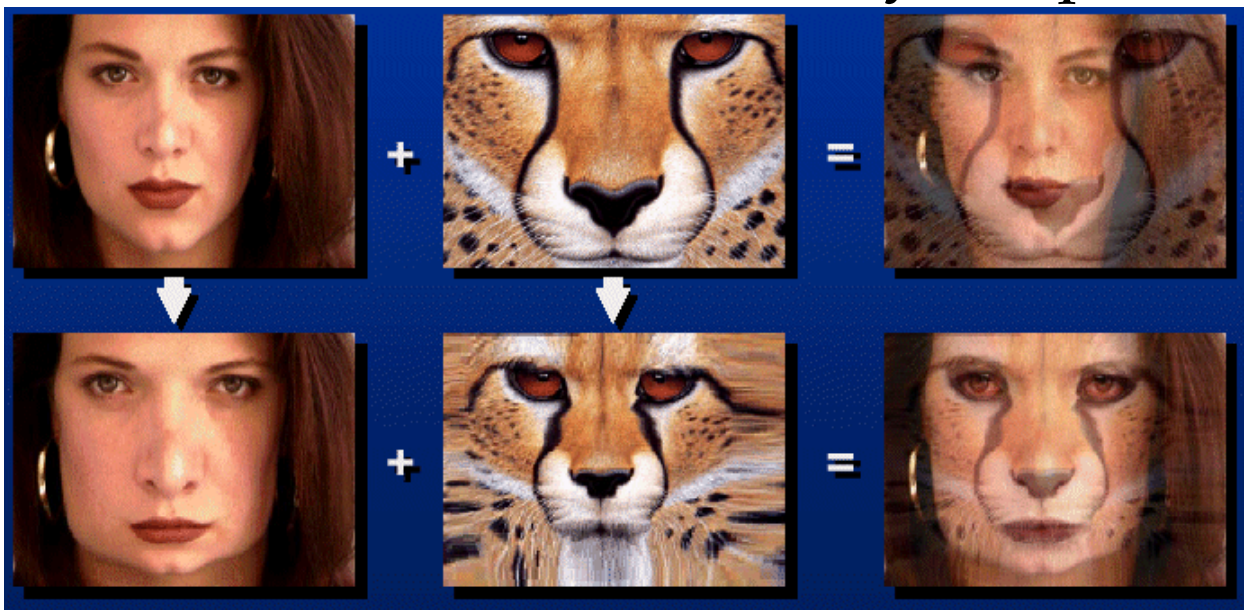


Image Morphing: Select Features

1) Generating correspondence points (by hand)

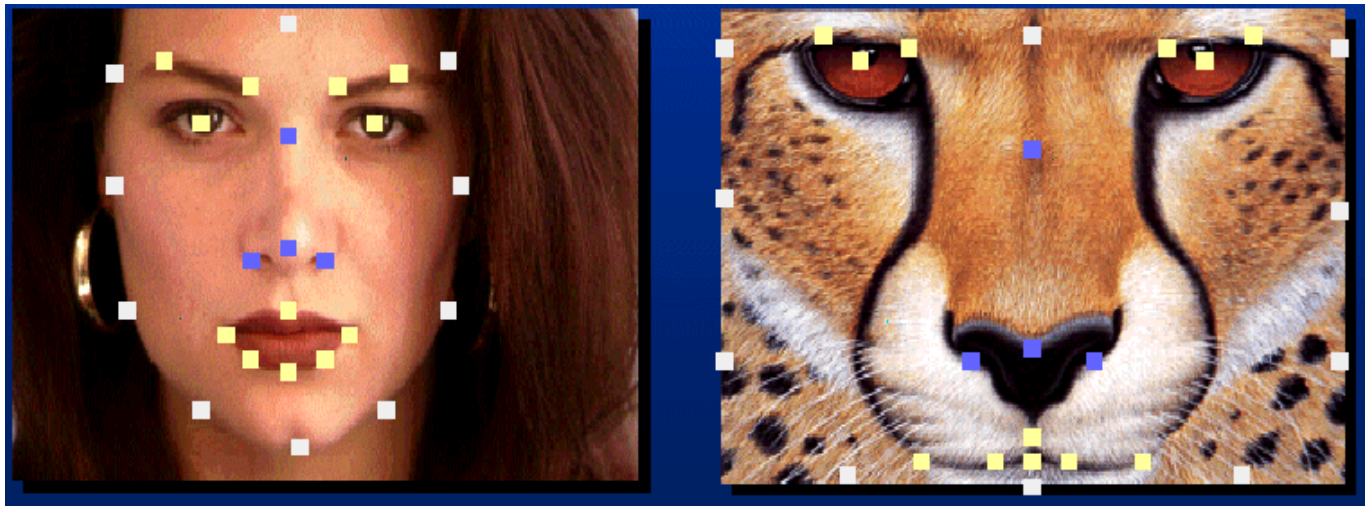


Image Morphing: Triangulation

2) Compute triangulation between the feature points
   a) Use Delaunay triangulation (use library)
   b) Make sure both images have the same triangulation!
      a) Use the intermediate shape to generate the triangulation (and use the fact you know correspondences of vertices).

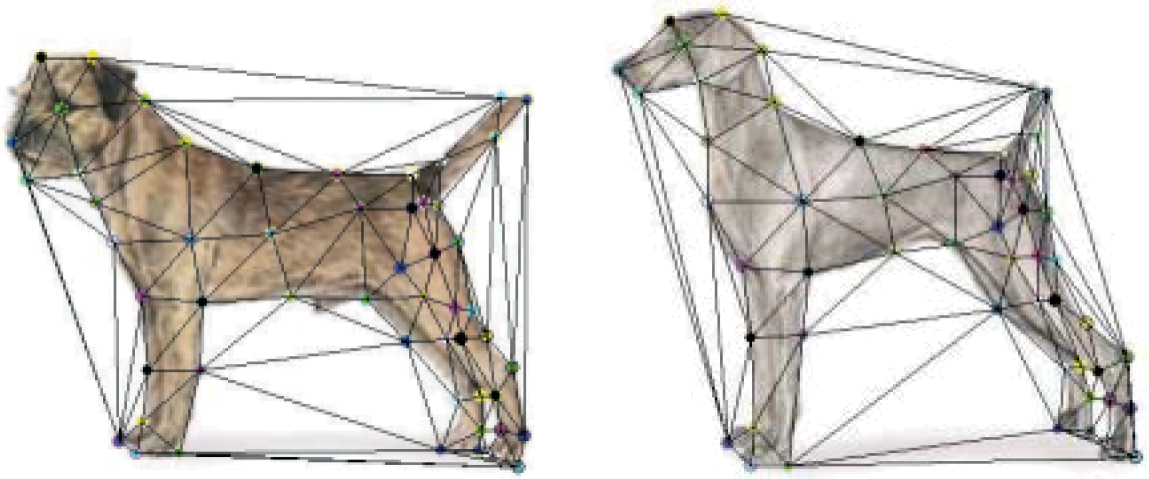
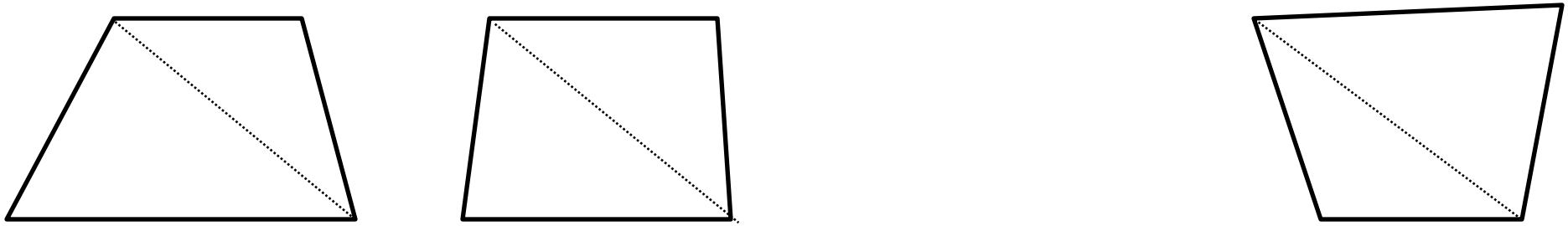


Image Morphing: Generate Intermediate Shape

2) Compute a weighted average shape

- Assume t = [0,1]
- Simple linear interpolation of each feature pair
- (1-t)*p1+t*p0 for corresponding features p0 and p1

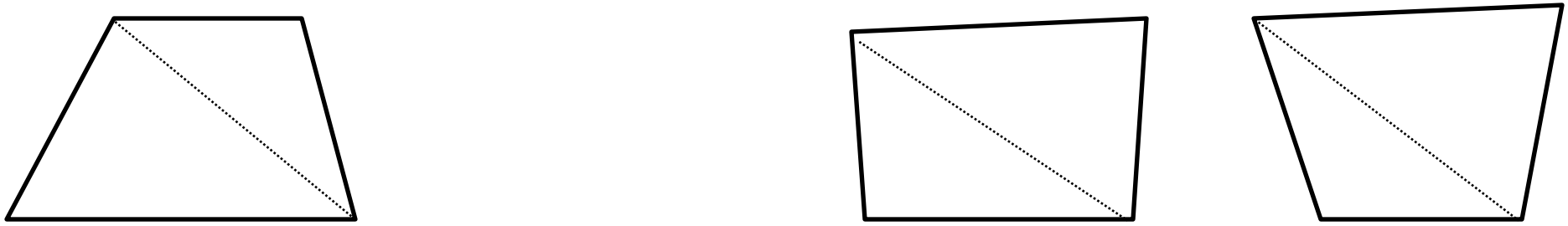


t=0.4

Image Morphing: Generate Intermediate Shape

2) Compute a weighted average shape

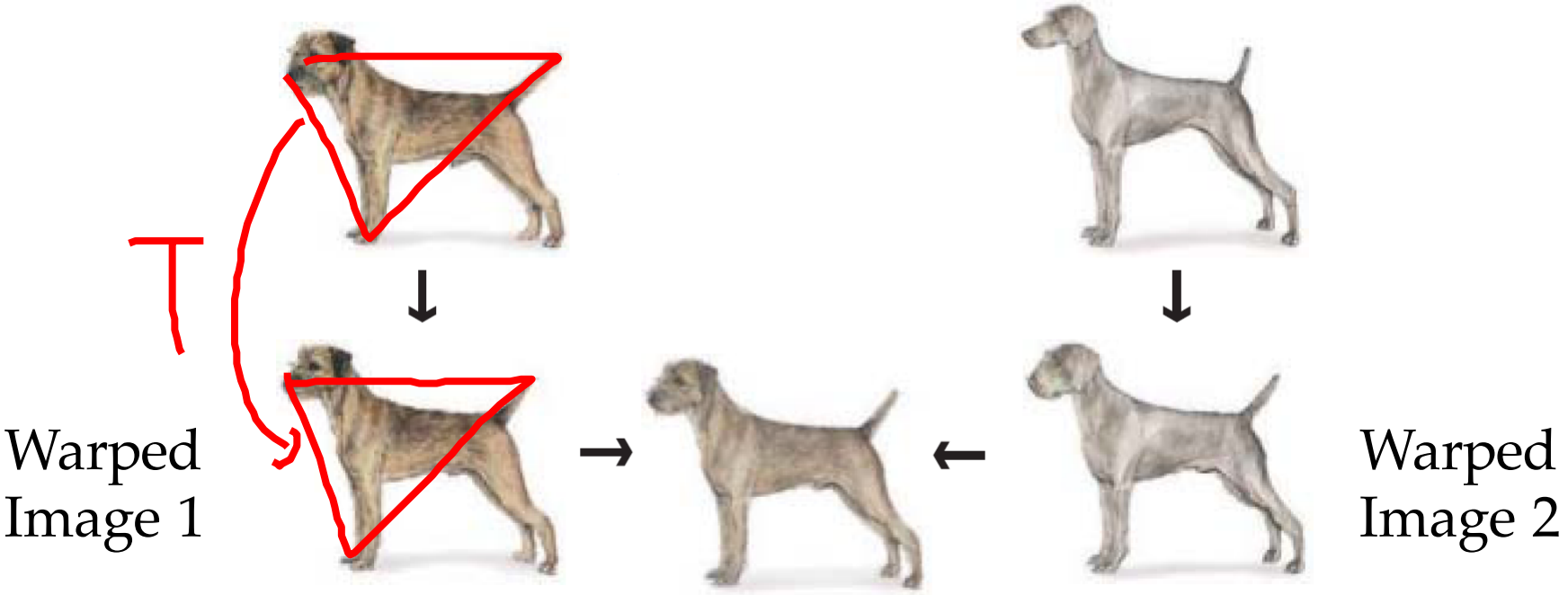
- Assume t = [0,1]
- Simple linear interpolation of each feature pair
- (1-t)*p1+t*p0 for corresponding features p0 and p1



t=0.7

In your implementation, compute the intermediate locations of the corresponding points (i.e., vertices) and then triangulate!!

Image Morphing: Generate Warped Image



3) Generate warped image 1 and 2:
source: original image 1, and 2
target: ***t***-intermediate shape

Image Morphing: Generate Warped Image with Triangulation

Barycentric coordinates are invariant to affine warping!!

- Determine which triangle it is in
- Compute its barycentric coordinates
- Find equivalent point in equivalent triangle in original image, and copy over its intensity value (use those barycentric coordinates)

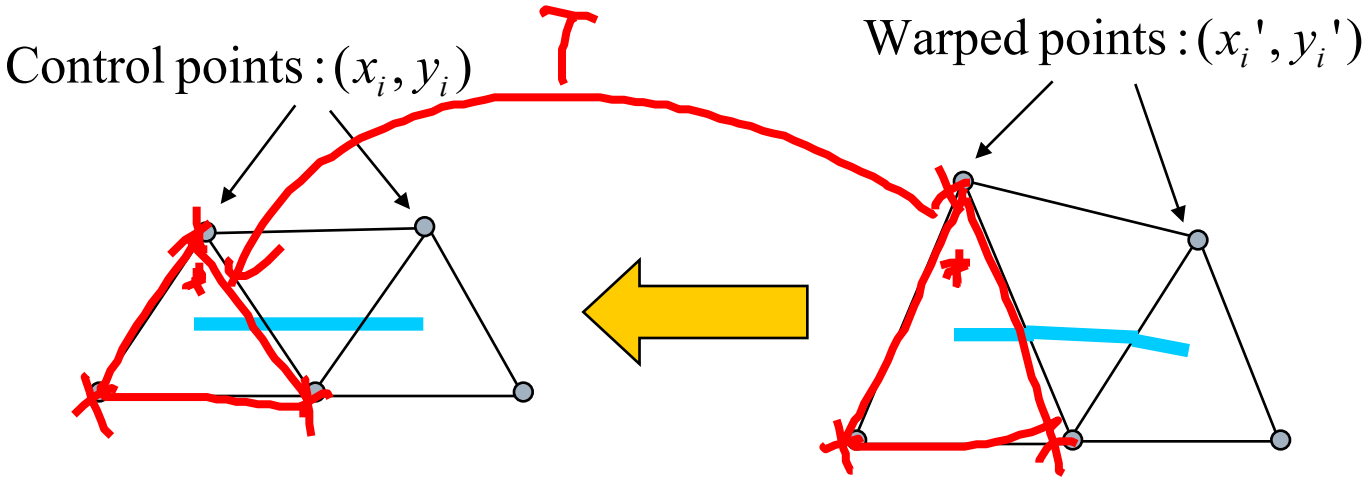
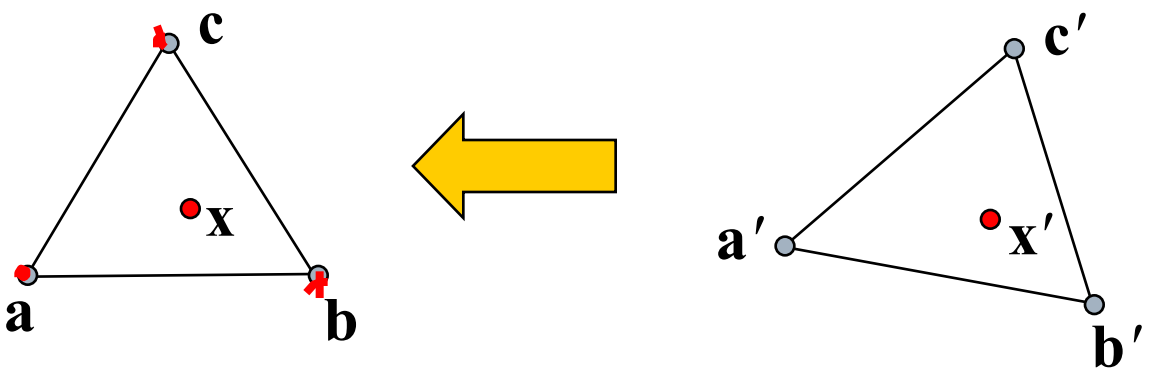


Image Morphing: Generate Warped Image with Triangulation

Barycentric coordinates are invariant to affine warping!!

- Determine which triangle it is in
- Compute its barycentric coordinates
- Find equivalent point in equivalent triangle in original image, and copy over its intensity value

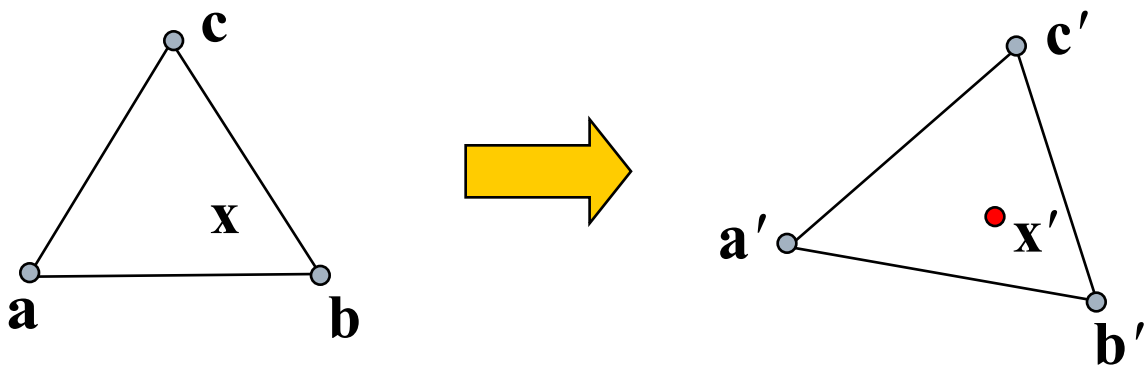


$$\mathbf{x} = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$$

$$\mathbf{x}' = \alpha\mathbf{a}' + \beta\mathbf{b}' + \gamma\mathbf{c}'$$

$$\alpha + \beta + \gamma = 1$$

Barycentric Coordinates



$$\mathbf{x} = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$$

$$\alpha + \beta + \gamma = 1$$

$$\mathbf{x}' = \alpha\mathbf{a}' + \beta\mathbf{b}' + \gamma\mathbf{c}'$$

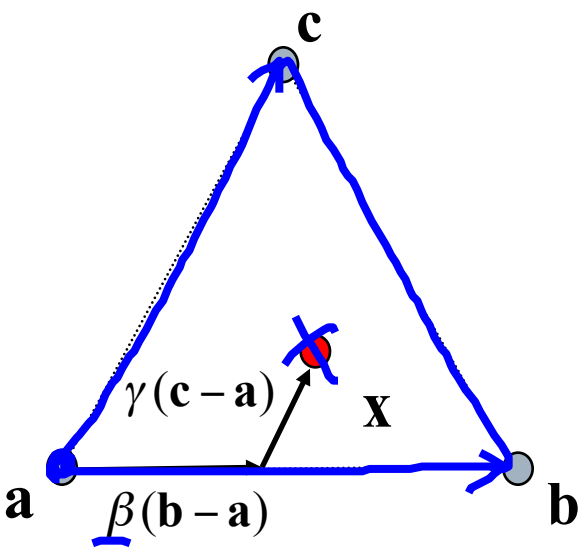
How do we know if a point is inside of a triangle?

$\mathbf{x}$ is inside the triangle if

$0 \le \alpha \le 1$ and $0 \le \beta \le 1$

In your implementation, use a scipy Delaunay method to determine the triangle encasing that vertex!

Barycentric Coordinates



$$\begin{aligned}\mathbf{x} &= \mathbf{a} + \beta(\mathbf{b}-\mathbf{a}) + \gamma(\mathbf{c}-\mathbf{a}) \\ &= (1-\beta-\gamma)\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c} \\ &= \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}\end{aligned}$$

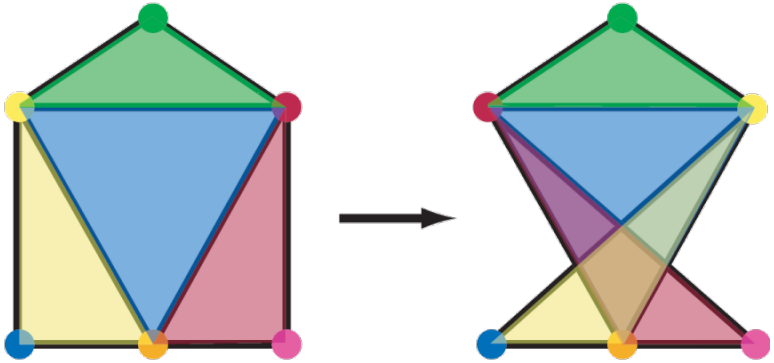
$$\mathbf{x} = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$$

$$\alpha + \beta + \gamma = 1$$

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix}$$

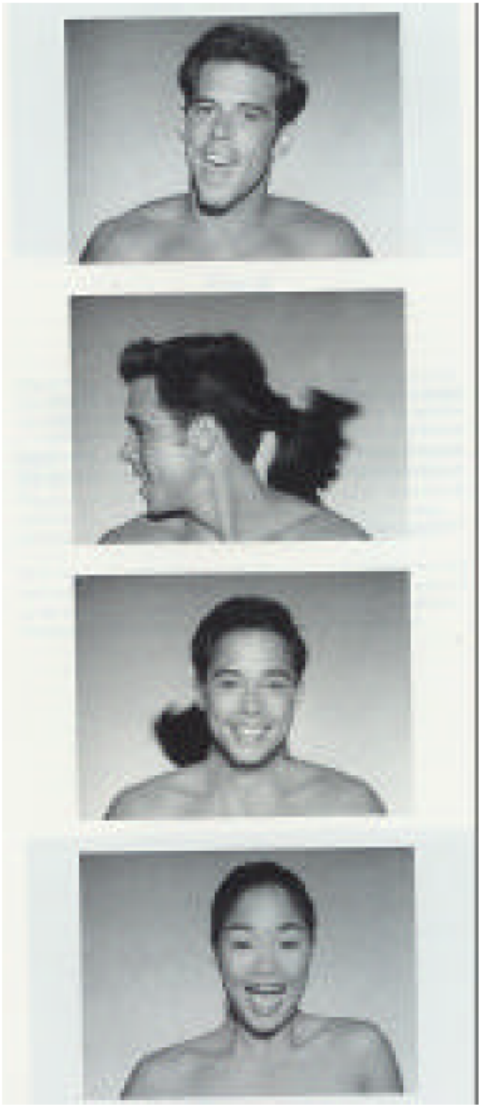
Three linear equations in 3 unknowns

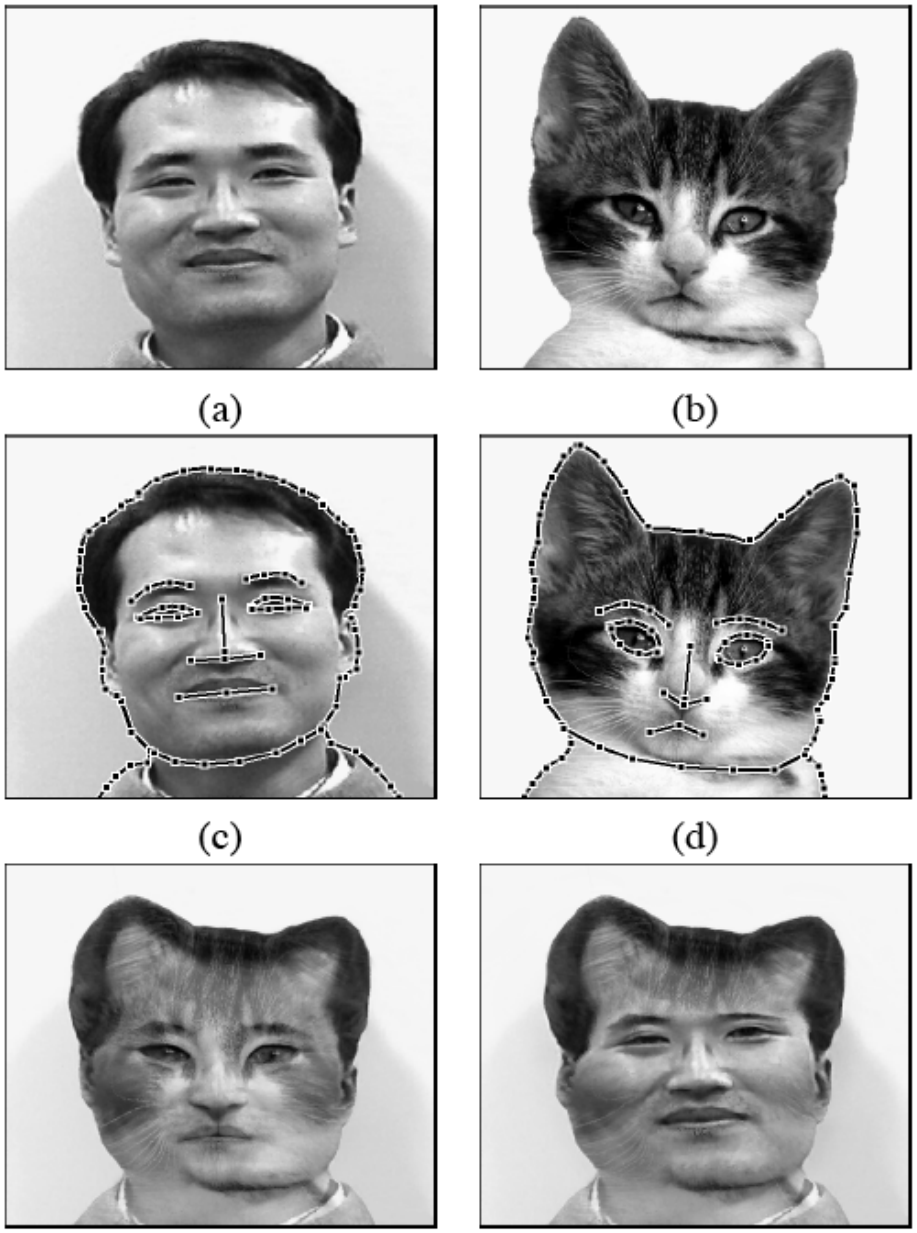
Other Issues

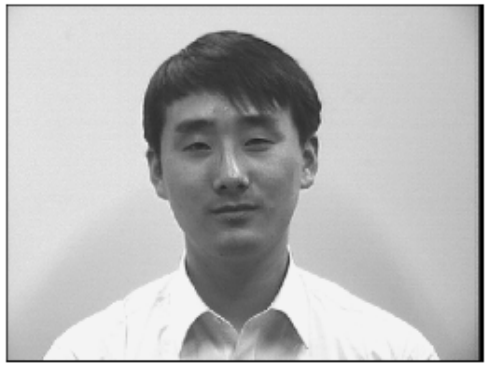


- Beware of folding
  - You are probably trying to do something 3D-ish
- Morphing can be generalized into 3D
  - If you have 3D data, that is!
- Extrapolation can sometimes produce interesting effects
  - Caricatures

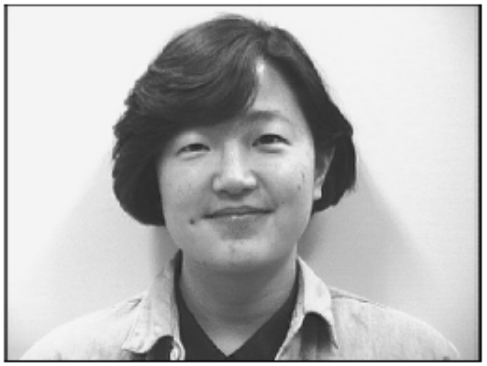
Dynamic Scene



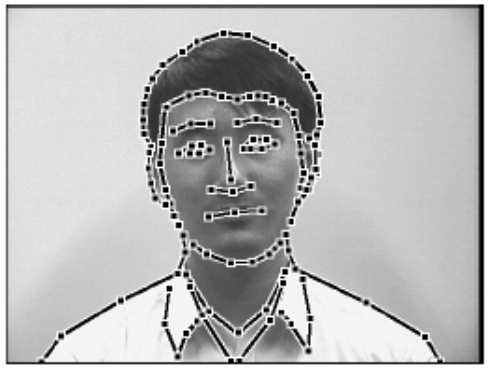

(a)
(b)
(c)
(d)



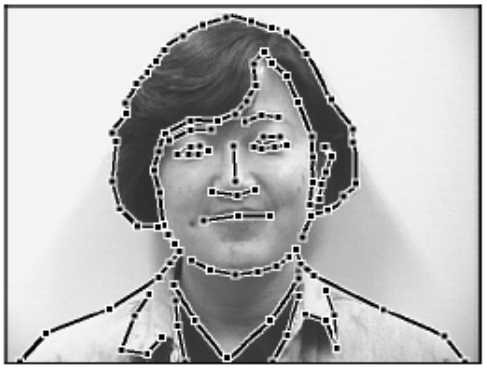
(a)



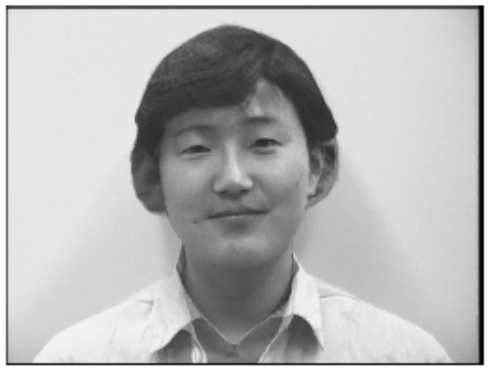
(b)

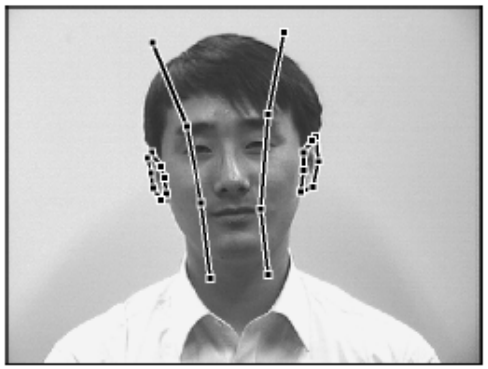


(c)



(d)





Project #3: Image Morphing

- Given two photos, produce a 60-frame morph animation

Recent Work in Face Morphing

- https://www.youtube.com/watch?v=ohmajJTcpNk