

# Image Morphing



Computational Photography  
Derek Hoiem, University of Illinois

# Project 3: Gradient Domain Editing

Now posted!

General concept: Solve for pixels of new image that satisfy constraints on the gradient and the intensity

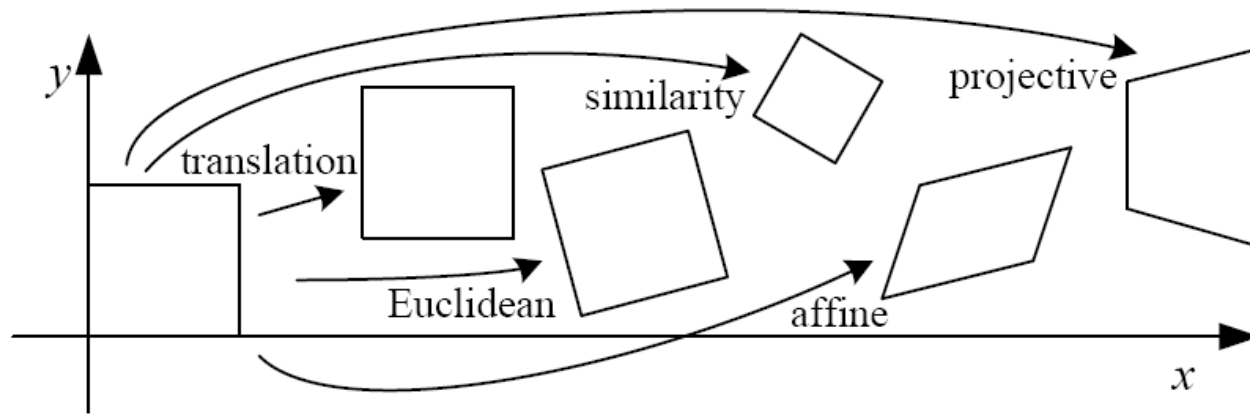
- Constraints can be from one image (for filtering) or more (for blending)






# Image Morphing



Computational Photography  
Derek Hoiem, University of Illinois

# 2D image transformations



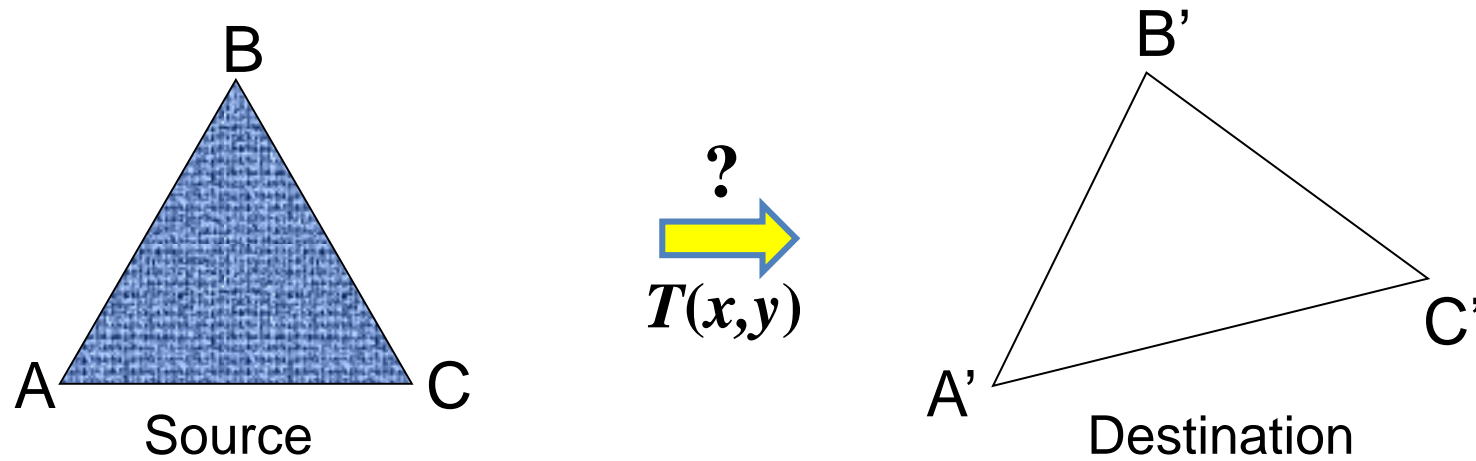
Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} \mathbf{I} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	2	orientation + ...	
rigid (Euclidean)	$\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	3	lengths + ...	
similarity	$\begin{bmatrix} s\mathbf{R} & \mathbf{t} \end{bmatrix}_{2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

These transformations are a nested set of groups

- Closed under composition and inverse is a member

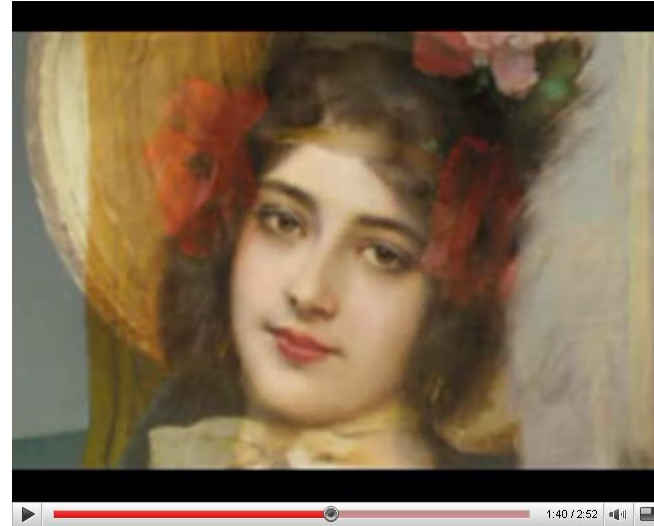
# Take-home Question

Suppose we have two triangles:  $ABC$  and  $A'B'C'$ . What transformation will map  $A$  to  $A'$ ,  $B$  to  $B'$ , and  $C$  to  $C'$ ? How can we get the parameters?



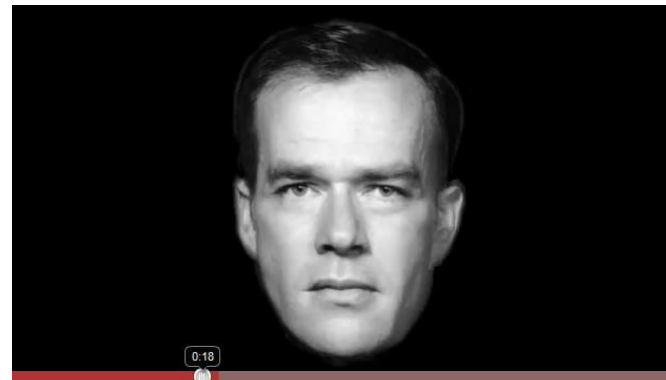
# Today: Morphing

## Women in art



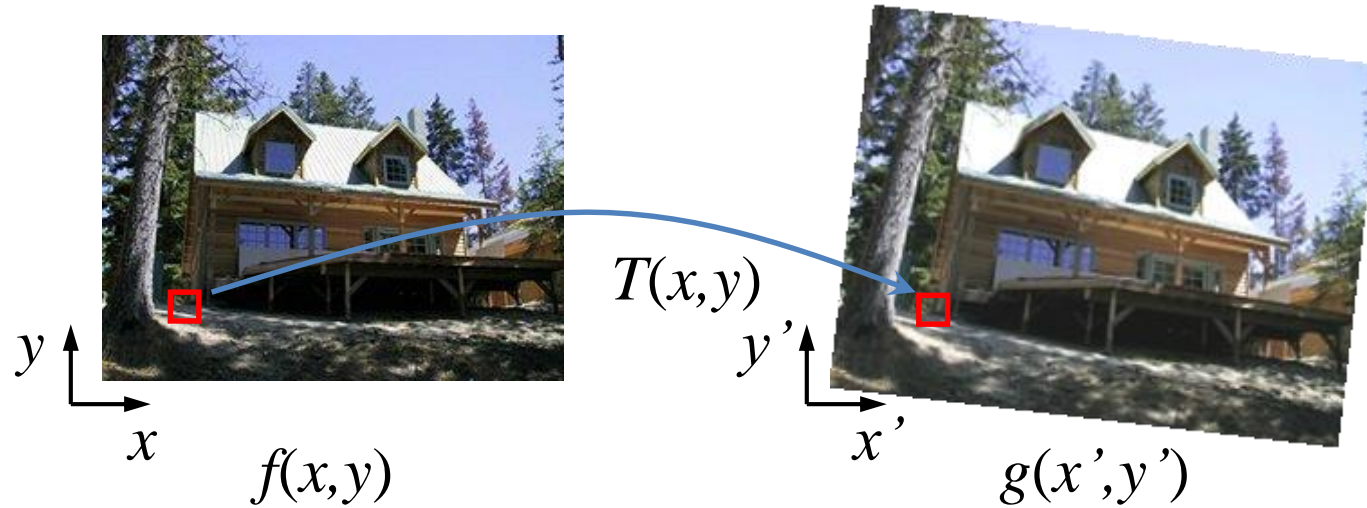
<http://youtube.com/watch?v=nUDIoN-Hxs>

## Aging



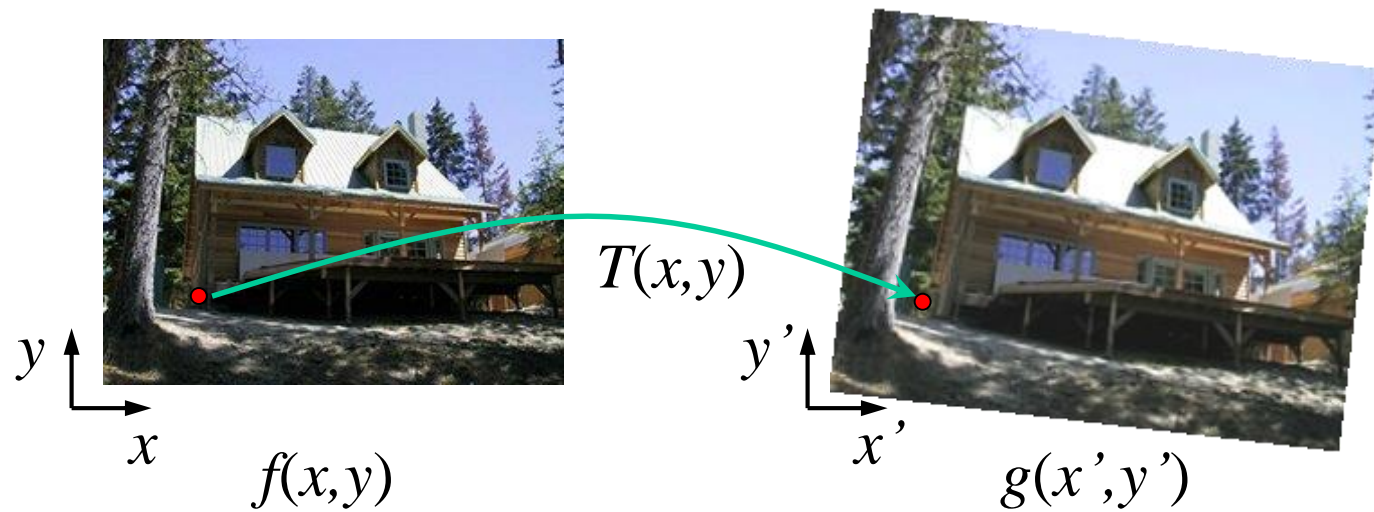
<http://www.youtube.com/watch?v=LOGKp-uvj00>

# Texturing in transformed coordinates



Given a coordinate transform  $(x',y') = T(x,y)$  and a source image  $f(x,y)$ , how do we compute a transformed image  $g(x',y') = f(T(x,y))$ ?

# Forward mapping

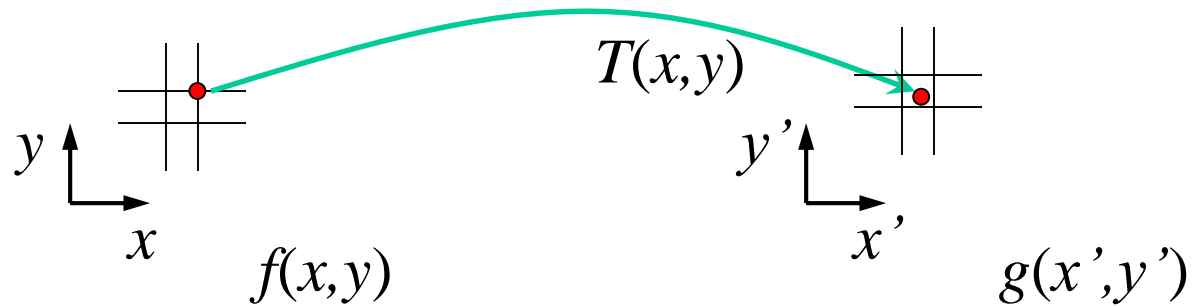


Send each pixel  $f(x,y)$  to its corresponding location  
 $(x',y') = T(x,y)$  in the second image



# Forward mapping

What is the problem with this approach?



Send each pixel  $f(x,y)$  to its corresponding location

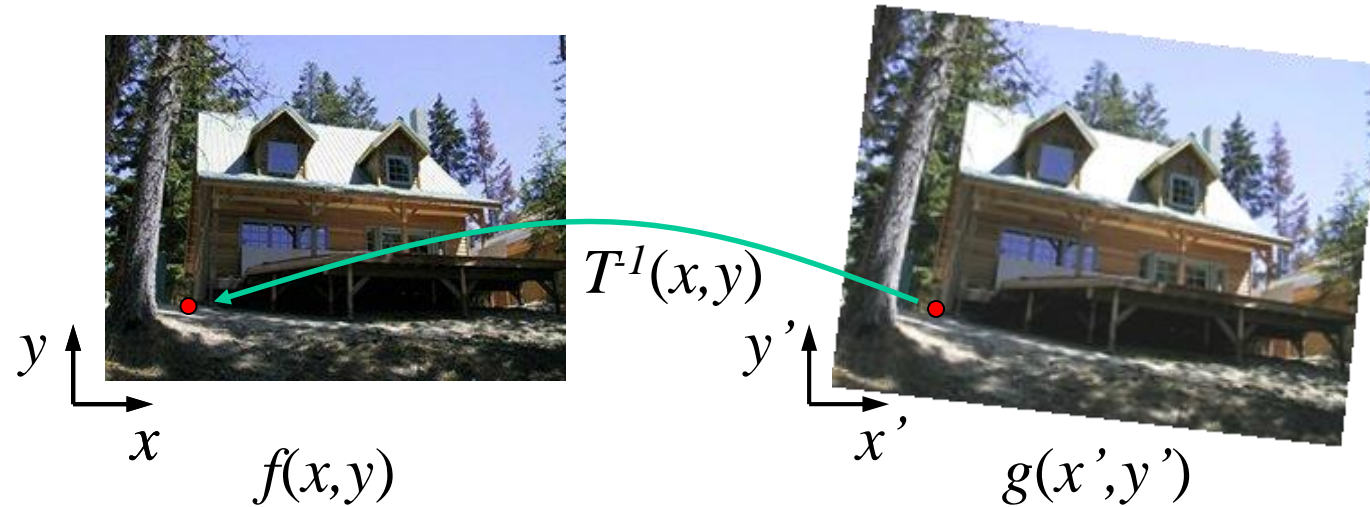
$(x',y') = T(x,y)$  in the second image

Q: what if pixel lands “between” two pixels?

A: distribute color among neighboring pixels  $(x',y')$

– Known as “splatting”

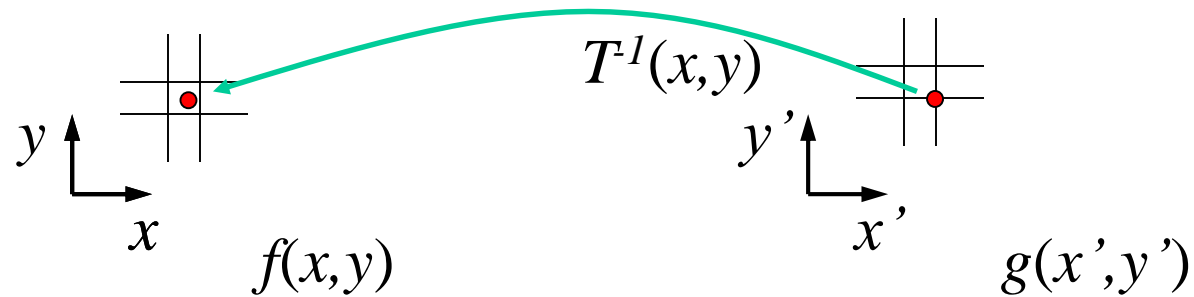
# Inverse mapping



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 mapping



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

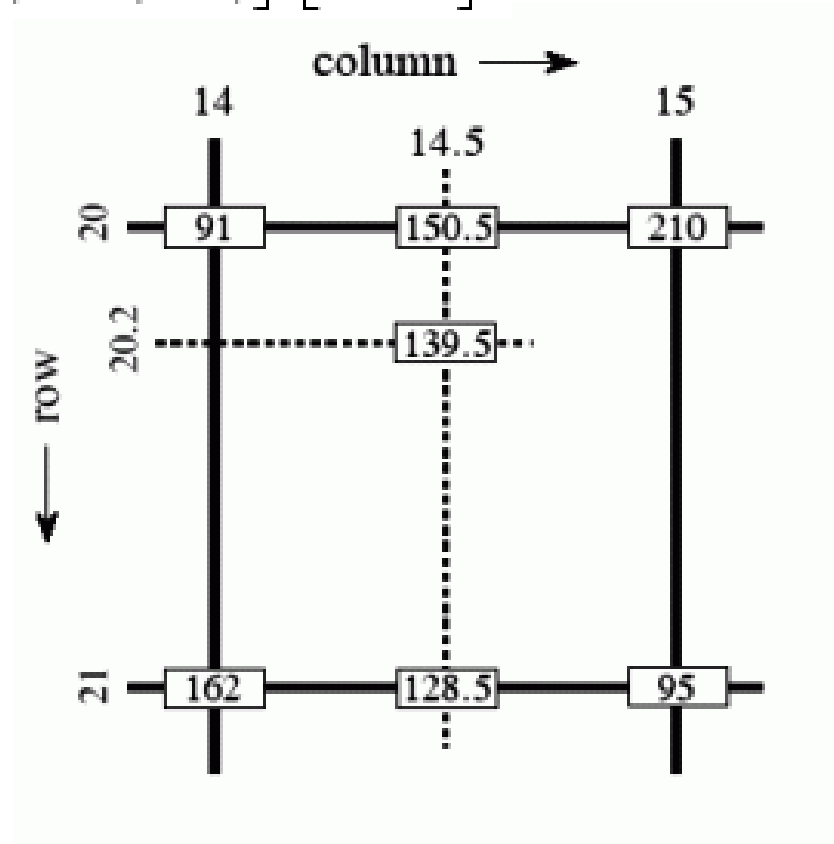
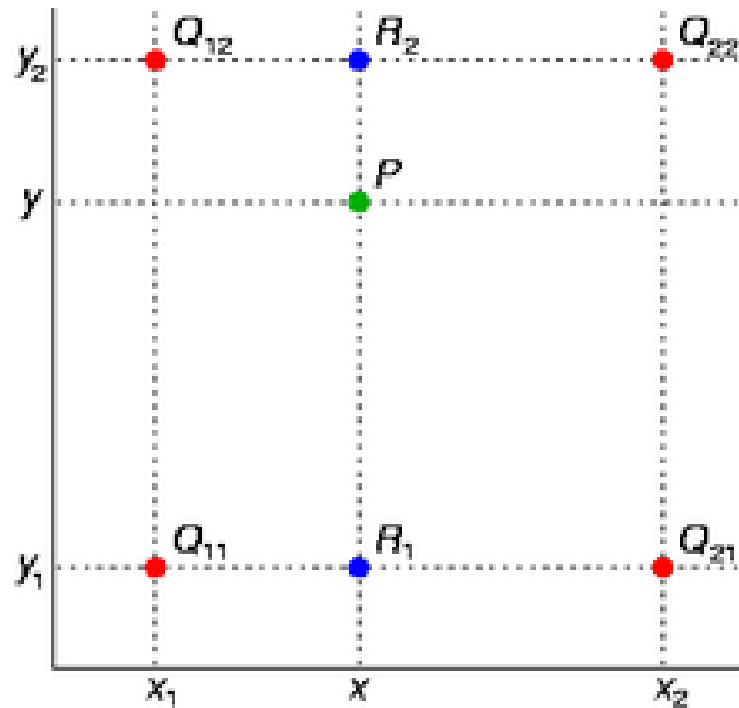
Q: what if pixel comes from “between” two pixels?

A: *Interpolate* color value from neighbors

- nearest neighbor, bilinear, Gaussian, bicubic
- E.g. `interpolate.interp2` or `ndimage.map_coordinates` in Python `scipy`

# Bilinear Interpolation

$$f(x, y) \approx \begin{bmatrix} 1 - x & x \end{bmatrix} \begin{bmatrix} f(0, 0) & f(0, 1) \\ f(1, 0) & f(1, 1) \end{bmatrix} \begin{bmatrix} 1 - y \\ y \end{bmatrix}.$$



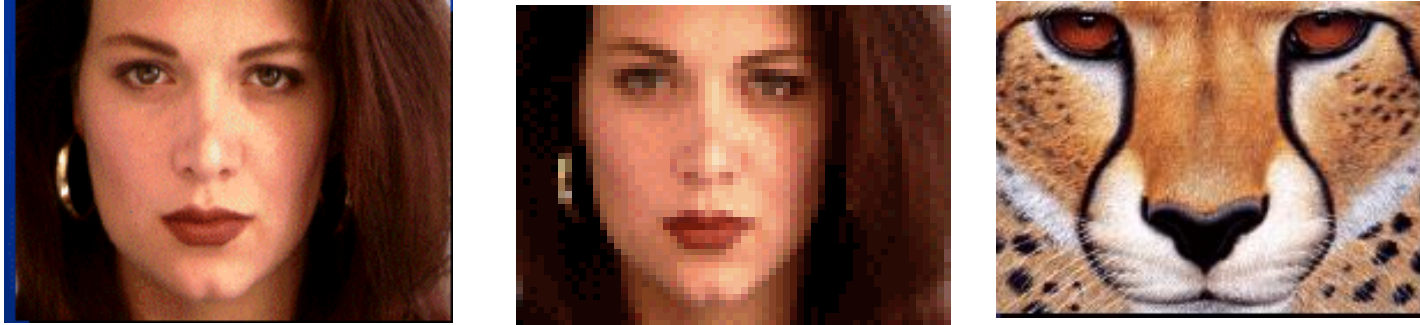
# Forward vs. inverse mapping

Q: which is better?

A: Usually inverse—eliminates holes

- however, it requires an invertible warp function

# Morphing = Object Averaging

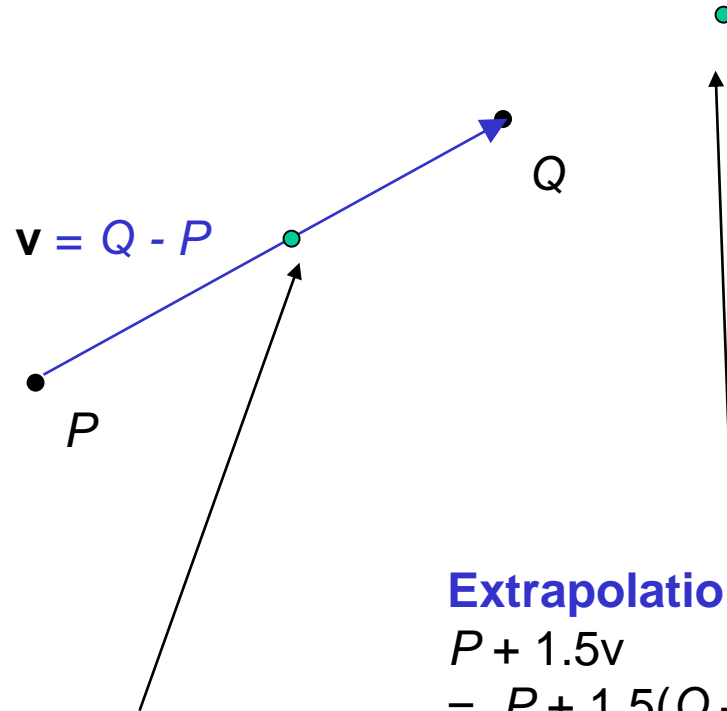


The aim is to find “an average” between two objects

- 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  $t$

# Averaging Points

What's the average of P and Q?



## Linear Interpolation

New point:  $P + t*(Q-P)$

Or equivalently:  $(1-t)P + tQ$

$0 < t < 1$

$$\begin{aligned} P + 0.5v \\ &= P + 0.5(Q - P) \\ &= 0.5P + 0.5Q \end{aligned}$$

**Extrapolation:**  $t < 0$  or  $t > 1$

$$\begin{aligned} P + 1.5v \\ &= P + 1.5(Q - P) \\ &= -0.5P + 1.5Q \quad (t=1.5) \end{aligned}$$

P and Q can be anything:

- points on a plane (2D) or in space (3D)
- Colors in RGB (3D)
- Whole images (m-by-n D)... etc.

# Idea #1: Cross-Dissolve



Interpolate whole images:

$$\text{Image}_{\text{halfway}} = (1-t) \cdot \text{Image}_1 + t \cdot \text{Image}_2$$

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

But what if the images are not aligned?



# Idea #2: Align, then cross-dissolve



Align first, then cross-dissolve

- Alignment using global warp – picture still valid

# Dog Averaging



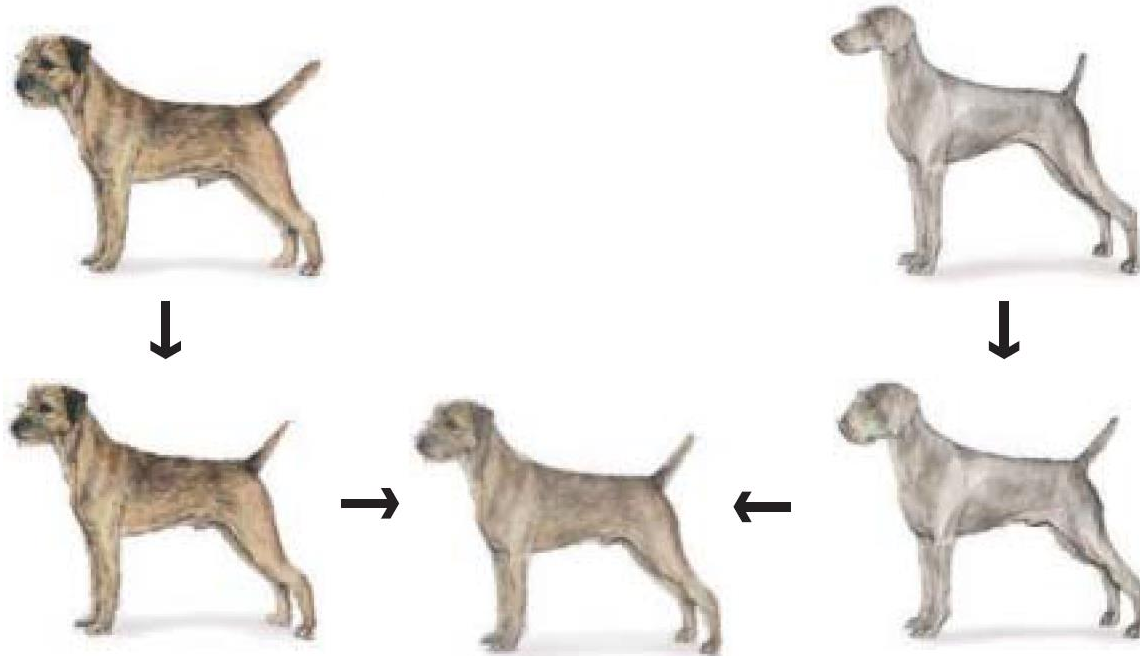
## What to do?

- 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

# Idea #3: Local warp, then cross-dissolve



## Morphing procedure

*For every frame  $t$ ,*

1. Find the average shape (the “mean dog” 😊)
  - local warping
2. Find the average color
  - Cross-dissolve the warped images

# Local (non-parametric) Image Warping



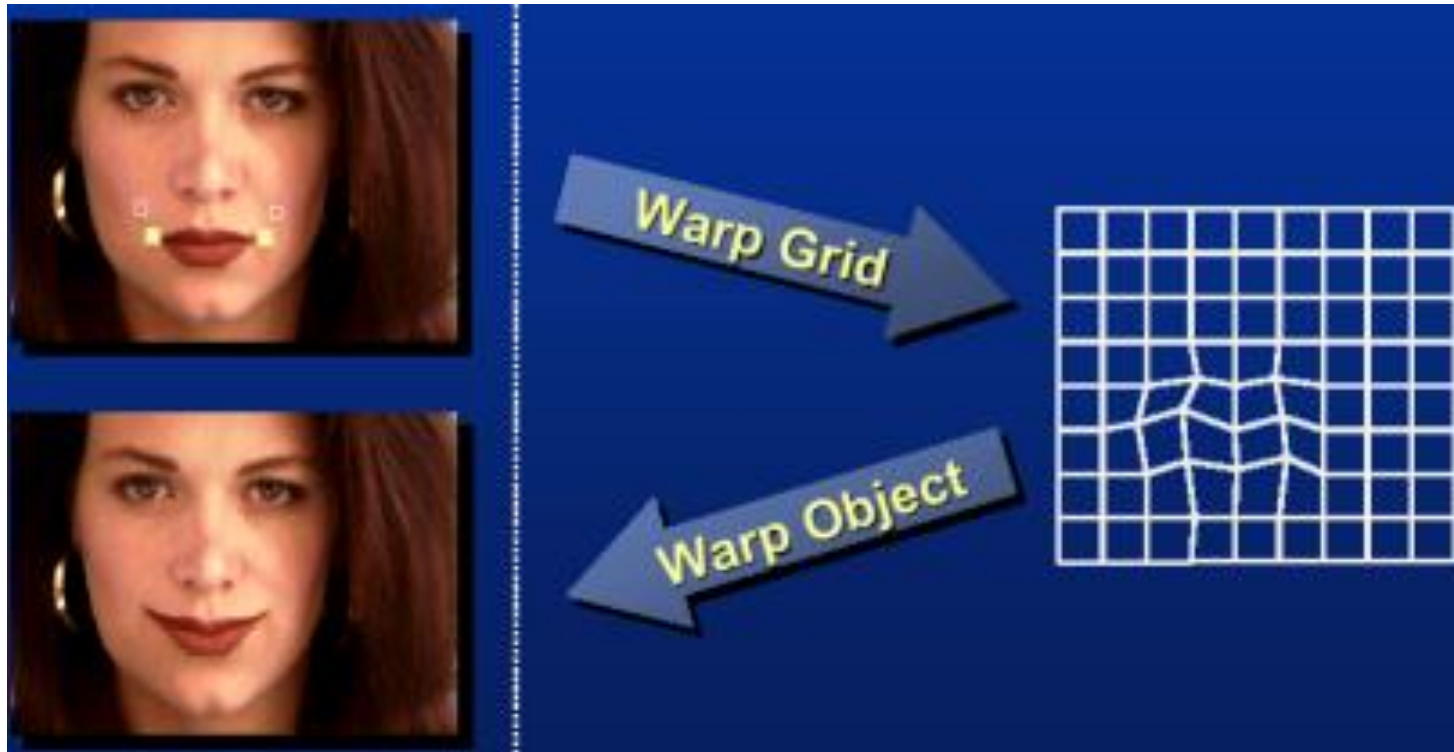
Need to specify a more detailed warp function

- 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

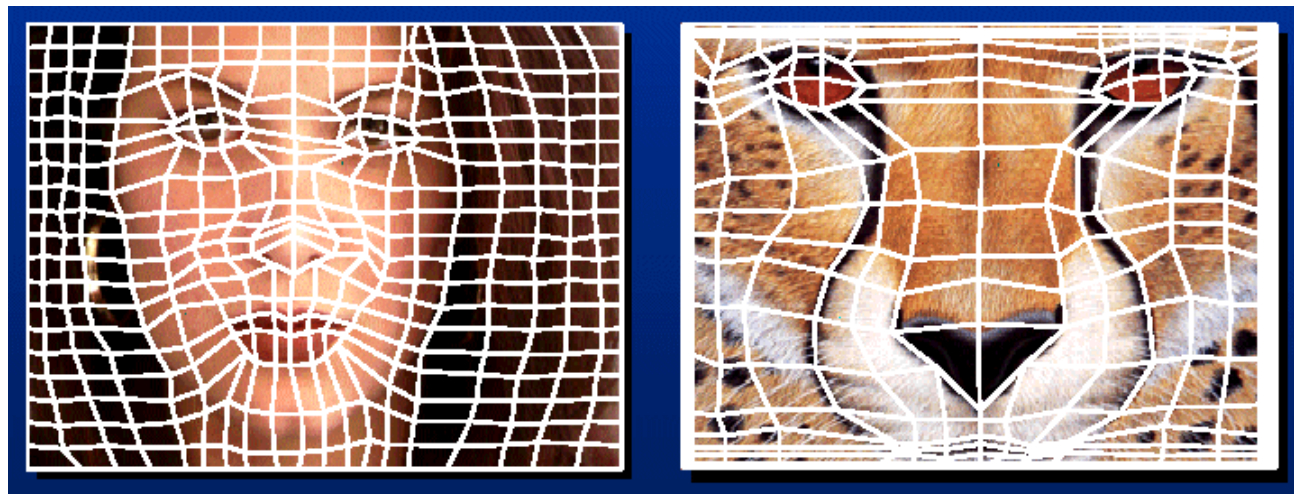


# Warp specification - dense

How can we specify the warp?

Specify corresponding *spline control points*

- *interpolate* to a complete warping function



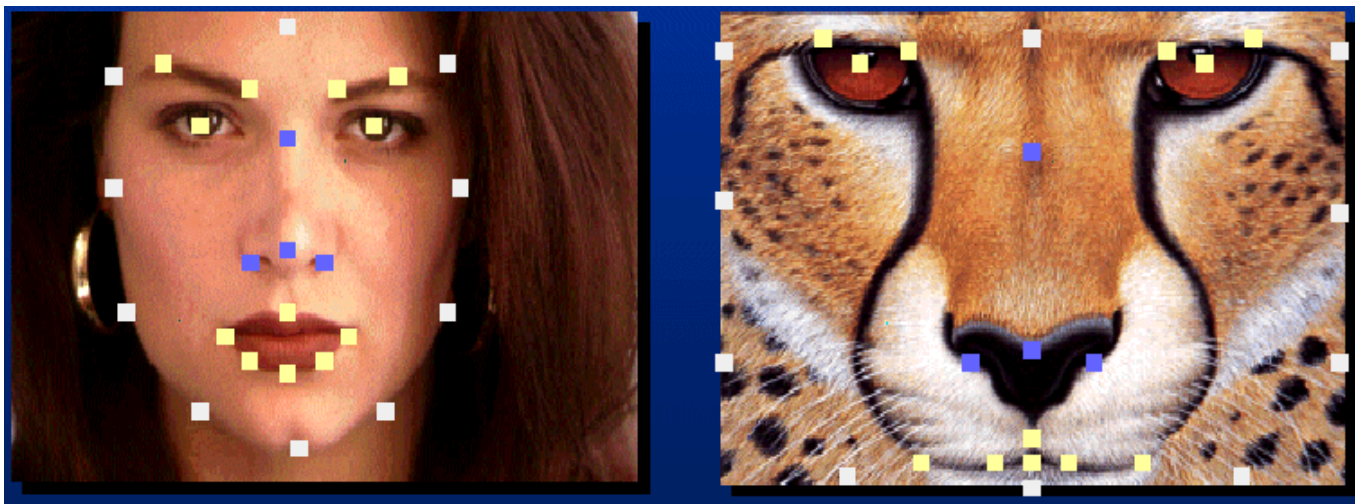
But we want to specify only a few points, not a grid

# Warp specification - sparse

How can we specify the warp?

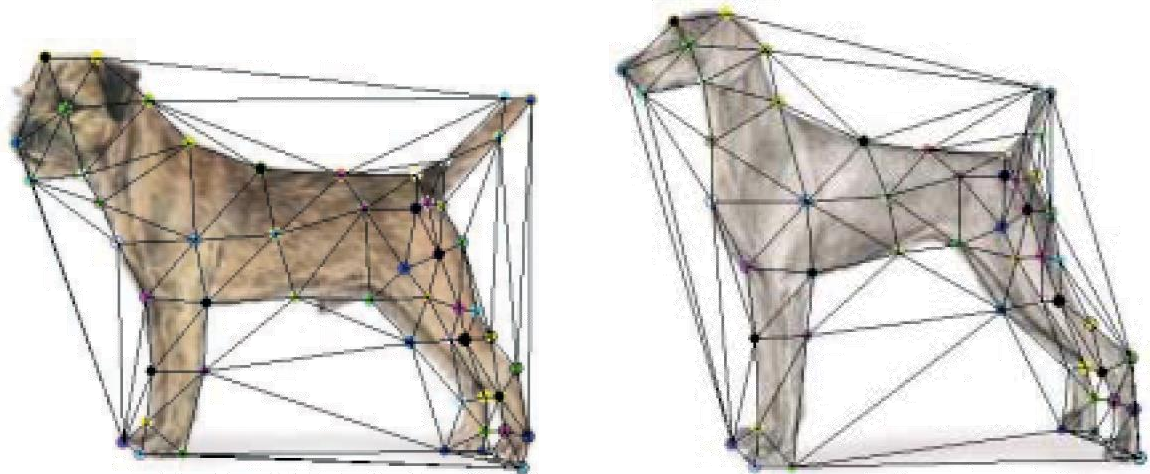
Specify corresponding *points*

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



How do we go from feature points to pixels?

# Triangular Mesh



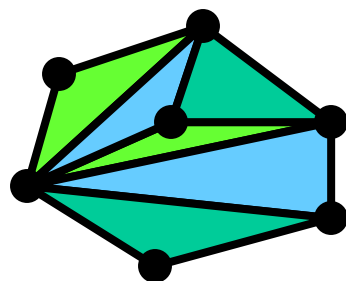
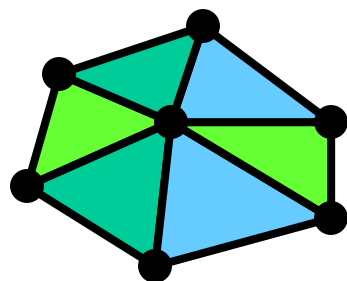
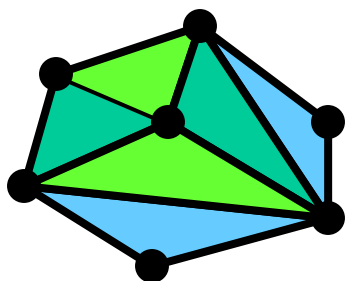
1. Input correspondences at key feature points
2. Define a triangular mesh over the points
  - Same mesh (triangulation) in both images!
  - Now we have triangle-to-triangle correspondences
3. Warp each triangle separately from source to destination
  - Affine warp with three corresponding points (just like take-home question)



# 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.

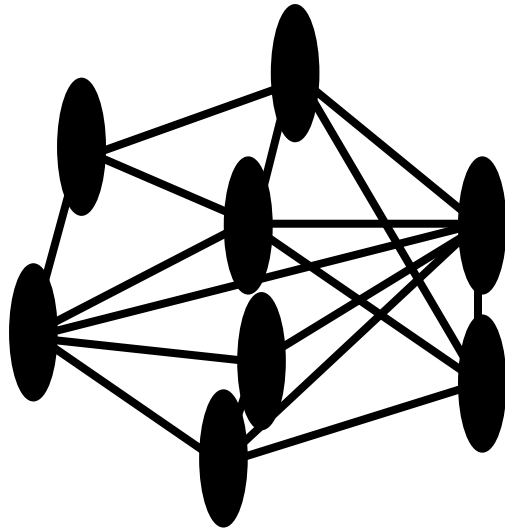
There are an exponential number of triangulations of a point set.



# An $O(n^3)$ Triangulation Algorithm

Repeat until impossible:

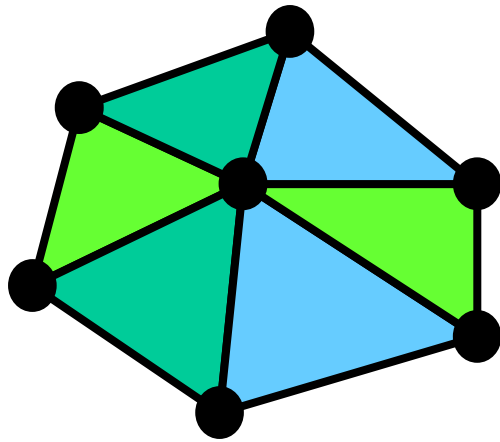
- Select two sites.
- If the edge connecting them does not intersect previous edges, keep it.



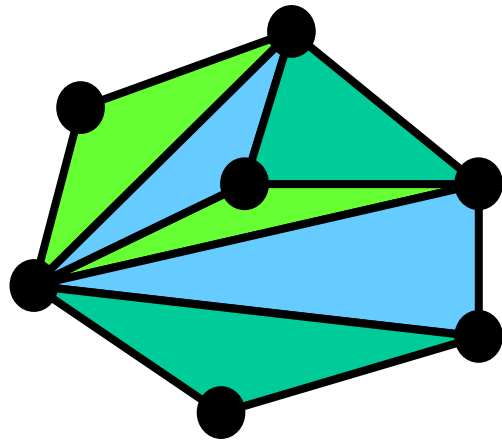
# “Quality” Triangulations

Let  $\alpha(T_i) = (\alpha_{i_1}, \alpha_{i_2}, \dots, \alpha_{i_3})$  be the vector of angles in the triangulation  $T$  in increasing order:

- A triangulation  $T_1$  is “better” than  $T_2$  if the smallest angle of  $T_1$  is larger than the smallest angle of  $T_2$
- Delaunay triangulation is the “best” (maximizes the smallest angles)



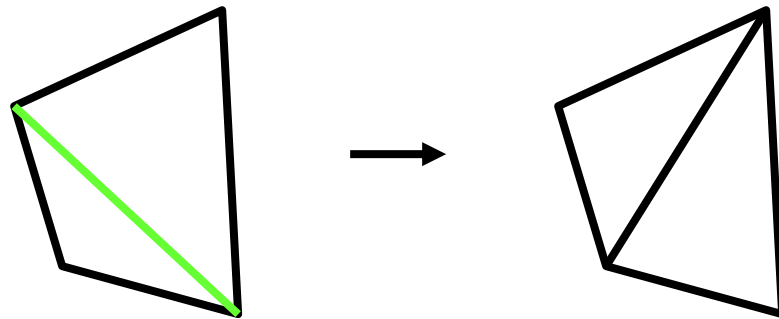
good



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.

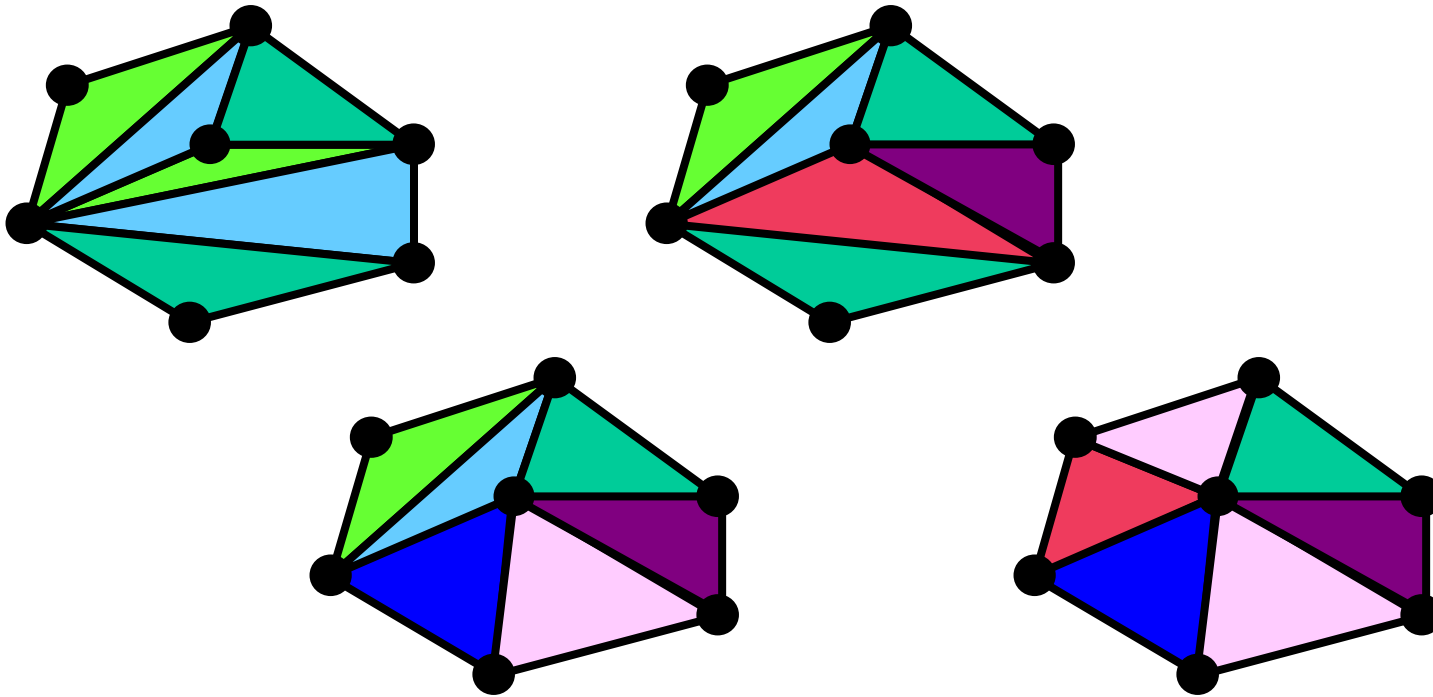


If an edge flip improves the triangulation, the first edge is called "*illegal*".

# Naïve Delaunay Algorithm

Start with an arbitrary triangulation. Flip any illegal edge until no more exist.

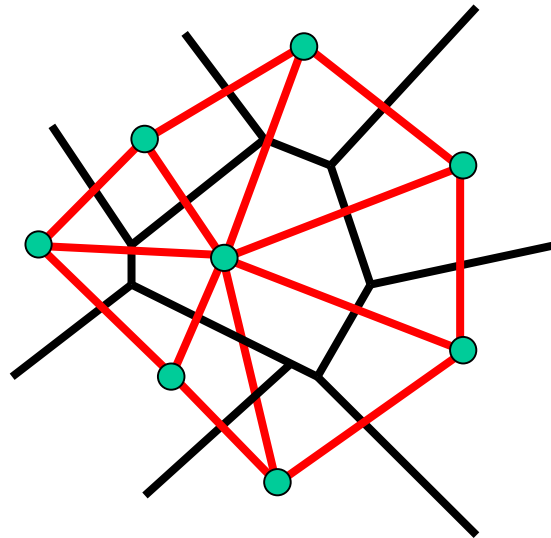
Could take a long time to terminate.



# Delaunay Triangulation by Duality

Draw the dual to the Voronoi diagram by connecting each two neighboring sites in the Voronoi diagram.

- The DT may be constructed in  $O(n \log n)$  time



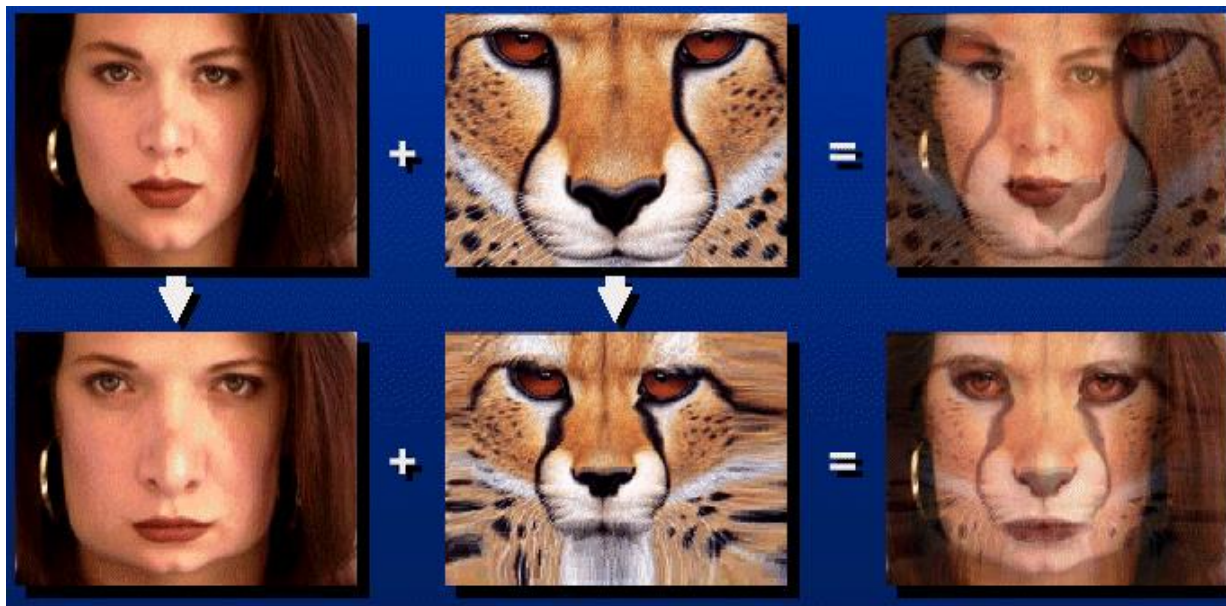
Demos: <http://www.cs.cornell.edu/home/chew/Delaunay.html>

<http://alexbeutel.com/webgl/voronoi.html>

# Image Morphing

How do we create a morphing sequence?

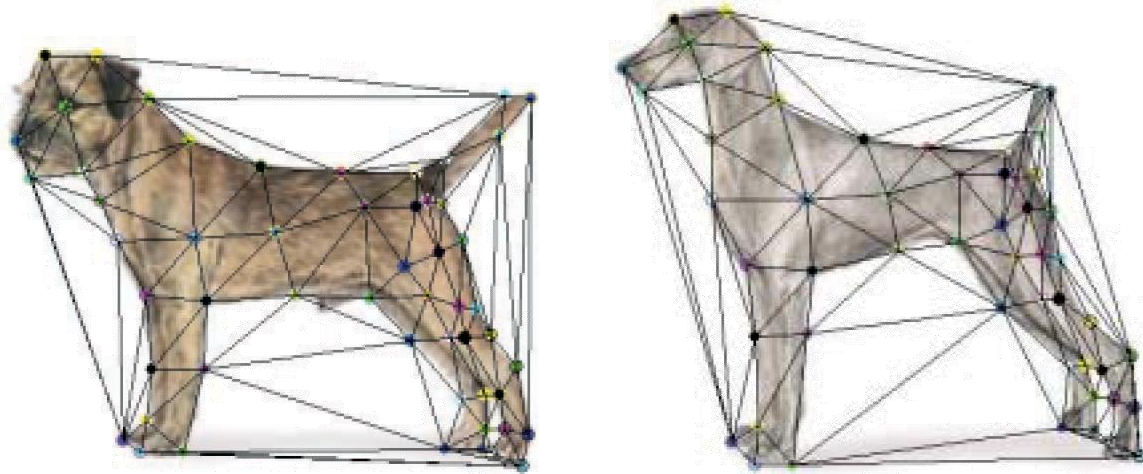
1. Create an intermediate shape (by interpolation)
2. Warp both images towards it
3. Cross-dissolve the colors in the newly warped images



# Warp interpolation

How do we create an intermediate shape at time  $t$ ?

- Assume  $t = [0,1]$
- Simple linear interpolation of each feature pair
  - $(1-t)*p_1+t*p_0$  for corresponding features  $p_0$  and  $p_1$





# 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)  $\alpha = 1.0$

# Dynamic Scene



Black or White (MJ):

<http://www.youtube.com/watch?v=R4kLKv5gtxc>

Willow morph: <http://www.youtube.com/watch?v=uLUyuWo3pG0>

# Summary of morphing

1. Define corresponding points
2. Define triangulation on points
  - Use same triangulation for both images
3. For each  $t = 0:\text{step}:1$ 
  - a. Compute the average shape (weighted average of points)
  - b. For each triangle in the average shape
    - Get the affine projection to the corresponding triangles in each image
    - For each pixel in the triangle, find the corresponding points in each image and set value to weighted average (optionally use interpolation)
  - c. Save the image as the next frame of the sequence

# Next classes

Pinhole camera: start of perspective geometry

Single-view metrology: measure 3D distances  
from an image