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 haven’t a clue!
  - But we can often fake something reasonable
    - Usually required user/artist input
Image Morphing, Thin Plate Spline

http://grail.cs.washington.edu/projects/photomontage/

Image Morphing, Thin-Plate Spline Model

slides courtesy A. Efros, J. Shi
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:
  \[ \text{Image}_{\text{halfway}} = (1-t) \times \text{Image}_1 + t \times \text{image}_2 \]
- 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
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* $t$,
  1. Find the average shape (the “mean dog” 😊)
    - local warping
  2. Find the average color
    - Cross-dissolve the warped images
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
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?
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!
   - Just like texture mapping
A triangulation of a set of points in the plane is a partition of the convex hull into 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.
Let $\alpha(T) = (\alpha_1, \alpha_2, \ldots, \alpha_{3t})$ be the vector of angles in the triangulation $T$ in increasing order.

A triangulation $T_1$ will be “better” than $T_2$ if $\alpha(T_1) > \alpha(T_2)$ lexicographically.

The Delaunay triangulation is the “best”
- Maximizes smallest angles

“Quality” Triangulations
Boris Nikolaevich Delaunay  
(March 15, 1890 – July 17, 1980)

http://higeom.math.msu.su/history/delone_r.html
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*. 
**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 Matlab’s `delaunay` function uses.
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
How do we create an intermediate warp at time $t$?

- Assume $t = [0,1]$
- Simple linear interpolation of each feature pair
- $(1-t)p_1 + tp_0$ for corresponding features $p_0$ and $p_1$
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!
   - Just like texture mapping
Problem:
- Given corresponding points in two images, how do we warp one into the other?

Two common solutions
1. Piece-wise linear using triangle mesh
2. Thin-plate spline interpolation
Interpolation using Triangles

Control points: \((x_i, y_i)\)

Warped points: \((x'_i, y'_i)\)

Region of interest enclosed by triangles.
Moving nodes changes each triangle
Just need to map regions between two triangles
Barycentric Co-ordinates

\[ x = \alpha a + \beta b + \gamma c \]

\[ \alpha + \beta + \gamma = 1 \]

How do we know if a point is inside of a triangle?
Three linear equations in 3 unknowns – to compute barycentric coord.
Interpolation using Triangles

- To find out where each pixel in new image comes from in old image
  - Determine which triangle it is in
  - Compute its barycentric co-ordinates
  - Find equivalent point in equivalent triangle in original image
- Only well defined in region of `convex hull’ of control points
Define a smooth mapping function \((x', y') = f(x, y)\) such that

\[ f(x_i, y_i) = (x_i', y_i') \quad \text{for all } i = 1..n \]

- It maps each point \((x, y)\) onto \((x', y')\) and does something smooth in between.
- Defined everywhere, even outside convex hull of control points
Thin-Plate Spline Interpolation

- Function has form

\[ f(x, y) = (f_x(x, y), f_y(x, y)) \]

\[ f_x(x, y) = a_x + b_x x + \sum_{i=1}^{i=n} w_{xi} r_i^2 \log r_i \]

\[ f_y(x, y) = a_y + b_y y + \sum_{i=1}^{i=n} w_{yi} r_i^2 \log r_i \]

where \( r_i^2 = (x - x_i)^2 + (y - y_i)^2 \)

The parameters \((a_x, b_x, w_{xi}, a_y, b_y, w_{yi})\)
are found by solving the linear equations given by

\[ f(x_i, y_i) = (x_i', y_i') \]
Sparse and irregular positioned feature points, and smooth interpolation
Let’s consider two sets of points for which we assume the correspondences to be known (a). The TPS warping allows a perfect alignment of the points and the bending of the grid shows the deformation needed to bring the two sets on top of each other (b). Note that in the case of TPS applied to coordinate transformation we actually use two splines, one for the displacement in the x direction and one for the displacement in the y direction. The displacement in each direction is considered as a height map for the points and a spline is fit as in the case of scattered points in 3D space. And finally the two resulting transformations are combined into a single mapping.
First the equation:

\[
f(x, y) = a_1 + a_x x + a_y y + \sum_{i=1}^{p} w_i U \left( \| (x_i, y_i) - (x, y) \| \right)
\]
Say the blue circles are the source image features, and red crosses are the target image features, how do we “back-warp” all the image features in the target image back to the source image? We could compute a TPS model that maps “red” to “blue”, and apply it to the rest of the target image pixels.

\[
f(x, y) = a_1 + a_x x + a_y y + \sum_{i=1}^{p} w_i U(||(x_i, y_i) - (x, y)||)
\]
How do we estimate the TPS parameters?

We would need two functions:
1) `tps_model = est_tps(source_pts, target_pts);`
2) `morphed_im = morph(im_source, tps_model);`

\[
f(x, y) = a_1 + a_x x + a_y y + \sum_{i=1}^{p} w_i U(||(x_i, y_i) - (x, y)||)
\]
TPS special case translation only

\[ f(x, y) = a_1 + a_x x + a_y y + \sum_{i=1}^{p} w_i U \left( \| (x_i, y_i) - (x, y) \| \right) \]
$f(x, y) = a_1 + a_x x + a_y y + \sum_{i=1}^{p} w_i U(||(x_i, y_i) - (x, y)||)$
TPS parameters full case

\[ f(x, y) = a_1 + a_x x + a_y y + \sum_{i=1}^{p} w_i U \left( \| (x_i, y_i) - (x, y) \| \right) \]
TPS full model

\[ f(x, y) = a_1 + a_x x + a_y y + \sum_{i=1}^{p} w_i U (\| (x_i, y_i) - (x, y) \|) \]

\[ K_{ij} = U (\| (x_i, y_i) - (x_j, y_j) \|). \]

\[ \text{ith row of } P \text{ is } (1, x_i, y_i), \]

\[
\begin{bmatrix}
K & P \\
PT & O
\end{bmatrix}
\begin{bmatrix}
w \\
a
\end{bmatrix}
= 
\begin{bmatrix}
v \\
o
\end{bmatrix}
\]
Building Texture Models

- For each example, extract texture vector
- Normalise vectors (as for eigenfaces)
- Build eigen-model

\[ \text{Texture, } g \]

Warp to mean shape

\[ g = \bar{g} + P_g b_g \]
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
Michael Jackson, Black and White video.