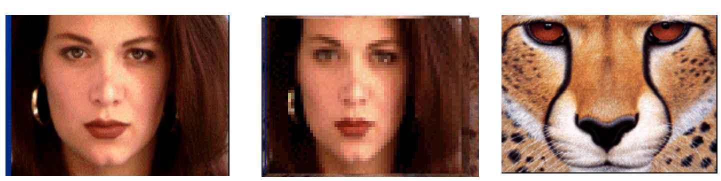
Image Morphing

15-463: Rendering and Image Processing
Alexei Efros

Morphing = Object Averaging

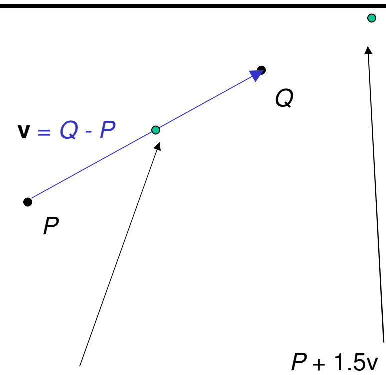


The aim is to find "an average" between two objects
- Not an average of two images of objects…
- …but an image of the average object!
- How can we make a smooth transition in time?
  - Do a "weighted average" over time t

How do we know what the average object looks like?
- We haven't a clue!
- But we can often fake something reasonable
  - Usually required user/artist input

Averaging Points

What's the average of P and Q?



$P + 0.5\mathbf{v}$
$= P + 0.5(Q - P)$
$= 0.5P + 0.5\,Q$

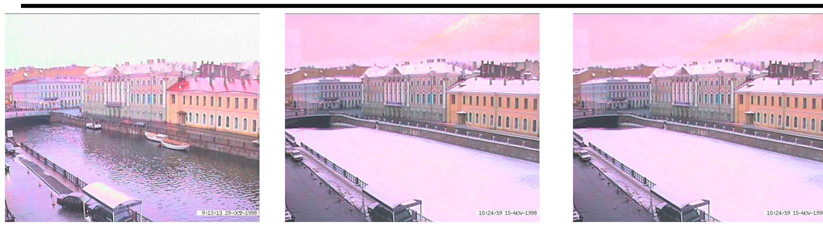
$P + 1.5\mathbf{v}$
$= P + 1.5(Q - P)$
$= -0.5P + 1.5\,Q$
(extrapolation)

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:

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

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

- Note similarity to alpha blending!

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

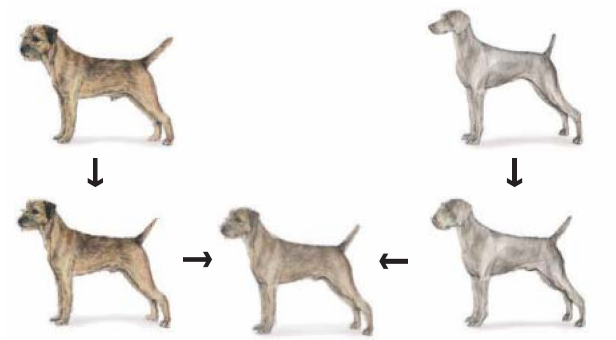
What to do?

- Cross-dissolve doesn't work
- Global alignment doesn't work
  - Cannot be done with a global transformation (e.g. prospective)
- 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

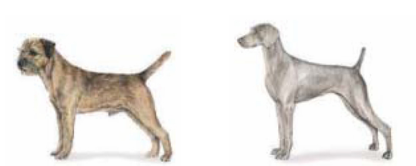


Morphing procedure:

for every t,

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

Local (non-parametric) Image Warping



Need to specify a more detailed warp function

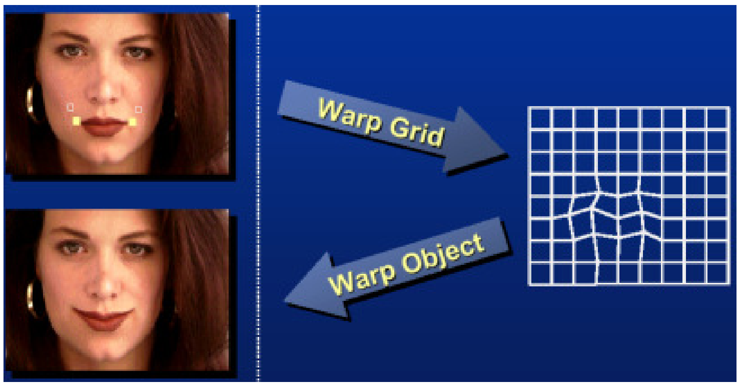
- Global warps were functions of a few (2,4,8) parameters
- Non-parametric warps u(x,y) and v(x,y) can be defined independently for every single location x,y!
- Once we know vector field u,v we can easily warp each pixel (use backward warping with interpolation)

Optical flow is just such a vector field

- Will it work for these dogs?
- Probably not… Need user control.

Image Warping – non-parametric

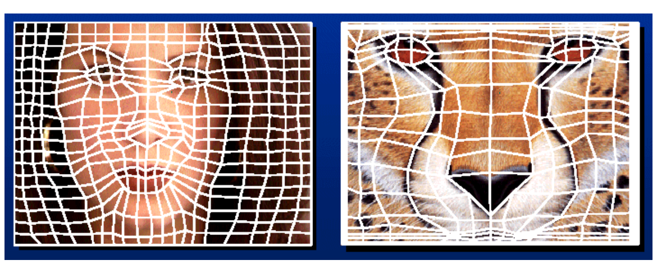
Move control points to specify a spline warp
Spline produces a smooth vector field



Warp specification

How can we specify the warp?

3. Specify corresponding *spline control points*
    - *interpolate* to a complete warping function

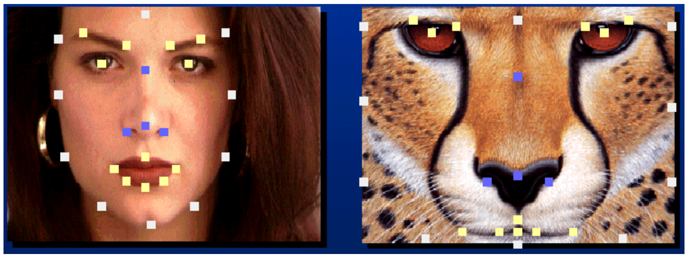


But we want to specify only a few points, not a grid

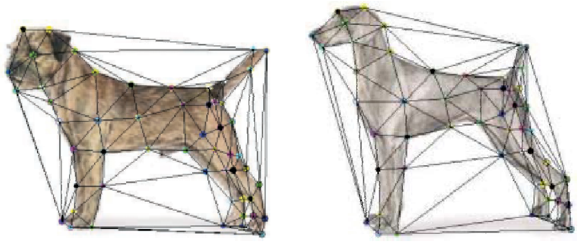
Warp specification

How can we specify the warp?

1. Specify corresponding *points*
   - *interpolate* to a complete warping function
   - How do we do it?



Triangular Mesh

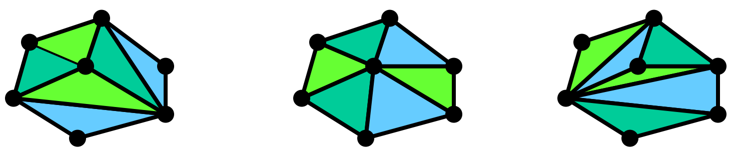


1. Input correspondences at key feature points
2. Define a triangular mesh over the points
   - Same mesh in both images!
   - Now we have triangle-to-triangle correspondences
3. Warp each triangle separately from source to destination
   - How do we warp a triangle?
   - 3 points = affine warp!
   -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

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.

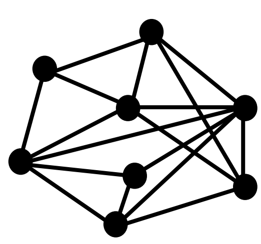
There are an exponential number of triangulations of a point set.



An O(n^3) Triangulation Algorithm

Repeat until impossible:

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



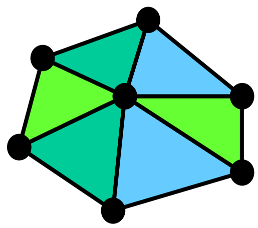
“Quality” Triangulations

Let $\alpha(T) = (\alpha_1, \alpha_2, ..., \alpha_{3t})$ be the vector of angles in the triangulation T in increasing order.

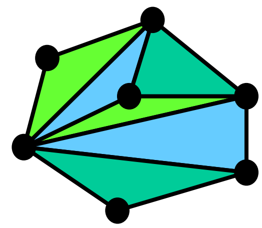
A triangulation T_1 will be “better” than T_2 if $\alpha(T_1) > \alpha(T_2)$ lexicographically.

The Delaunay triangulation is the “best”

- Maximizes smallest angles



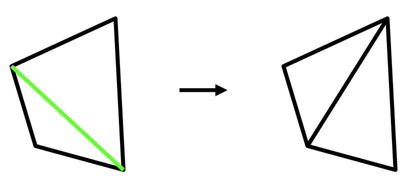
good



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.

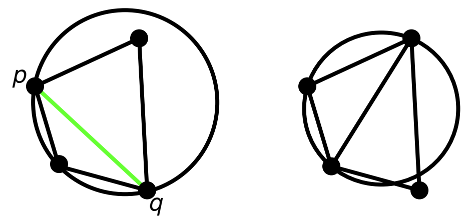


If an edge flip improves the triangulation, the first edge is called *illegal*.

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.

O(n^4) Delaunay Triangulation Algorithm

Repeat until impossible:

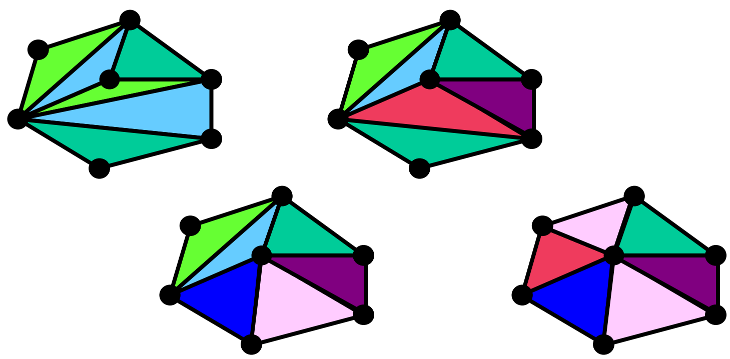
- Select a triple of sites.
- If the circle through them does not contain other sites, keep the triangle whose vertices are the triple.

Naïve Delaunay Algorithm

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

Requires proof that there are no local minima.

Could take a long time to terminate.

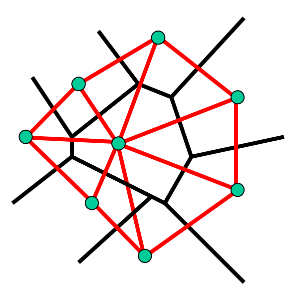


Delaunay Triangulation by Duality

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.

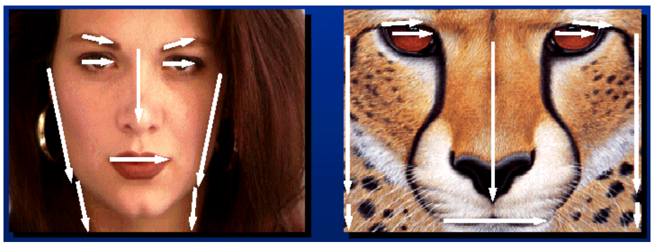
Corollary: The DT may be constructed in O($n$log$n$) time.



Warp specification

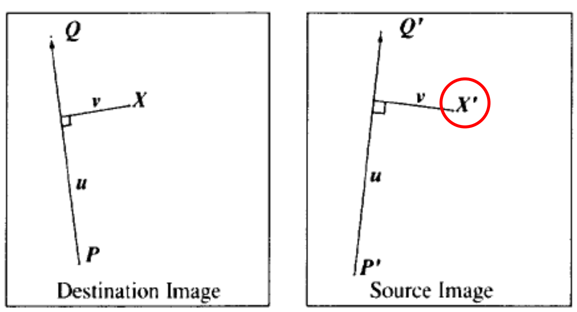
How can we specify the warp?

2. Specify corresponding *vectors*
   - *interpolate* to a complete warping function
   - The Beier & Neely Algorithm



Beier&Neely (SIGGRAPH 1992)

Single line-pair PQ to P'Q':



$$u = \frac{(X - P) \cdot (Q - P)}{\|Q - P\|^2} \quad (1)$$

$$v = \frac{(X - P) \cdot Perpendicular(Q - P)}{\|Q - P\|} \quad (2)$$

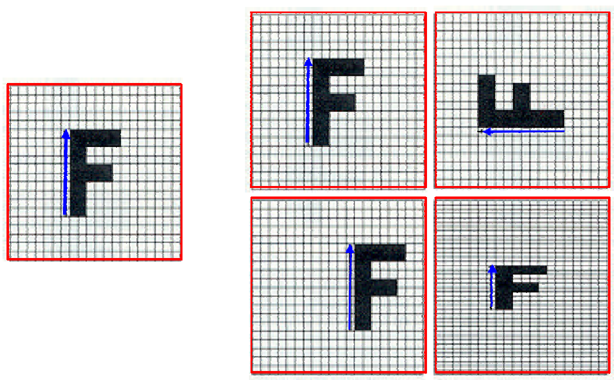
$$X' = P' + u \cdot (Q' - P') + \frac{v \cdot Perpendicular(Q' - P')}{\|Q' - P'\|} \quad (3)$$

Algorithm (single line-pair)

For each X in the destination image:

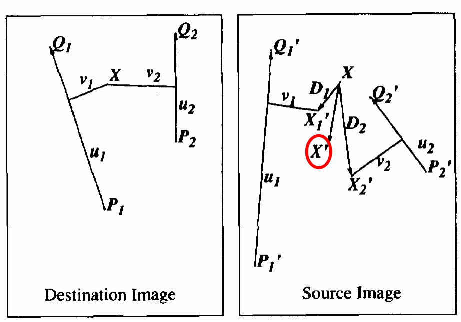
1. Find the corresponding u,v
2. Find X' in the source image for that u,v
3. destinationImage(X) = sourceImage(X')

Examples:



Multiple Lines

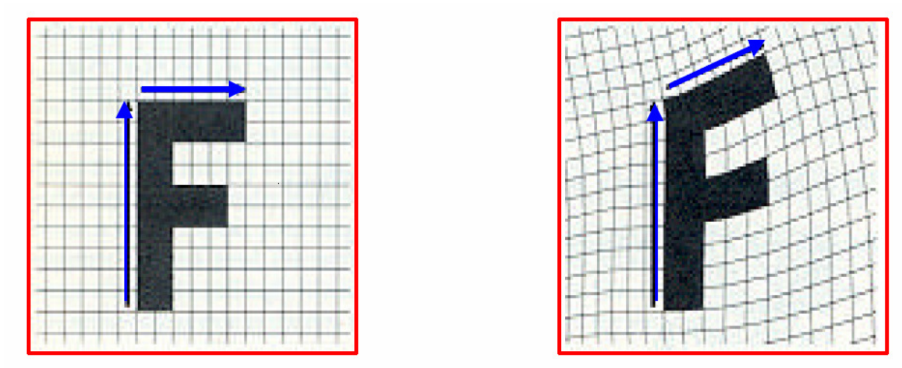
$$D_i = X_i' - X_i$$



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

Length = length of the line segment, dist = distance to line segment
a, p, b – constants. What do they do?

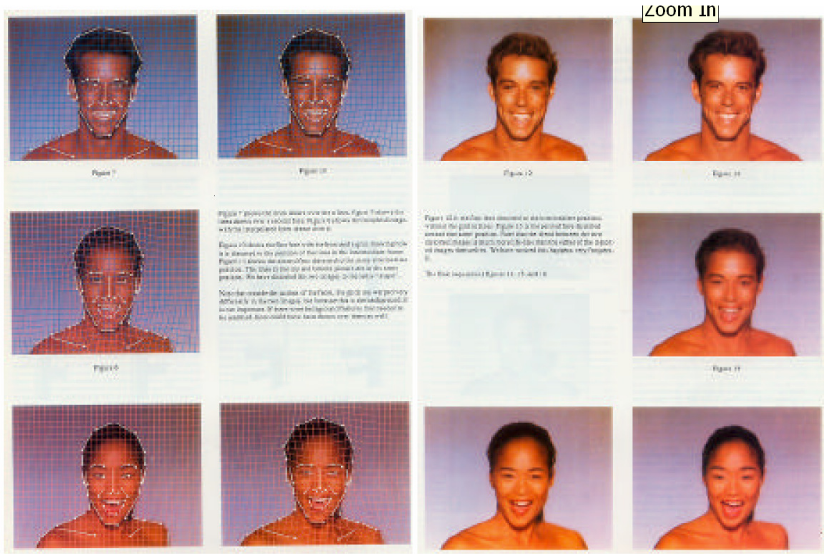
Resulting warp (complex!)



Full Algorithm

For each pixel X in the destination
  $DSUM = (0,0)$
  $weightsum = 0$
  For each line $P_i\, Q_i$
    calculate u,v based on $P_i\, Q_i$
    calculate X'_i based on u,v and $P_i'Q_i'$
    calculate displacement $D_i = X_i' - X_i$ for this line
    $dist$ = shortest distance from X to $P_i\, Q_i$
    $weight = (length^p / (a + dist))^b$
    $DSUM$ += $D_i * weight$
    $weightsum$ += $weight$
  $X' = X + DSUM / weightsum$
  destinationImage(X) = sourceImage(X')

Results

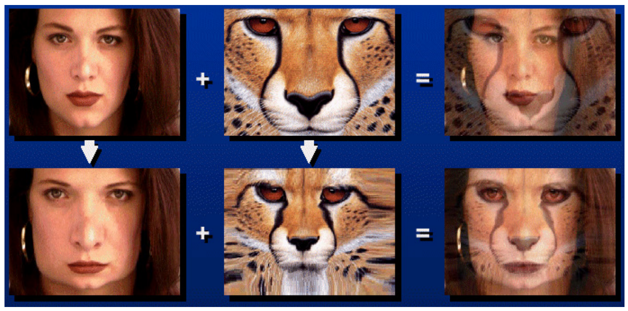


Dynamic Scene

Image Morphing

We know how to warp one image into the other, but how do we create a morphing sequence?

1. Create an intermediate warping field (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 warp at time t?

For optical flow?

- Easy. Interpolate each flow vector
- That's how interframe interpolation is done

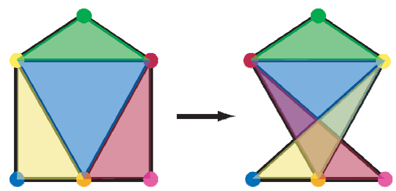
For feature point methods

- Simple linear interpolation of each feature pair (e.g. 0.5p1+0.5p0 for the middle warp)

For Beier-Neely?

- Can do the same for line end-points
- But what could happen?
- A line rotating 180 degrees will become 0 length in the middle
- One solution is to interpolate line mid-point and orientation angle
- Not very intuitive

Other Issues



Beware of folding

- Can happen in any of the methods
- 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

Video Matching