Image Morphing, Triangulation

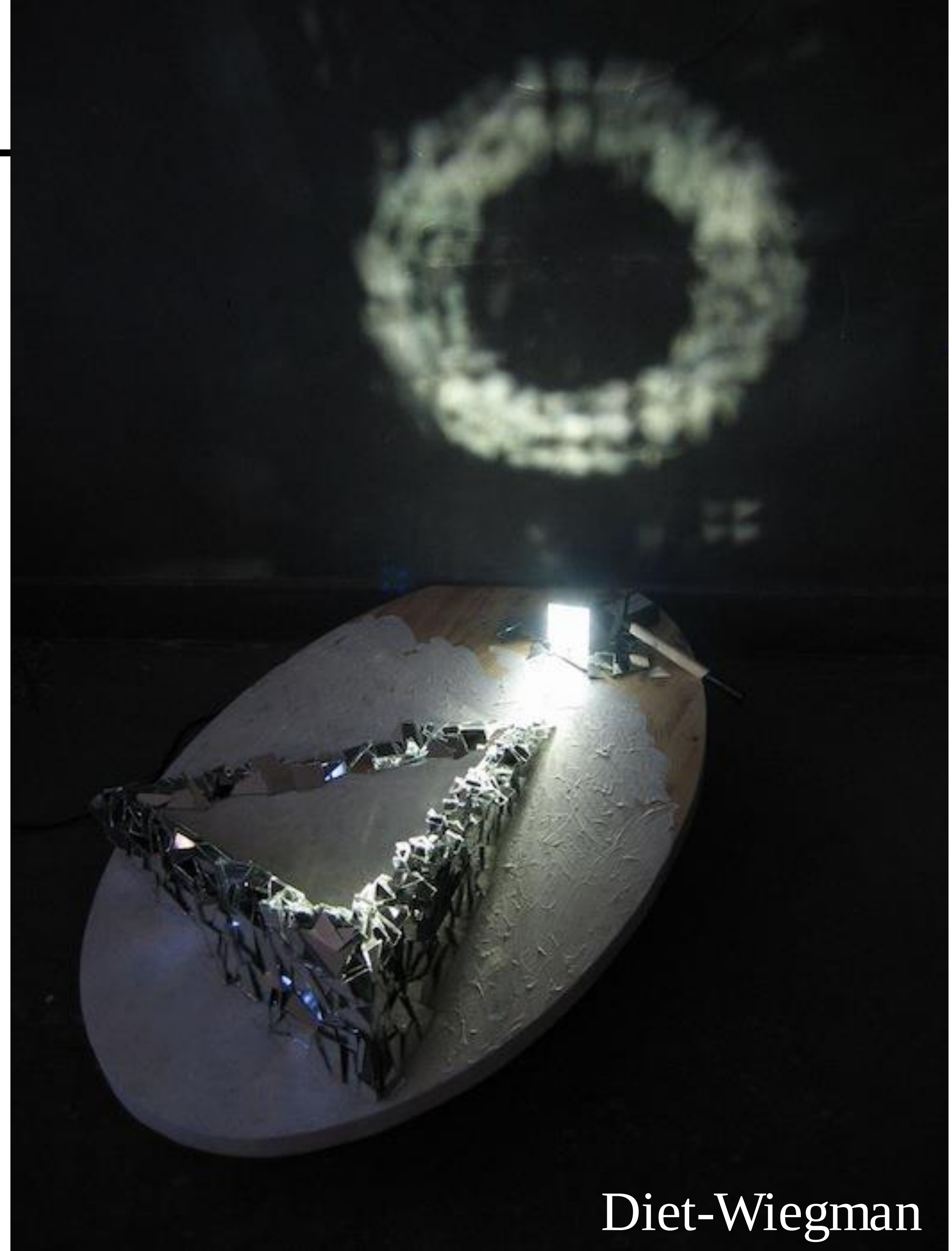
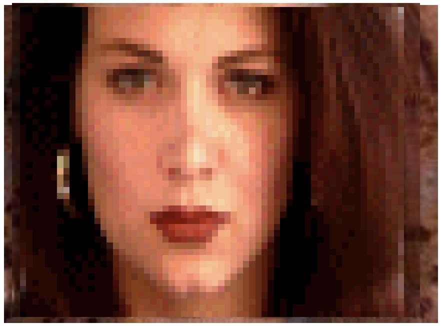


Image morphing:

Lecture notes borrowed from A. Efros, T. Cootes

Morphing = Object Averaging


The aim is to find "an average" between two objects

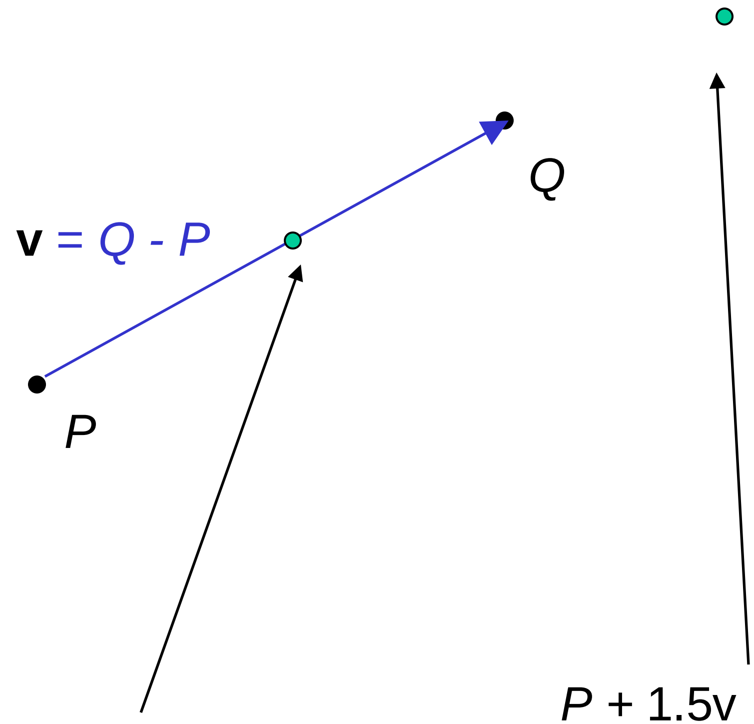
- Not an average of two images of objects…
- …but an image of the average object!
- How can we make a smooth transition in time?
  - Do a "weighted average" over time t

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:

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

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

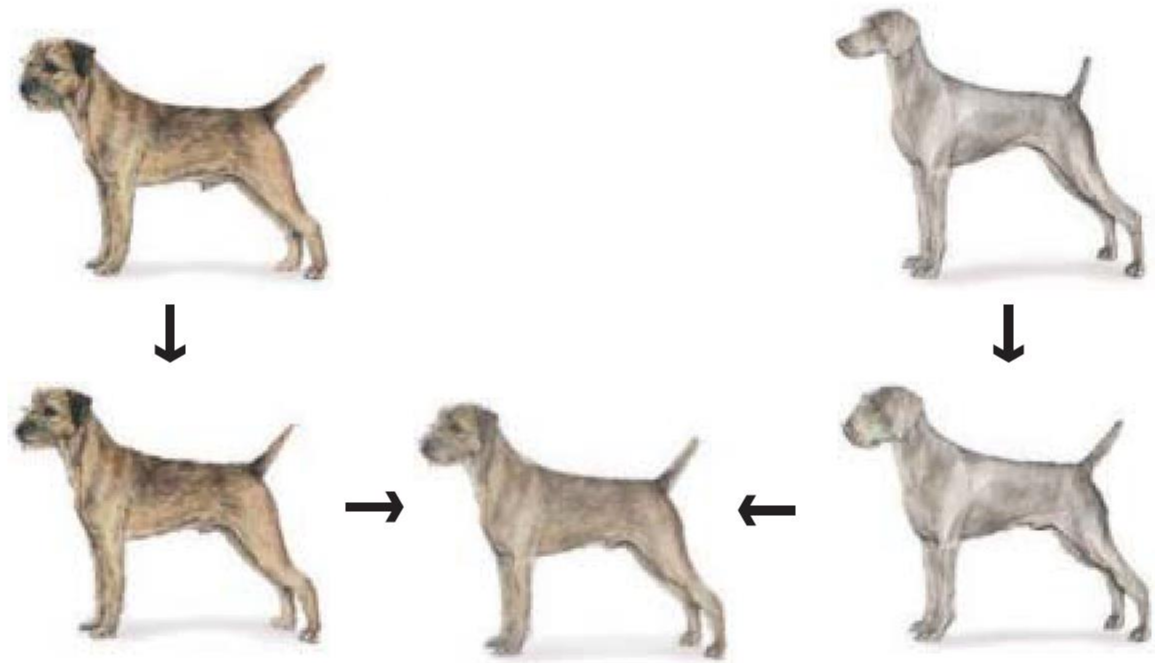
What to do?

- 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

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

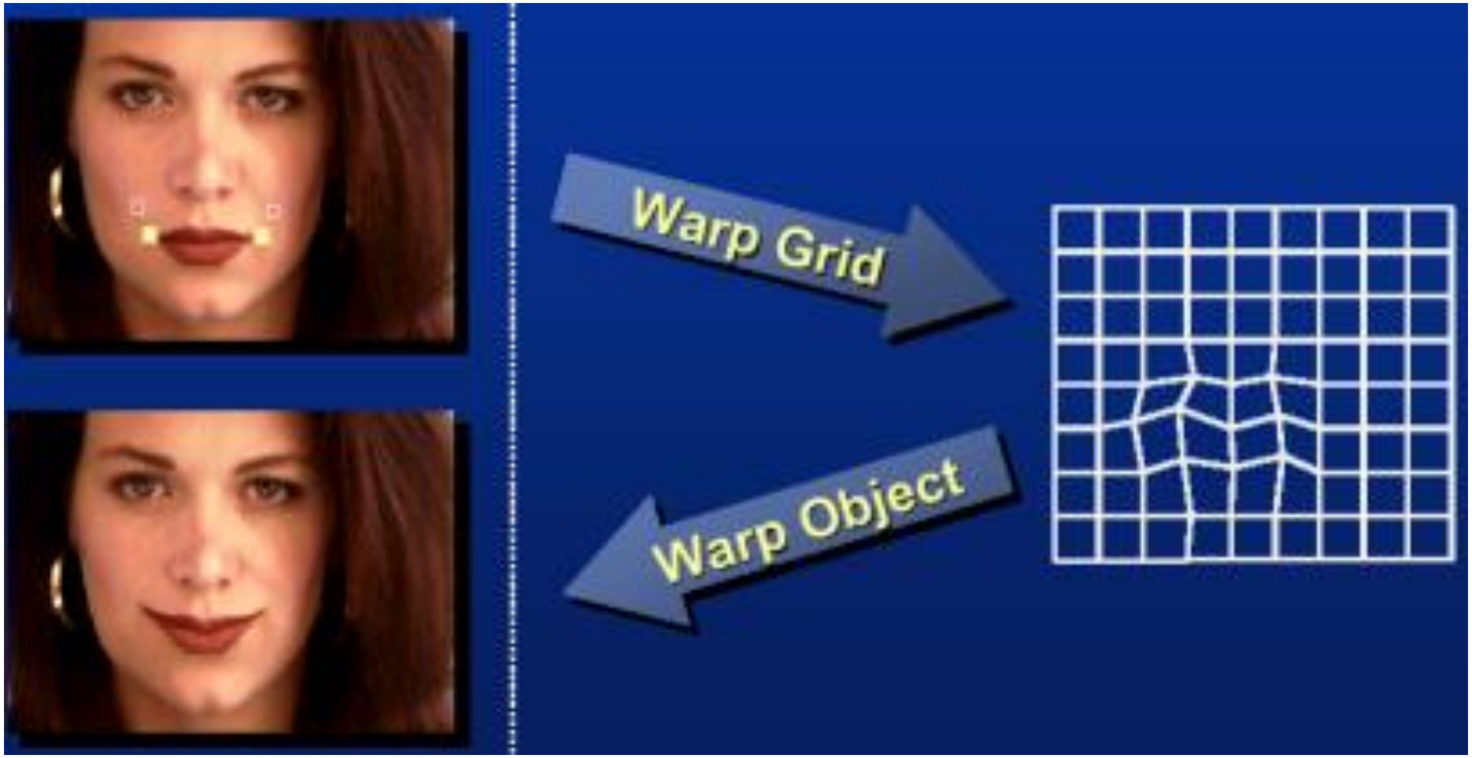
Need to specify a more detailed warp function

- 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)

Image Warping – non-parametric

Move control points to specify a spline warp

Spline produces a smooth vector field

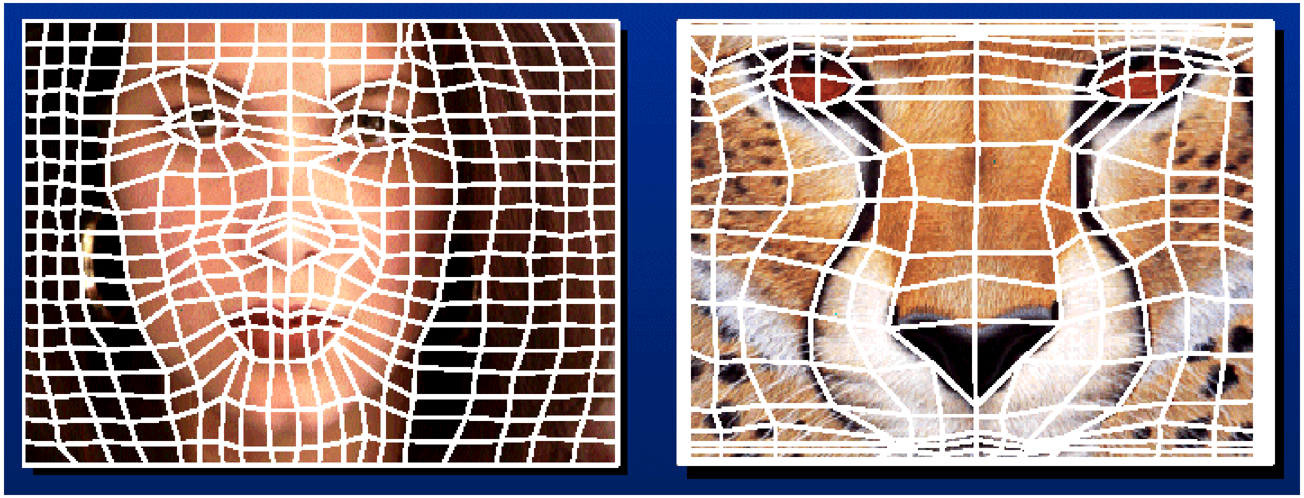


Warp specification - dense

How can we specify the warp?

Specify corresponding *spline control points*

- *interpolate* to a complete warping function



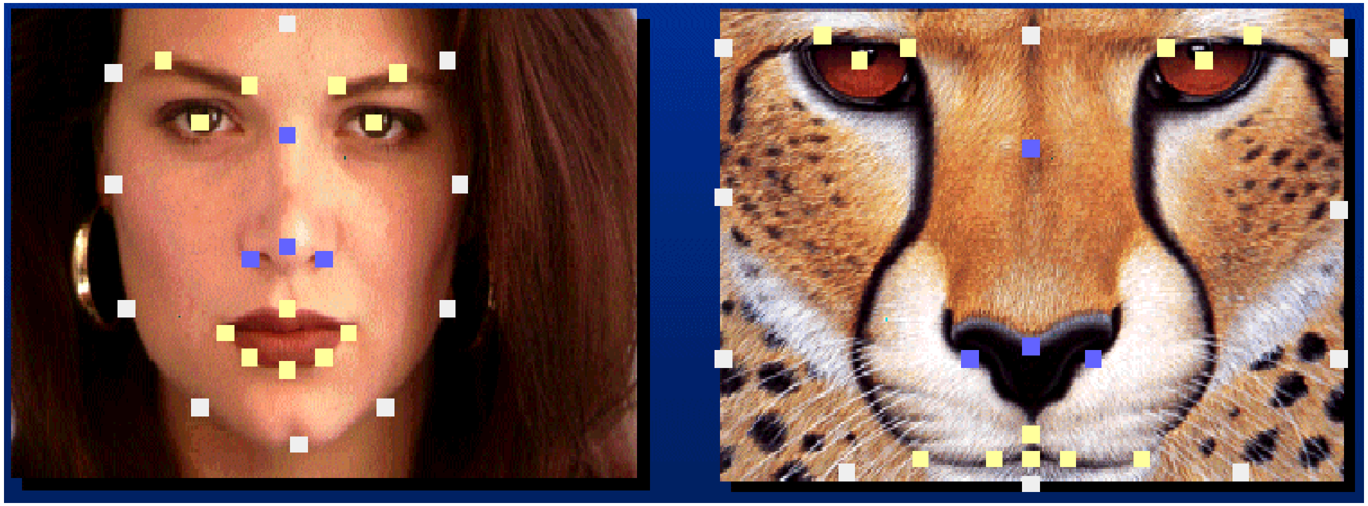
But we want to specify only a few points, not a grid

Warp specification - sparse

How can we specify the warp?

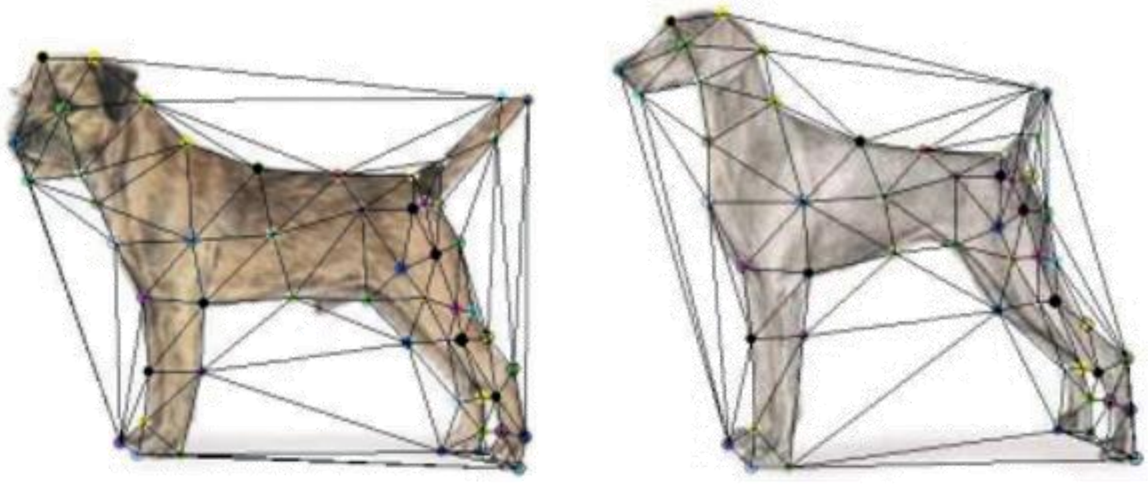
Specify corresponding *points*

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



How do we go from feature points to pixels?

Triangular Mesh

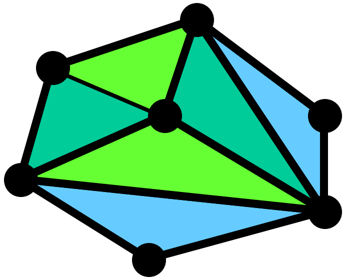
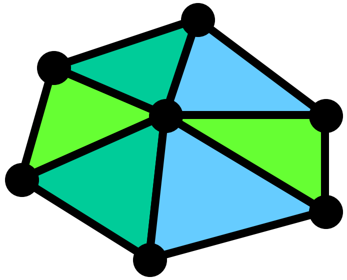
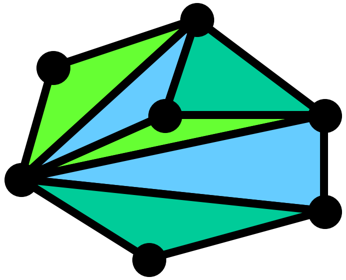


1. Input correspondences at key feature points
2. Define a triangular mesh over the points
   - Same mesh in both images!
   - Now we have triangle-to-triangle correspondences
3. Warp each triangle separately from source to destination
   - How do we warp a triangle?
   - 3 points = affine warp!
   -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

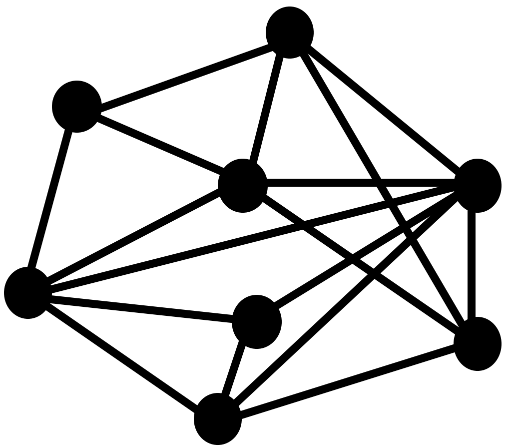
There are an exponential number of triangulations of a point set.

An O(n^3) Triangulation Algorithm

Repeat until impossible:

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



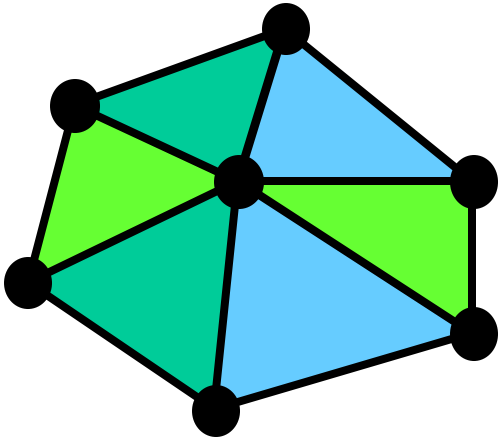
"Quality" Triangulations

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

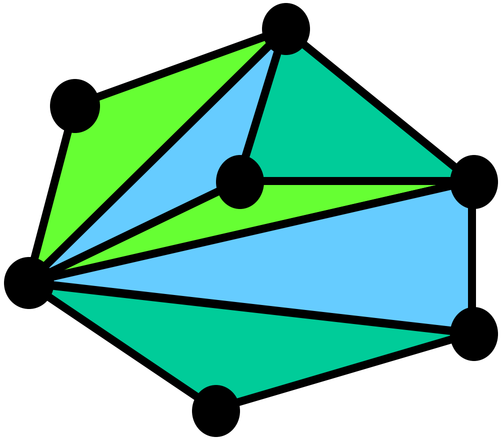
A triangulation T_1 will be "better" than T_2 if $\alpha(T_1) > \alpha(T_2)$ lexicographically.

The Delaunay triangulation is the "best"

- Maximizes smallest angles

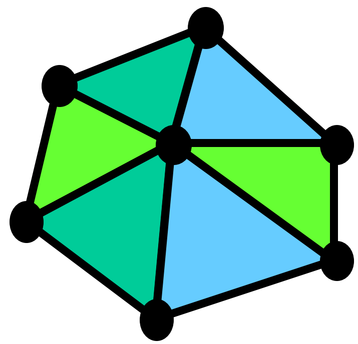


good

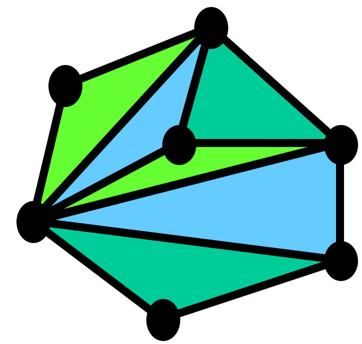


bad

Boris Nikolaevich Delaunay (March 15, 1890 – July 17, 1980)



Delaunay

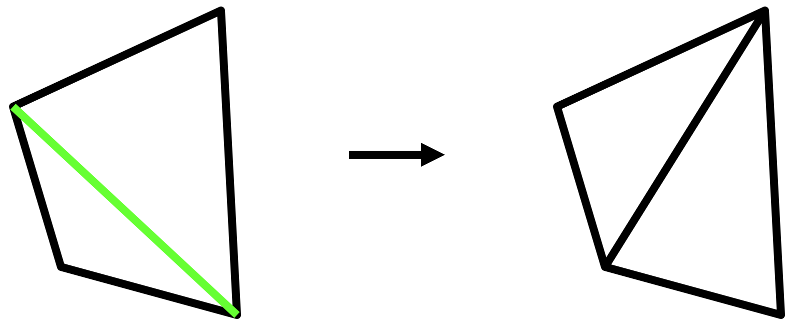


bad

http://higeom.math.msu.su/history/delone_r.html

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.

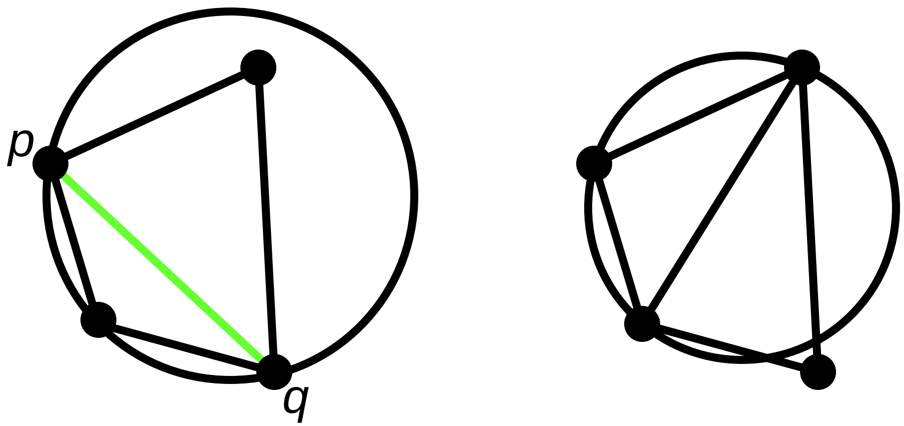


If an edge flip improves the triangulation, the first edge is called *illegal*.

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.

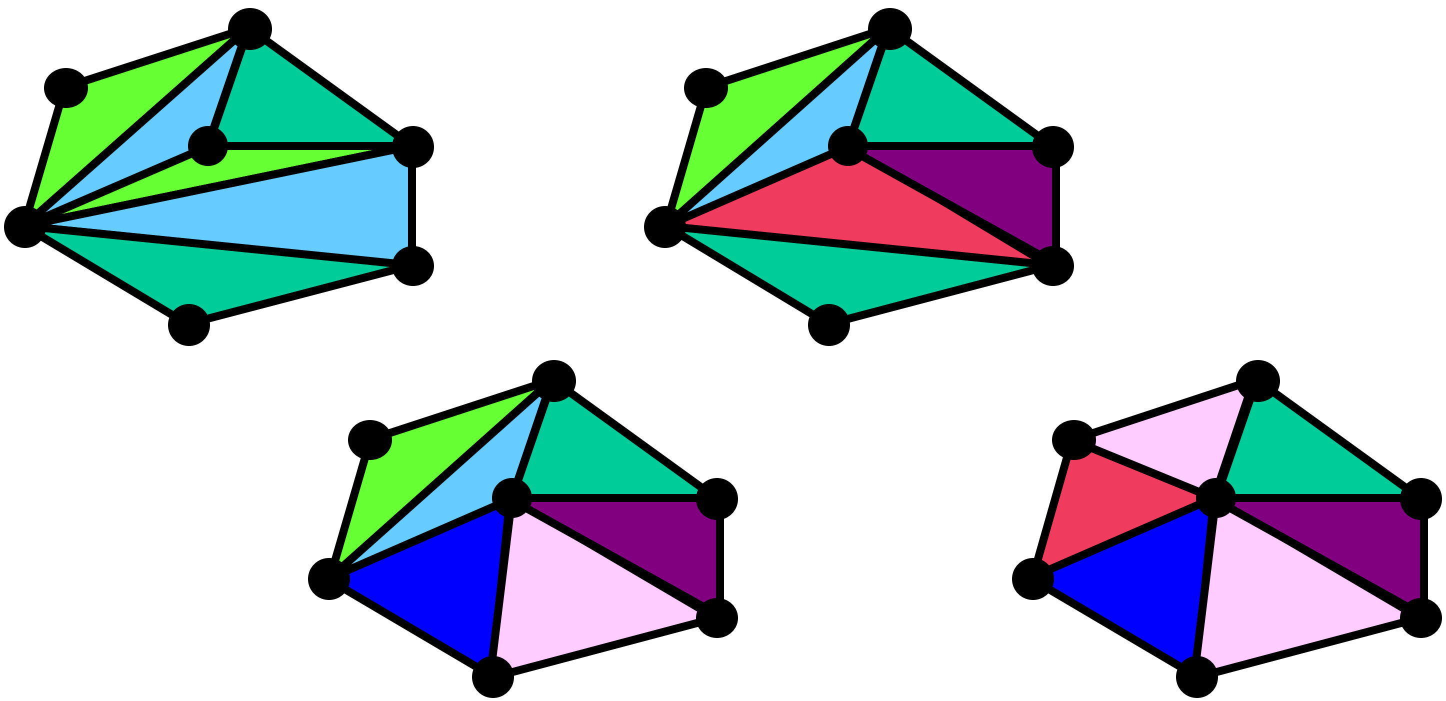
Corollary: A triangle is Delaunay iff the circle through its vertices is empty of other sites.

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.



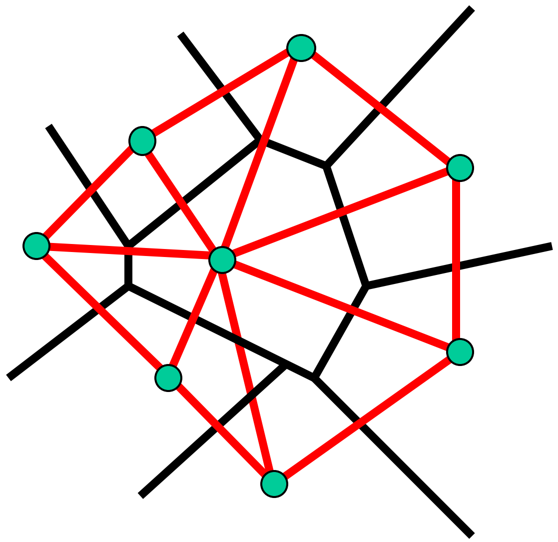
Delaunay Triangulation by Duality

General position assumption: There are no four co-circular points.

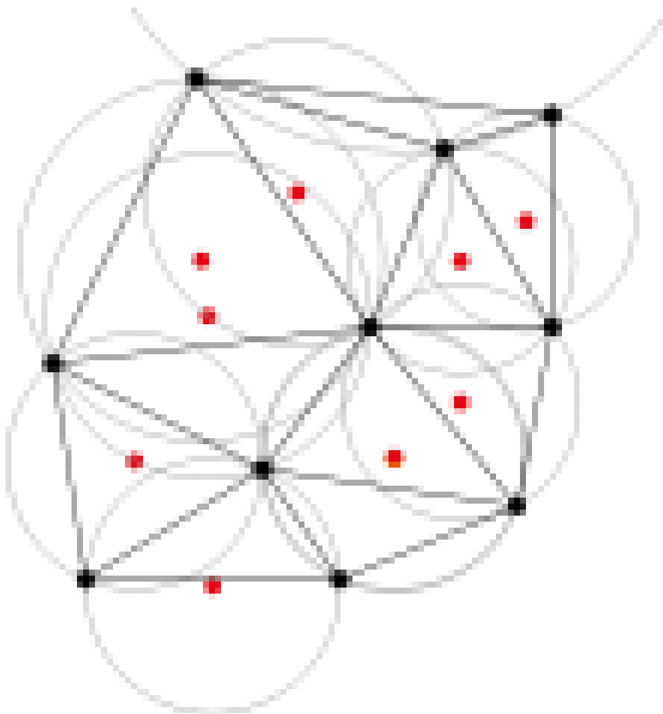
Draw the dual to the Voronoi diagram by connecting each two neighboring sites in the Voronoi diagram.

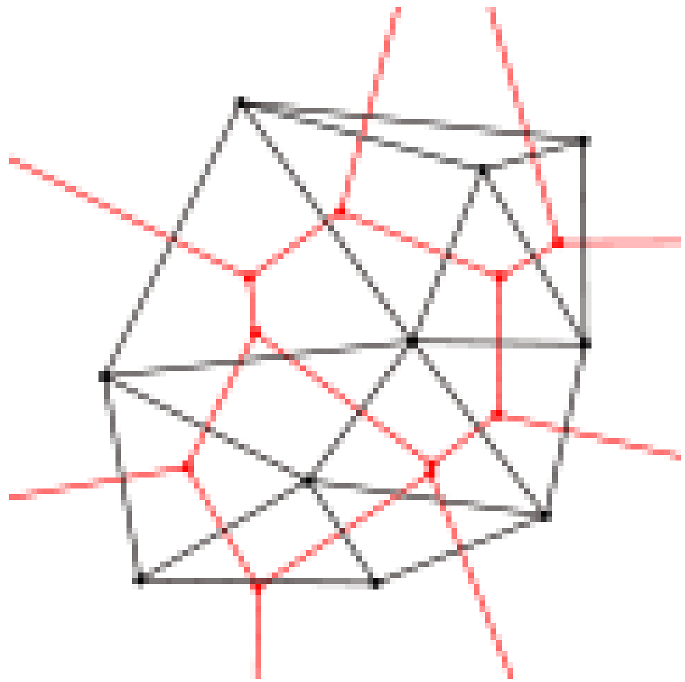
Corollary: The DT may be constructed in O(*n*log*n*) time.

This is what Matlab's `delaunay` function uses.



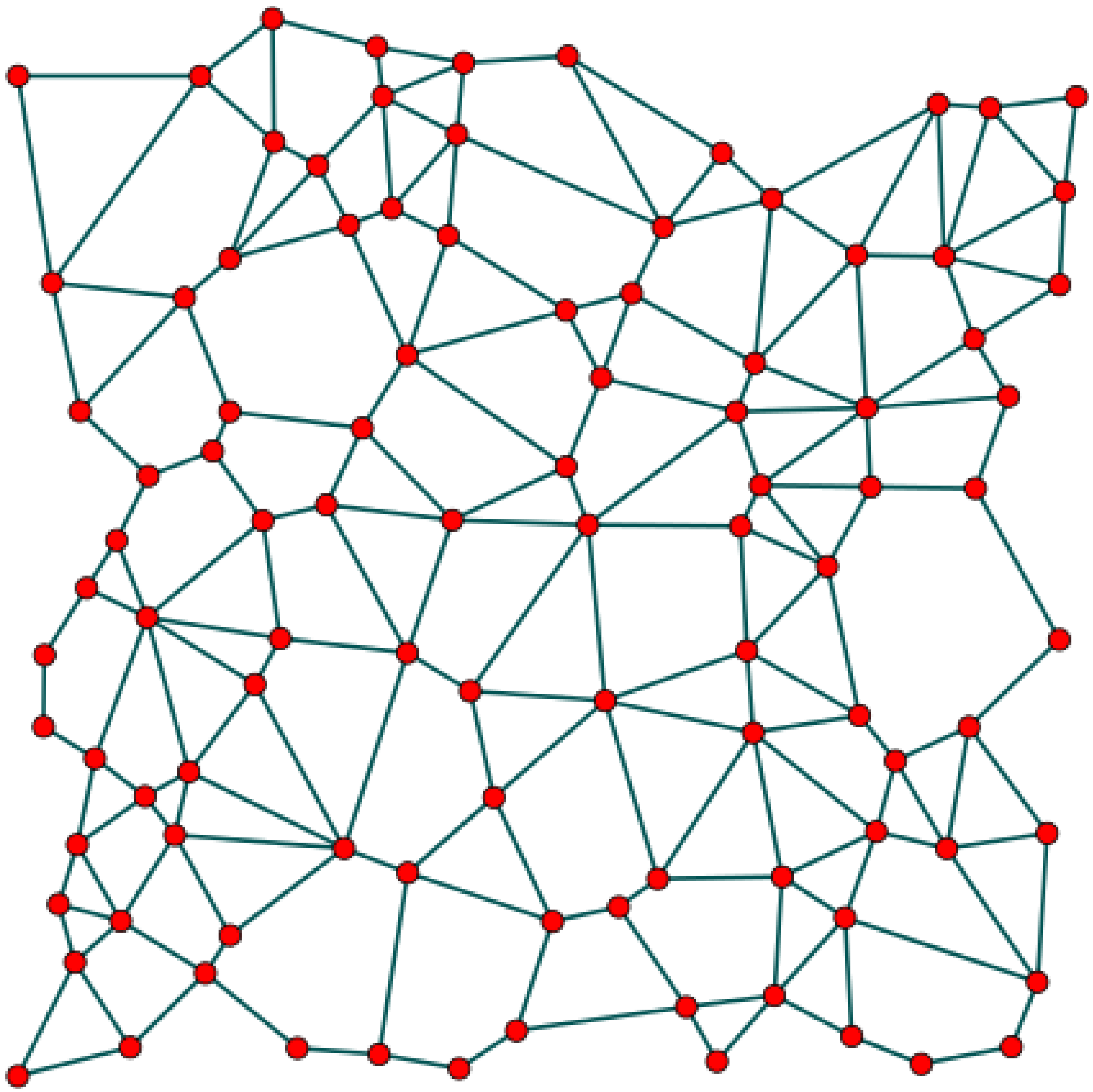
A circle circumscribing any Delaunay triangle does not contain any other input points in its interior.

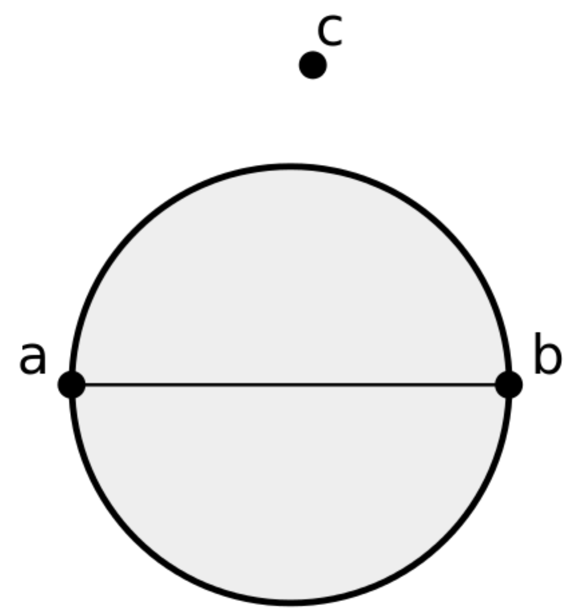




If a circle passing through two of the input points doesn't contain any other of them in its interior, then the segment connecting the two points is an edge of a Delaunay triangulation of the given points.

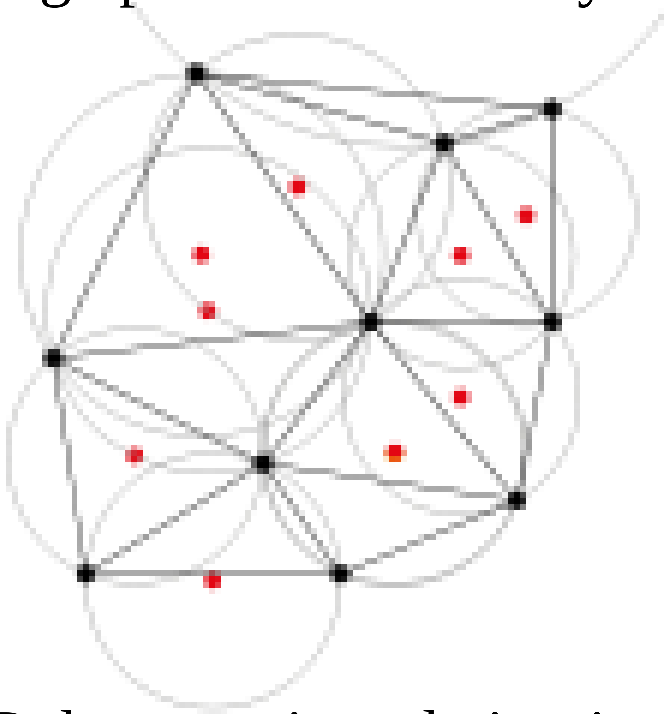
Gabriel Graph is a subset of Delaunay triangle

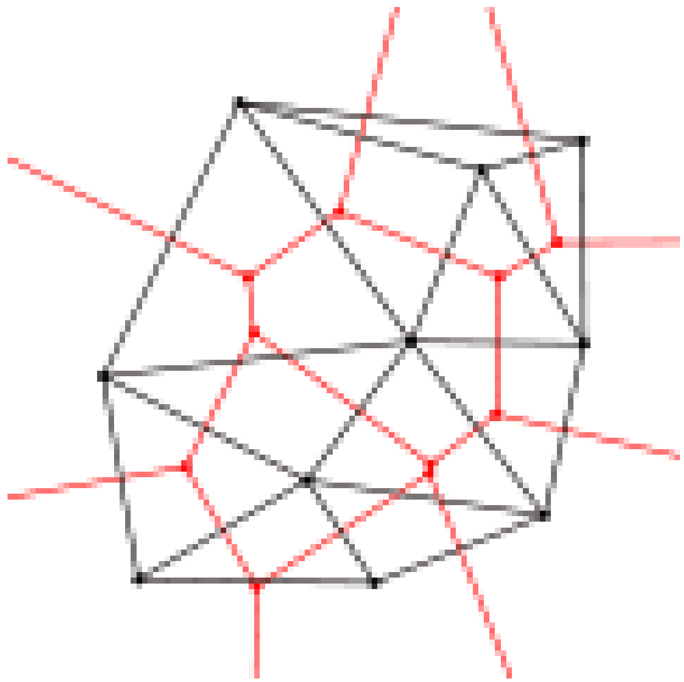




Points a and b are Gabriel neighbours, as c is outside their diameter circle.

The closest neighbor b to any point p is on an edge bp in the Delaunay triangulation since the nearest neighbor graph is a subgraph of the Delaunay triangulation.





The Delaunay triangulation is a geometric spanner: the shortest path between two vertices, along Delaunay edges, is known to be no longer than $\frac{4\pi}{3\sqrt{3}} \approx 2.418$ times the Euclidean distance between them.

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

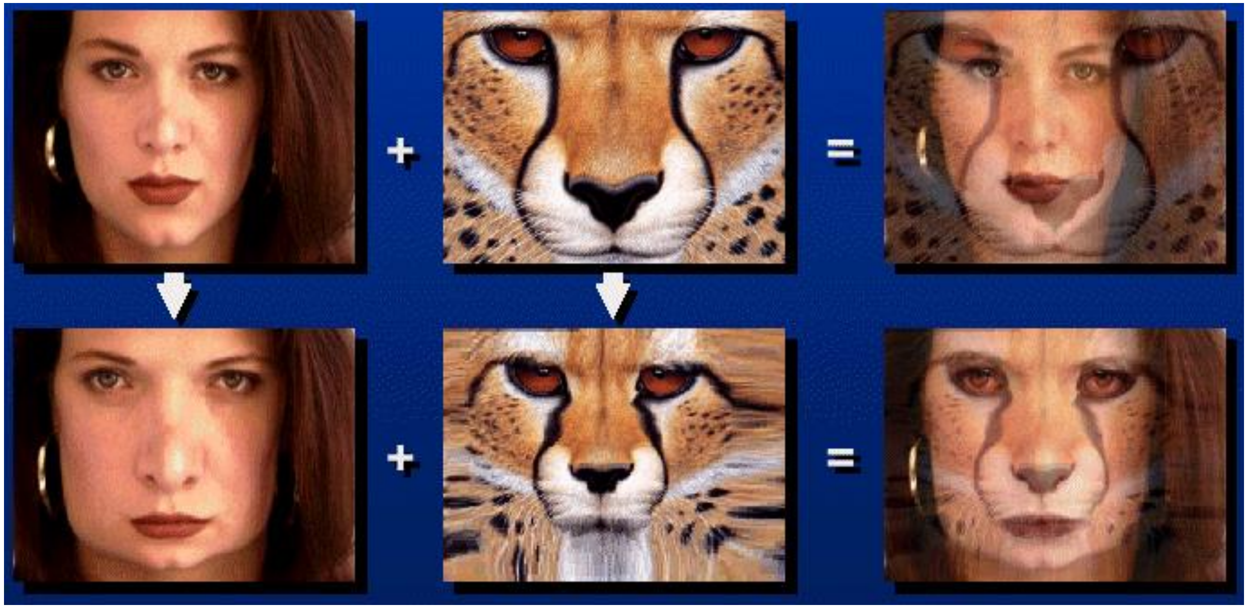
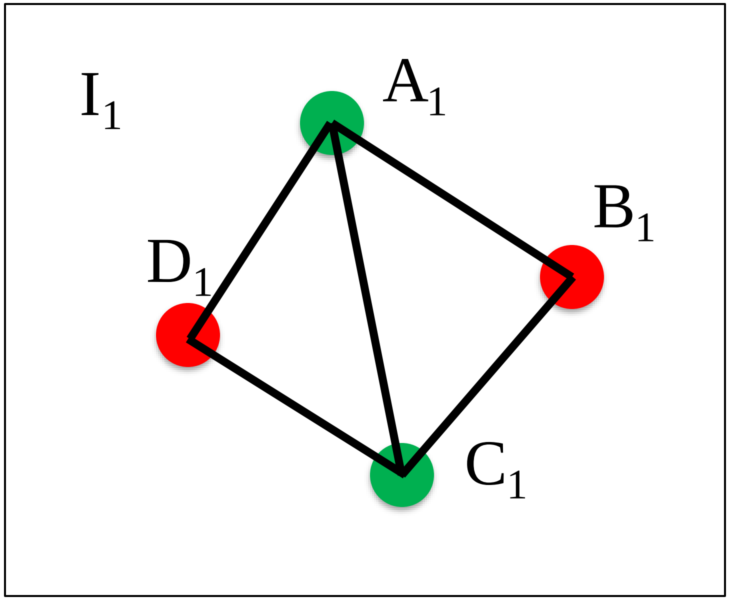
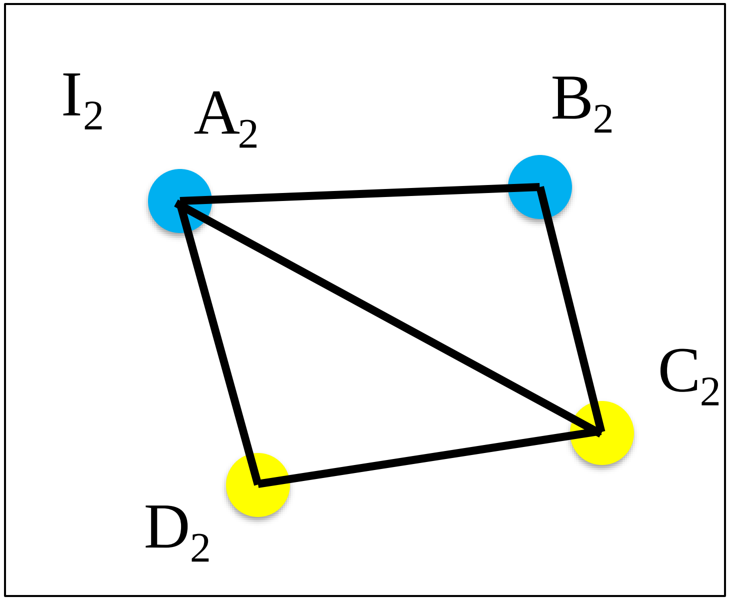


Image Morphing





Corresponding points:

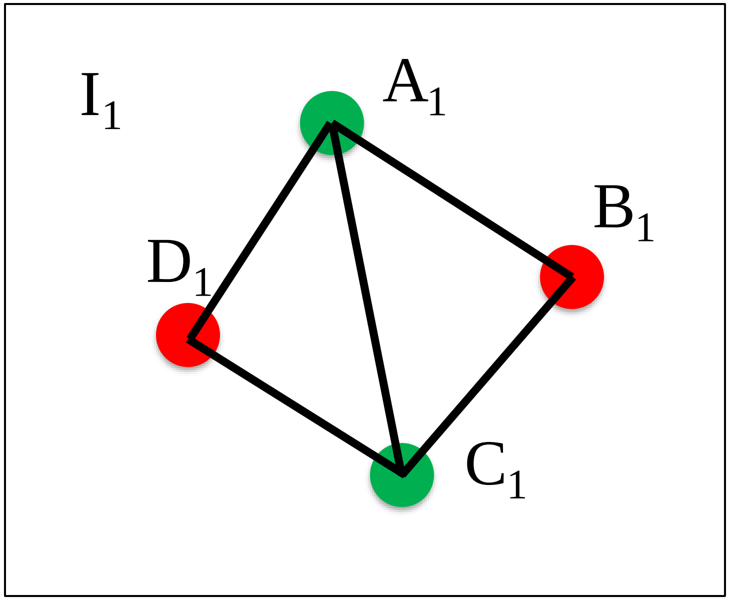
- A1 – A2, B1 – B2, C1 – C2, D1 – D2

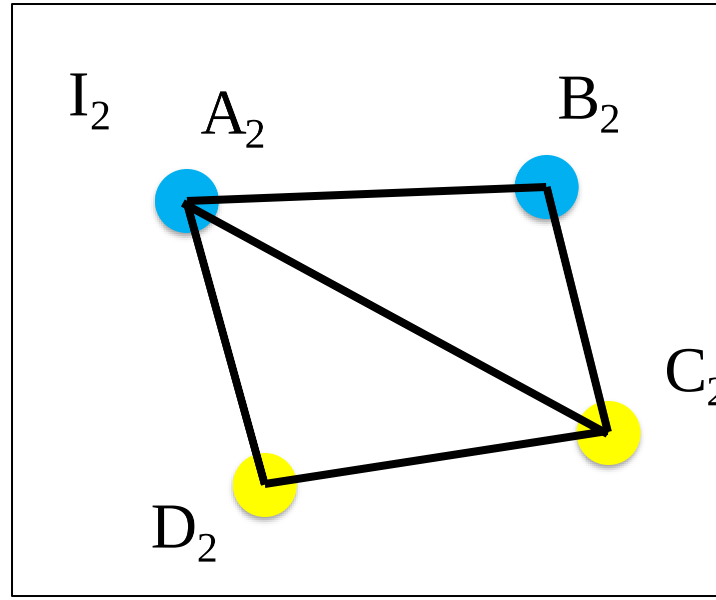
Step1: Create an intermediate shape (by interpolation)

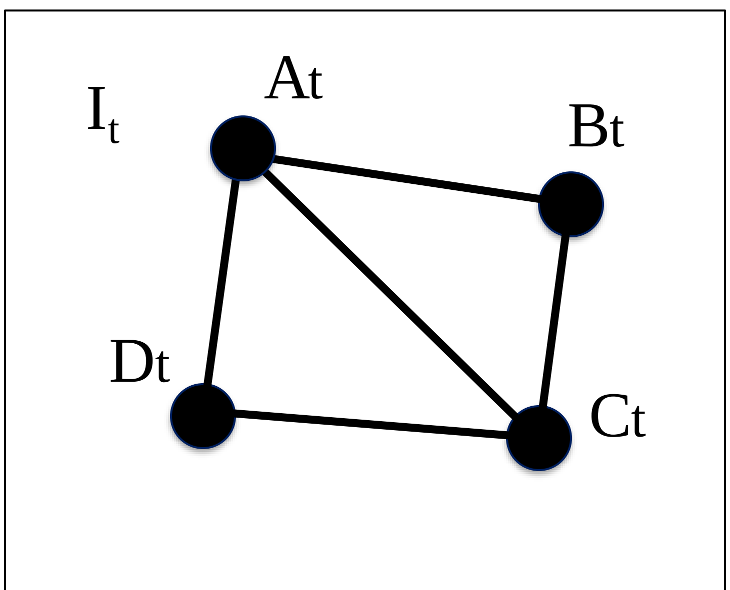
Step2: Warp both images towards the shape

Step3: Cross-dissolve the color

Image Morphing: Intermediate Shape







$$A_t = tA_1 + (1-t)A_2$$
$$B_t = tB_1 + (1-t)B_2$$
$$C_t = tC_1 + (1-t)C_2$$
$$D_t = tD_1 + (1-t)D_2$$
$$0 \leq t \leq 1$$

Image Morphing: Warping

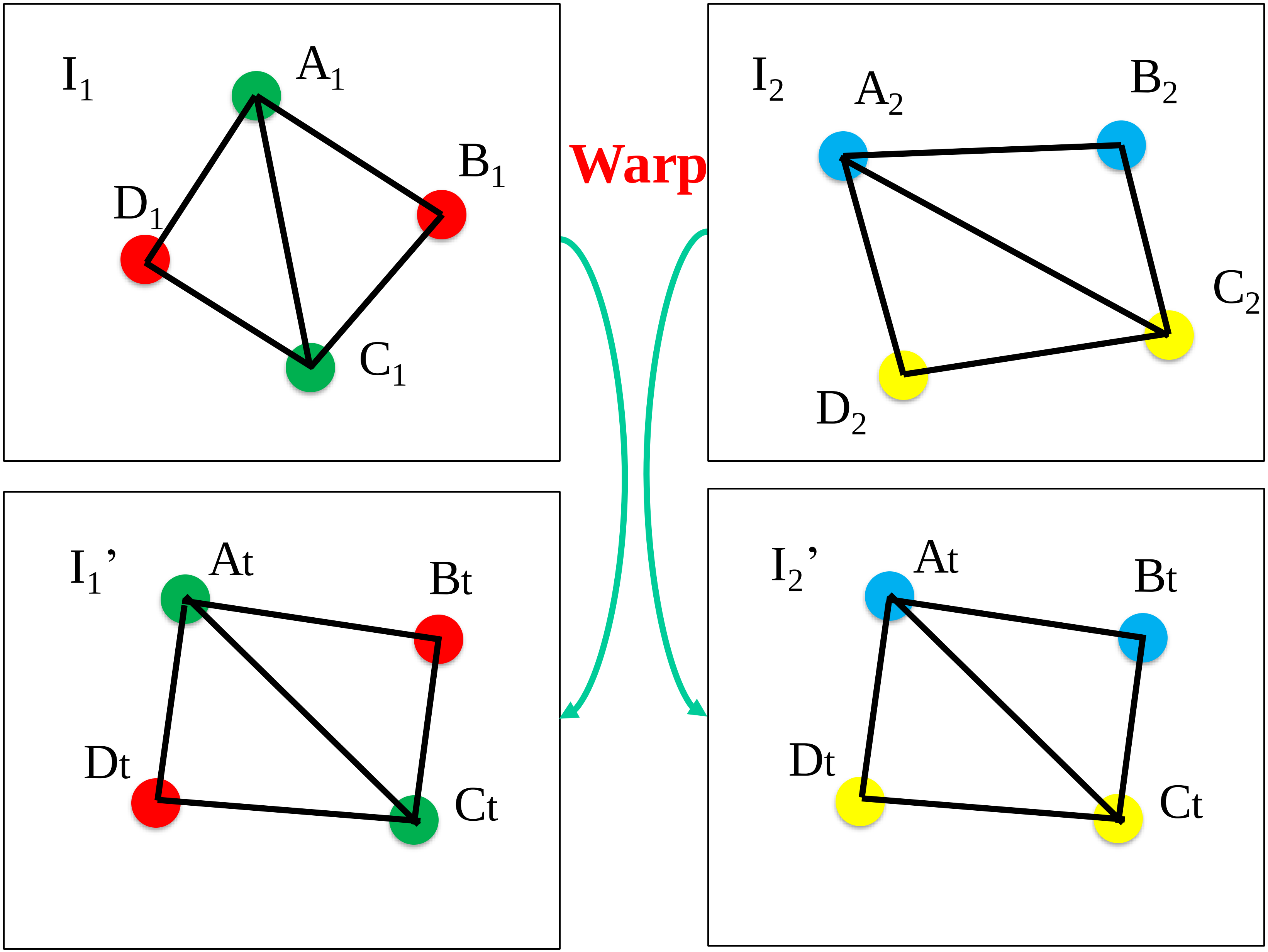
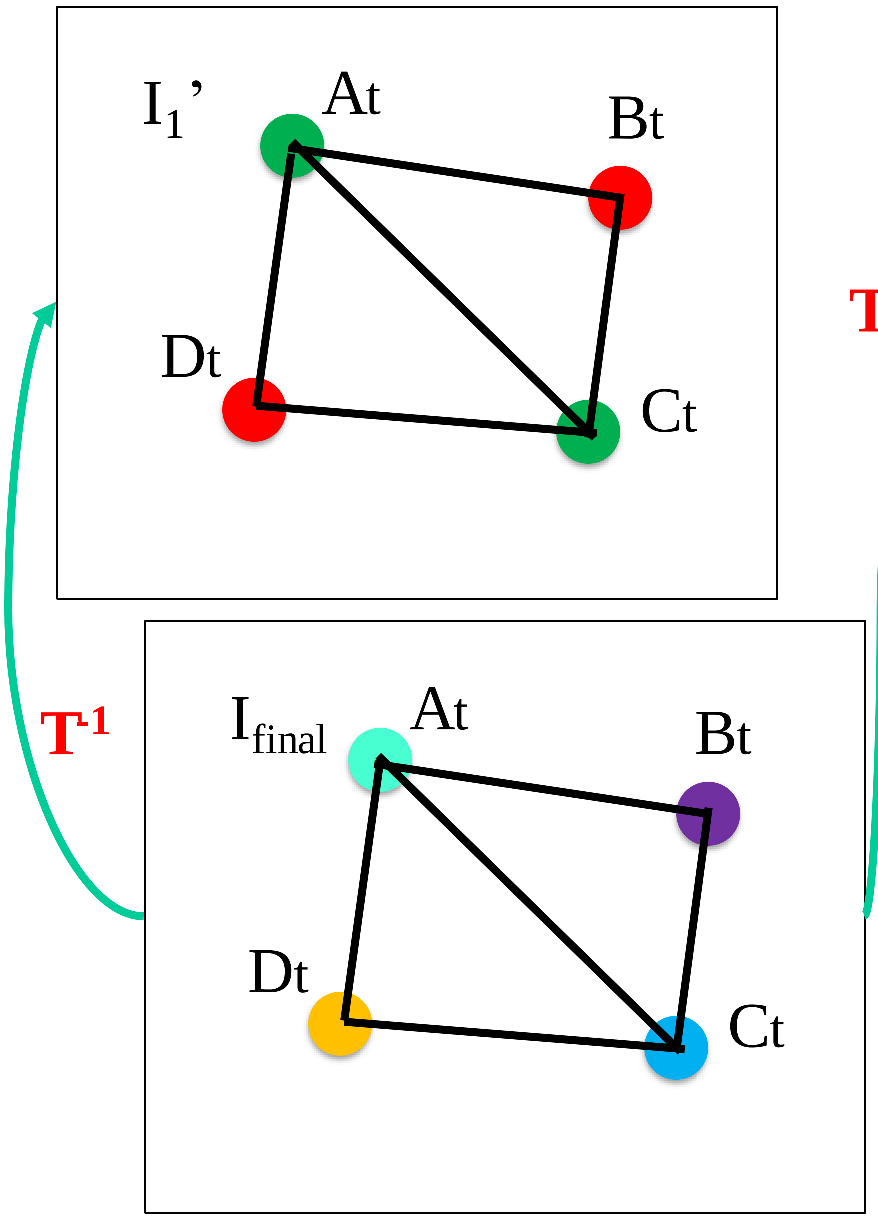
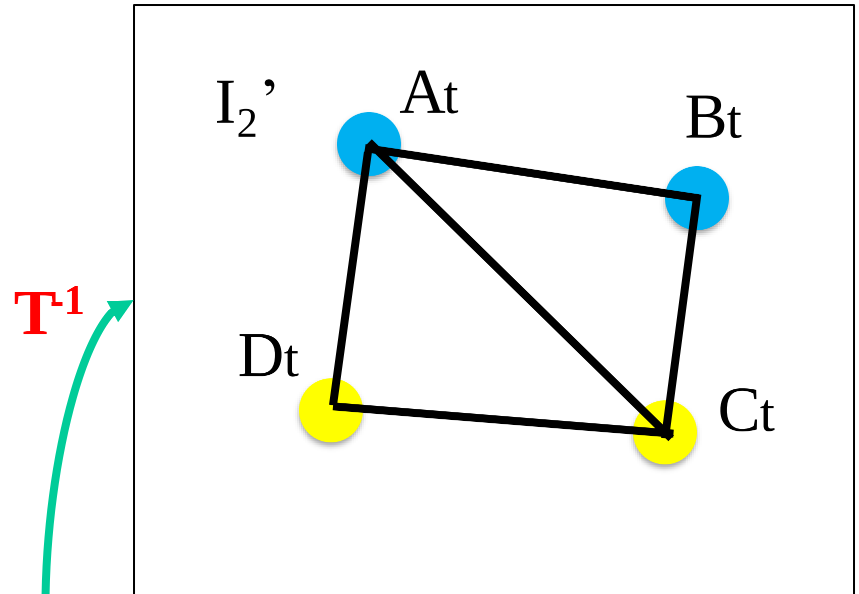


Image Morphing: Cross Dissolve Colors





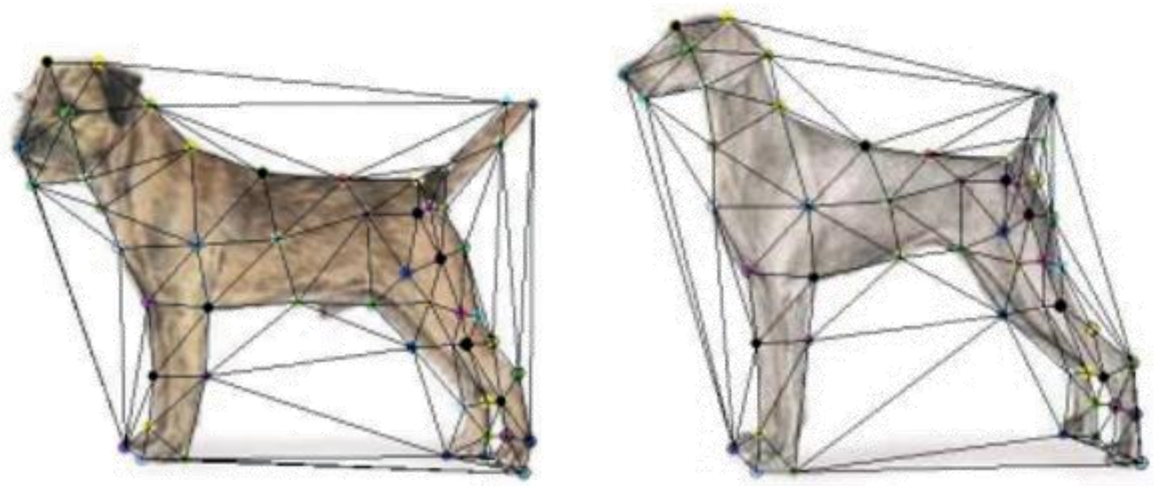
Cross-dissolve the colors by **inverse triangle warping**

- A_t: Cyan = Green + Blue
- B_t: Purple = Red + Blue
- C_t: Blue = Green + Yellow
- Dt: Orange = Red + Yellow

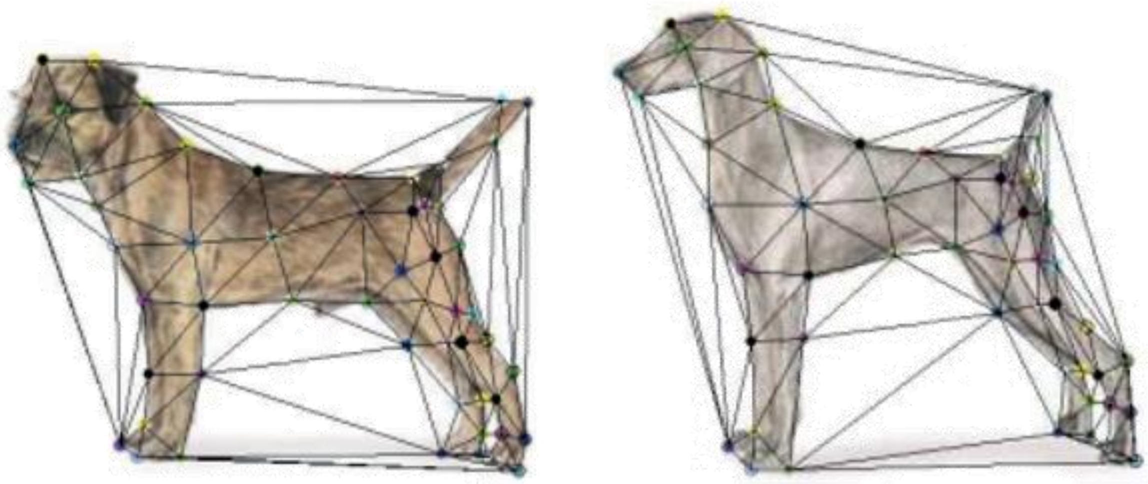
Warp interpolation

How do we create an intermediate warp at time t?

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



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

Warping texture

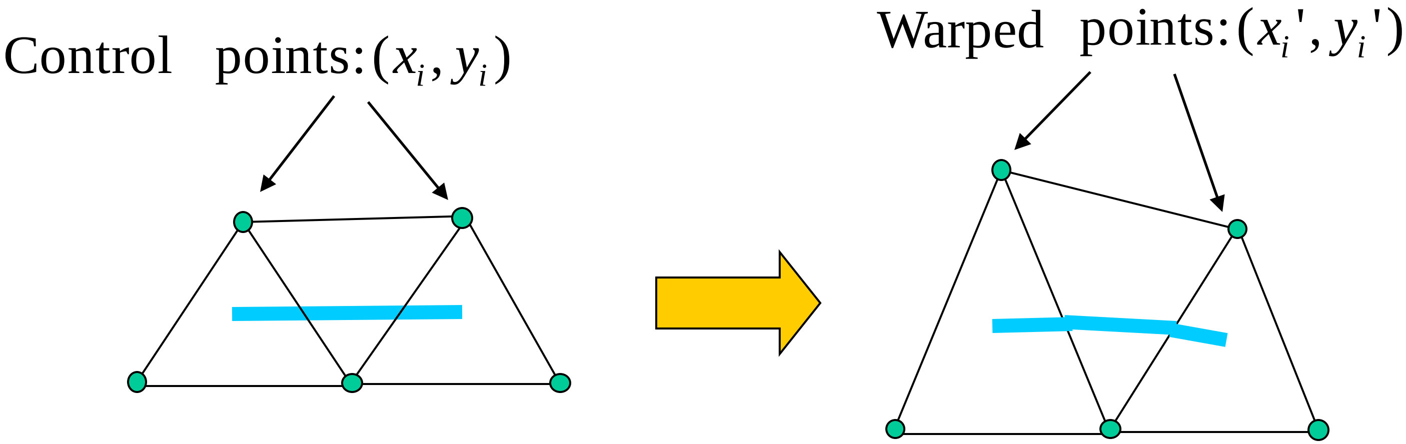
Problem:

- 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

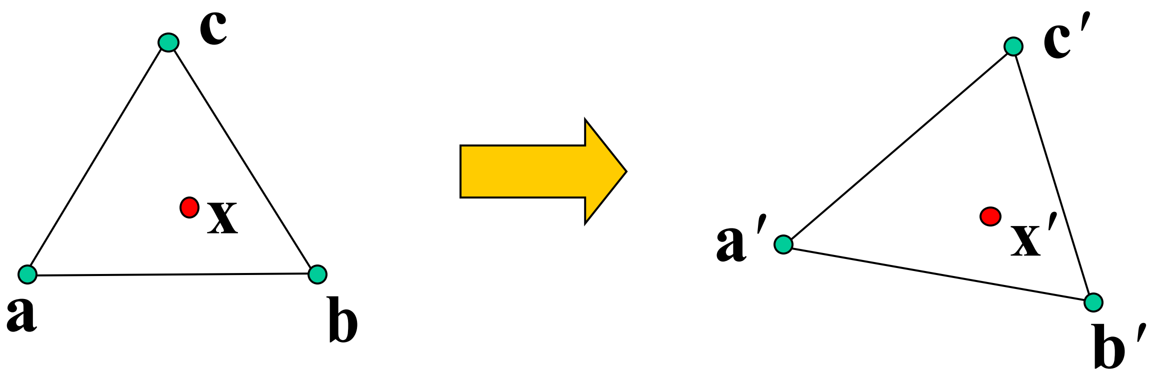


Region of interest enclosed by triangles.

Moving nodes changes each triangle

Just need to map regions between two triangles

Barycentric Co-ordinates



$$\mathbf{x} = a\mathbf{a} + b\mathbf{b} + g\mathbf{c}$$

$$\mathbf{x}' = a\mathbf{a}' + b\mathbf{b}' + g\mathbf{c}'$$

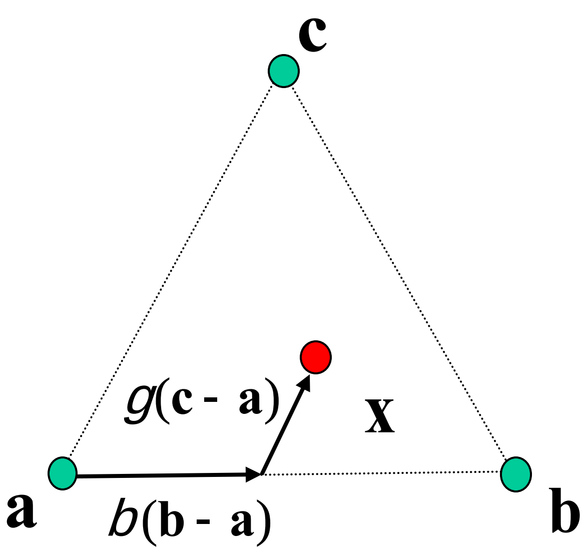
$$a + b + g = 1$$

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

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

$0 \leq á \leq 1$ and $0 \leq â \leq 1$

Barycentric Co-ordinates



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

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

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

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

Three linear equations in 3 unknowns

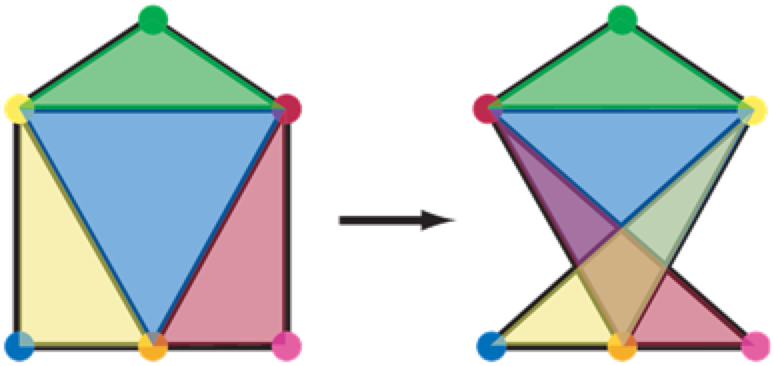
Interpolation using Triangles

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

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