# Digital Image Processing (CSE/ECE 478)

# Lecture # 16: Image morphing and Chamfer Matching

Avinash Sharma

Center for Visual Information Technology (CVIT),

IIIT Hyderabad

# Today's Lecture

- Image Morphing
- Chamfer Matching

# 500 years of female portrait

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

# 0 to 65 and back in a minute


https://www.youtube.com/watch?v=L0GKp-uvjO0

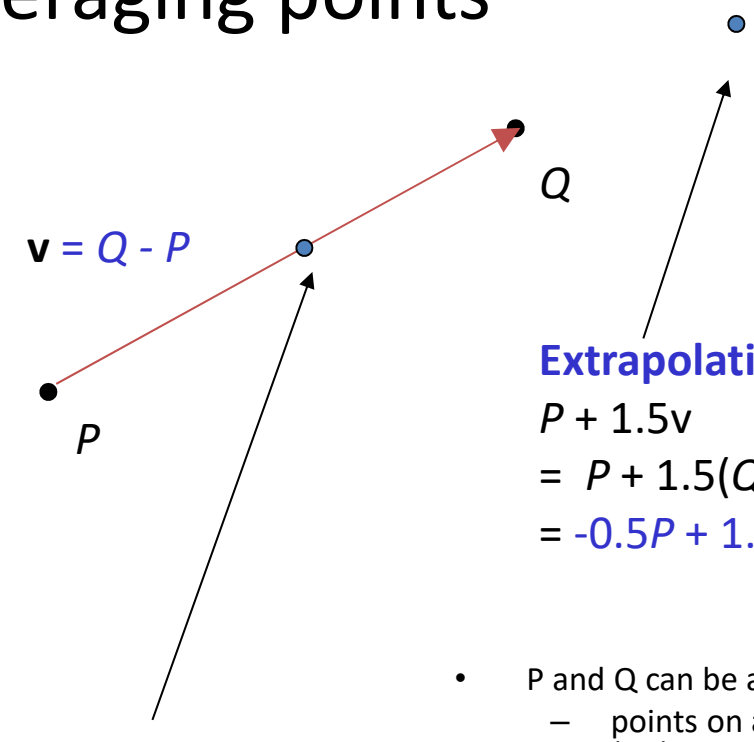
# Averaging vs Morphing



- 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: *(1-t)P + tQ*

*0<t<1*

*P* + 0.5v

= *P* + 0.5(*Q* – *P*)

= 0.5*P* + 0.5 *Q*

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

*P* + 1.5v

= *P* + 1.5(*Q* – *P*)

= -0.5*P* + 1.5 *Q (t=1.5)*

- 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:
- $Image_{halfway} = (1-t)*Image_1 + t*image_2$
- This is called **cross-dissolve** in film industry

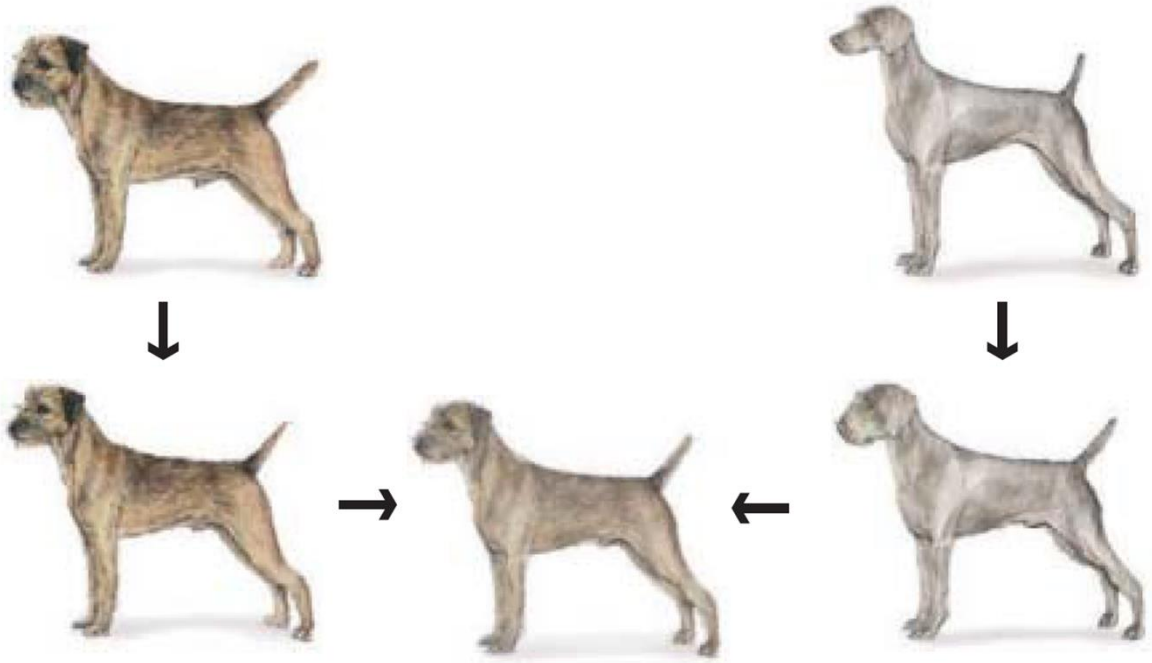
- But what if the images are not aligned?

# Idea-2: align, then cross dissolve

# 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



- *For every frame t,*

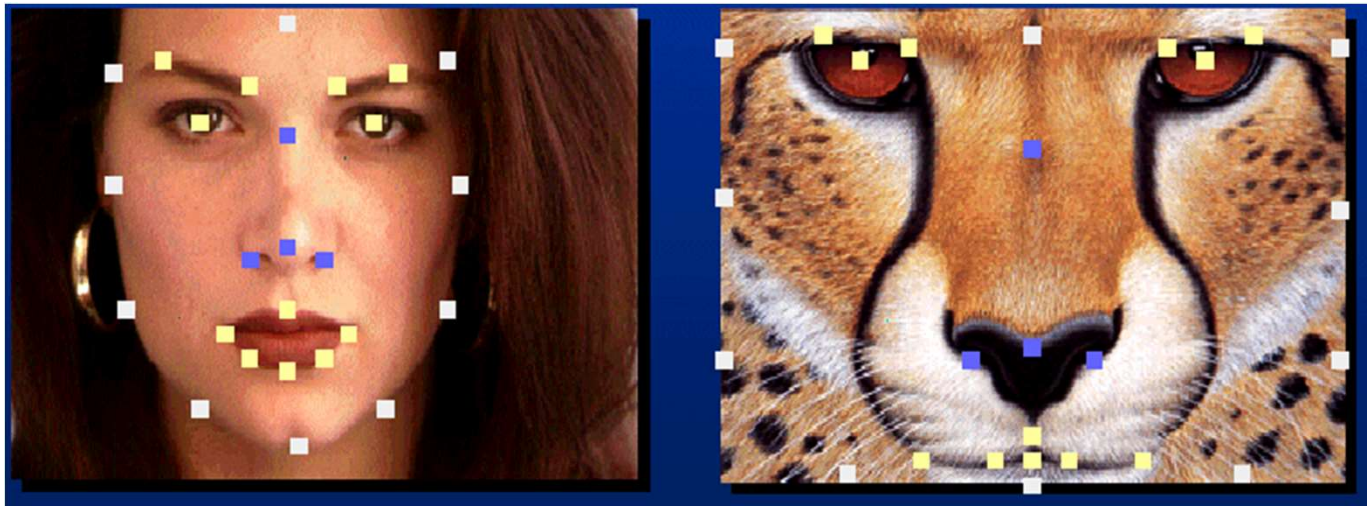
1. Find the average shape (the “mean dog”☺)
   - local warping
2. Find the average color
   - Cross-dissolve the warped images

# Warp specification

- How can we specify the warp?

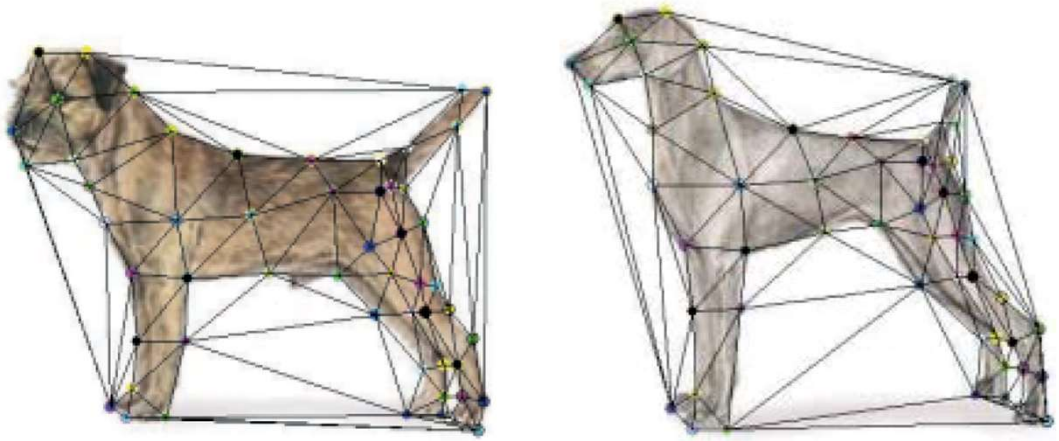
  Specify corresponding *points*

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

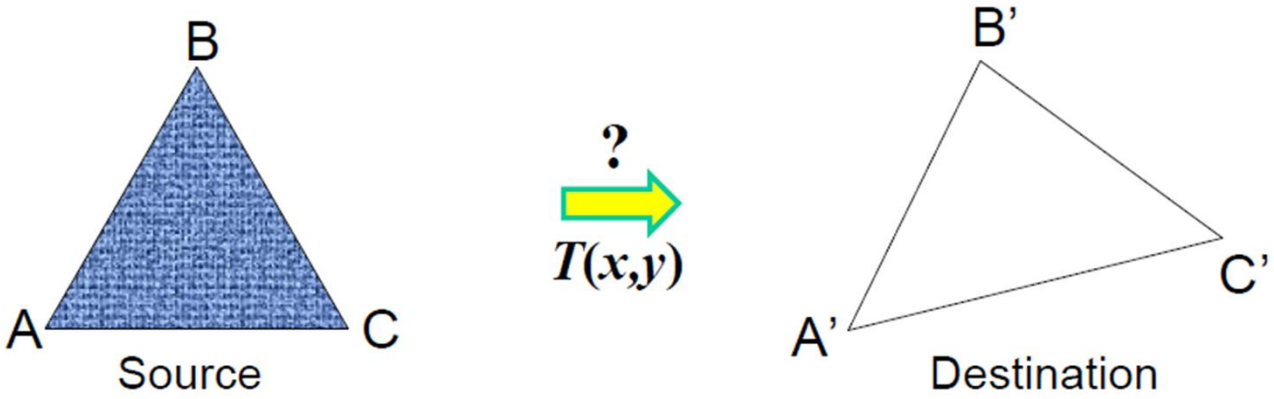


# Triangular Mesh

1. Input correspondences at key feature points
2. Define a triangular mesh over the points
   - Same mesh (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



# 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?

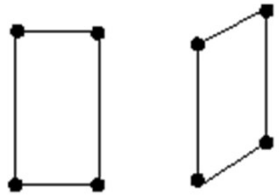
# Affine transformation

- Preserves collinearity (all points on a line, remain on a line)
  - Parallel lines remain parallel
  - Does not necessarily preserve angles between lines or distances between points (**any triangle can be transformed into another using affine transformation**)
  - Preserve ratios of distances between points lying on a straight line

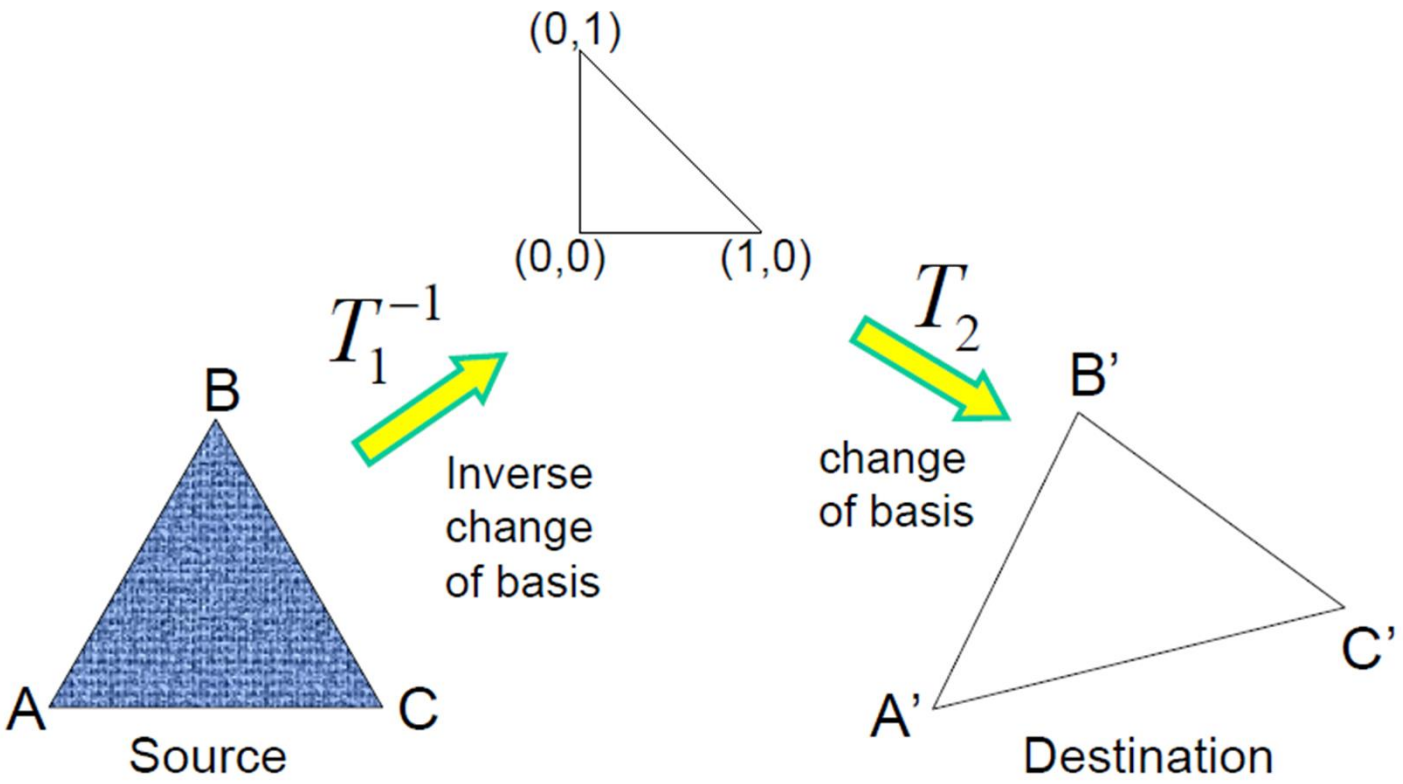
- Desired transformation as combination of simpler ones

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

$\boldsymbol{T}$



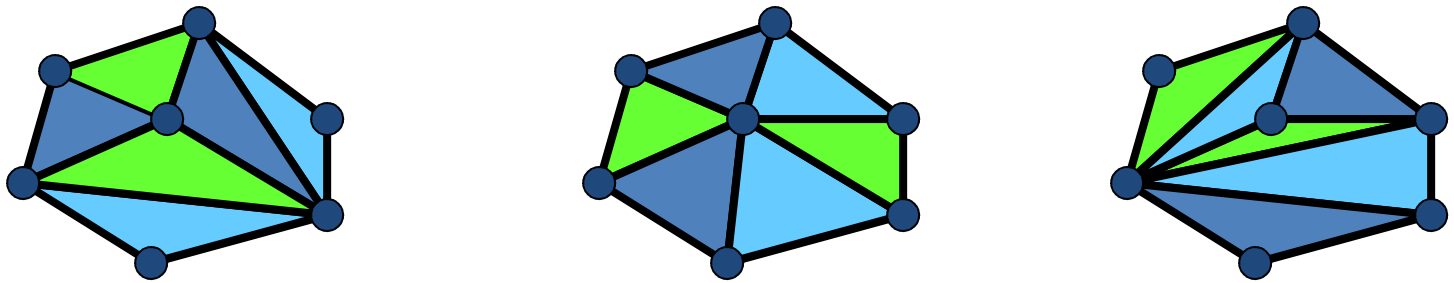
# Warping Triangles



# Triangulations

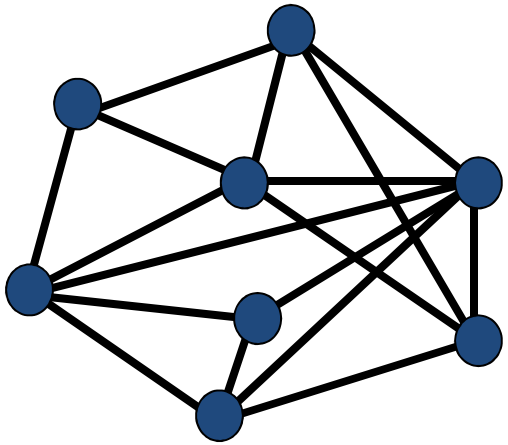
•A *triangulation* of set of points in the plane is a *partition* of the convex hull to triangles whose vertices are the points, and do not contain other points.

•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.



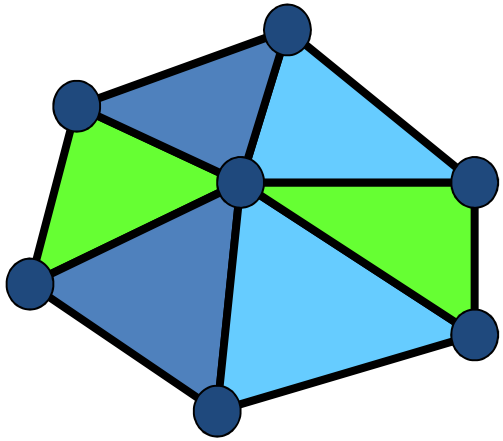
# “Quality” Triangulations

• Let $\alpha(T_i) = (\alpha_{i1}, \alpha_{i2}, .., \alpha_{i3})$ be the vector of angles in the triangulation $T$ in increasing order:

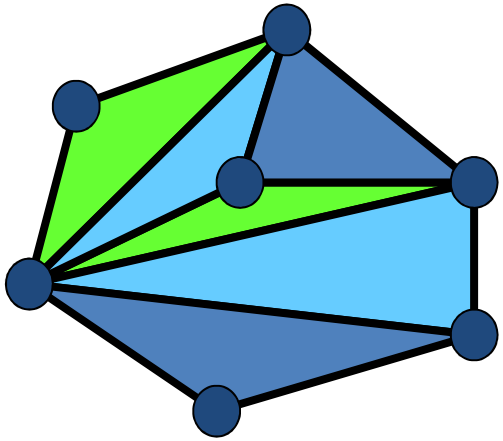
• A triangulation $T_1$ is “better” than $T_2$ if the smallest angle of $T_1$ is larger than the smallest angle of $T_2$

• Delaunay triangulation is the “best” (maximizes the smallest angles)

good

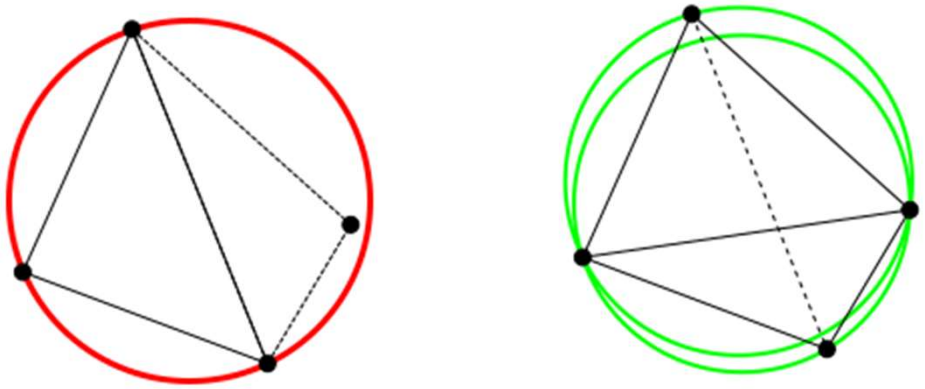


bad

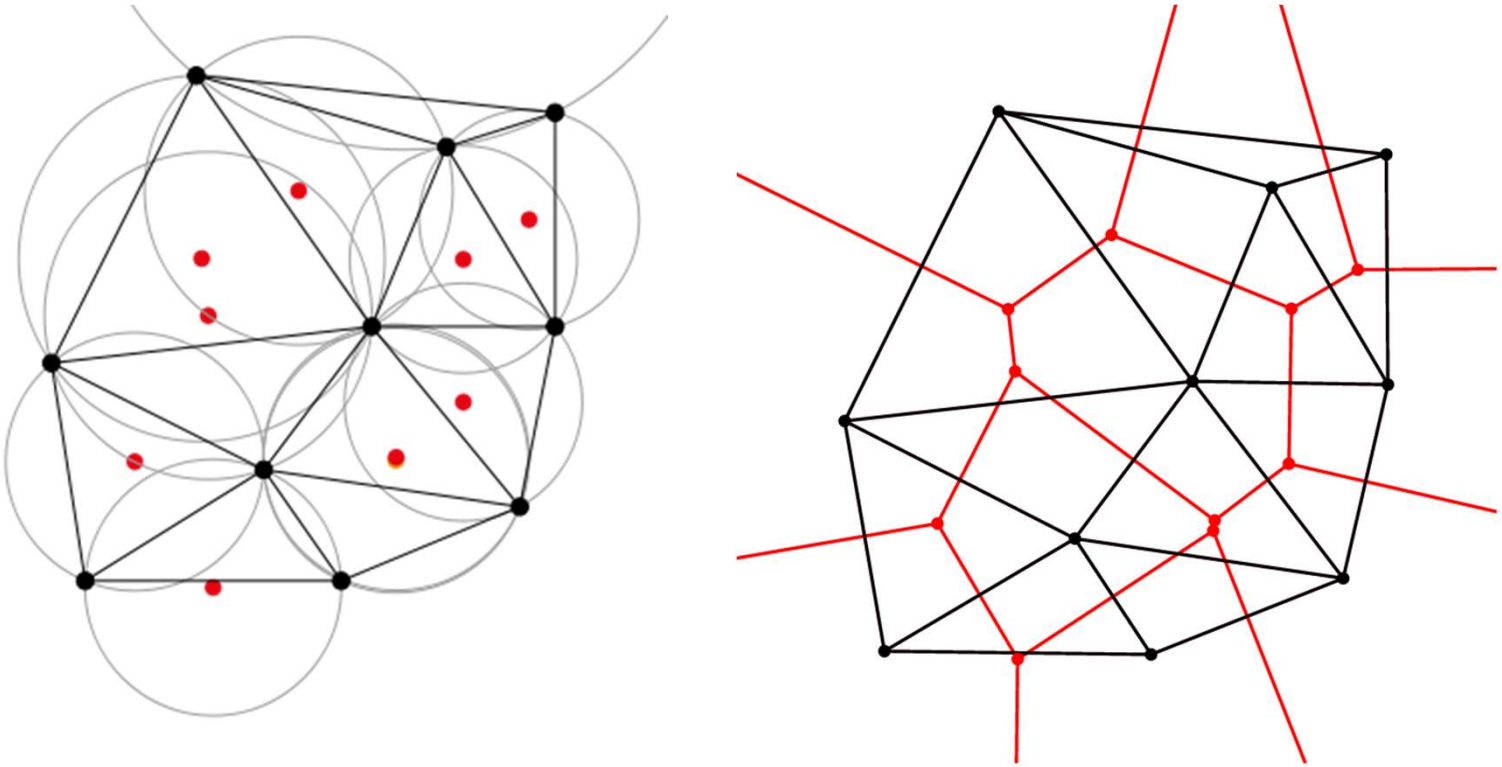


# Delaunay triangulation

A Delaunay triangulation for a given set P of discrete points in a plane is a triangulation DT(P) such that no point in P is inside the circumcircle of any triangle in DT(P).



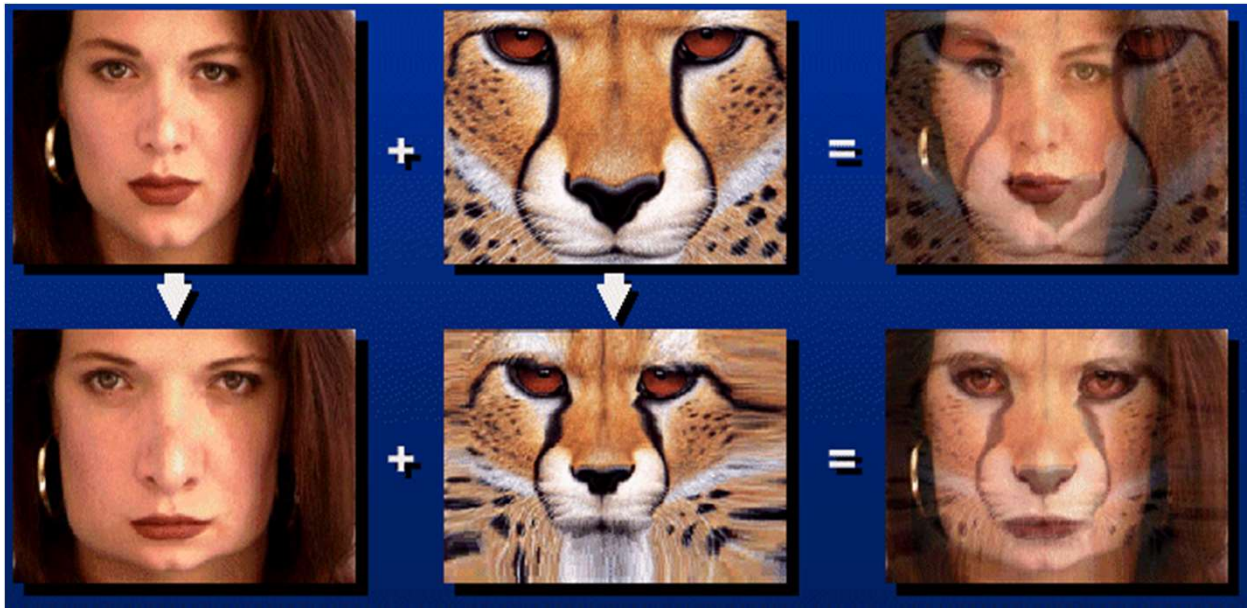
# Delaunay triangulation



# Image Morphing

How do we create a morphing sequence?

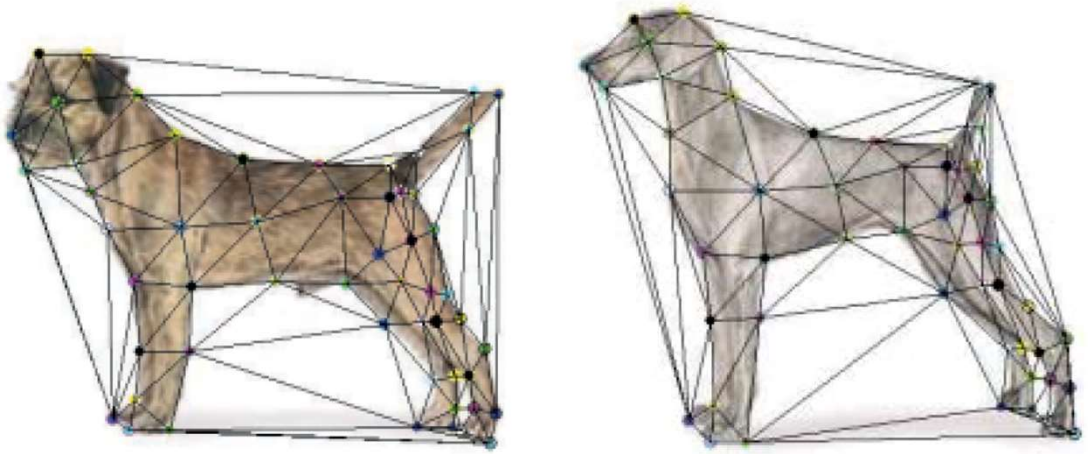
1. Create an intermediate shape (by interpolation)
2. Warp both images towards it
3. Cross-dissolve the colors in the newly warped images



# 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)*p1+t*p0 for corresponding features p0 and p1



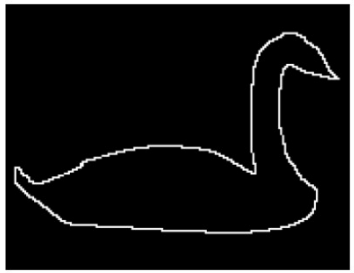
# Summary of Morphing

1. Define corresponding points
2. Define triangulation on points
   - Use same triangulation for both images
3. For each t in 0: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

# Today's Lecture

- Image Morphing
- Chamfer Matching

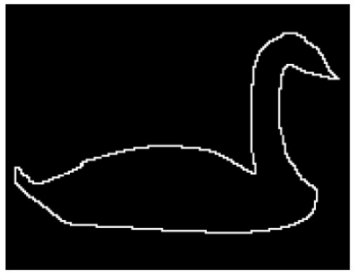
# Contour based shape matching



Template shape

Query Image

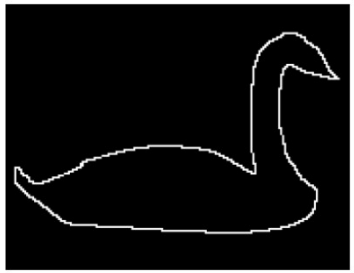
# Contour based shape matching



Template shape

Query Image

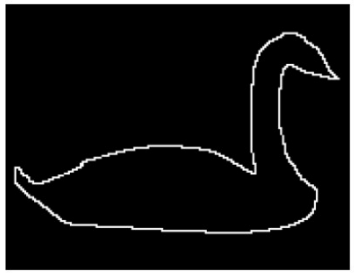
# Contour based shape matching



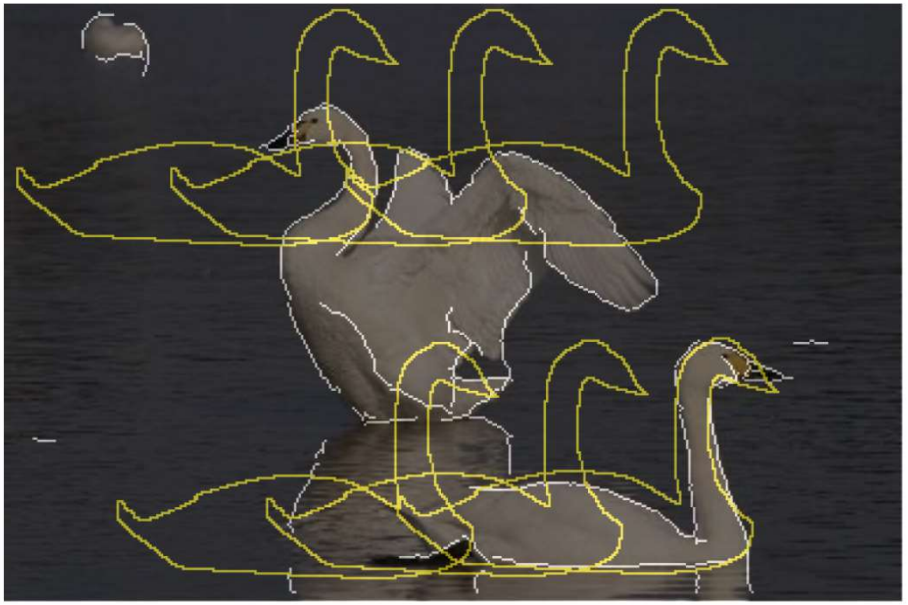
Template shape

Query Image

# Contour based shape matching

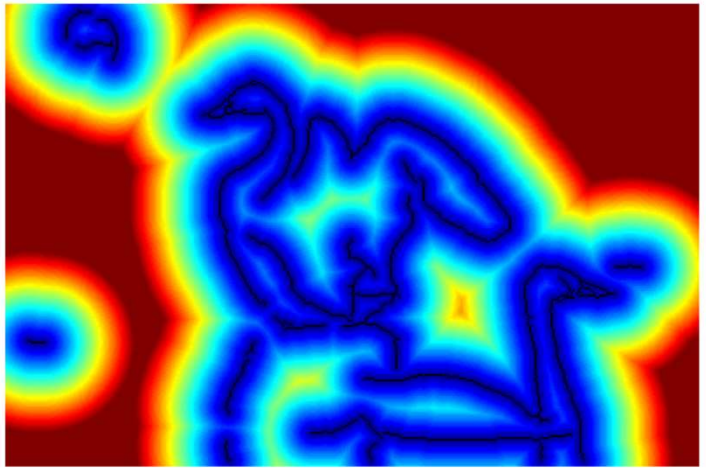


Template shape

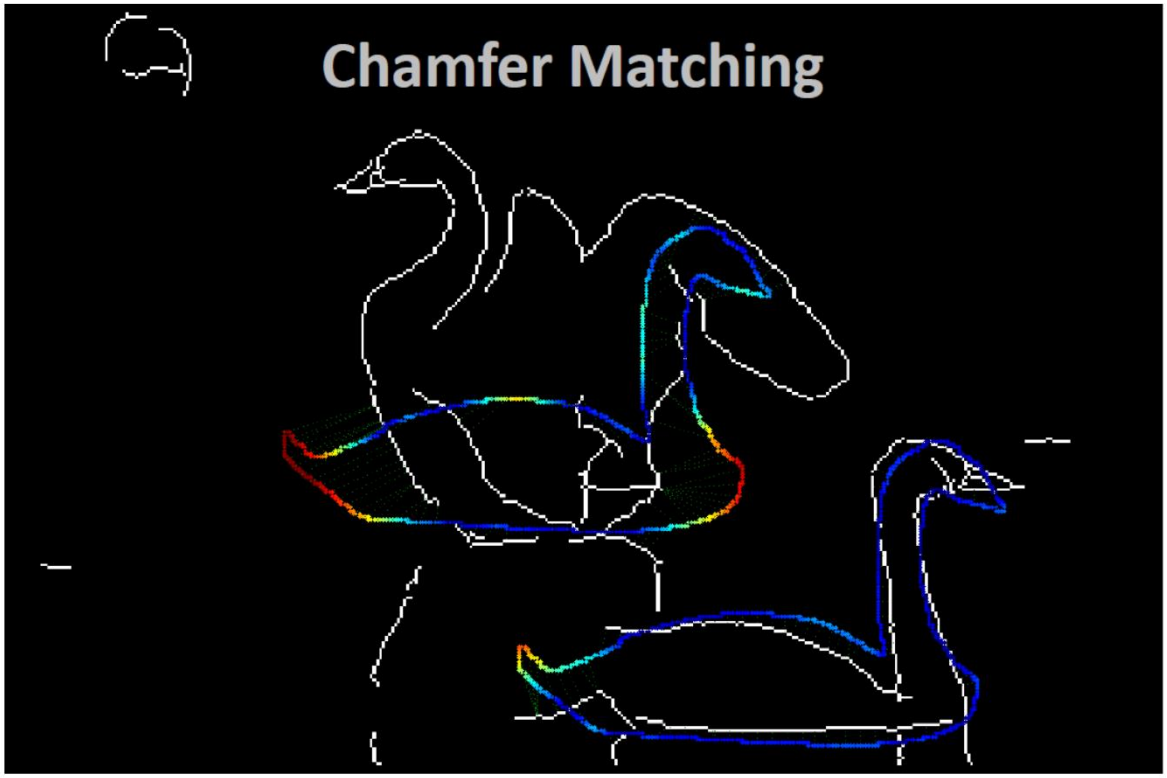


Query Image

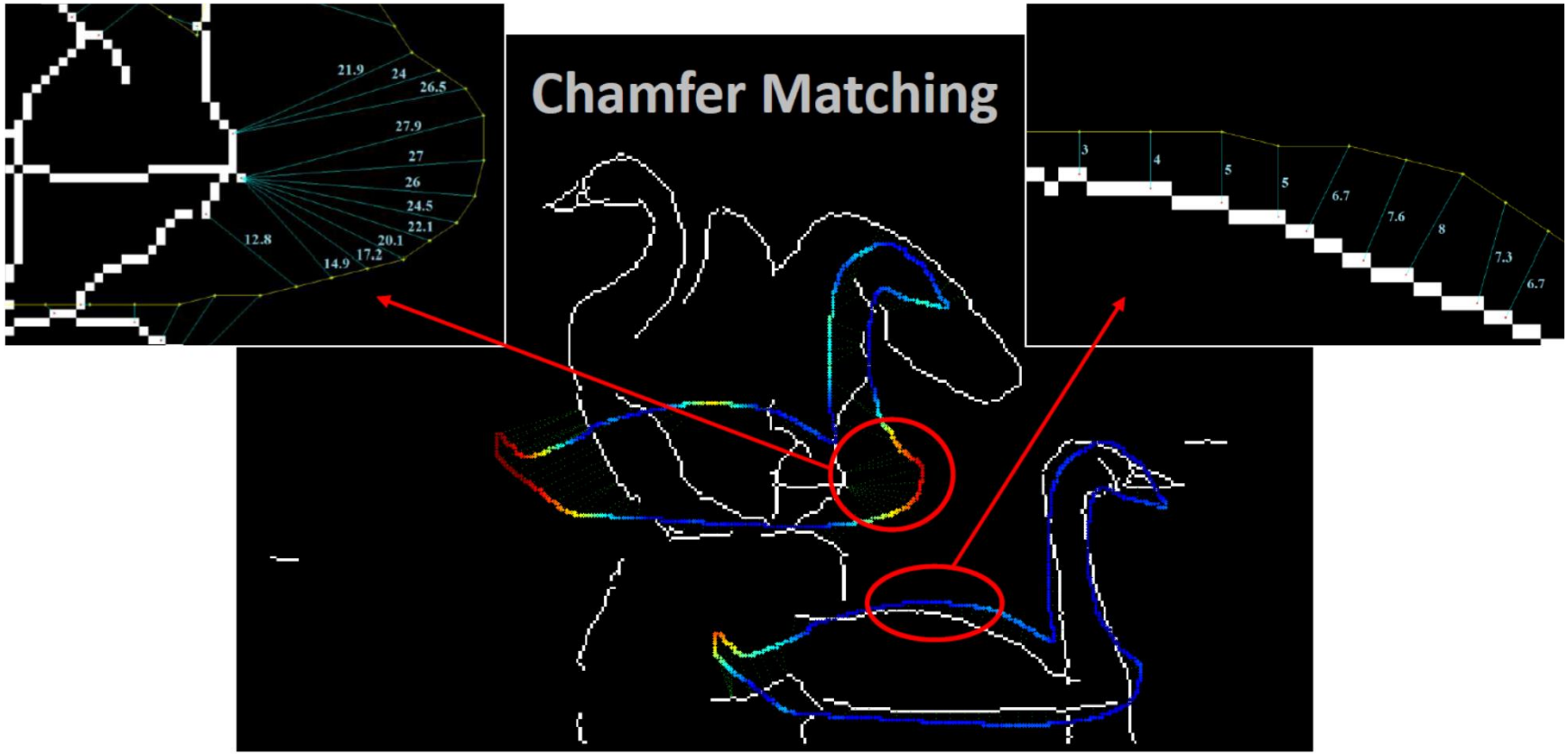
# Distance Transform



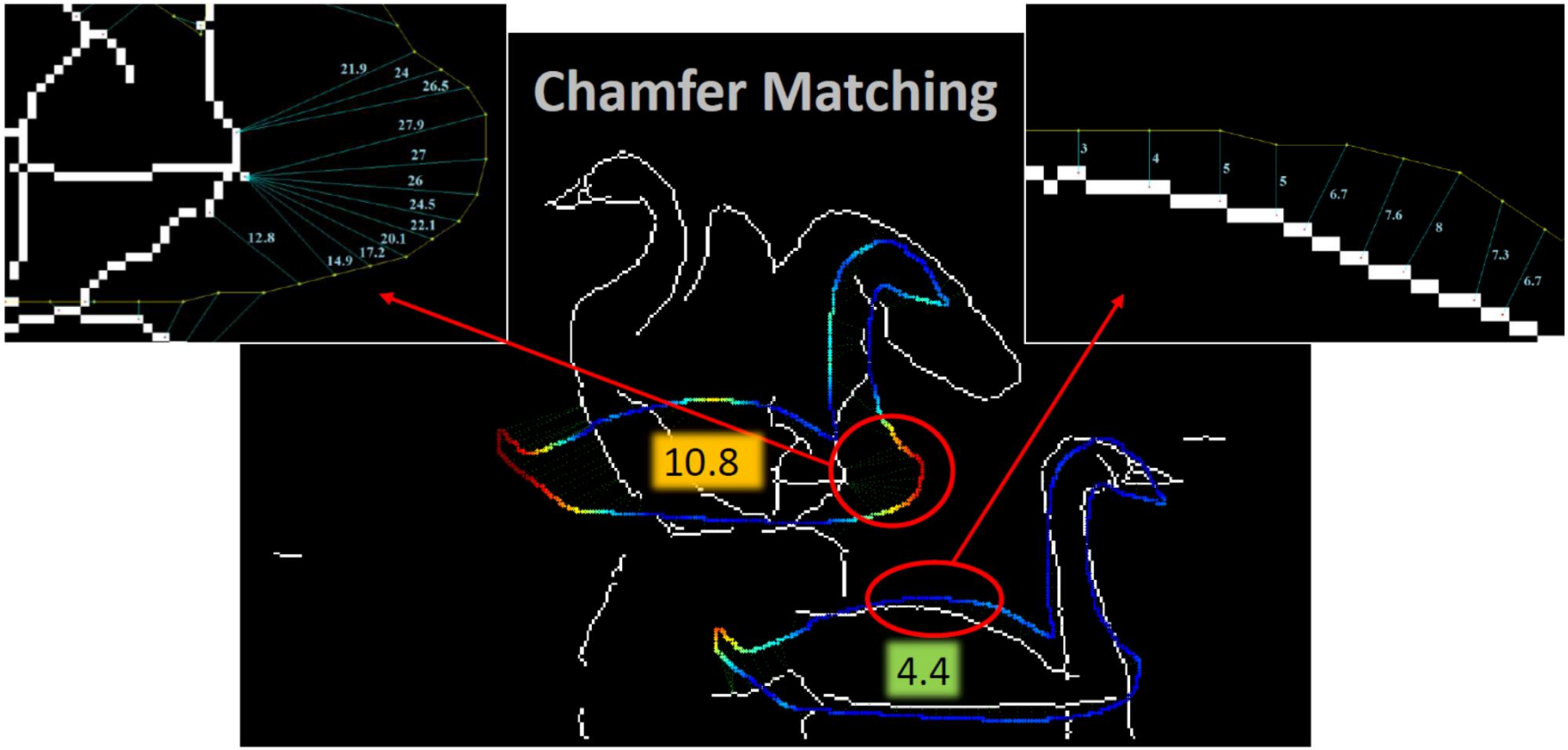
# Chamfer Matching



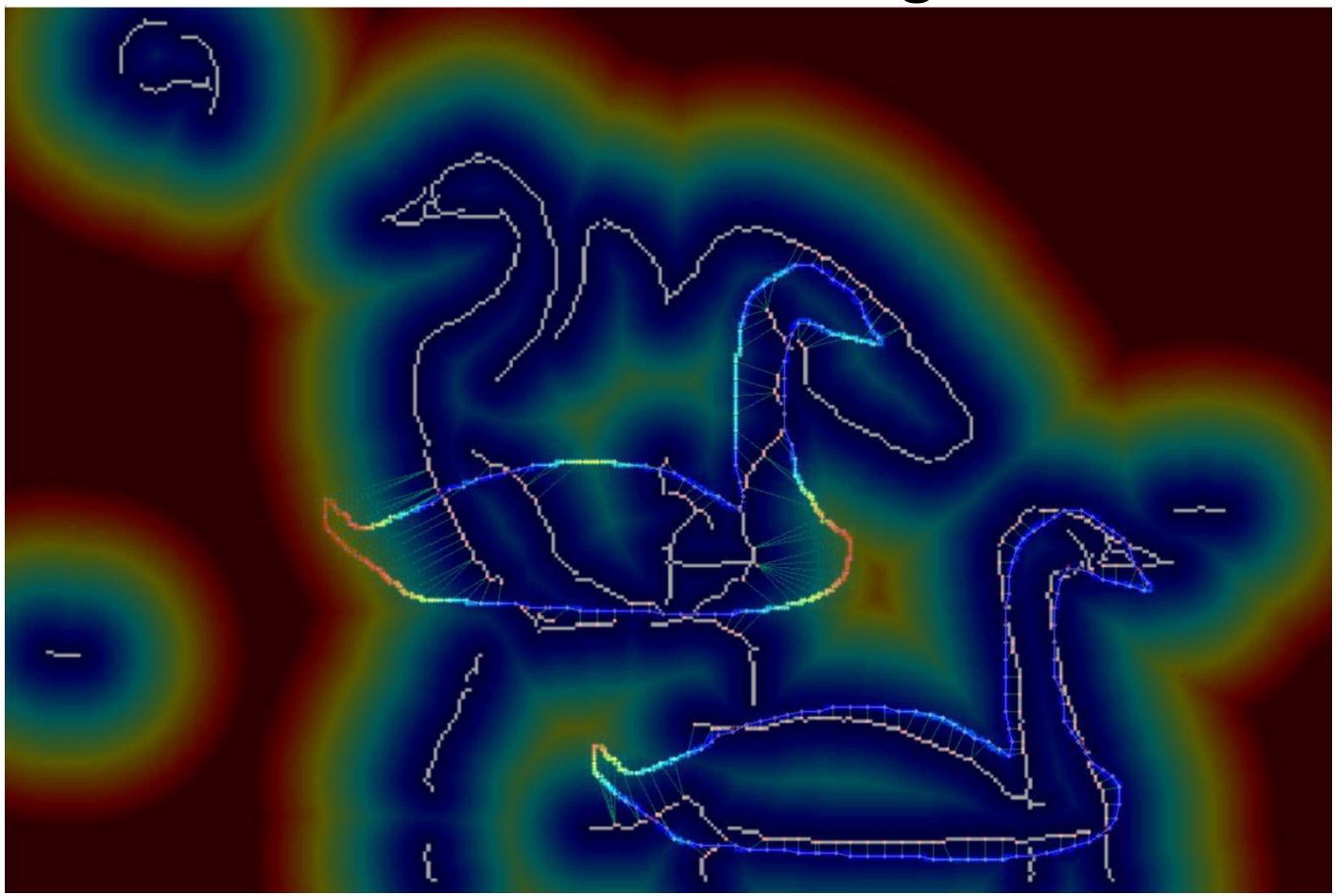
# Chamfer Matching



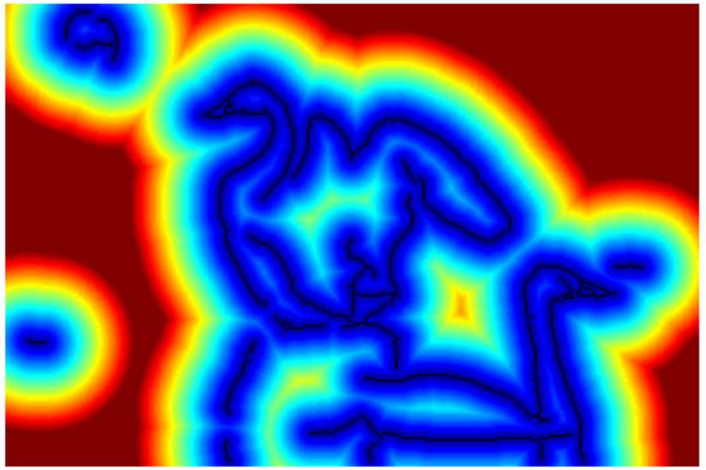
# Chamfer Matching



# Chamfer Matching

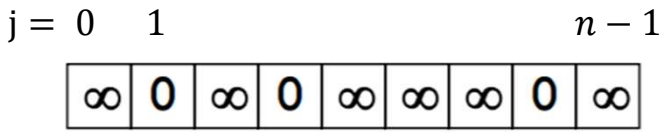


# How to efficiently compute DT?

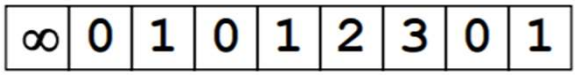


# Distance Transform – 1D case

- 1D case, L 1 norm: | x 1 – y 1| + | x 2 - y 2|
  - Two passes:
    - Find closest point on left
    - Find closest on right if closer than one on left
    - O(n) algorithm

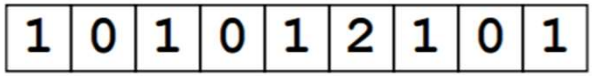



1. Initialize



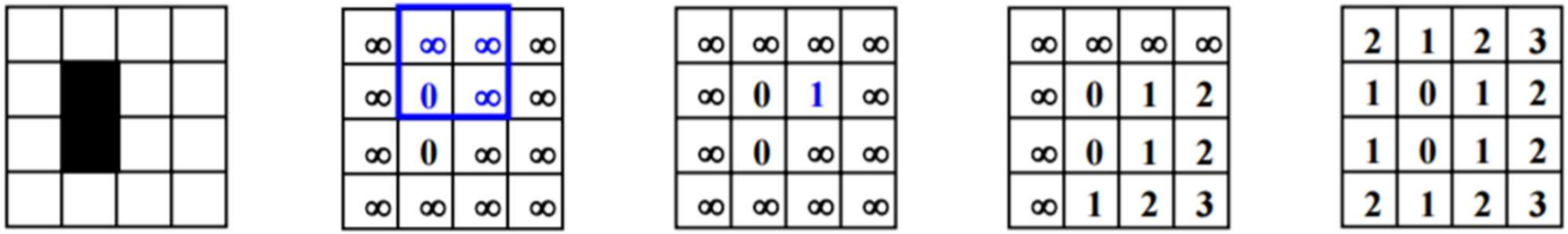

2. Forward pass (j=1 to n-1) D[j] ← min(D[j],D[j-1]+1)

3. Backward pass (j=n-2 to 0) D[j] ← min(D[j],D[j+1]+1)



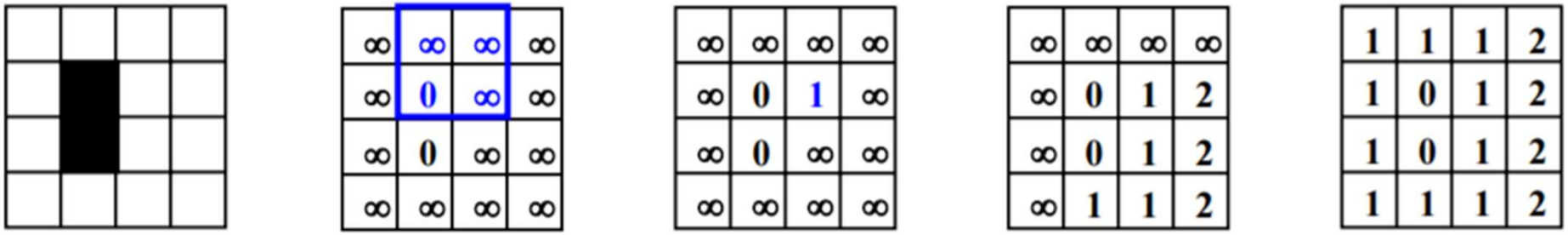
# Distance Transform – 2D

- 2D case similar to 1D
  - Initialize
  - Forward Pass (left and top)
  - Backward Pass (right and below)



# Distance Transform – 2D

- Similar extension to 8 neighbours (Chessboard distance)
  - Initialize
  - Forward Pass (left, top, top-left)
  - Backward Pass (right, below, right-below)

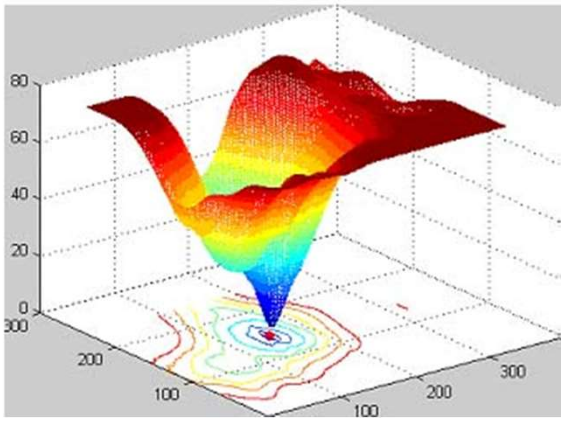



What about Euclidian Distance??

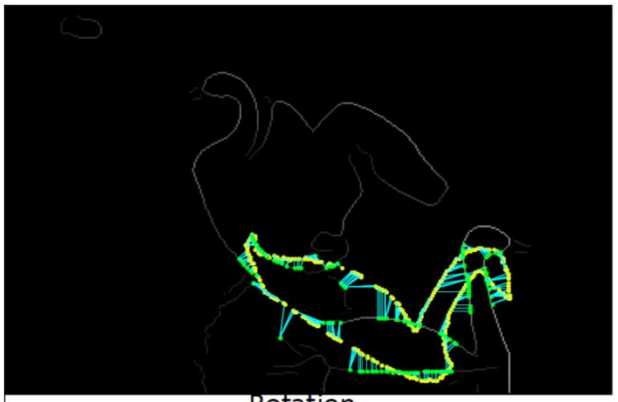
Courtesy: Dan Huttenlocher

# Chamfer Matching Overview

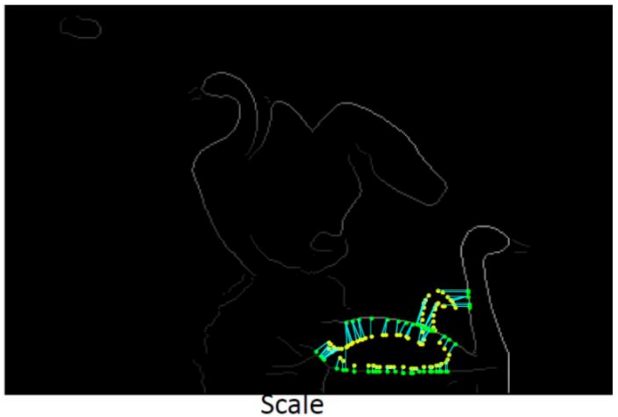
- Detect edges in query image
- Slide template over query image edge map
- Find closest edge pixel in image for each shifted template pixel
- At each location, compute average distance from each pixel in template to closest edge in image
- Lowest cost is the best match



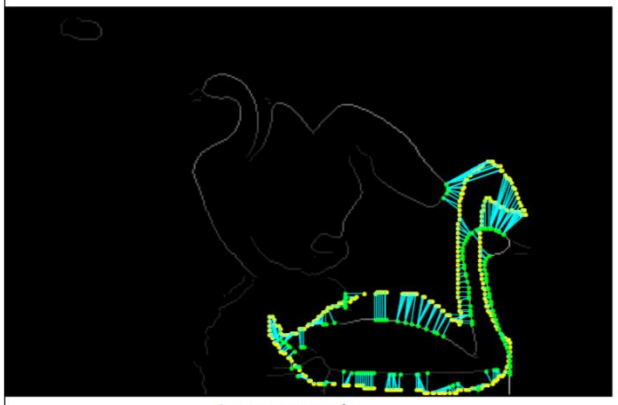
# Limitations



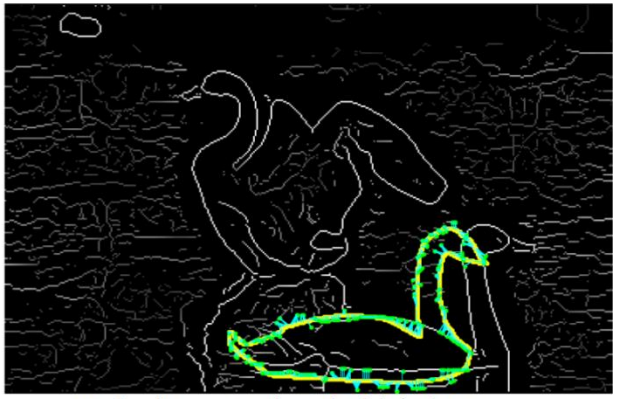
Rotation



Scale

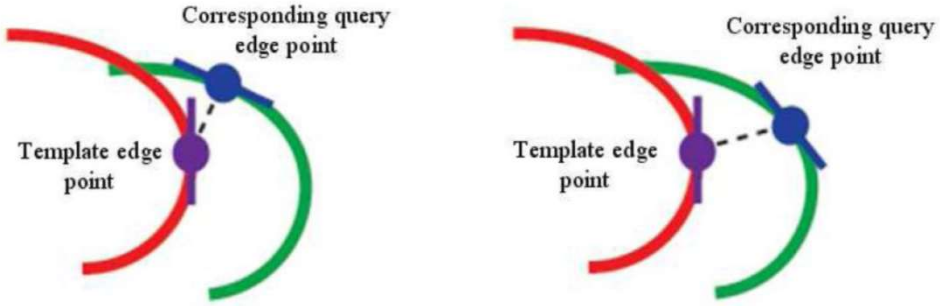


Aspect ratio



Bad edge map threshold / Clutter

# Possible improvement



# THANK YOU