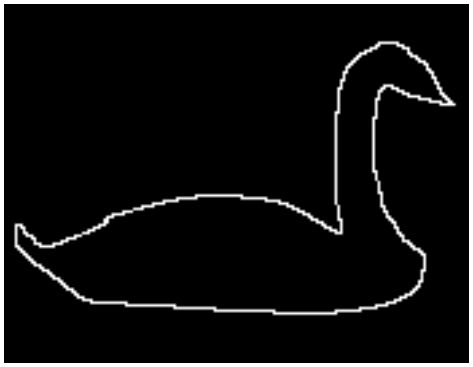
Chamfer Matching
(contour based shape matching)

Nicu Știurcă
12-5-2013

Contour based shape matching

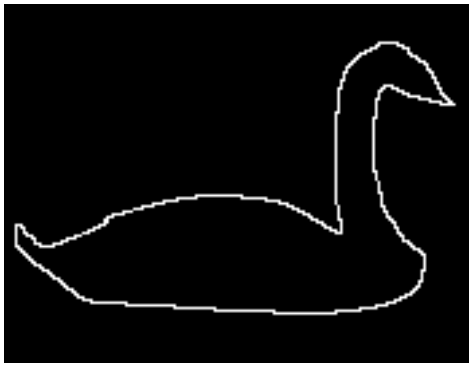


template shape

query image

How to find the template shape in the query image?

Contour based shape matching

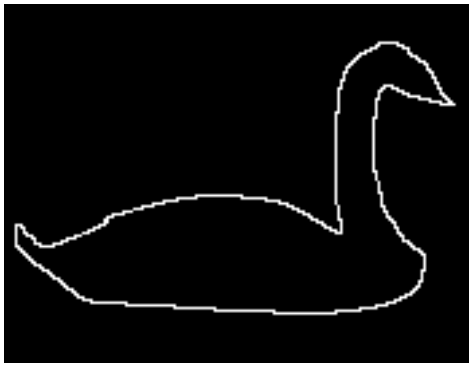


template shape

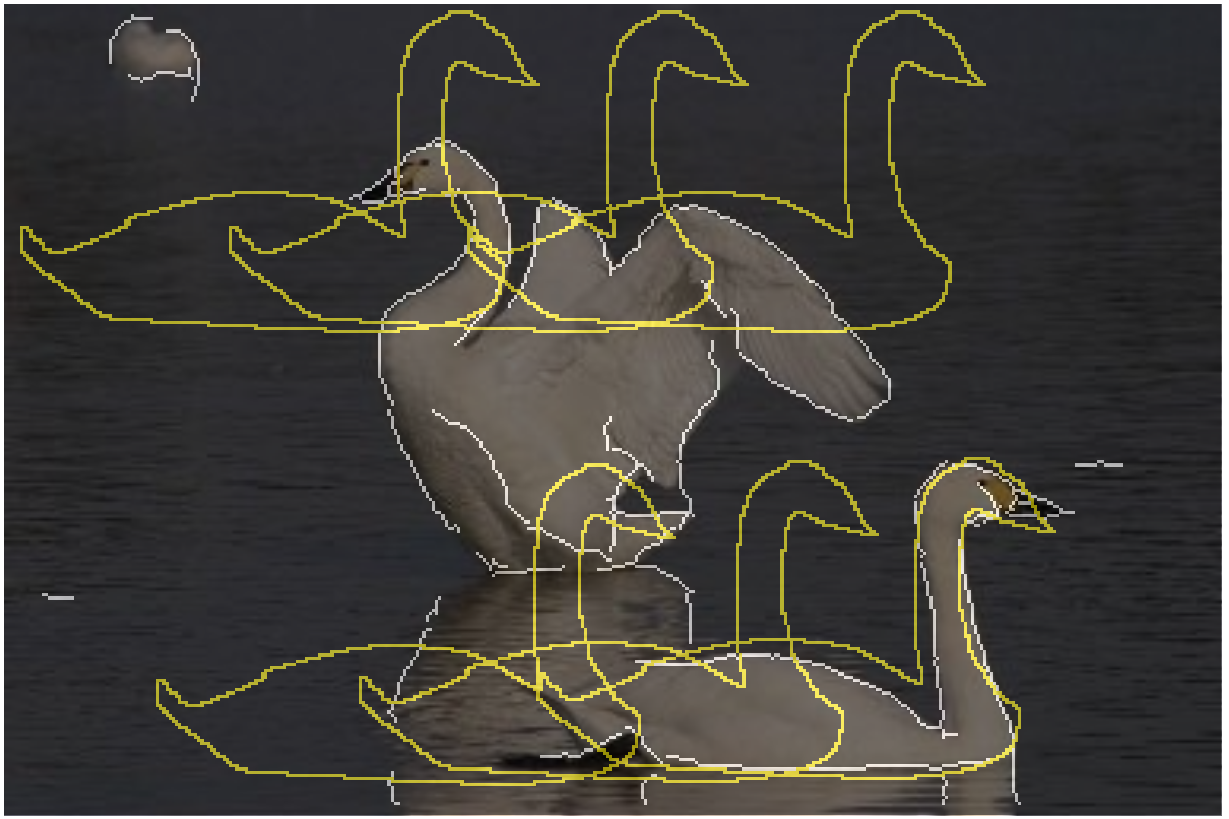
query image

Detect edges in query image, binary edge or edge with soft-magnitude

Contour based shape matching



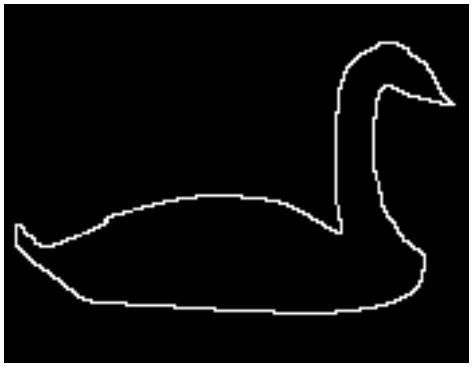
template shape



query image

Slide template over query image edge map

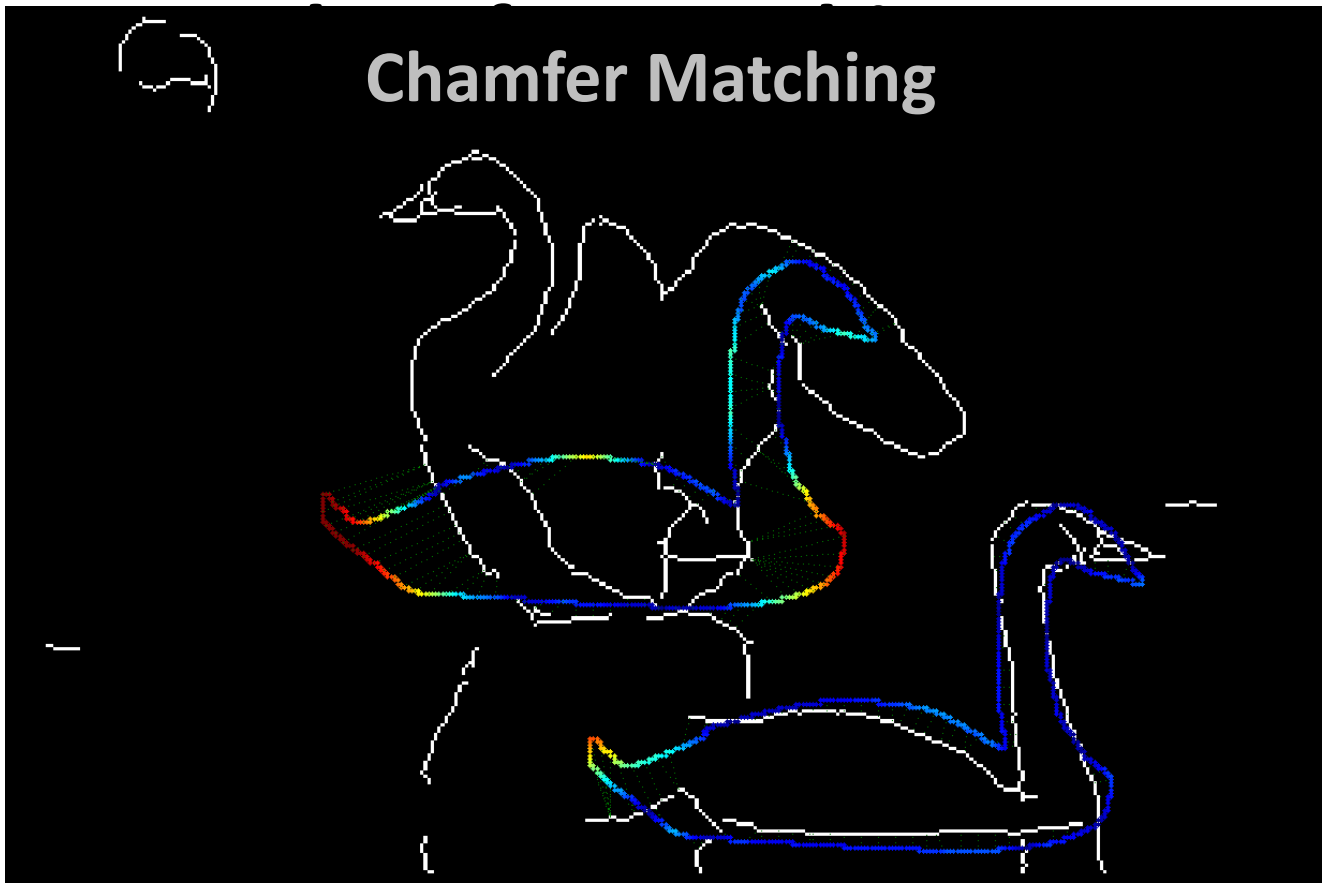
Contour based shape matching



template shape

query image

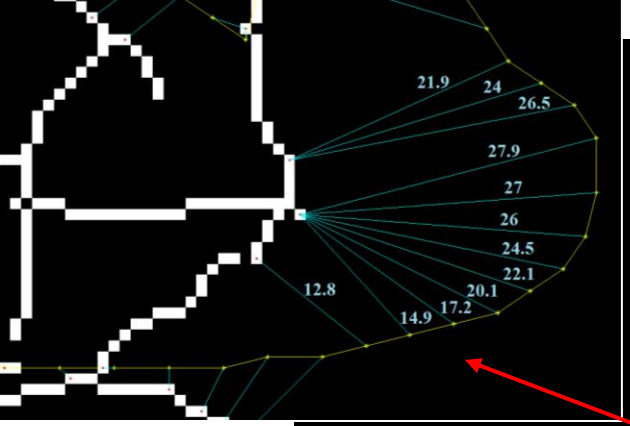
At each location, compute distance from each pixel *p* in template to closest edge in image *q*. (red is large distance, blue is low)
Location with the lowest *average* cost match wins
(over template pixels)

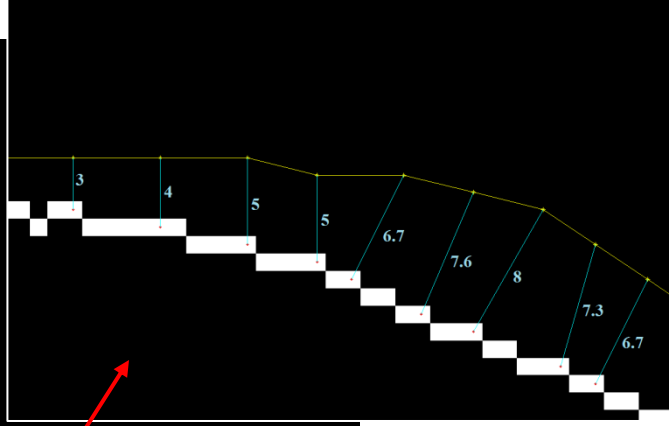


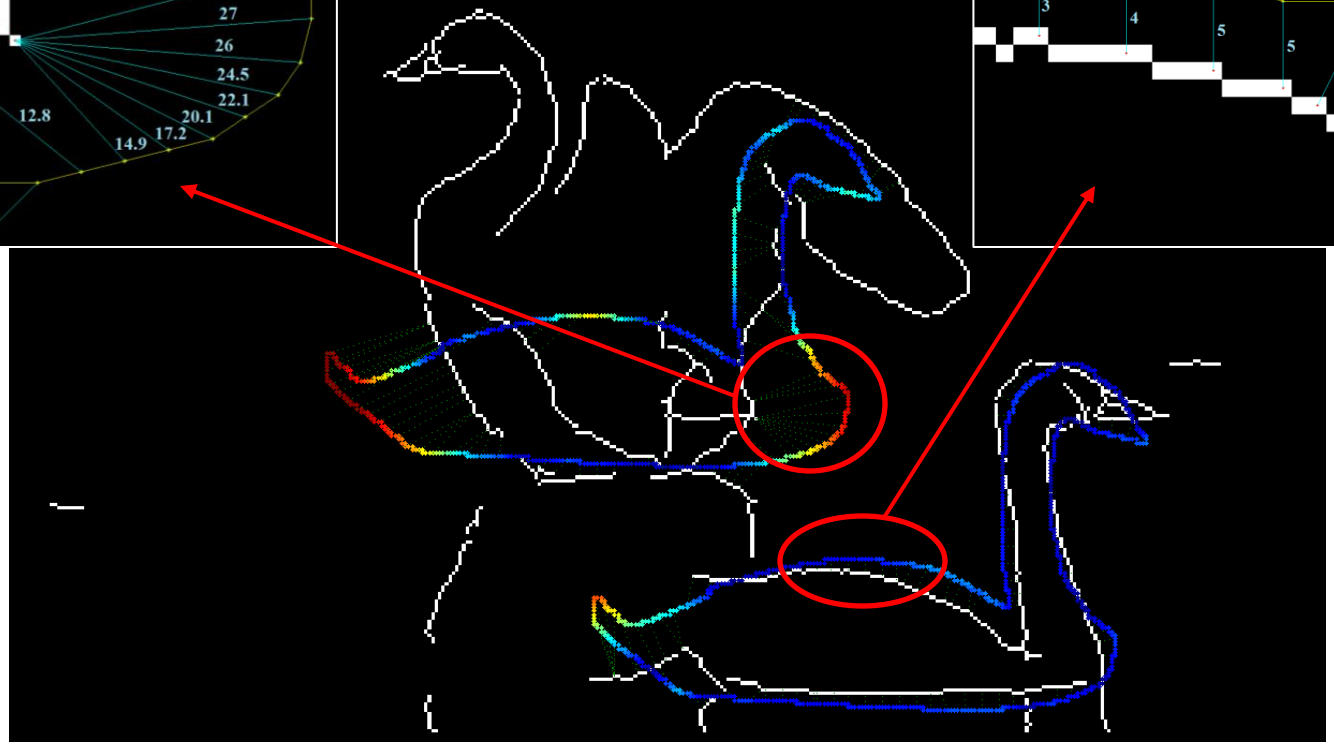
1. Slide template *T* by *(u,v)* over query image edge map *E*

$$T(u,v) = \{(x+u, y+v) \mid (x,y) \in T\}$$

Chamfer Matching







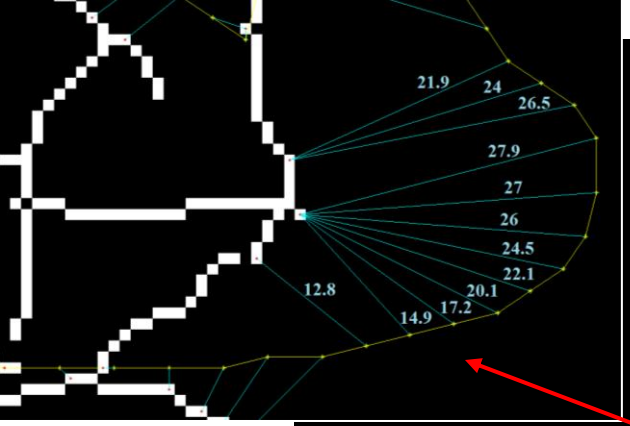
1. Slide template *T* by *(u,v)* over query image edge map *E*
2. **Matching cost of each pixel *p* in the shifted template *T(u,v)* is its shortest distance to any edge pixel *q* in the edge map *E***

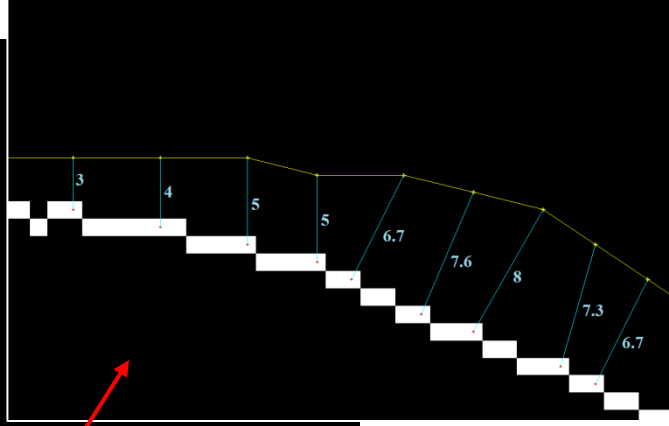
$$c(p) = \min_{q \in E} \|p - q\|_2$$

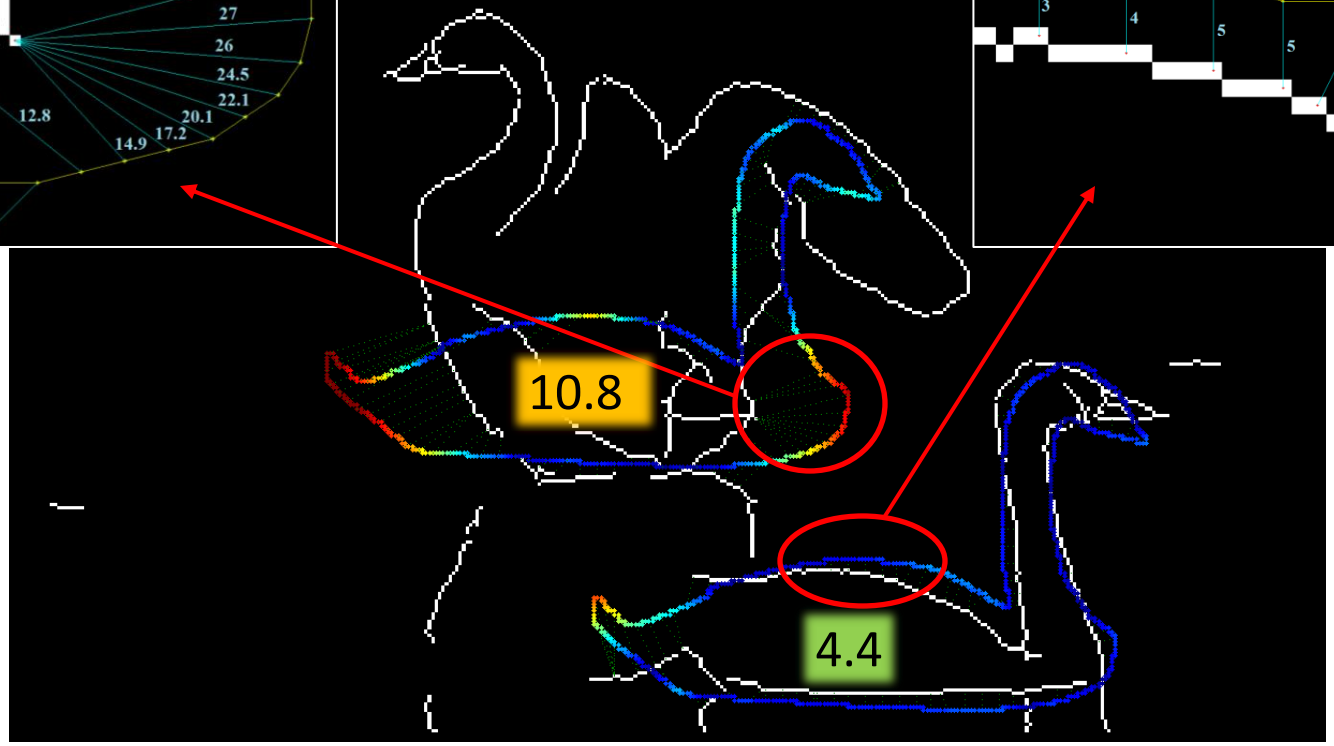
Brute force computation takes $\mathcal{O}(n \| T \|)$

Using distance transform, it takes $\mathcal{O}(n) + \mathcal{O}(\| T \|)$

Chamfer Matching







1. Slide template *T* by *(u,v)* over query image edge map *E*
2. Matching cost of each pixel *p* in the shifted template *T(u,v)*
   is its shortest distance to any edge pixel *q* in the edge map *E*
3. **Total cost of the shifted template is the average cost of each shifted template pixel**

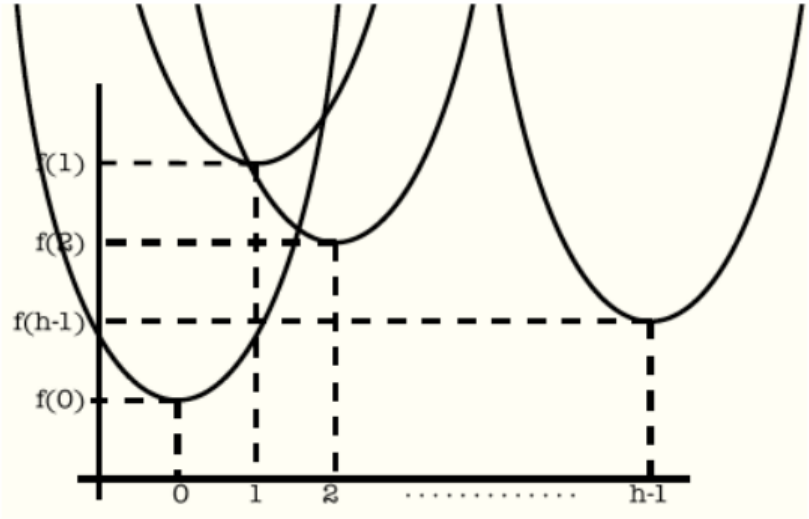
$$Cost(u,v) = \frac{1}{\|T\|} \sum_{p \in T(u,v)} \min_{q \in E} \|p - q\|_2$$

Recall: generalized distance transform

Our target: $$c(p) = \min_{q \in E} \|p - q\|_2$$

We can compute the following for all p in linear time:

$$D_f(p) = \min_{p \in G} \left(\|p - q\|^2 + f(p) \right)$$

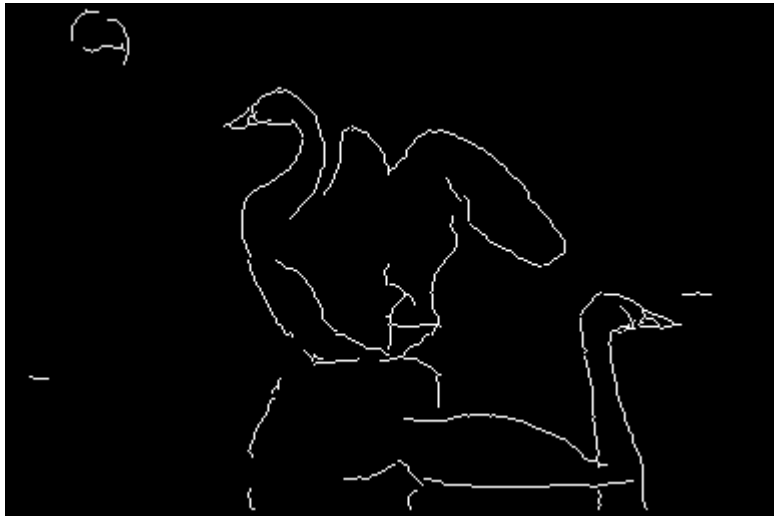


Recall: generalized distance transform

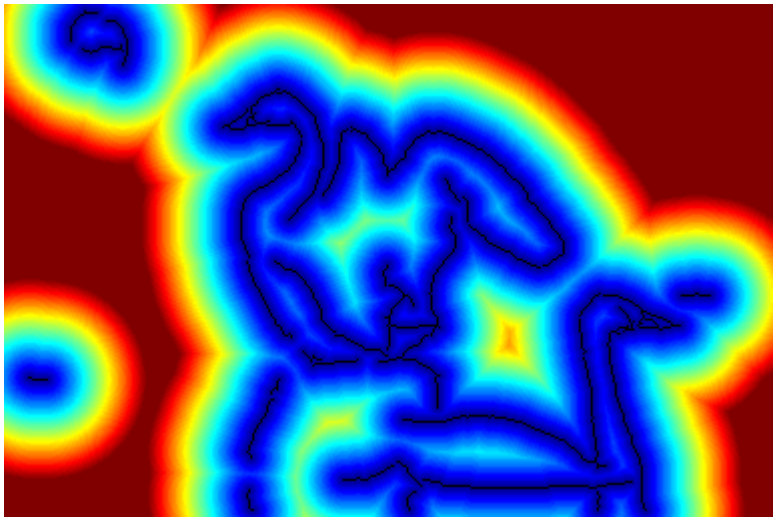
Algorithm $DT(f)$

1. $k \leftarrow 0$ (* Index of rightmost parabola in lower envelope *)
2. $v[0] \leftarrow 0$ (* Locations of parabolas in lower envelope *)
3. $z[0] \leftarrow -\infty$ (* Locations of boundaries between parabolas *)
4. $z[1] \leftarrow +\infty$
5. **for** $q = 1$ **to** $n - 1$ (* Compute lower envelope *)
6. $\quad s \leftarrow ((f(q) + q^2) - (f(v[k]) + v[k]^2))/(2q - 2v[k])$
7. $\quad$ **if** $s \leq z[k]$
8. $\quad\quad$ **then** $k \leftarrow k - 1$
9. $\quad\quad\quad$ **goto** 6
10. $\quad\quad$ **else** $k \leftarrow k + 1$
11. $\quad\quad\quad v[k] \leftarrow q$
12. $\quad\quad\quad z[k] \leftarrow s$
13. $\quad\quad\quad z[k+1] \leftarrow +\infty$
14. $k \leftarrow 0$
15. **for** $q = 0$ **to** $n - 1$ (* Fill in values of distance transform *)
16. $\quad$ **while** $z[k+1] < q$
17. $\quad\quad k \leftarrow k + 1$
18. $\quad \mathcal{D}_f(q) \leftarrow (q - v[k])^2 + f(v[k])$

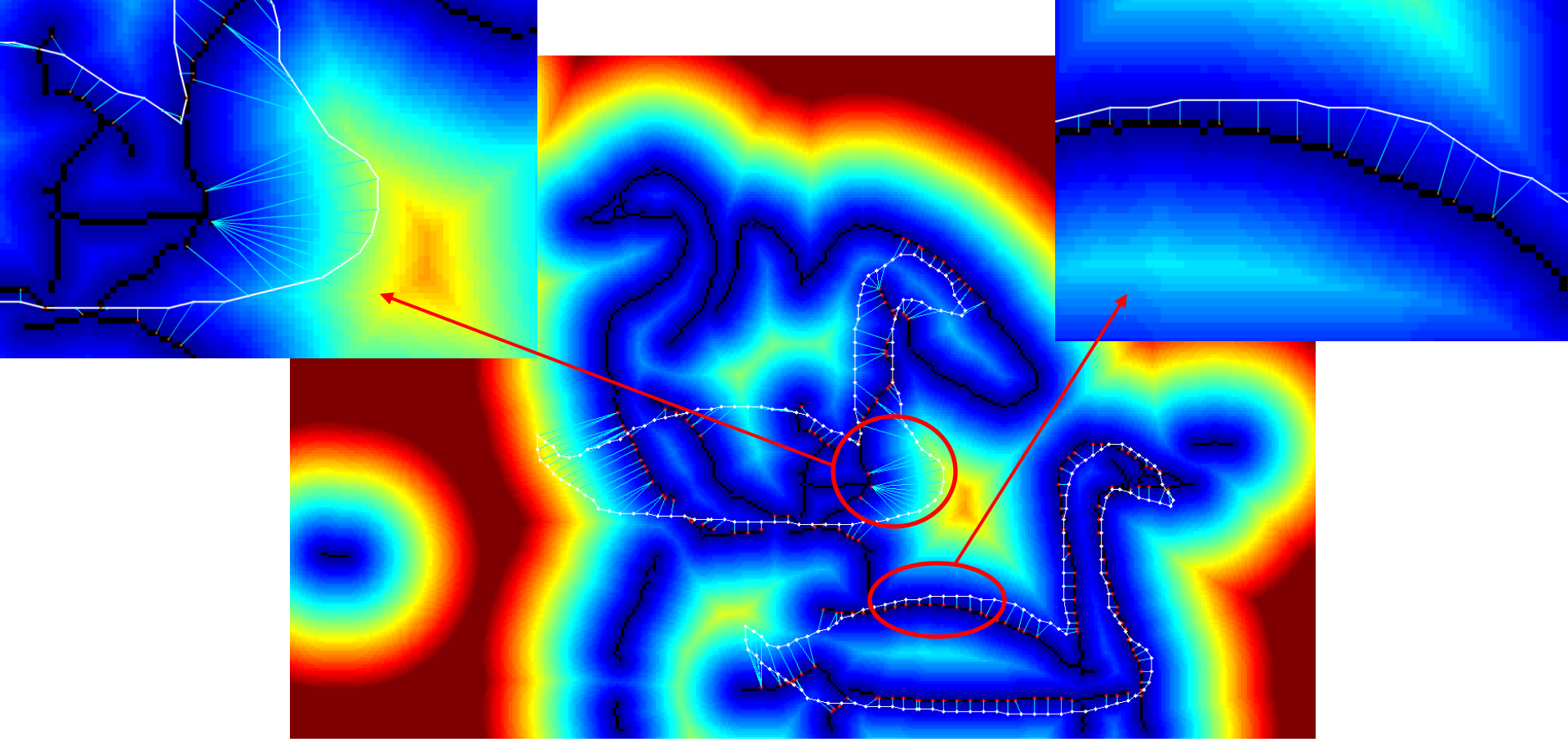
Recall: generalized distance transform



E



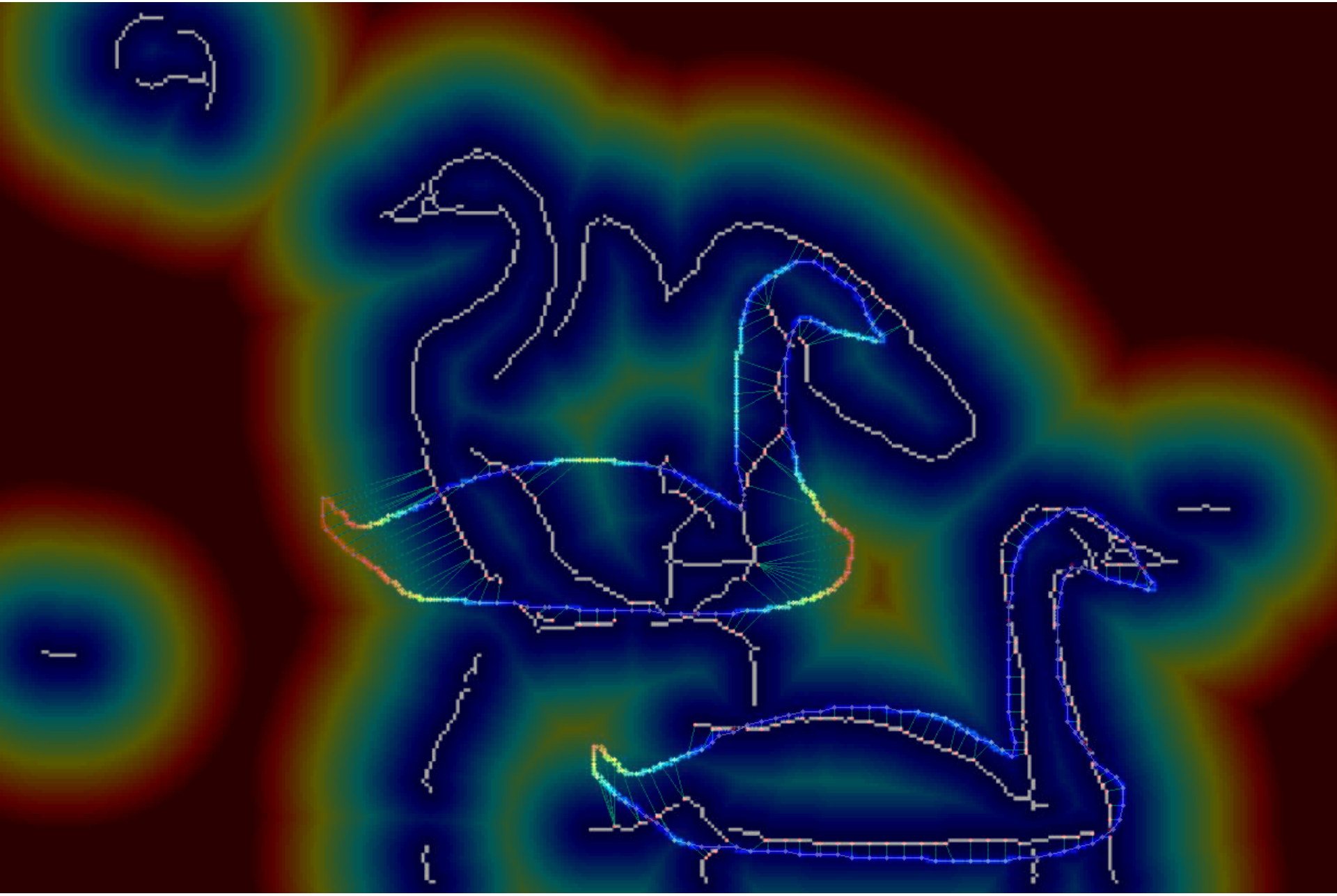
$$c(p) = \min_{q \in E} \|p - q\|_2$$



Now finding the cost of each point is just a look up!

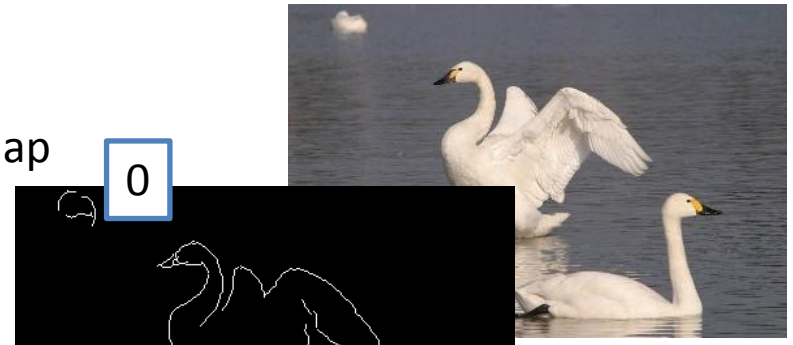
Evaluation time for each shift is just $O(\|T\|)$, and therefore the total running time is: (*n* is the number of pixels in the query image)

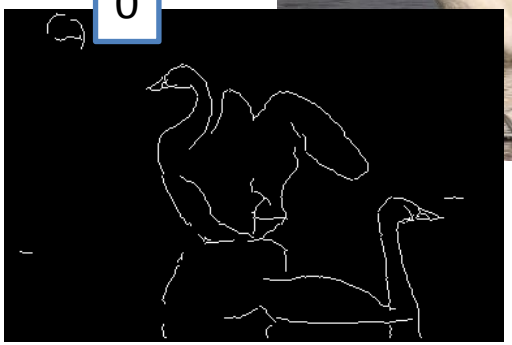
$$\mathcal{O}(n \| T \| \mathcal{O}(\min_{q \in E} \| p - q \|_2)) = \mathcal{O}(n) + \mathcal{O}(n \| T \|)$$

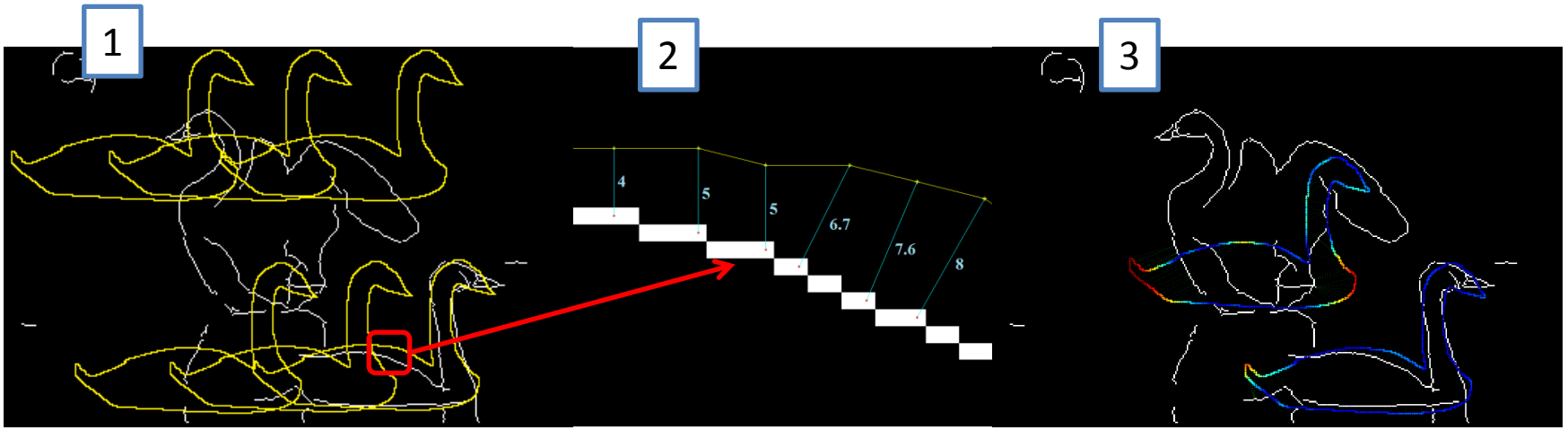


Chamfer Matching Review

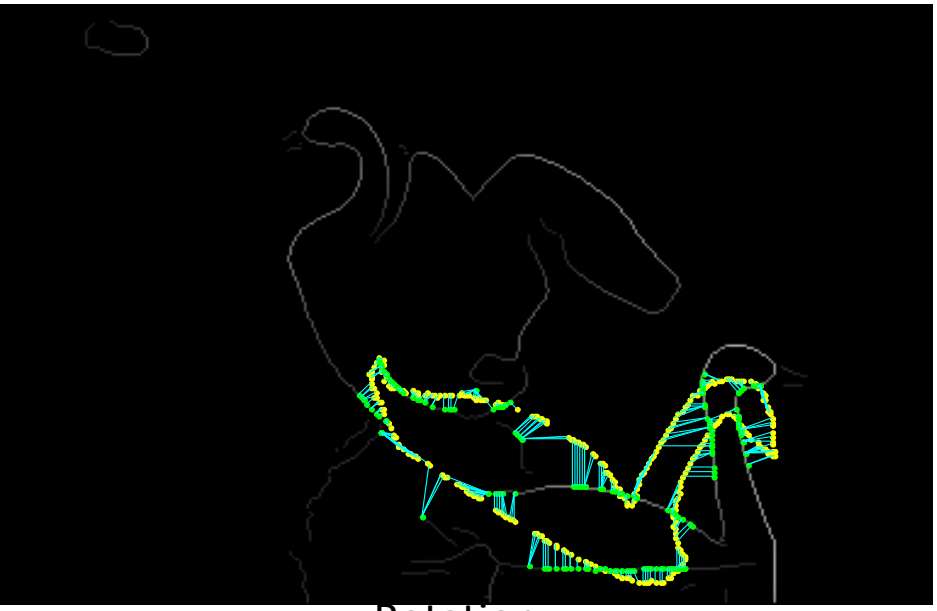
0. Detect edges in query image
1. Slide template over query image edge map
2. Find closest edge pixel in image for each shifted template pixel
3. At each location, compute average distance from each pixel in template to closest edge in image
4. Lowest cost match wins



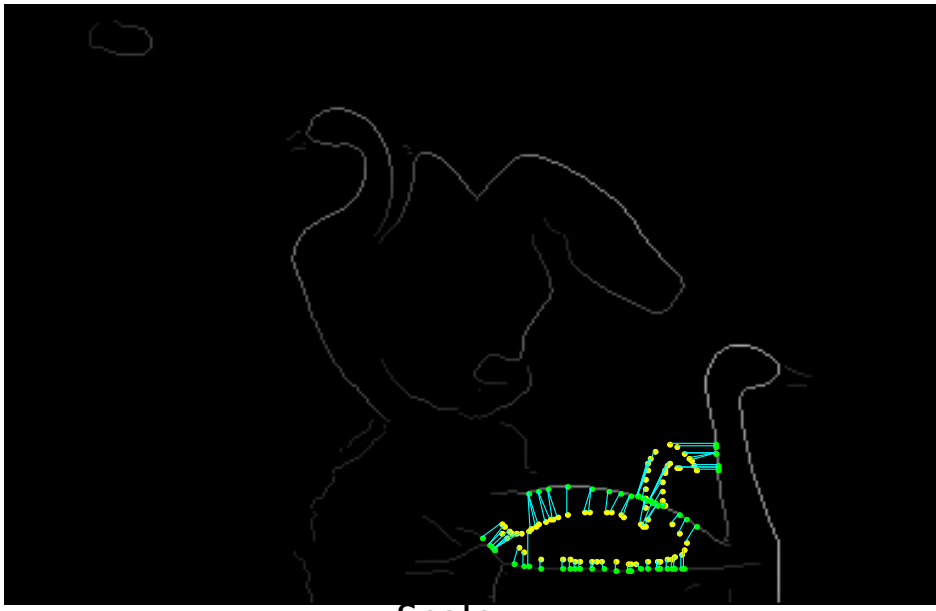




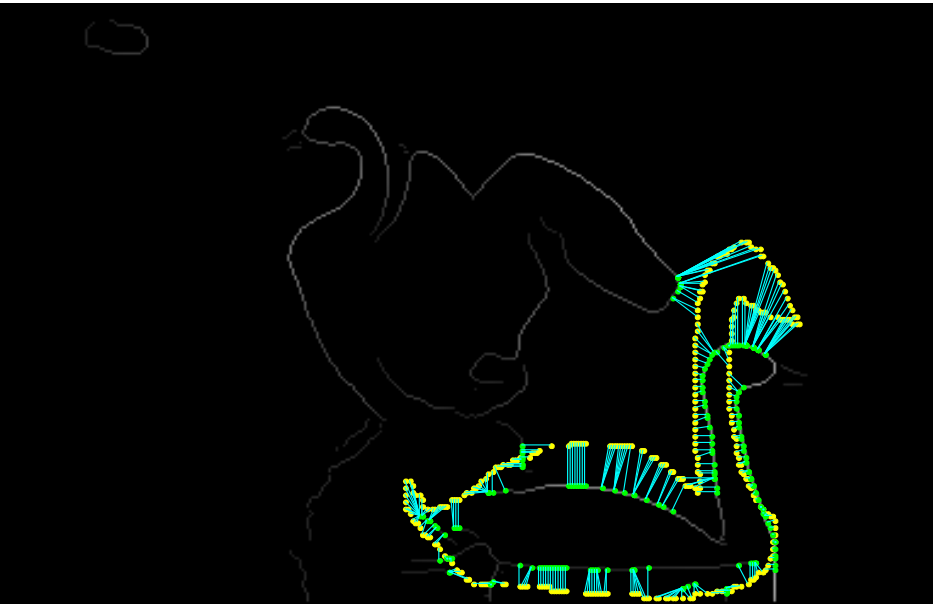
Weaknesses of Chamfer Matching?



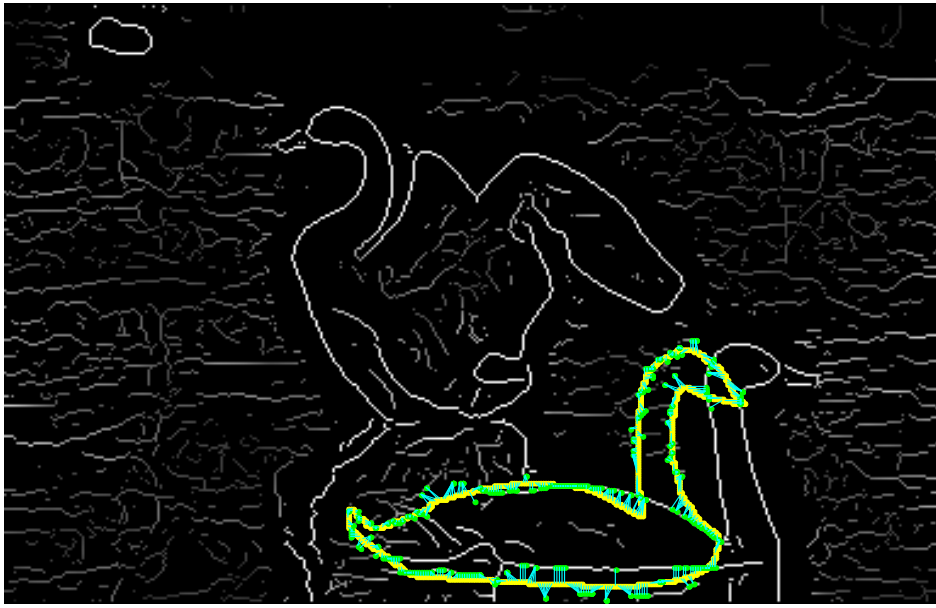
Rotation



Scale



Aspect ratio

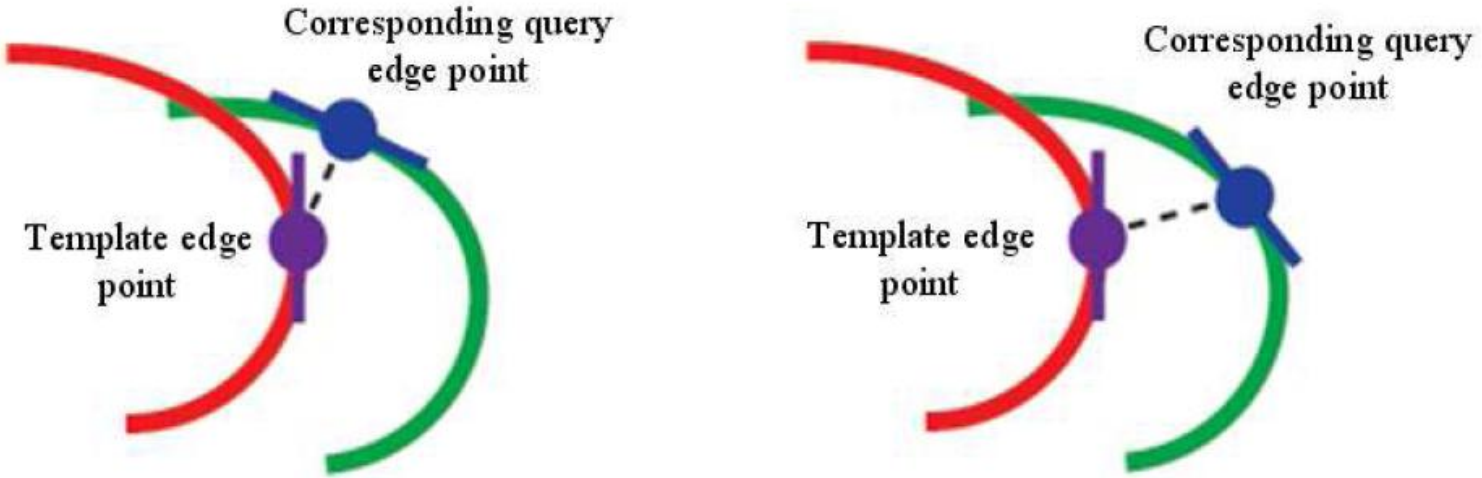


Bad edge map threshold / Clutter

Directional Chamfer Matching

Improvement over regular chamfer matching:

1. **Add edge orientation cost**: penalize template points whose nearest edge points are different orientation. Increases robustness to clutter

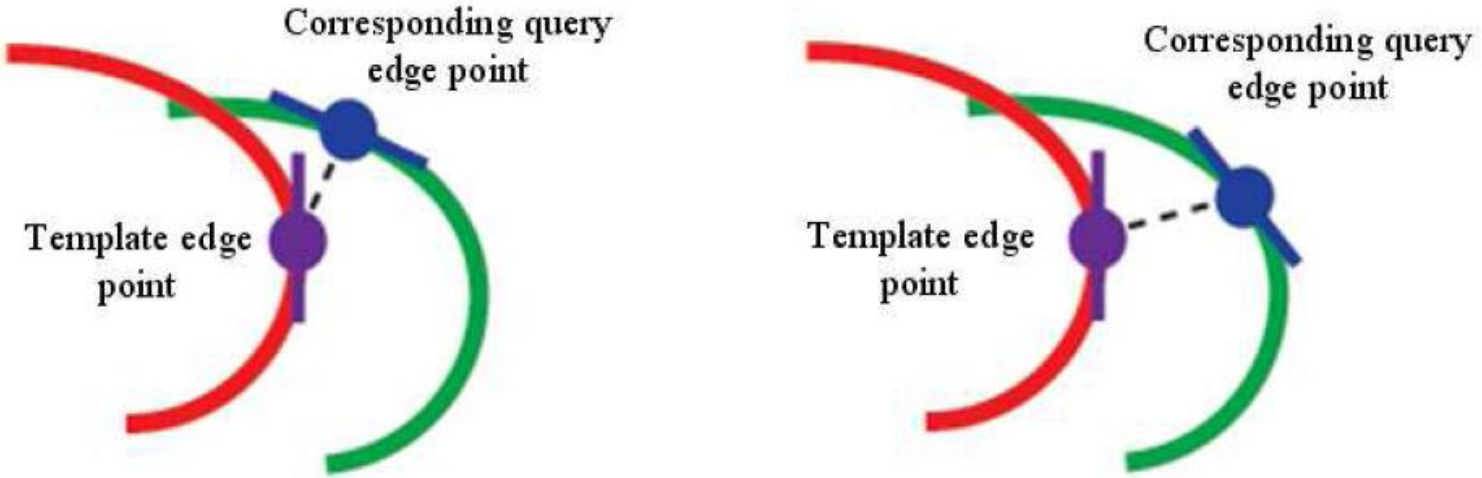


Directional Chamfer Matching

1. **Add edge orientation cost**: the cost for every pixel in the shifted template becomes:

$$c(p) = \min_{q \in E}(\|p - q\| + \lambda|\varphi(p) - \varphi(q)|)$$

where $\varphi(p)$ and $\varphi(q)$ are the orientation of the edge at pixel *p* and *q*.

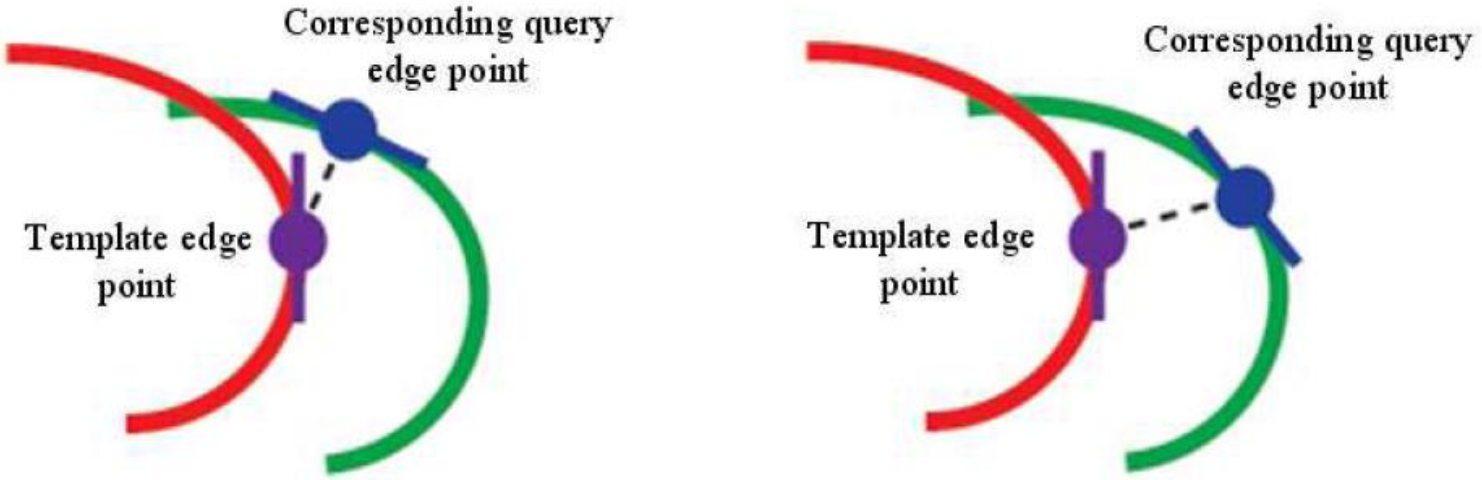


Directional Chamfer Matching

1. **Add edge orientation cost**: the cost for every pixel in the shifted template becomes:

$$c(p) = \min_{q \in E}(\|p - q\| + \lambda|\varphi(p) - \varphi(q)|)$$

where $\varphi(p)$ and $\varphi(q)$ are the orientation of the edge at pixel *p* and *q*.



New cost is separable in orientation:

$$c(p) = \min_{\varphi_i \in \Phi}(\min_{q \in E_{\varphi_i}}(\| p - q \|) + \lambda \,|\, \varphi(p) - \varphi_i \,|)$$

Where Φ is the set of *r* orientation bins in [0, π), and E_{φ_i} is the subset of edges in *E* that have orientation φ_i.

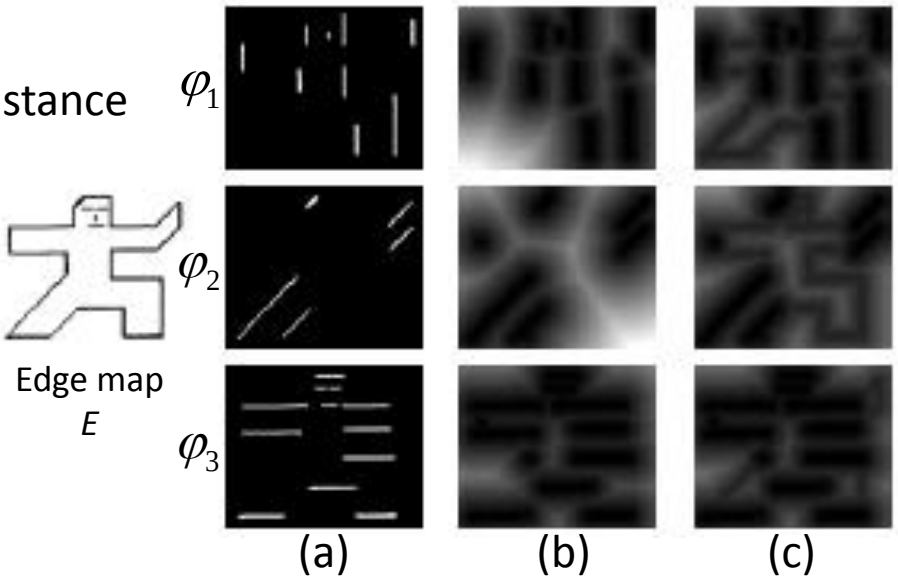
Directional Chamfer Matching

1. Add edge orientation cost: the cost for every pixel in the shifted template becomes:

$$c(p) = \min_{\varphi_i \in \Phi} \left(\min_{q \in E_{\varphi_i}} (\| p - q \|) + \lambda \, | \varphi(p) - \varphi_i | \right)$$

Need to add orientation dimension to Distance Transform to maintain efficiency.

a) Discretize orientation φ into *k* bins in range $[0, \pi)$. Split edge map by orientation.
b) Compute 2D distance transform for each edge orientation φ_i separately.
c) Use DP per-pixel to compute DT over orientations

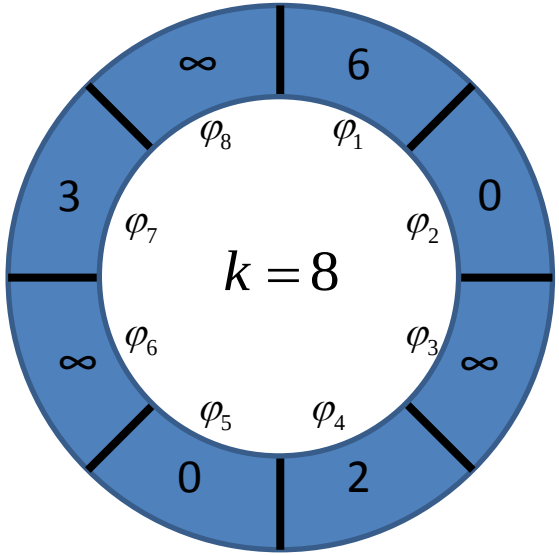


Running time: $O(kn)$, where k is the number of discretized orientations and n is the number of pixels in the query image.

Detour: Circular Distance Transform

Note that distances on φ have to be computed modulus π.

$$|\varphi(p)-\varphi(q)|=\min(|\varphi(p)-\varphi(q)|,||\varphi(p)-\varphi(q)|-\pi|)$$



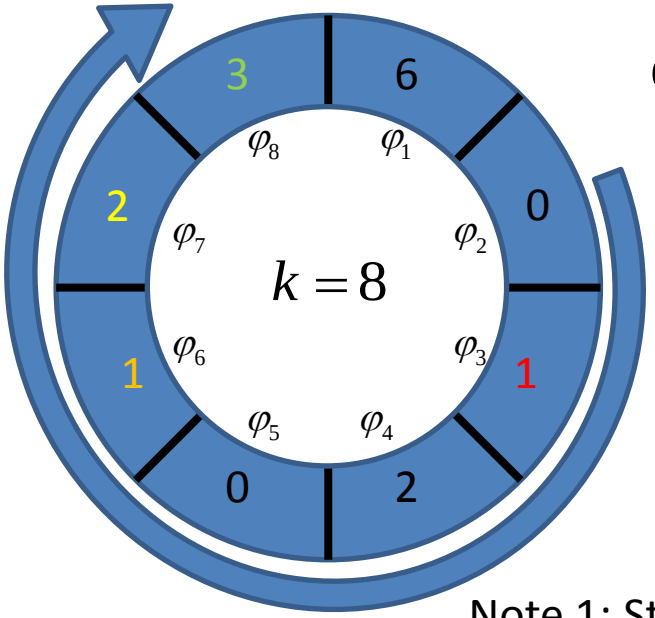
Clockwise dynamic programming pass

1. For each pixel, arrange the *k* values in a circular queue

Detour: Circular Distance Transform

Note that distances on φ have to be computed modulus π.

$$|\varphi(p)-\varphi(q)|=\min(|\varphi(p)-\varphi(q)|,||\varphi(p)-\varphi(q)|-\pi|)$$



Clockwise dynamic programming pass

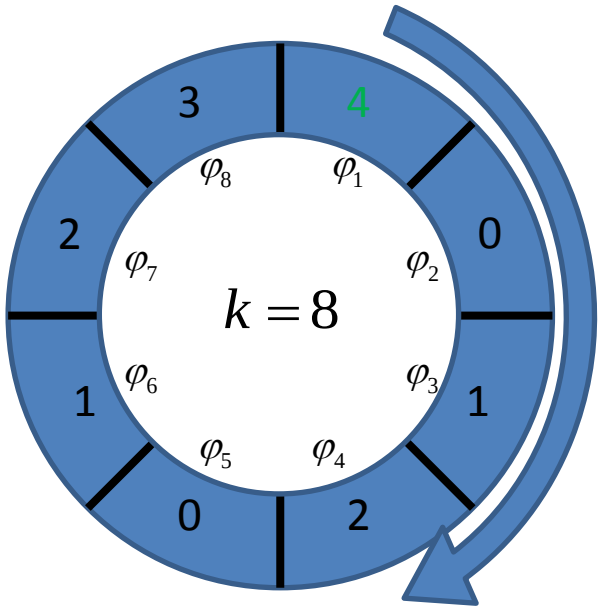
1. For each pixel, arrange the *k* values in a circular queue
2. Loop for *i* = 2 to *k*
3. $\varphi_i \leftarrow \min(\varphi_i, \varphi_{i-1}+\lambda)$

Note 1: Steps 2-3 are just like linear 1D distance transform

Detour: Circular Distance Transform

Note that distances on φ have to be computed modulus π.

$$|\varphi(p)-\varphi(q)|=\min(|\varphi(p)-\varphi(q)|,||\varphi(p)-\varphi(q)|-\pi|)$$



Clockwise dynamic programming pass

1. For each pixel, arrange the *k* values in a circular queue
2. Loop for *i* = 2 to *k*
3. $\varphi_i \leftarrow \min(\varphi_i, \varphi_{i-1}+\lambda)$
4. Continue looping from *i* = 1 until φ_i doesn't change

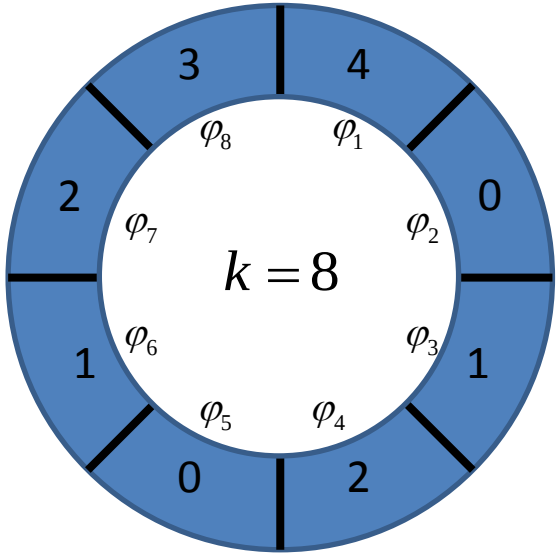
Note 1: Steps 2-3 are just like linear 1D distance transform

Note 2: Step 4 requires at most *k*/2 additional iterations

Detour: Circular Distance Transform

Note that distances on φ have to be computed modulus π.

$$|\varphi(p)-\varphi(q)|=\min(|\varphi(p)-\varphi(q)|,||\varphi(p)-\varphi(q)|-\pi|)$$



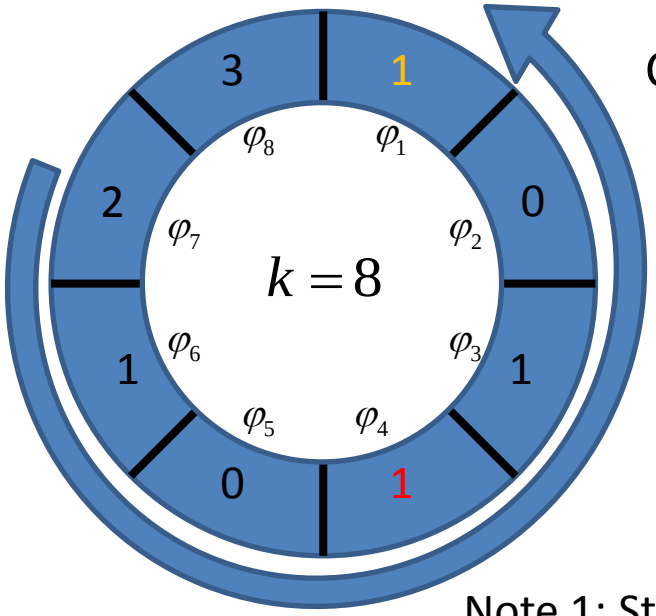
Counter-Clockwise dynamic programming pass

1. For each pixel, arrange the *k* values in a circular queue

Detour: Circular Distance Transform

Note that distances on φ have to be computed modulus π.

$$|\varphi(p)-\varphi(q)|=\min(|\varphi(p)-\varphi(q)|,||\varphi(p)-\varphi(q)|-\pi|)$$



Counter-Clockwise dynamic programming pass

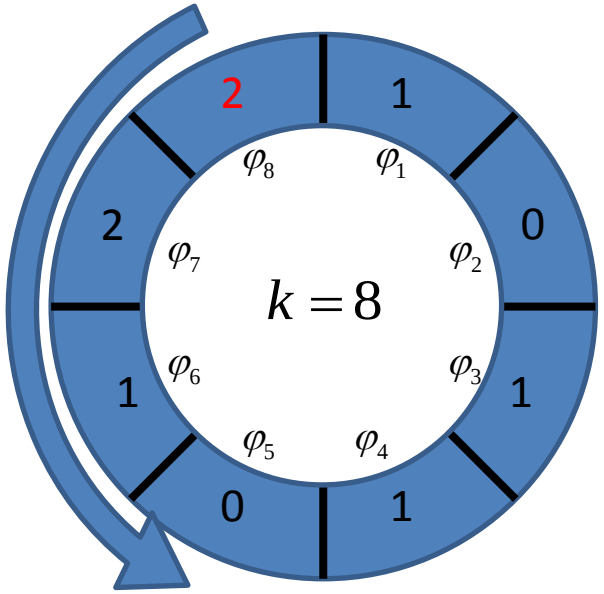
1. For each pixel, arrange the *k* values in a circular queue
2. Loop for *i* = *k*-1 to 1
3. $\varphi_i \leftarrow \min(\varphi_i, \varphi_{i+1}+\lambda)$

Note 1: Steps 2-3 are just like linear 1D distance transform

Detour: Circular Distance Transform

Note that distances on φ have to be computed modulus π.

$$|\varphi(p)-\varphi(q)|=\min(|\varphi(p)-\varphi(q)|,||\varphi(p)-\varphi(q)|-\pi|)$$



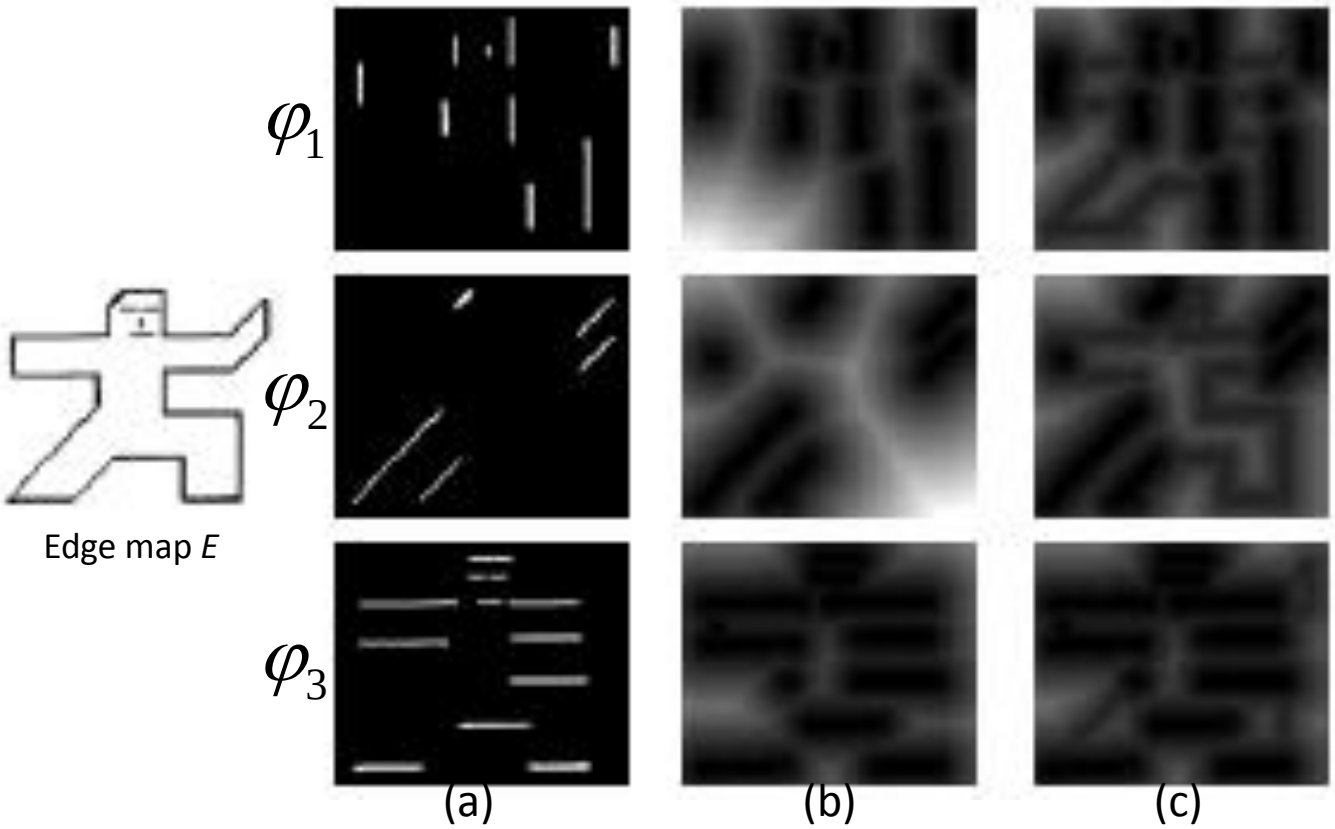
Counter-Clockwise dynamic programming pass

1. For each pixel, arrange the *k* values in a circular queue
2. Loop for *i* = *k*-1 to 1
3. $\varphi_i \leftarrow \min(\varphi_i, \varphi_{i+1}+\lambda)$
4. Continue looping from *i* = *k* until φ_i doesn't change

Note 1: Steps 2-3 are just like linear 1D distance transform
Note 2: Step 4 requires at most *k*/2 additional iterations

Directional Chamfer Matching

$$c(p) = \min_{\varphi_i \in \Phi} \left(\min_{q \in E_{\varphi_i}} (\| p - q \|) + \lambda \, | \varphi(p) - \varphi_i | \right)$$



How fast is it?

Recall standard Chamfer Cost is $\mathcal{O}(n) + \mathcal{O}(\| T \|)$

Oriented Chamfer Cost is $\mathcal{O}(kn) + \mathcal{O}(\| T \|)$

To find the best translation, rotation, scale, and aspect ratio, we need *nksa* iterations where

- n Size of image
- k Number of orientation bins
- s Number of scales
- a Number of aspect ratios

So total Chamfer Matching cost is $\mathcal{O}(kn) + \mathcal{O}(nksa \| T \|)$

That's linear in every variable, but still very big! Can we do better?

Fast + Directional Chamfer Matching

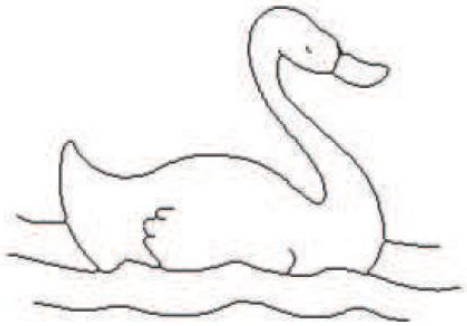
Some optimizations over oriented chamfer matching:

2. **Approximate template shape with line segments**: greatly reduces representation size of each template (eg, from ~1200 points for the goose to ~60 lines). Also, template need not be resampled if it is scaled up/down or aspect ratio changes.
3. **Distance transform integral**: Allow computing the cost of each line in $\mathcal{O}(1)$ time regardless of size. Also allows computing longest lines first for early pruning of bad matches.
4. **Optimized region search**: Cost function is smooth in spatial domain, and is bounded by nearby cost. Therefore, if cost at some location p is δ greater than maximum allowed cost ε, then all costs within δ of p are also greater than ε and can be pruned.

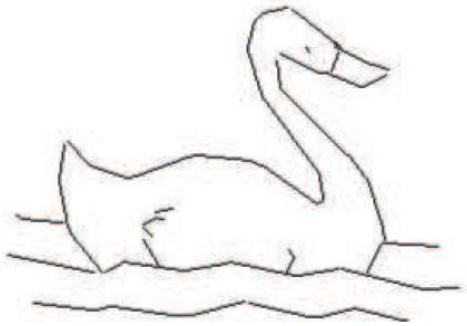
These optimizations allow ***sub-linear search***!

Fast + Directional Chamfer Matching

2. **Approximate template shape with line segments**



a template consists of 1242 points



approximation using 60 line segments

A variant of RANSAC algorithm:

1. Pick several points in *T* randomly. Each point and its orientation defines a line.
2. Support of each line is the points in *T* that are close to the line and are connected.
3. Line segment with greatest support is retained
4. Repeat 1-3 on remaining points until there are not enough points to support any new lines.

Fast + Directional Chamfer Matching

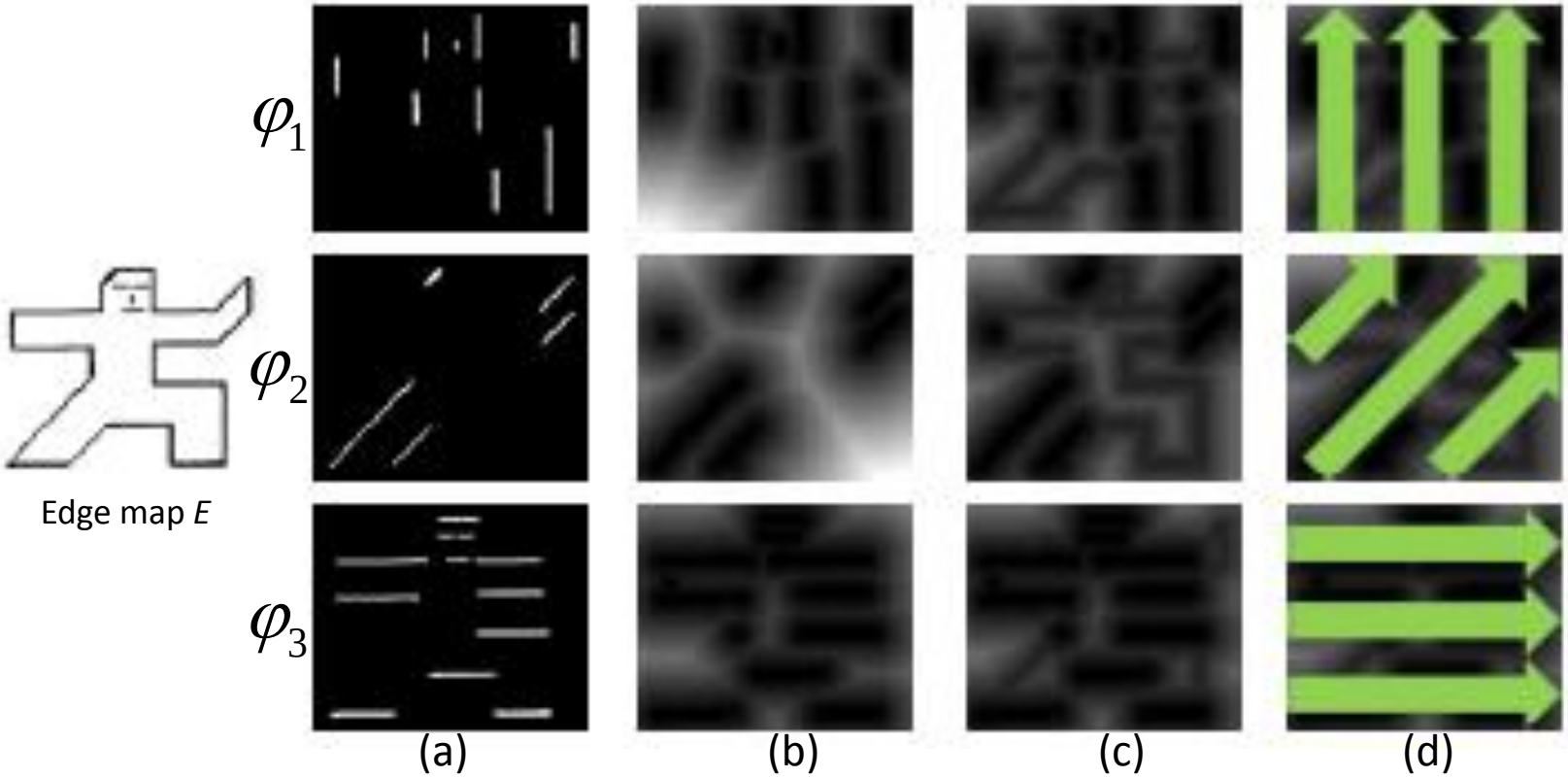
3. **Distance transform integral image**

Sum over lines in template

Sum over pixels in line

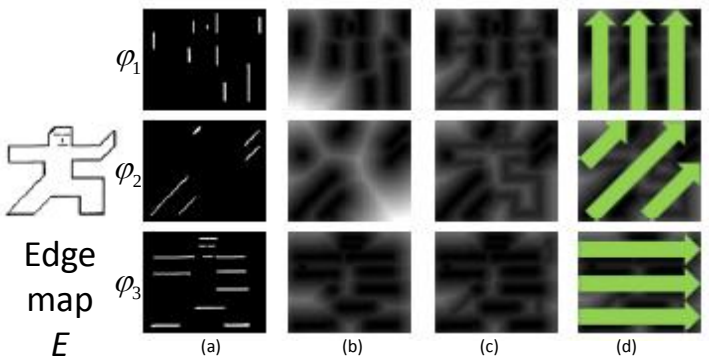
$$\mathrm{D}(T,E)=\frac{1}{\|T\|}\sum_{l_j\in L_T}\sum_{u_i\in l_j}\mathrm{DT}3_E(u_i,\phi(l_j))$$

Can be computed in $\mathcal{O}(1)$ using integral distance transform



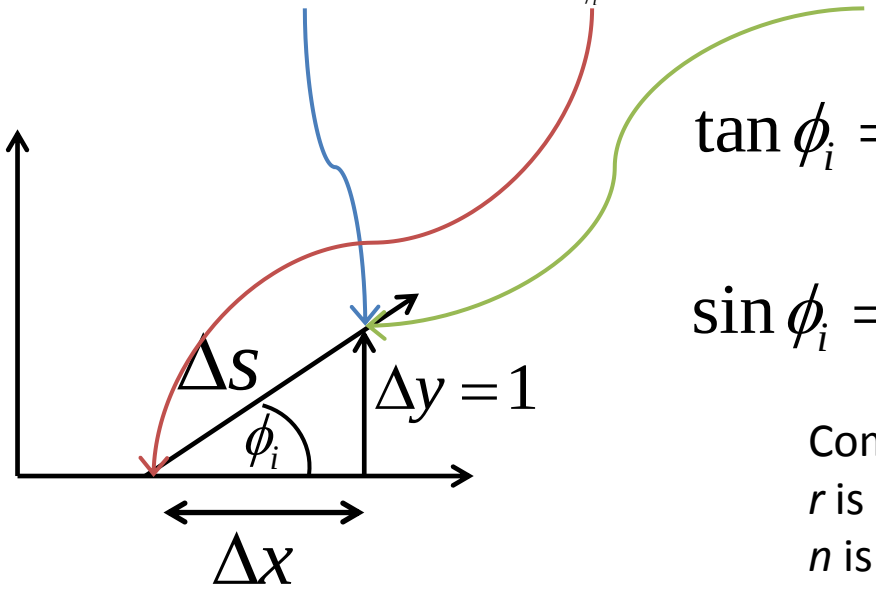
Edge map *E*

Computing the Integral Distance Transform
(more dynamic programming)



How to compute IDT (d) from DT (c):

1. For each orientation $\phi_i \in \Phi$:
2. Initialize IDT(x,y=1) = DT(x,y=1) for all x
3. For y = 2 to *#rows*:
4. For x = 1 to *#cols*:
5. $\mathrm{IDT}(x, y) = \mathrm{IDT}(x - \Delta x_{\phi_i}, y - 1) + \Delta s \cdot \mathrm{DT}(x, y)$



$$\tan \phi_i = \frac{1}{\Delta x} \qquad \Delta x = \cot \phi_i$$

$$\sin \phi_i = \frac{1}{\Delta s} \qquad \Delta s = \csc \phi_i$$

Complexity: $\mathcal{O}(rn)$

r is number of orientations

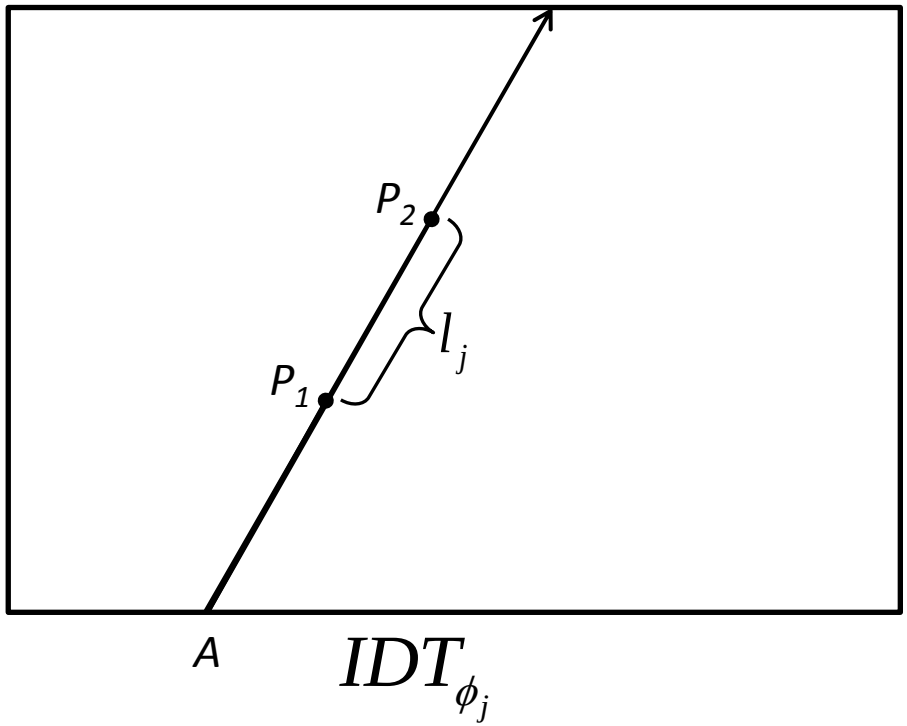
n is number of pixels in image

Fast + Directional Chamfer Matching

3. **Distance transform integral image**

$$\mathrm{D}(T,E) = \frac{1}{\|T\|} \sum_{l_j \in L_T} \sum_{u_i \in l_j} \mathrm{DT}3_E(u_i, \phi(l_j))$$

$$\mathrm{D}(T,E) = \frac{1}{\|T\|} \sum_{l_j \in L_T} [\mathrm{IDT}_{\phi_j}(P_2) - \mathrm{IDT}_{\phi_j}(P_1)]$$



Fast + Directional Chamfer Matching

4. **Optimized Region Search**

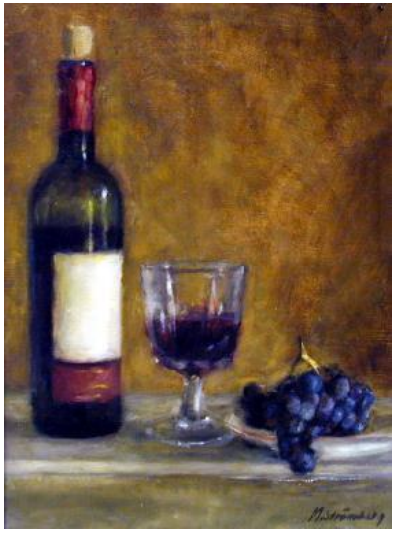
$$
\begin{aligned}
&Cost(u+\delta_x, v+\delta_y) \\
&\quad = \frac{1}{\|T\|} \sum_{p \in T(u,v)} \min_{q \in E} (\|p+\delta-q\| + \lambda|\varphi(p)-\varphi(q)|) \\
&\quad \leq \frac{1}{\|T\|} \sum_{p \in T(u,v)} \min_{q \in E} (\|p-q\| + |\delta| + \lambda|\varphi(p)-\varphi(q)|) \\
&\quad = Cost(u,v) + |\delta|
\end{aligned}
$$

This means *not every possible position needs to be evaluated*. To find matches with cost smaller than ε, and the match cost at point $p=(u,v)$ is $\psi>\varepsilon$, then there cannot be any matches with cost smaller than ε whose distance to p is less than $\delta=\psi-\varepsilon$. Thus we can skip evaluation this region.

A full search requires linear time, and optimized region search only requires sub-linear time (empirically).

Some Alternatives

Each edge pixel may have an “edgeness” score instead of a binary value to avoid bad thresholding.



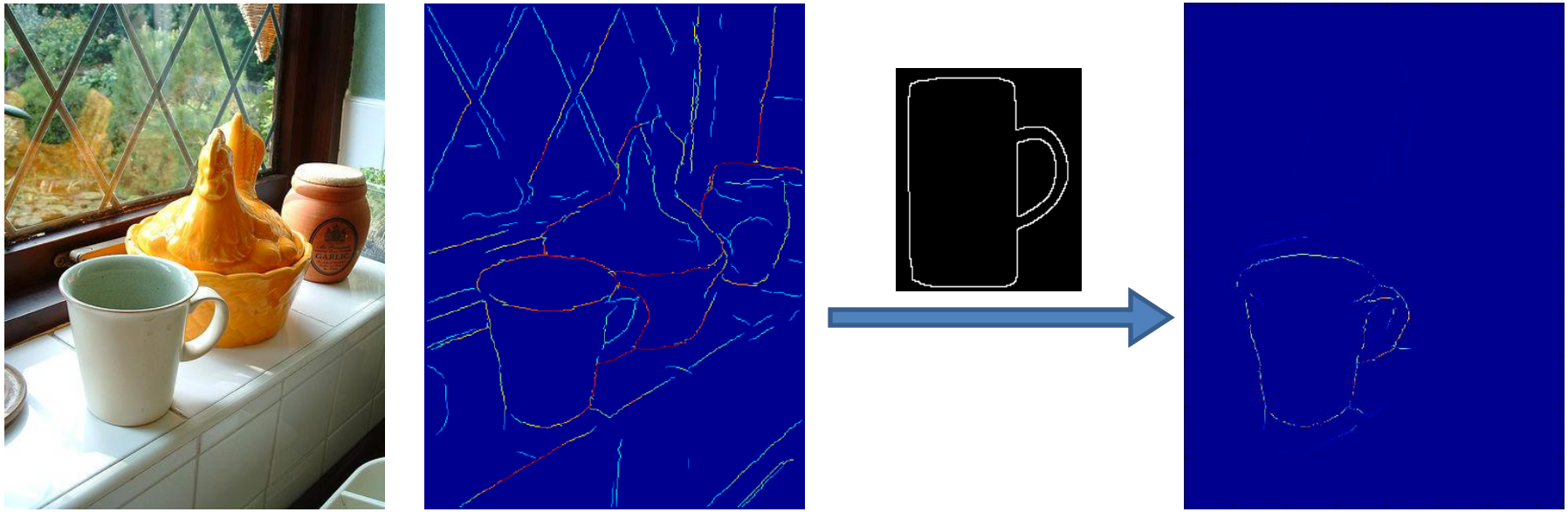
$$c(p) = \min_{q:f(q)>0} \left(\left(\frac{1}{f(q)^2} - 1 \right) + \|p - q\| + \lambda |\varphi(p) - \varphi(q)| \right)$$

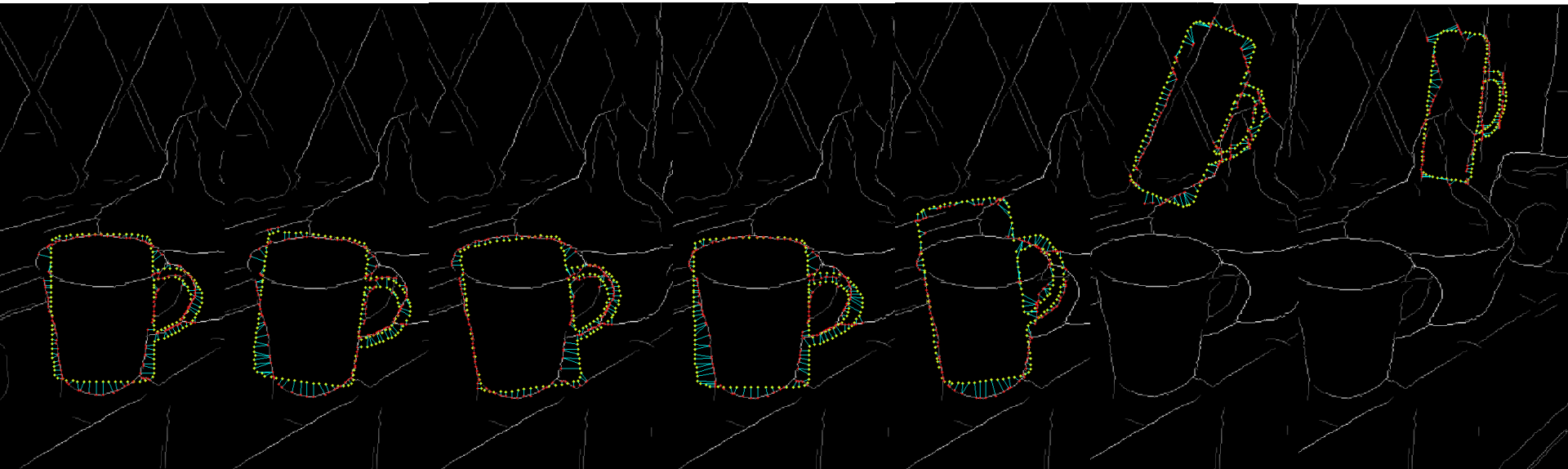
Where *f(q)* is the “edgeness” of pixel q, and *f(q)* is in [0,1].

Distance transform still applies.

Voting from low cost matches:

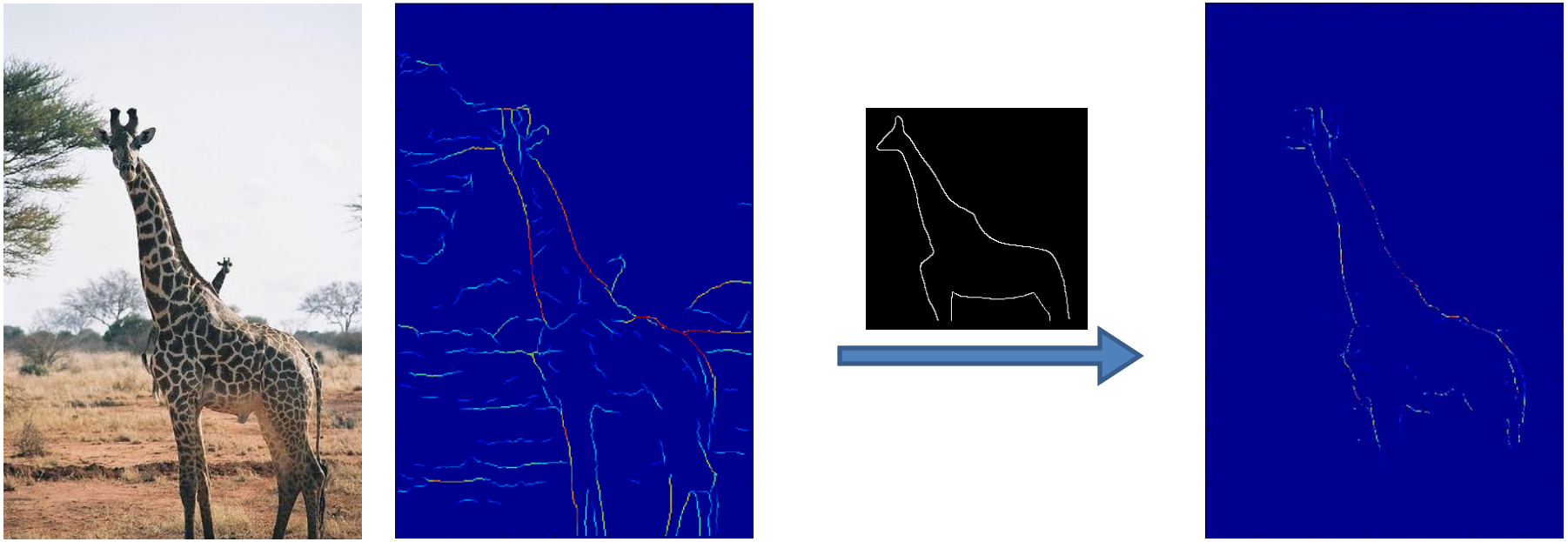
Each hypothesis votes for edge pixels in the query image that participates in the match.





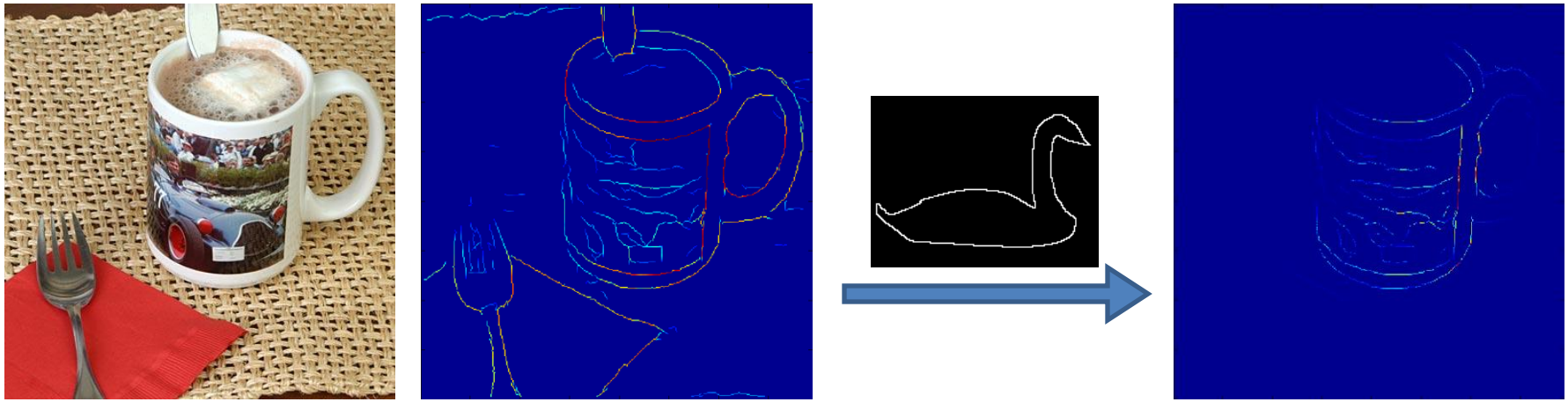
Voting from low cost matches:

Each hypothesis votes for edge pixels in the query image that participates in the match.



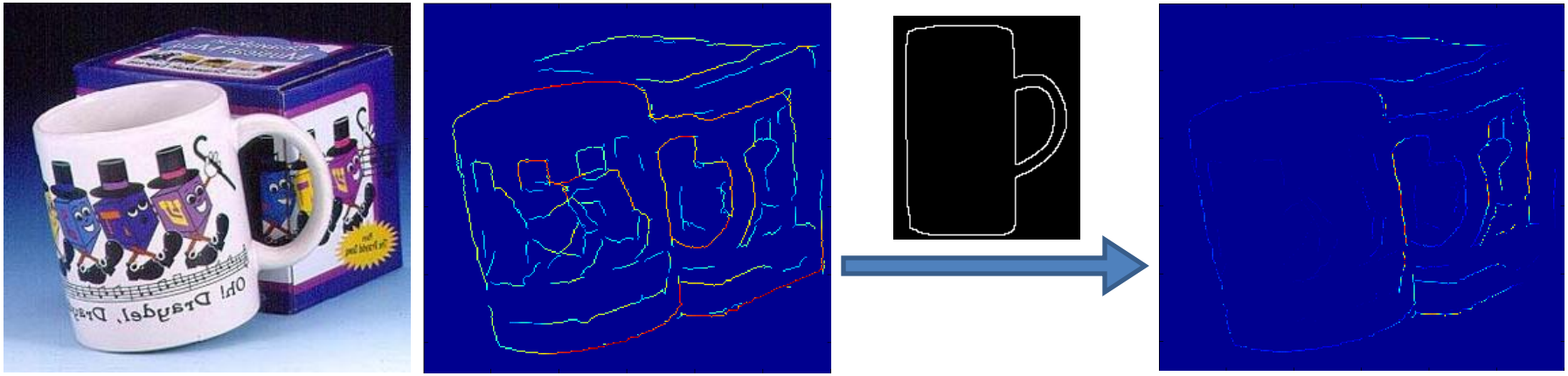
Voting from low cost matches:

Each hypothesis votes for edge pixels in the query image that participates in the match.



Voting from low cost matches:

Each hypothesis votes for edge pixels in the query image that participates in the match.



What results in high chance of accidental alignment?

How is chamfer matching different from other method we introduced?