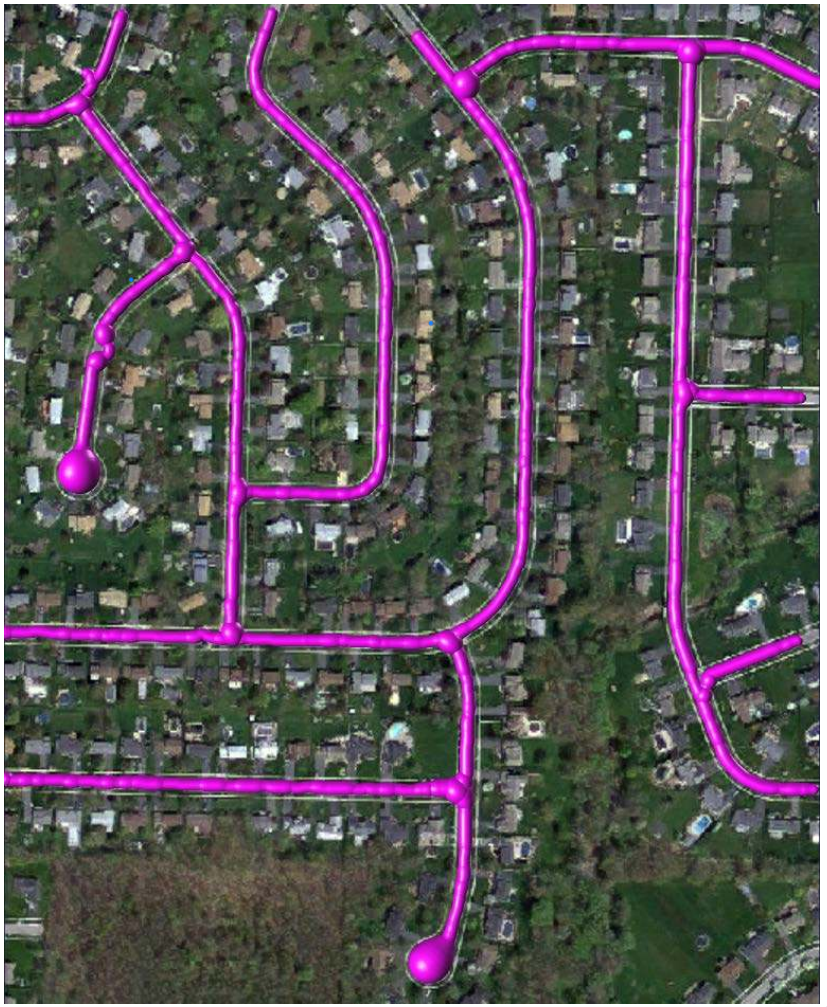
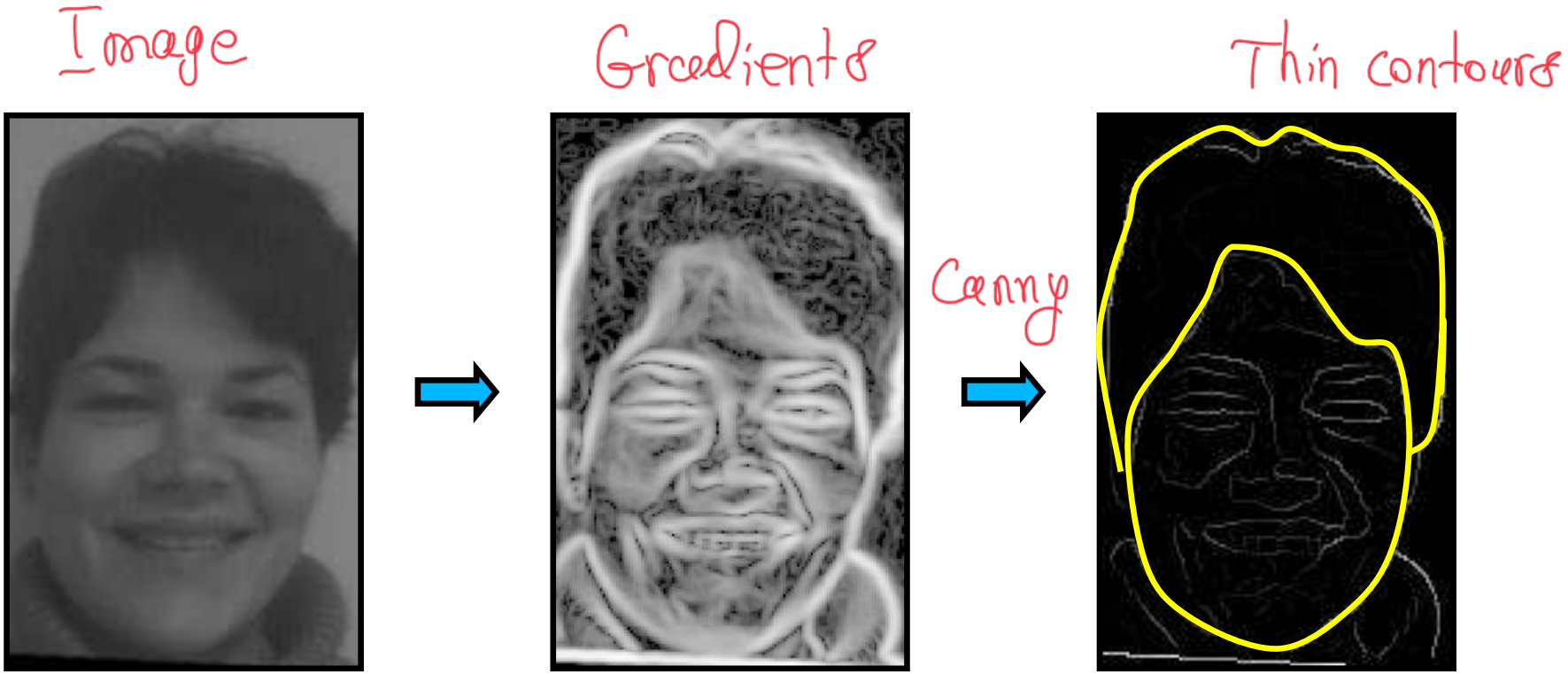
Delineation

Edge map => How they usable

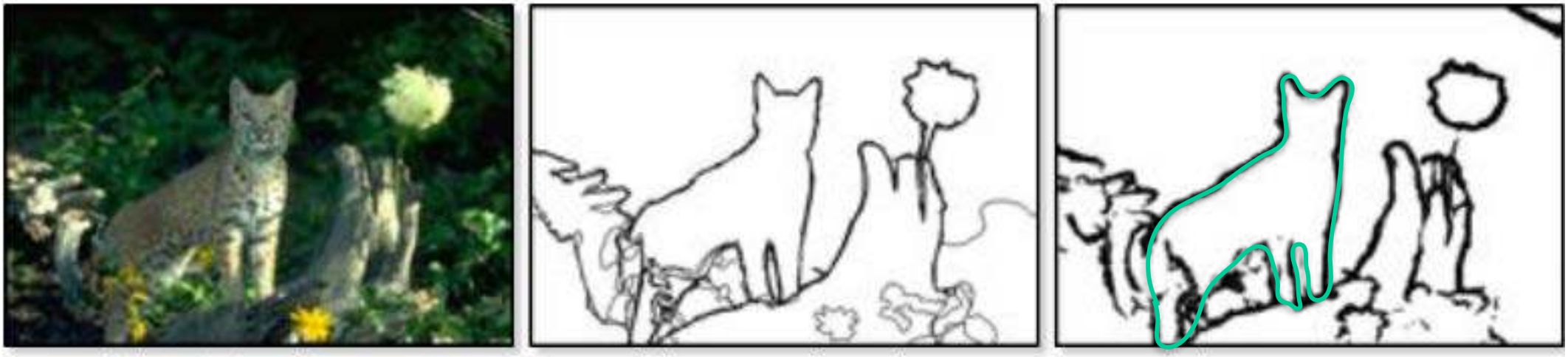
- Dynamic Programming
- Deformable Models
- Hough Transform
- Graph Based Approaches



From Gradients to Outlines



From Deep-Nets to Outlines



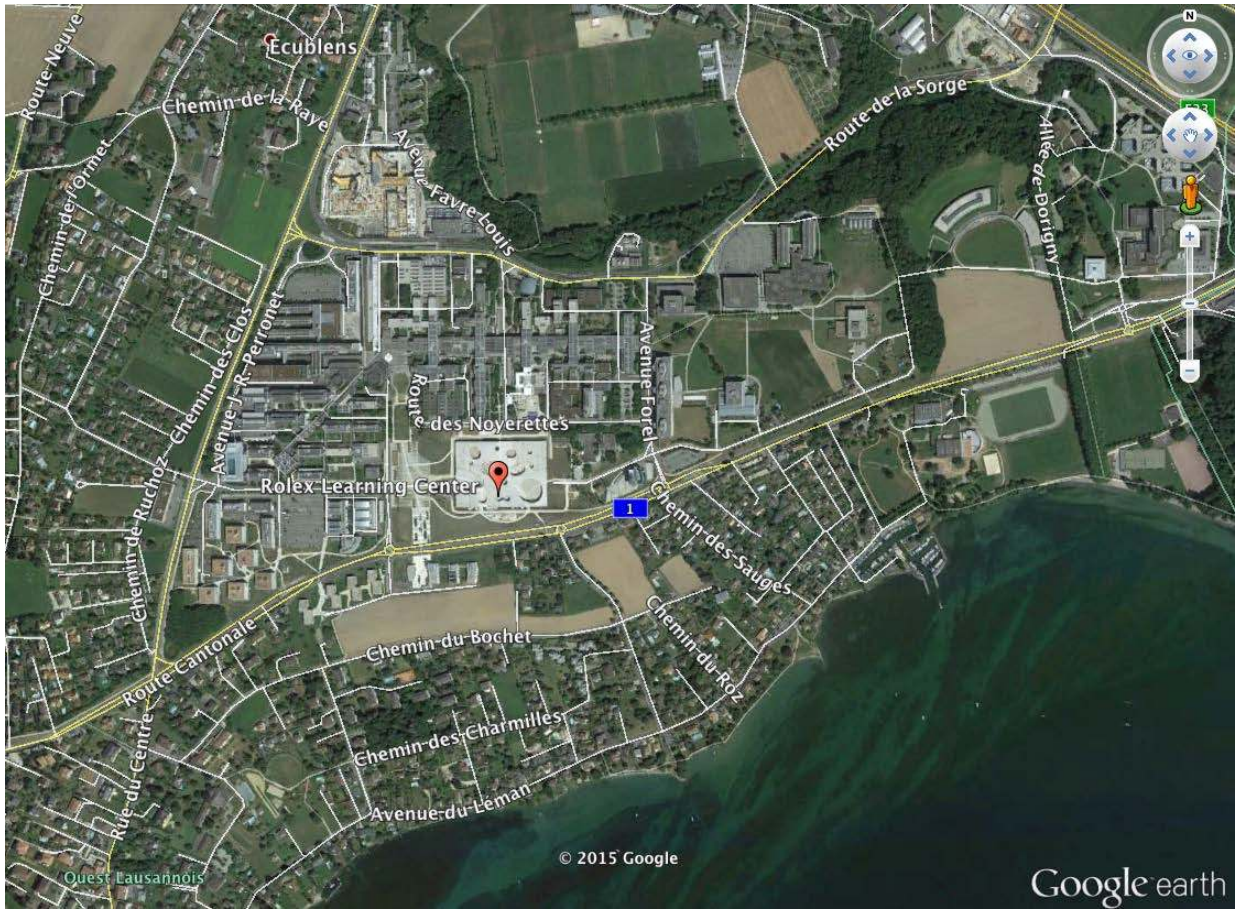
(a) original image (b) ground truth (c) HED: output

DeepNet

not complete contour

—> Still work to do!

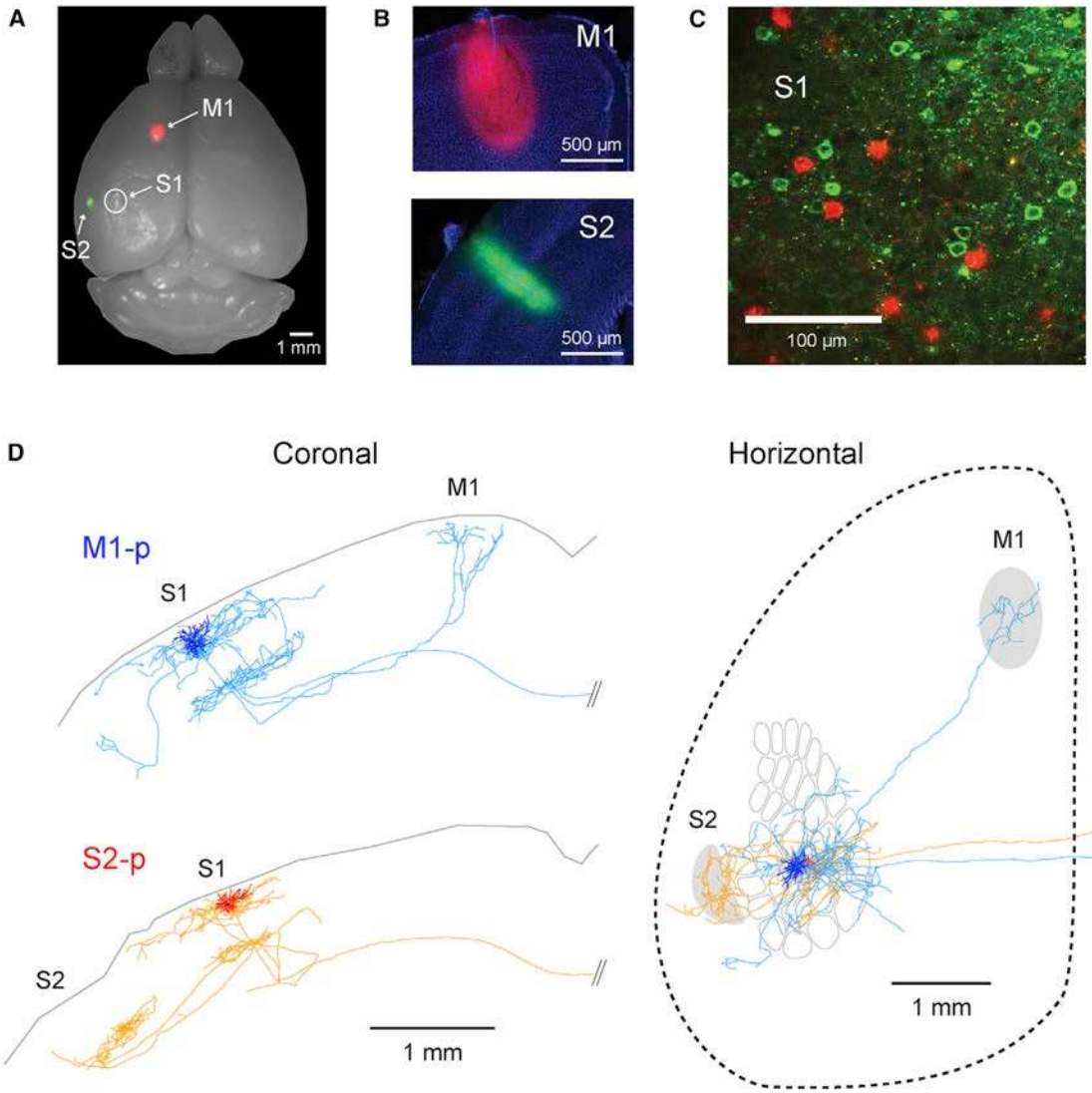
Mapping and Overlays



Connection

Connectivity matters!

Connectomics



Connectivity matter

—> Topology needed

Courtesy of C. Petersen

Analogy

Low level processing

- Uses Deep Nets to find the most promising locations to focus on.

High level processing

- Performs tree-based search when possible.

- Relies on reinforcement learning and other ML techniques to train.

Techniques

Semi-Automated Techniques: (We'll have human in the loop)

- Dynamic Programming
- Deformable Models

Fully Automated Techniques:

- Hough Transform
- Graph Based Approaches

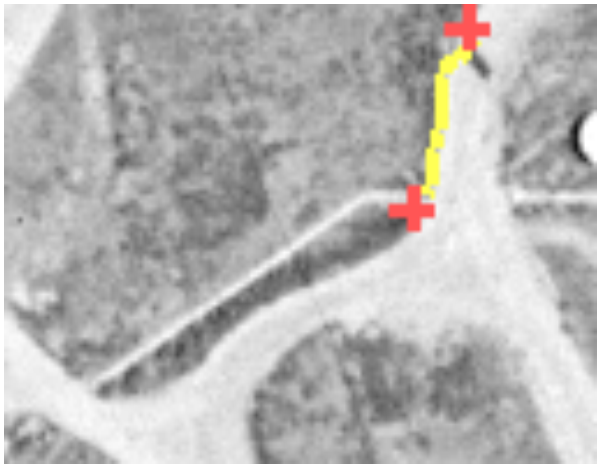
Reminder: Canny Limitations

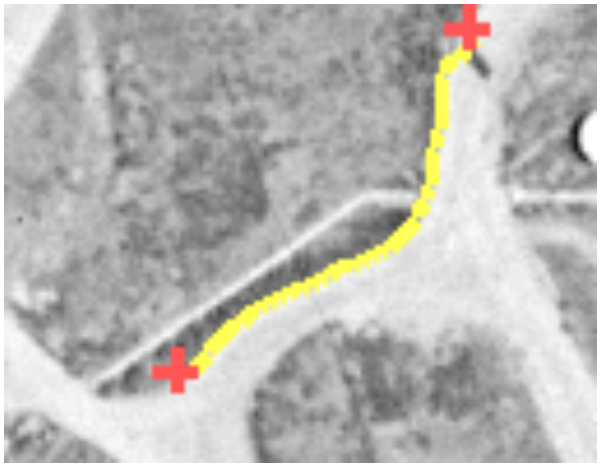
- There is no ideal value of σ!
- Deep nets can help but do not solve the problem.

Interactive Delineation

Image

Gradient

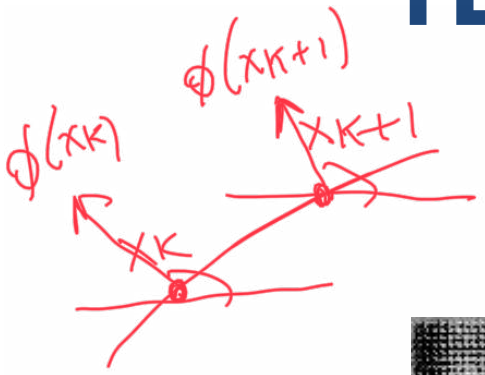




- The user provides the start and end points (red x).
- The algorithm does the rest (yellow line).

Live Wire in Action

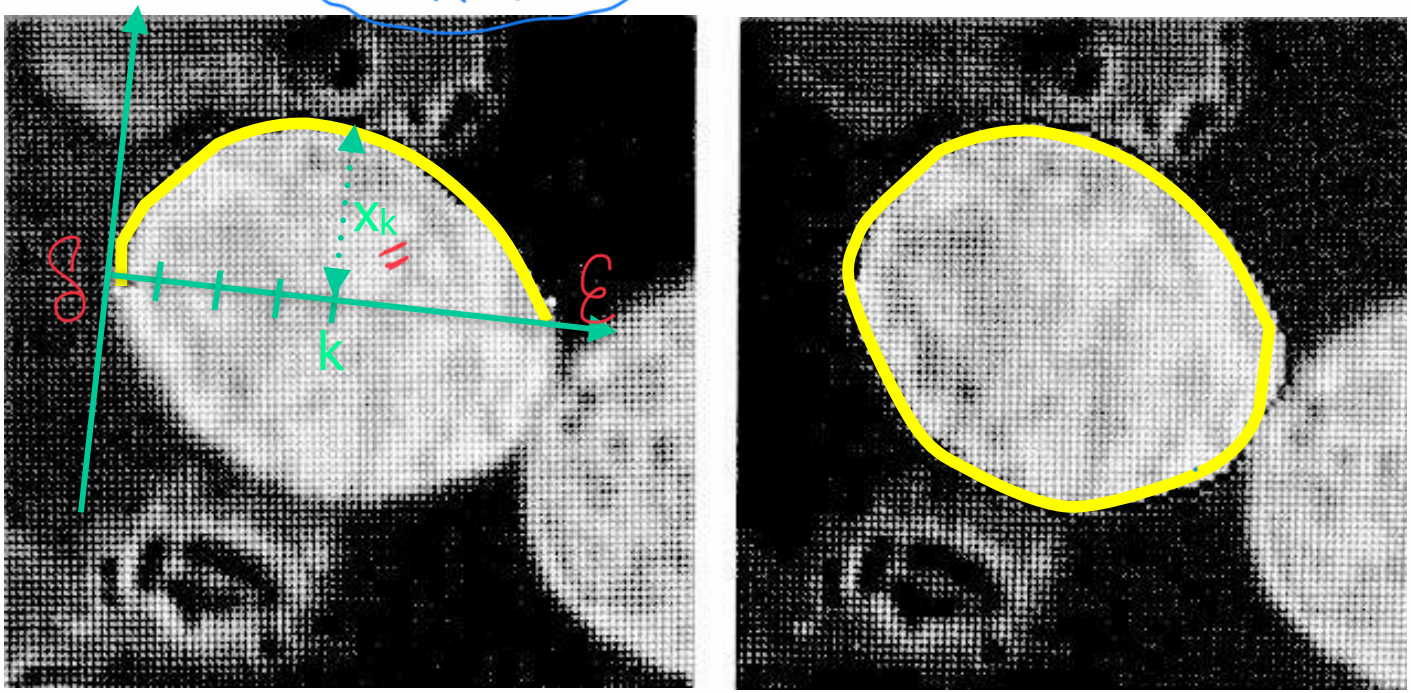
1D Dynamic Programming



$s < t < e \Rightarrow x = f(t)$

Curve more smooth

Markov order 1



maximize

objective function

$$h(x_1, x_2, \ldots, x_n) = -\sum_{k=1}^{n} g(x_k) + \sum_{k=1}^{n-1} r(x_k, x_{k+1})$$

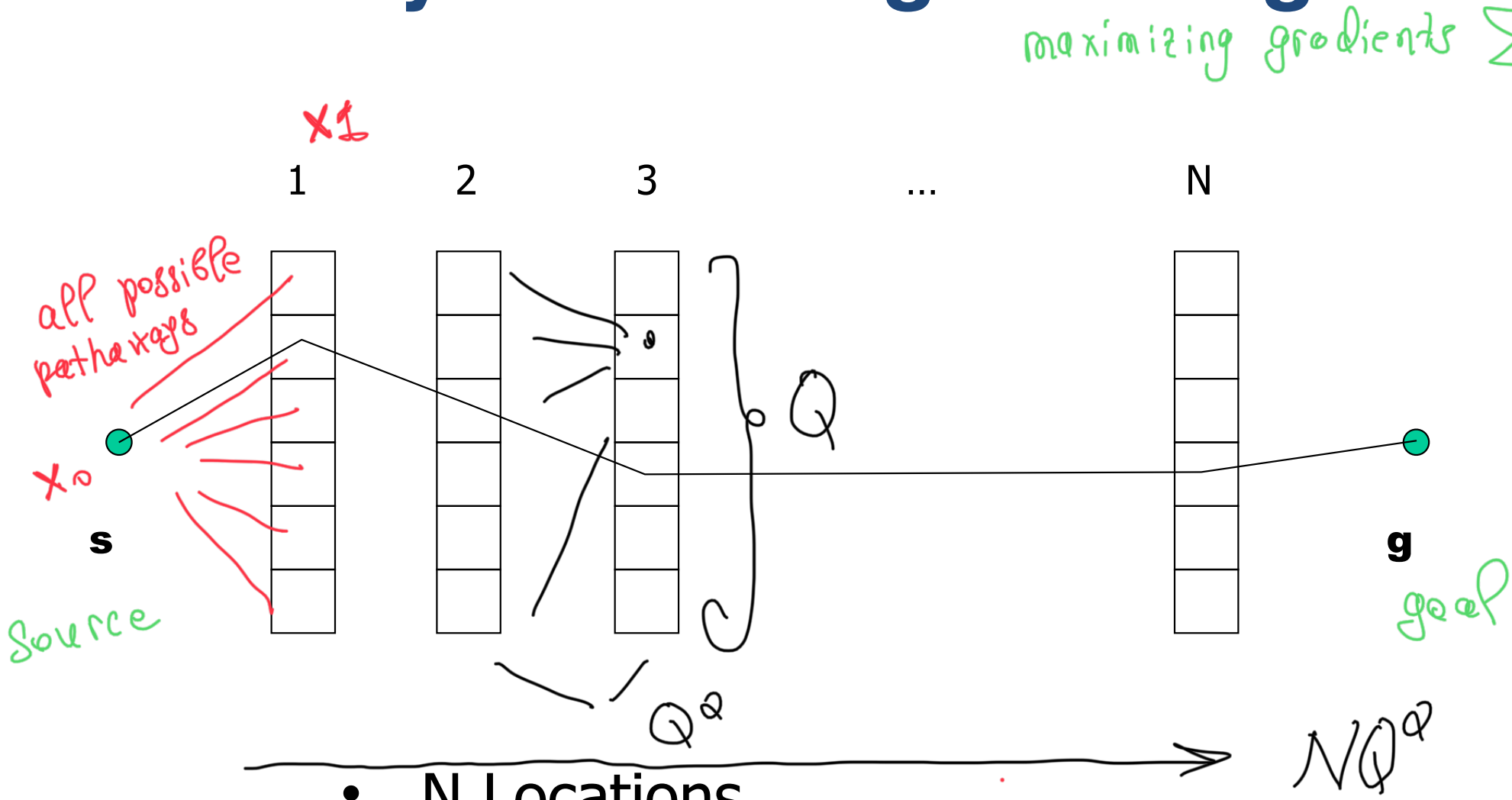
gradient defined at x_k

$$r(x_k, x_{k+1}) = diff(\phi(x_k), \phi(x_{k+1}))$$

difference between 2 angles

where ϕ denotes the gradient orientation.

1D Dynamic Programming



- N Locations
- Q Quantized values (pixel grid)

potential values of x

Exhaustive search

→ Global optimum $O(NQ^2)$

1D Dynamic Programming

To find

$$\min_{x_i} h(x_1, x_2, \ldots, x_n)$$

where

$$h(x_1, x_2, \ldots, x_n) = \mathrm{r}(s, x_1) + \sum_{i=1}^{n-1} \mathrm{r}(x_i, x_{i+1}) + \mathrm{r}(x_n, g)$$

define

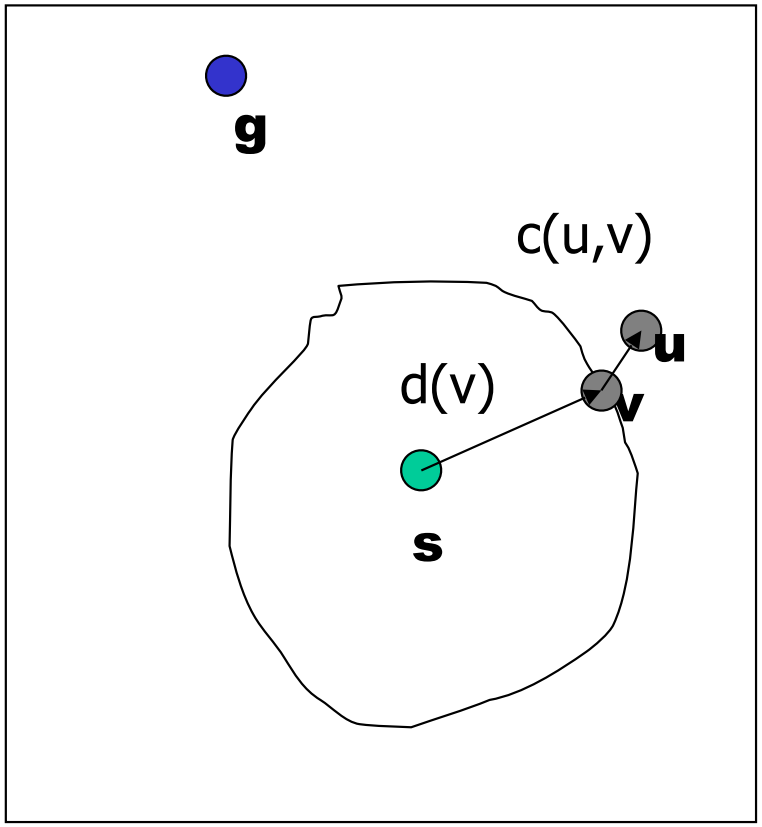
$$\begin{aligned}
f_1(x_2) &= \min_{x_1} (\mathrm{r}(s, x_1) + \mathrm{r}(x_1, x_2)) \\
f_2(x_3) &= \min_{x_2} (\mathrm{r}(x_2, x_3) + f_1(x_2)) \\
&\vdots \quad \vdots \\
f_{n-1}(x_n) &= \min_{x_{n-1}} (\mathrm{r}(x_{n-1}, x_n) + f_{n-2}(x_{n-1}))
\end{aligned}$$

$$\Rightarrow \min h(x_1, x_2, \ldots, x_n) = \min_{x_n} (\mathrm{r}(x_n, g) + f_{n-1}(x_n))$$

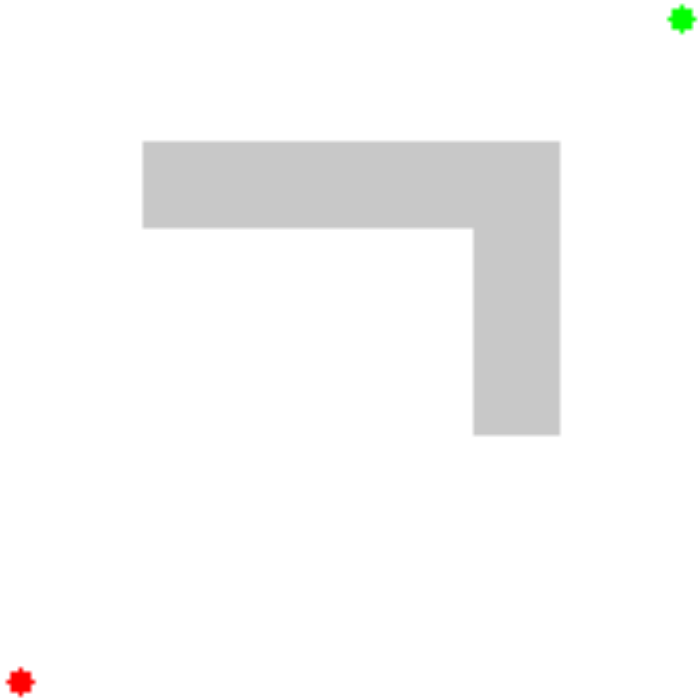
2D Dynamic Programming

Notations:

s	Start point
L	List of active nodes to nodes which we know best paths
c(u,v)	Local costs for link u -> v
d(v)	Total cost from s to v



Dijkstra Path Expansion



Open nodes represent "unvisited" nodes. Filled nodes are visited ones, with color representing the distance: The greener, the shorter the path. Nodes in all the different directions are explored uniformly, appearing more-or-less as a circular wavefront.

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

Dijkstra's Algorithm

Initialization :

$d(s) \leftarrow 0$ and $d(u) \leftarrow \infty$ for $u \neq s$

$T = \emptyset$

$v = s$

source

not connected yet

list of paths

progressively create fronts until g

end of currently shortest path

Loop until goal is reached :

$T \leftarrow T \cup \{v\}$

for all $v \rightarrow u$ edges such that $u \notin T$

if $d(v) + c(v,u) < d(u)$ (cost)

$d(u) \leftarrow d(v) + c(v,u)$

end

end

$v = \text{argmin}_{w \notin T} d(w)$

Maintain a sorted list of paths

Live Wire

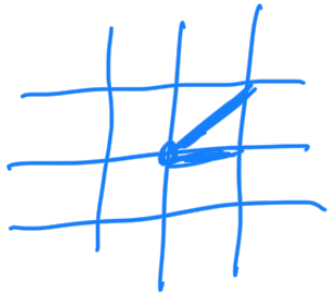
- Sorting is the expensive operation. Normally nlog(n), but can be reduced to log(n) if all costs are integer costs

- Local costs computed using gradient:

edge along which gradient is high ⇒ low cost (low)

gradient

$$c(u,v) = 255 - \tfrac{1}{2}\,(g(u) + g(v))$$



- Diagonal penalized by multiplying cost of non diagonal edges by:

$$\frac{1}{\sqrt{2}} = \frac{5}{7}$$

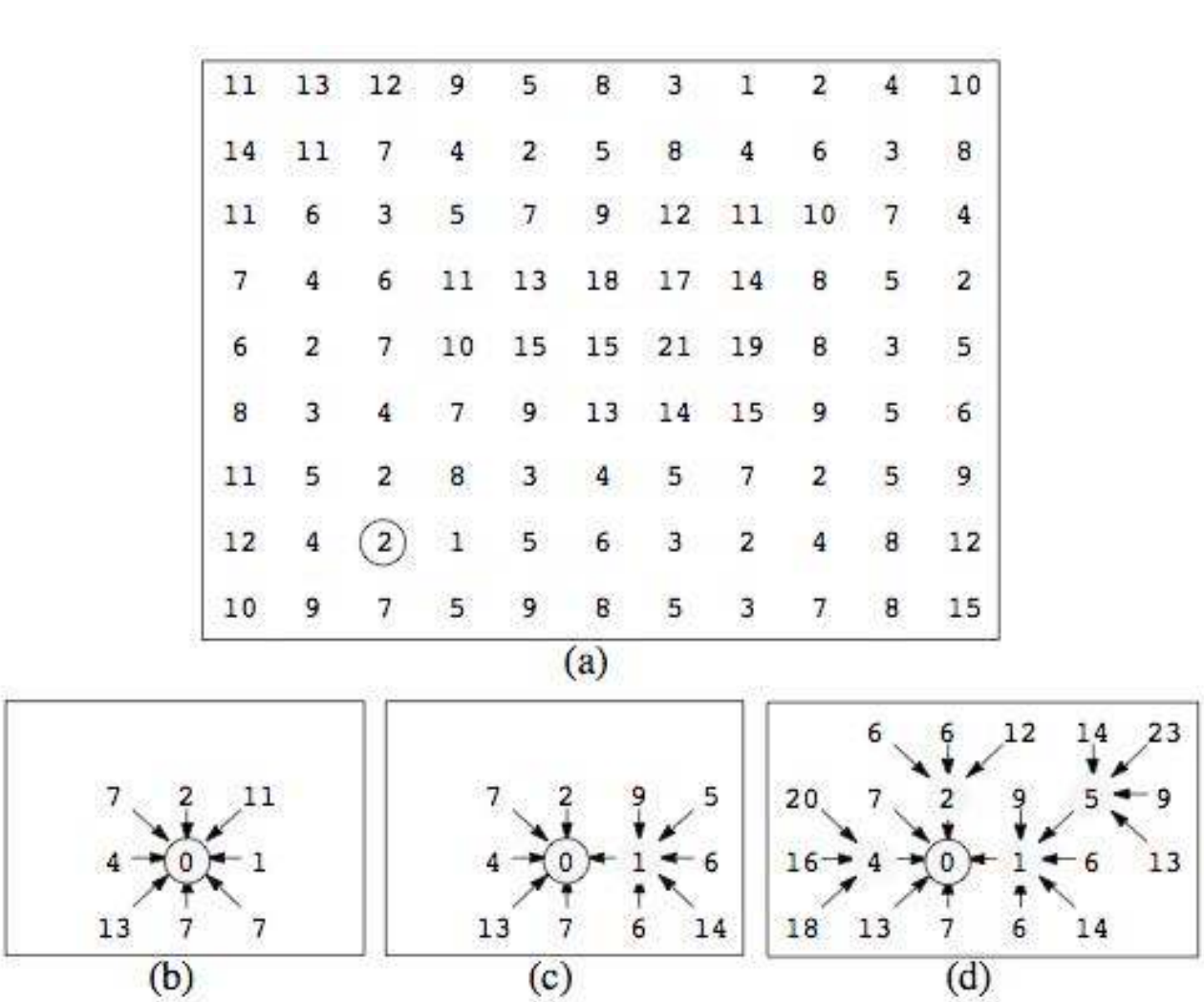
for longer edges (like diags), you need to reduce gradient

more edges ⇒ more cost

- Add a constant cost for each edge.

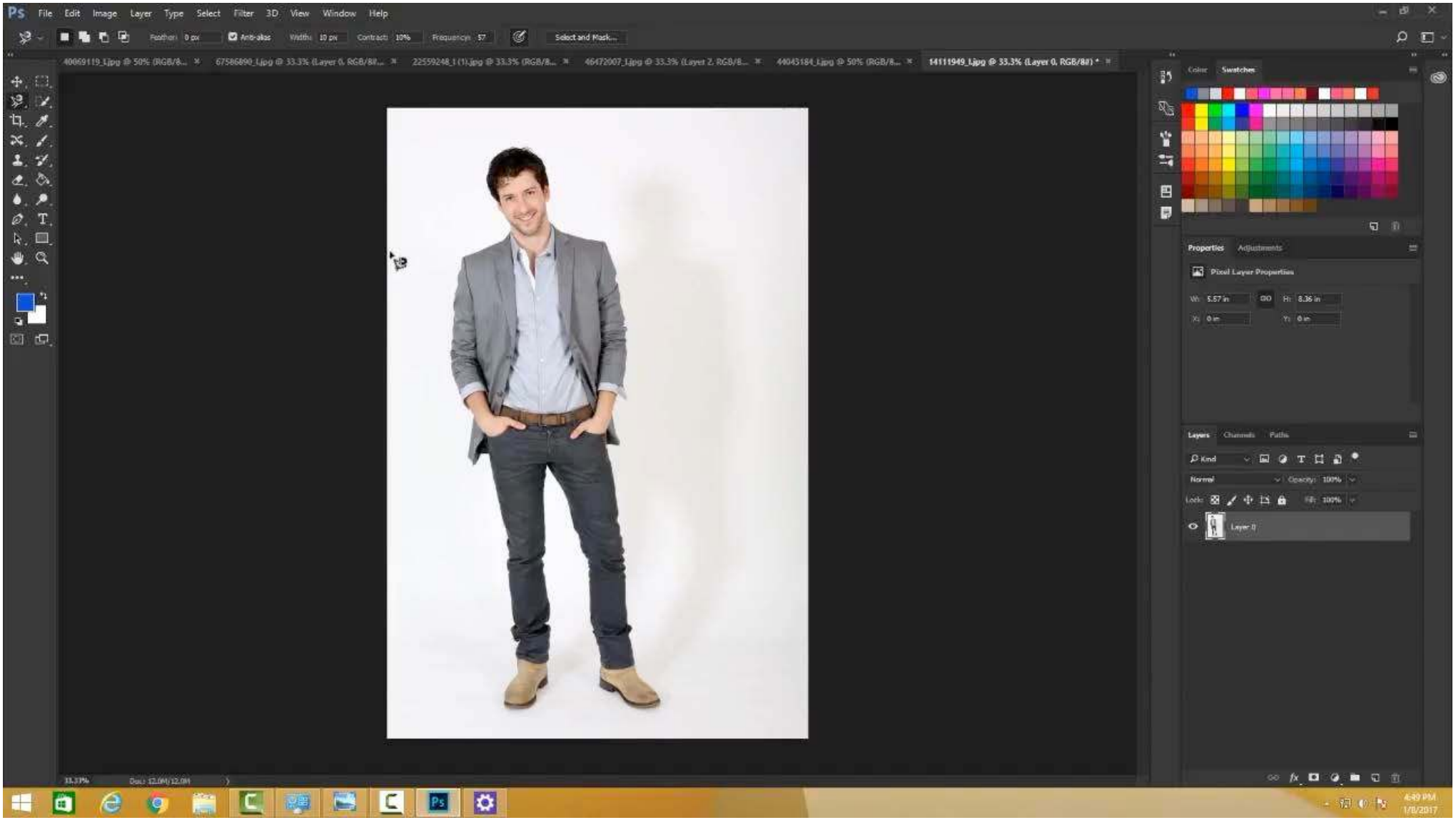
maybe gradient was small ≈ 0 (all edges should be equivalent)

Cost Expansion



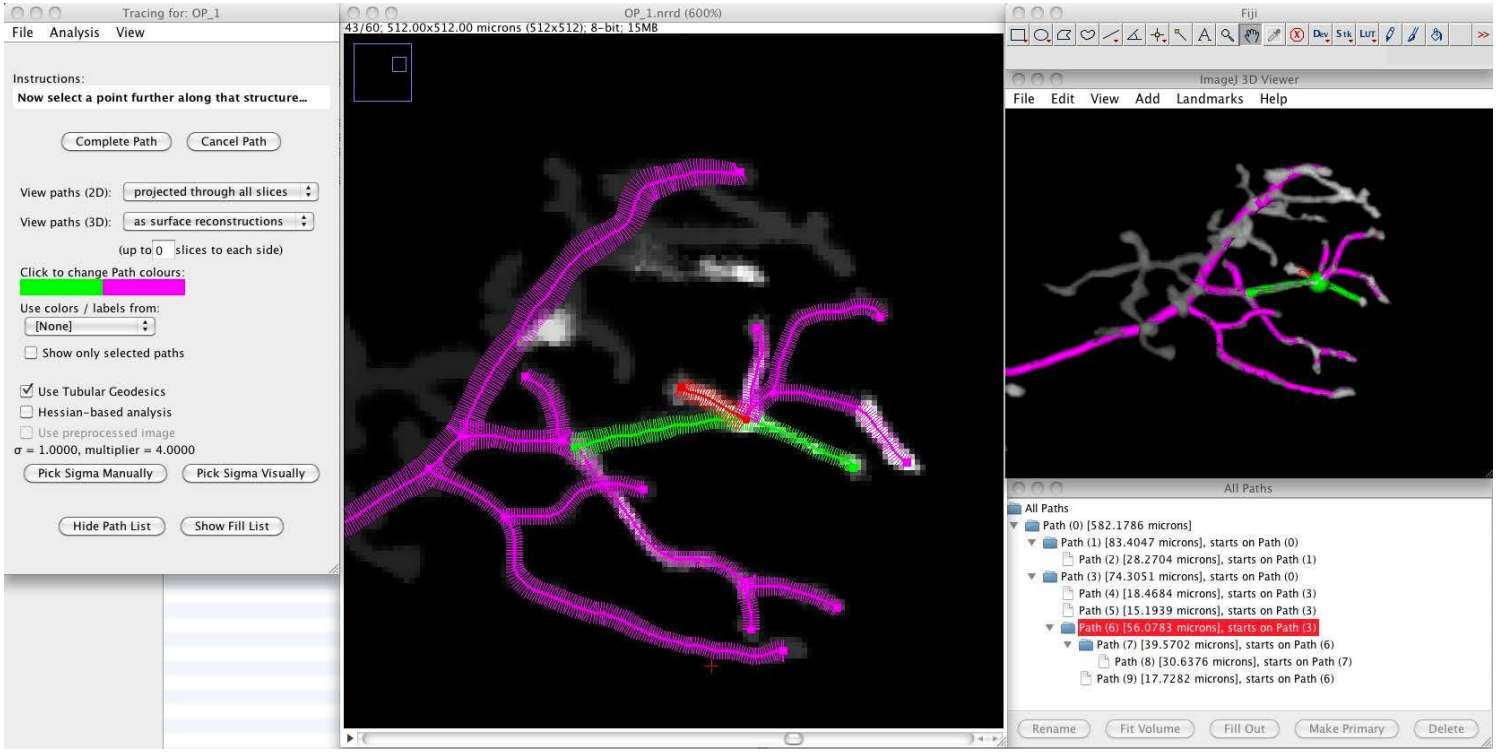
(a) Local cost map. (b) Seed point expanded. (c) 2 points expanded. (d) 5 points expanded. (e) 47 points expanded. (f) Completed cost path-pointer map with optimal paths shown from nodes with total costs 42 and 39.

Magnetic Lasso in Photoshop



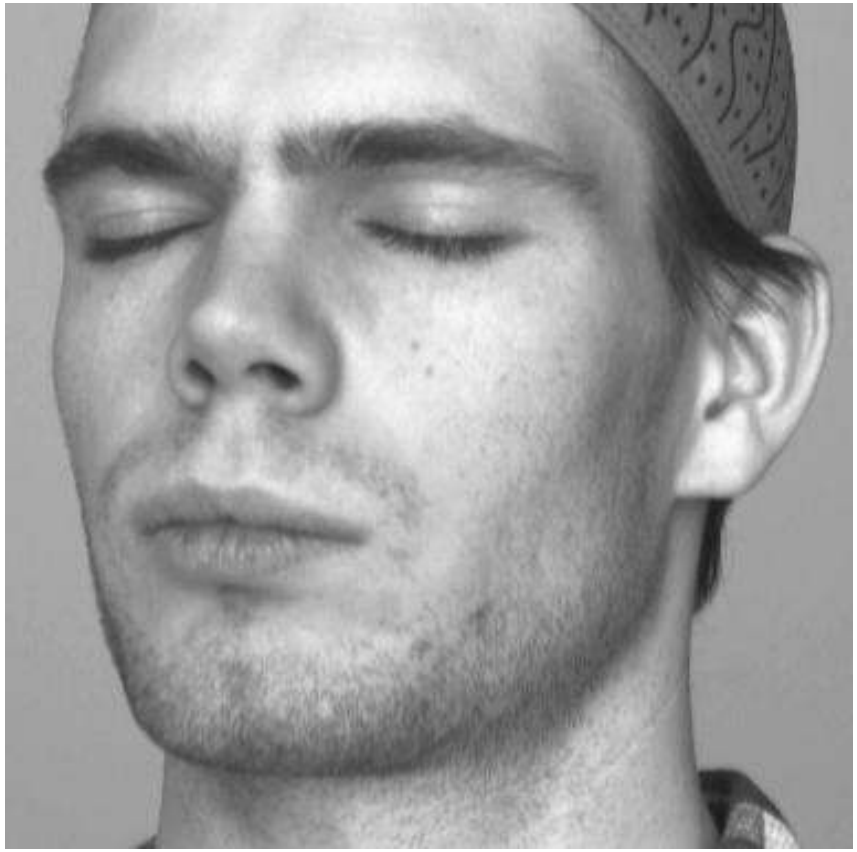
Integrating the LiveWire into a powerful interface that allows a user to correct mistakes yields a useful too.

Tracing Neurons



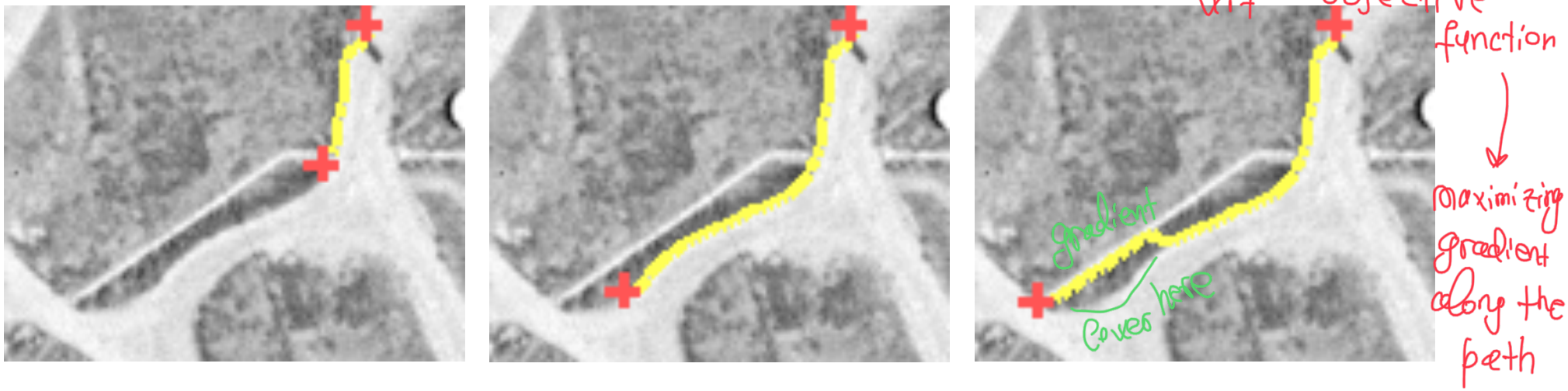
- In the biomedical world, images are 3D cubes of data.
- The approach extends naturally to tracking of 3D structures such as dendritic trees in the brain, blood vessels, etc ...

Face Image



Live Wire

Limitations



- The "optimal" path is not always the "best" one.
- Difficult to impose global constraints. (road is globally smooth)
  Markov order 1 ⇒ points and their immediate neighbors
- The cost grows exponentially with the dimension of the space in which we work.

Dynamic programming ⇒ global optimum (not as expressive as you want)

--> Must often look for local, as opposed to global, optimum using gradient descent techniques.

Techniques

Semi-Automated Techniques:

- Dynamic programming
- Deformable Models

Fully Automated Techniques:

- Hough transform
- Graph Based Approaches

Snakes

Draw Rough approximation (contour being deformed)

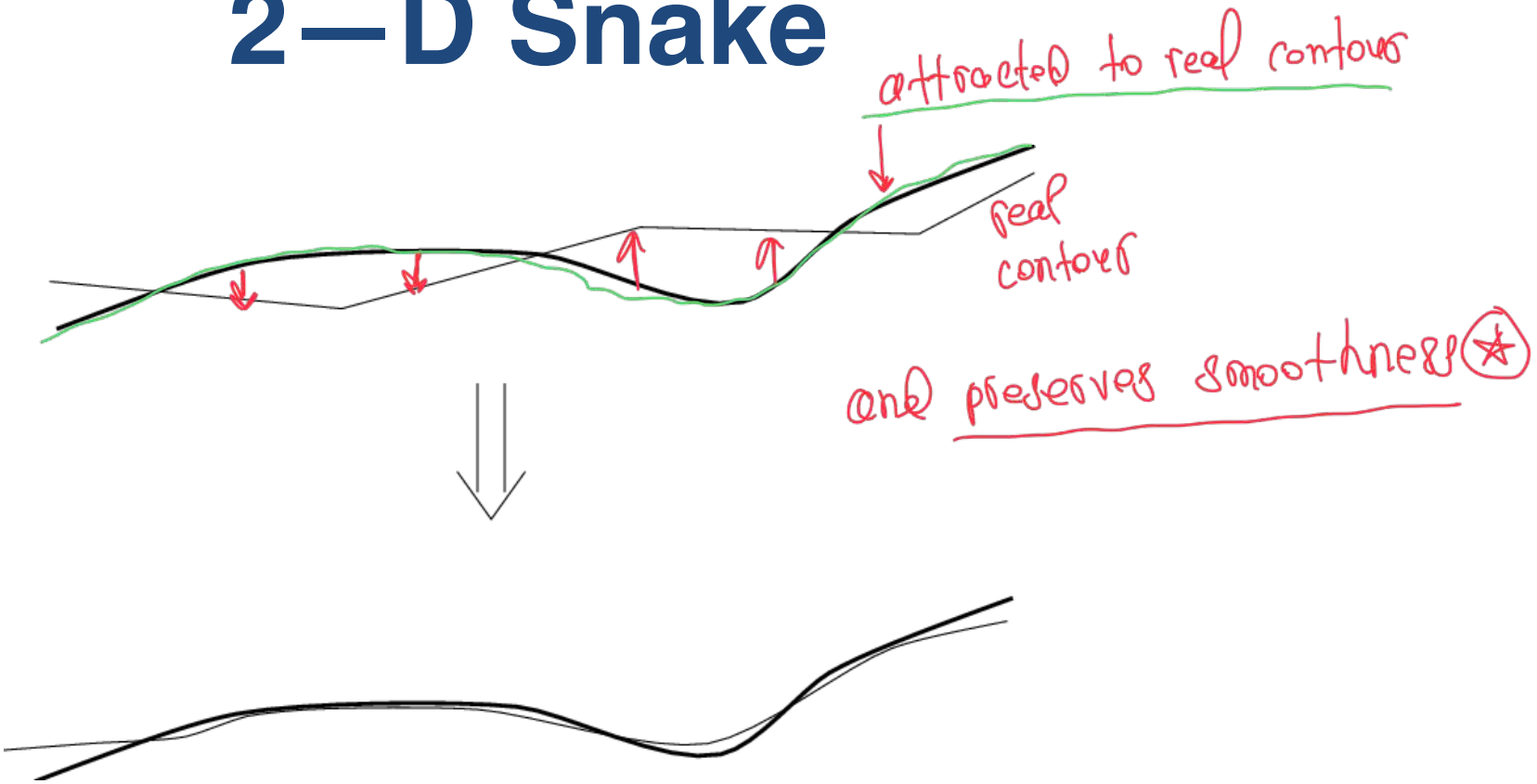
Contour is attracted to gradient &

When it deforms:

i) it remains smooth

ii) maximize gradient on the curve

2—D Snake



Deformable contours that

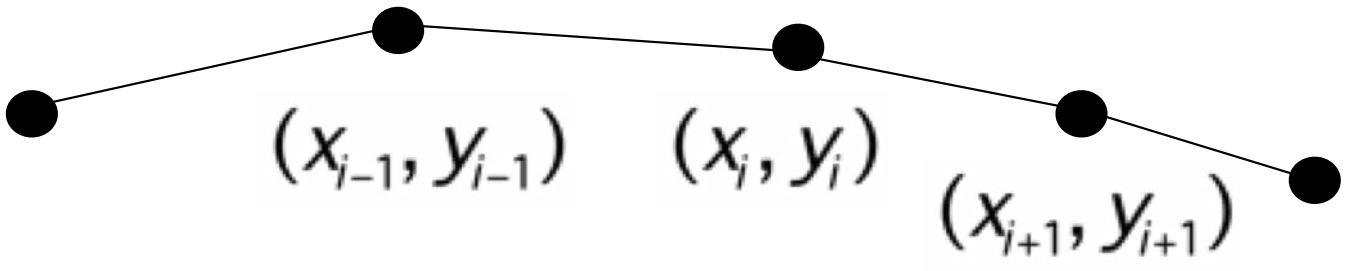
- Maximize the gradient along the curve;
- Minimize their deformation energy. → remains smooth

--> Interactive tools for contour detection that can be generalized to handle sophisticated models

Polygonal Approximation

defined by its vertices

Compromise between largest possible gradient and smallest possible curvature



Weighting coefficient

$$E = -\frac{\lambda}{N+1}\sum_{i=0}^{N} G(x_i, y_i) + -\frac{1}{N}\sum_{i=1}^{N}((2x_i - x_{i-1} - x_{i+1})^2 + (2y_i - y_{i-1} - y_{i+1})^2)$$

of 2nd derivatives

Finite difference approximation

$\frac{\partial x}{\partial s}$ $\frac{\partial y}{\partial s}$

Average gradient

Average sum of squared 2nd derivatives

$\approx$

Average sum of square curvature

Matrix Notation

along the curve
$\sum G(x,y)$ ⇝ gradients

$$
\begin{aligned}
E &= E_G + 1/2 X^t K X + 1/2 Y^t K Y \\
X &= [x_1, \ldots, x_N]^t \\
Y &= [y_1, \ldots, y_N]^t
\end{aligned}
$$

Sparse matrix

$$
K = \begin{bmatrix}
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & 1 & -4 & 6 & -4 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & 1 & -4 & 6 & -4 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & -4 & 6 & -4 & 1 & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 1 & -4 & 6 & -4 & 1 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & -4 & 6 & -4 & 1 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{bmatrix}
$$

Local Optimum

$$\frac{\partial E}{\partial X} = \left[\frac{\partial E}{\partial x_1} \cdots \frac{\partial E}{\partial x_n} \right]$$

we are looking for minimum of E

$$E = E_G + \frac{1}{2} X^T K X + \frac{1}{2} Y^T K Y$$

$$\frac{\delta E}{\delta X} = \frac{\delta E_G}{\delta X} + KX = 0$$

$$\frac{\delta E}{\delta Y} = \frac{\delta E_G}{\delta Y} + KY = 0$$

positive but not invertible

But K is not invertible!

Dynamics

compute $\{ X_0, X_1, \ldots, X_T$ / $y_0, y_1, \ldots, y_T$

rubber band

Embed curve in a viscous medium and solve at each step:

faster it goes, more slow down

$$0 = \frac{\partial E}{\partial X} + \alpha \frac{dX}{dt} = \frac{\partial E_G}{\partial X} + KX + \alpha \frac{dX}{dt}$$

$$0 = \frac{\partial E}{\partial Y} + \alpha \frac{dY}{dt} = \frac{\partial E_G}{\partial Y} + KY + \alpha \frac{dY}{dt}$$

Iterating

Deformed curves ⇒ computing X_t, Y_t

At every step:

Solving a differential equations

$\frac{dx}{dt}$: finite-difference expression of derivative

$$0 = \frac{\delta E_G}{\delta X} + KX_t + \alpha(X_t - X_{t-1}) \quad \Rightarrow \quad (K + \alpha I)X_t = \alpha X_{t-1} - \frac{\delta E_G}{\delta X}$$

$(K + \alpha I)$ makes invertible

Derivatives of gradient terms

$$0 = \frac{\delta E_G}{\delta Y} + KY_t + \alpha(Y_t - Y_{t-1}) \quad \Rightarrow \quad (K + \alpha I)Y_t = \alpha Y_{t-1} - \frac{\delta E_G}{\delta Y} .$$

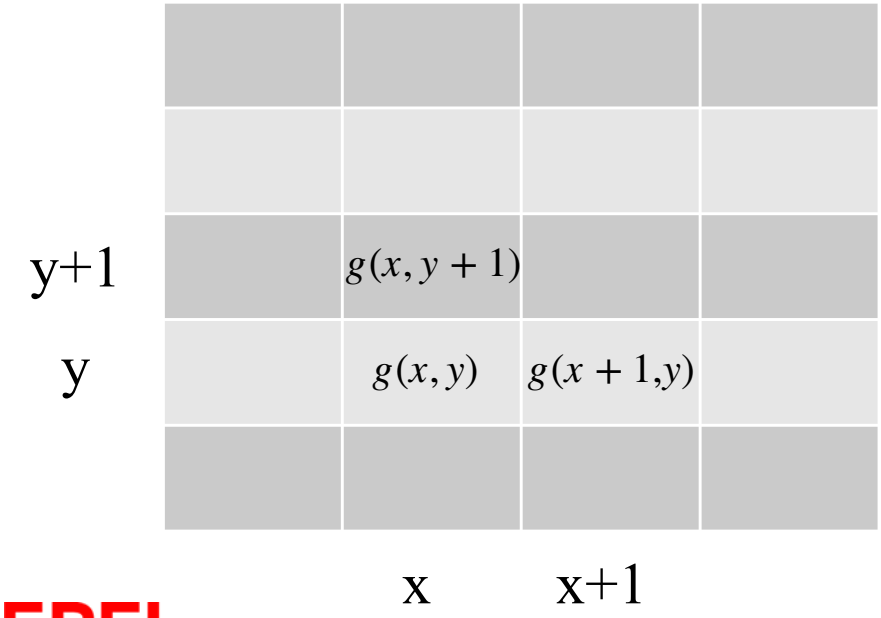
→ Solve two linear equations at each iteration.

Derivatives of the Image Gradient

$$E_G = -\frac{I}{N}\sum_{i=1}^{N} G(x_i, y_i)$$

$$\frac{\partial E_G}{\partial X} = \left[\frac{\partial E_G}{\partial x_1} \quad \cdots \quad \frac{\partial E_G}{\partial x_N}\right] \quad , \quad \frac{\partial E_G}{\partial Y} = \left[\frac{\partial E_G}{\partial y_1} \quad \cdots \quad \frac{\partial E_G}{\partial y_N}\right]$$

$$\frac{\partial E_G}{\partial x_i} = -\frac{I}{N}\frac{\partial G}{\partial x_i}(x_i, y_i) \quad , \quad \frac{\partial E_G}{\partial y_i} = -\frac{I}{N}\frac{\partial G}{\partial y_i}(x_i, y_i)$$

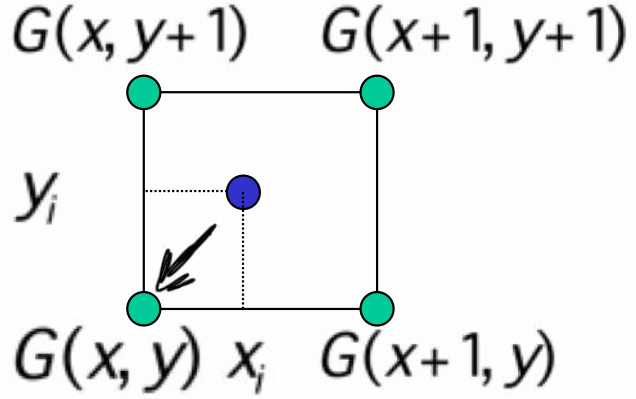


- We have values of g for integer values of x and y.
- But x_i and y_i are not integers.

—> We need to interpolate.

Bilinear Interpolation

You have gradients at 4 pts



$x \in \mathbb{Z}$

$x < x_i < x+1$

$0 < p, q < 1$

The more closer it's to one of these 4 pts, the more influence it gets from weight

$$p = x_i - x$$

$$q = y_i - y$$

bilinear interpolation

$$G(x_i, y_i) = (1-p)(1-q)G(x,y) + (1-p)qG(x,y+1) + p(1-q)G(x+1,y) + pqG(x+1,y+1)$$

$$\frac{\partial G}{\partial x_i} = (1-q)(G(x+1,y) - G(x,y)) + q(G(x+1,y+1) - G(x,y+1))$$

$$\frac{\partial G}{\partial y_i} = (1-p)(G(x,y+1) - G(x,y)) + p(G(x+1,y+1) - G(x+1,y))$$

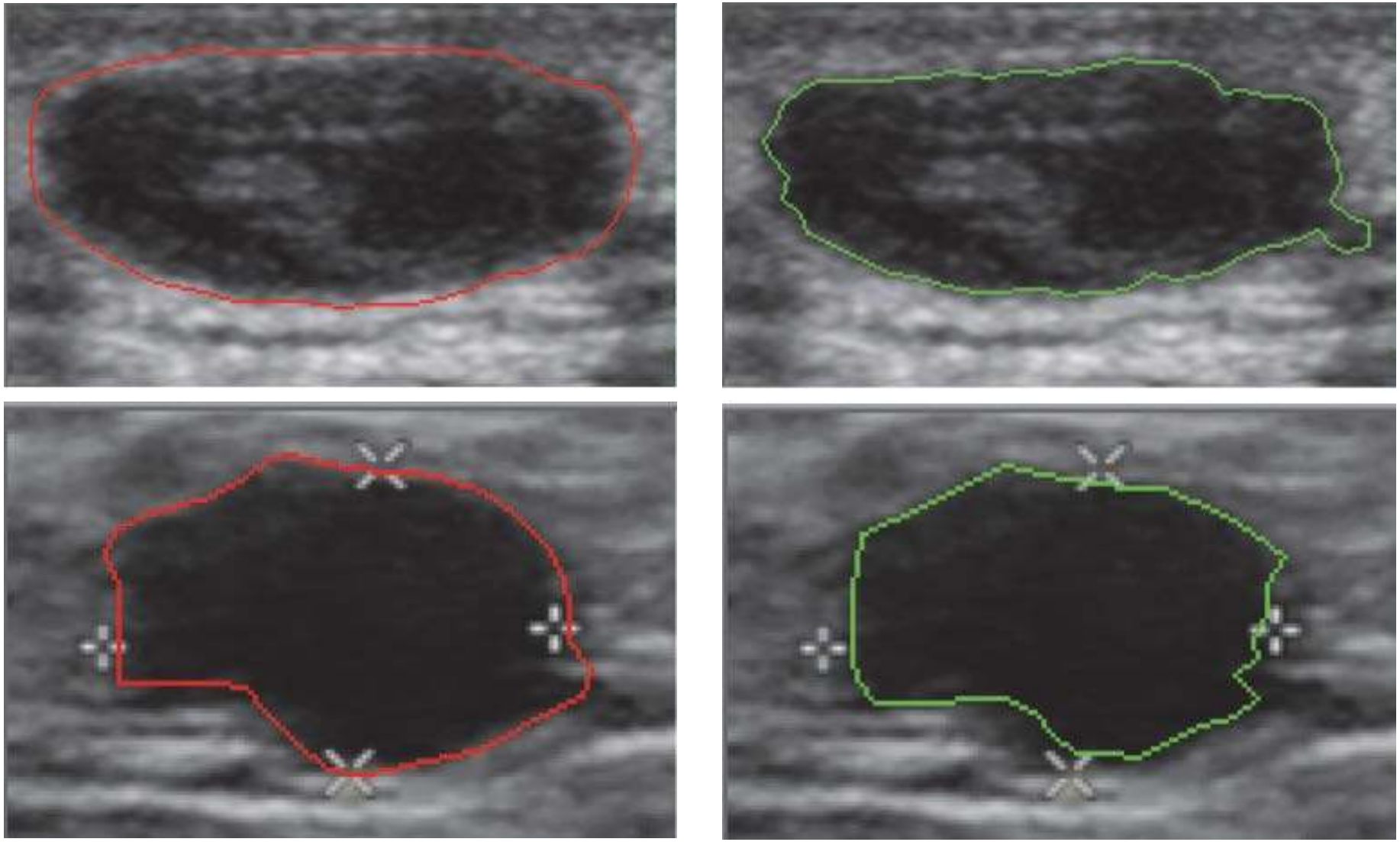
we only have values of G for integers, we can estimate for non-integers and compute derivatives

$$\frac{\partial G}{\partial x_i} = \frac{\partial G}{\partial p}$$

Open and Closed Snakes

Cysts Tumors in Ultrasound Images

↳ extremely noisy



Drawn by the physician.

Refined by the Computer.

with minimum human intervention

Luo et al., 2017

Network Snakes

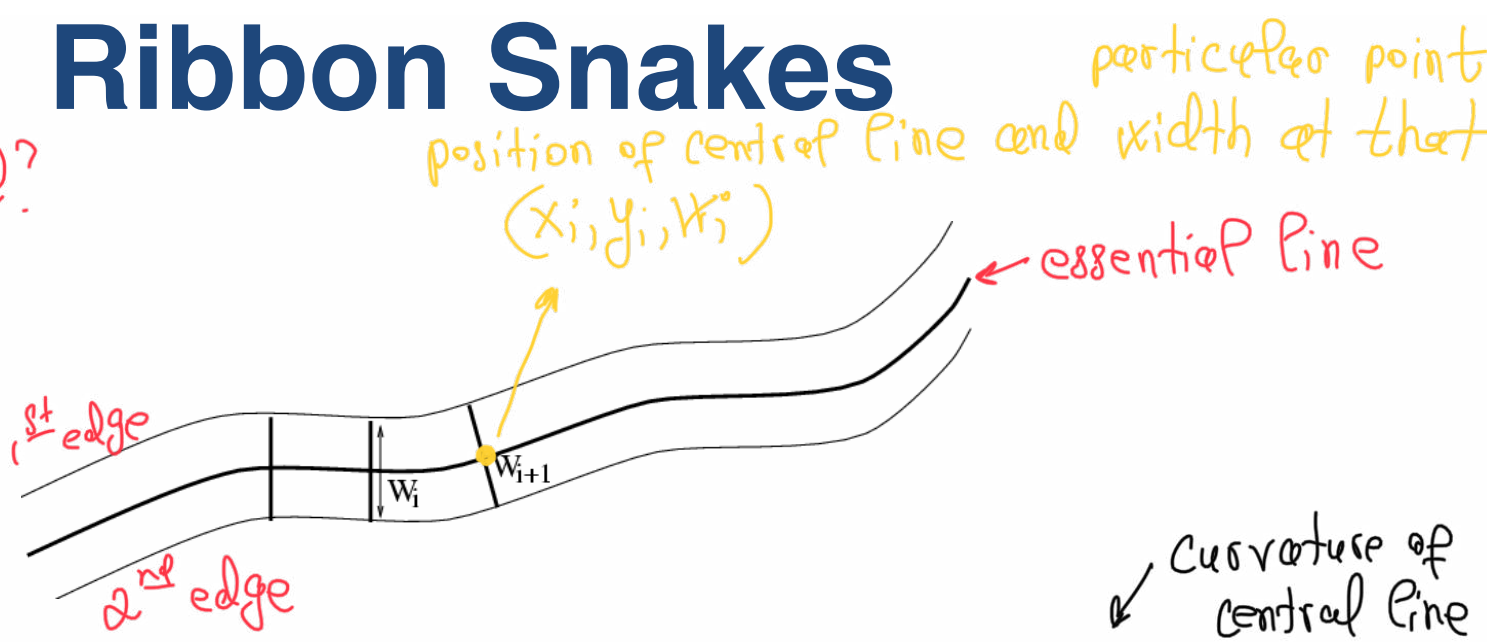
Cadastral maps

You have network of curves → deformed by themselves

--> Updated field boundaries. → who owns what?

Ribbon Snakes

How to represent road?



$$E = E_G + 1/2X^t K X + 1/2Y^t K Y + 1/2W^t K_W W$$

$$W = [w_1, \ldots, w_N]^t$$

$$K_W = \begin{bmatrix} . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & -1 & 2 & -1 & . & . & . & . & . \\ . & . & . & . & -1 & 2 & -1 & . & . & . & . \\ . & . & . & . & . & -1 & 2 & -1 & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . \end{bmatrix}$$

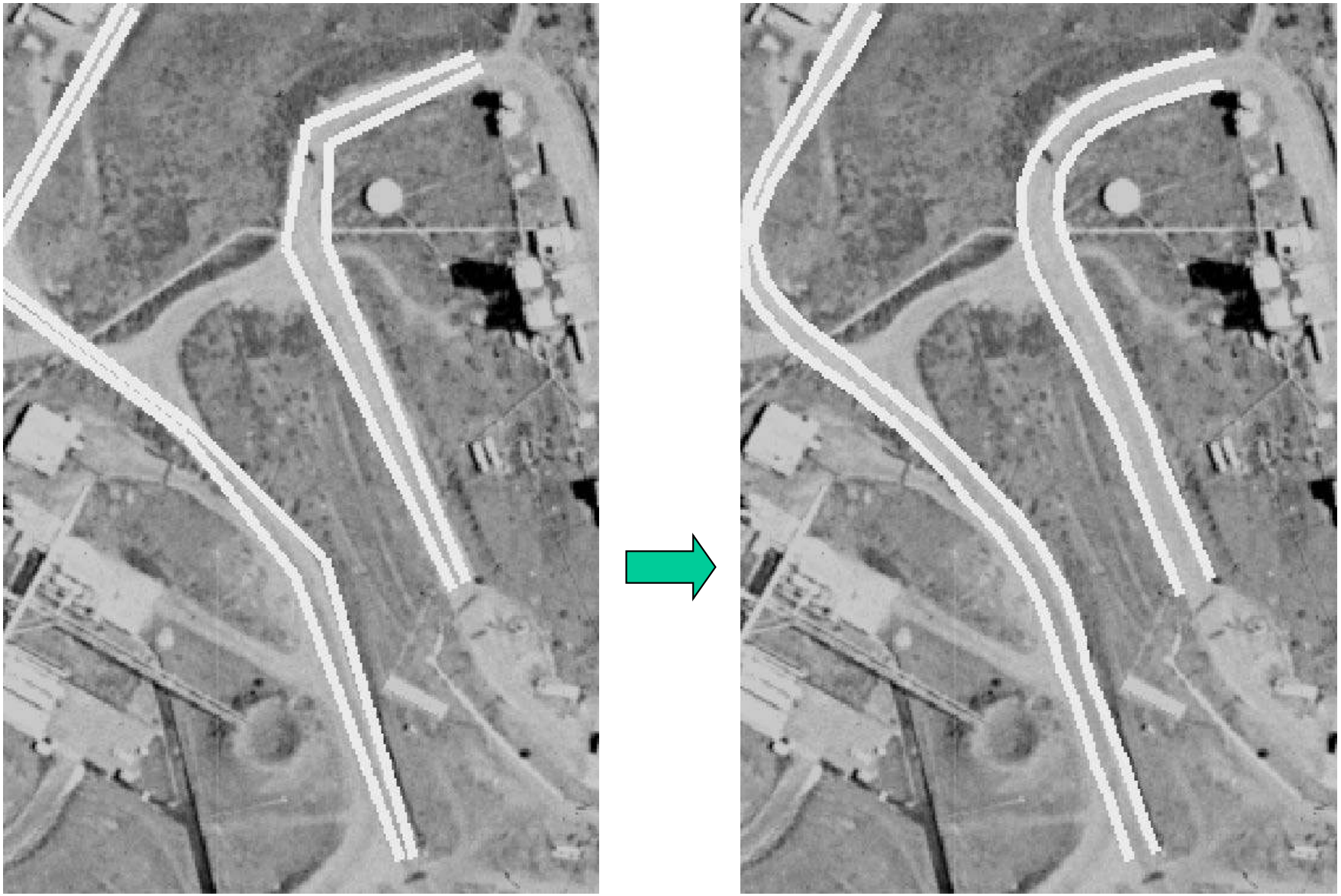
Dynamics Equations

3 eq's to solve at each iteration

$$
\begin{aligned}
(K + \alpha I) X_t &= \alpha X_{t-1} - \frac{\delta E_G}{\delta X} \\
(K + \alpha I) Y_t &= \alpha Y_{t-1} - \frac{\delta E_G}{\delta Y} \\
(K + \alpha I) W_t &= \alpha W_{t-1} - \frac{\delta E_G}{\delta W}
\end{aligned}
$$

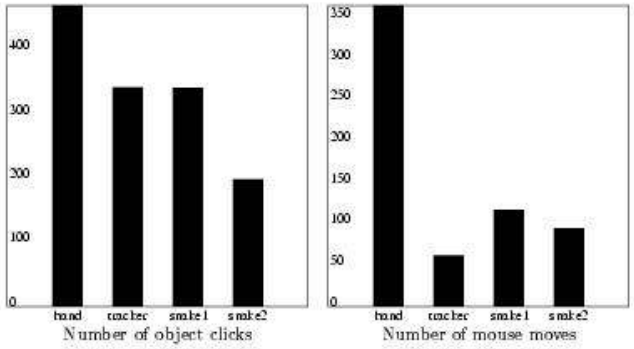
→ Solve three linear equations at each iteration.

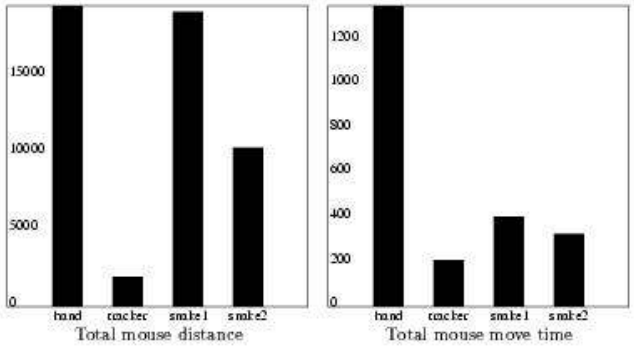
Delineating Roads

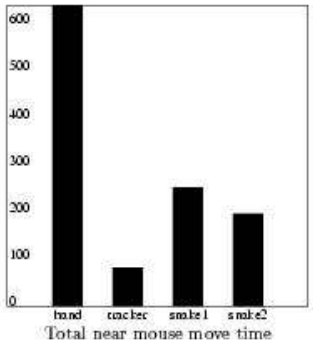


Delineating Roads

Evaluation

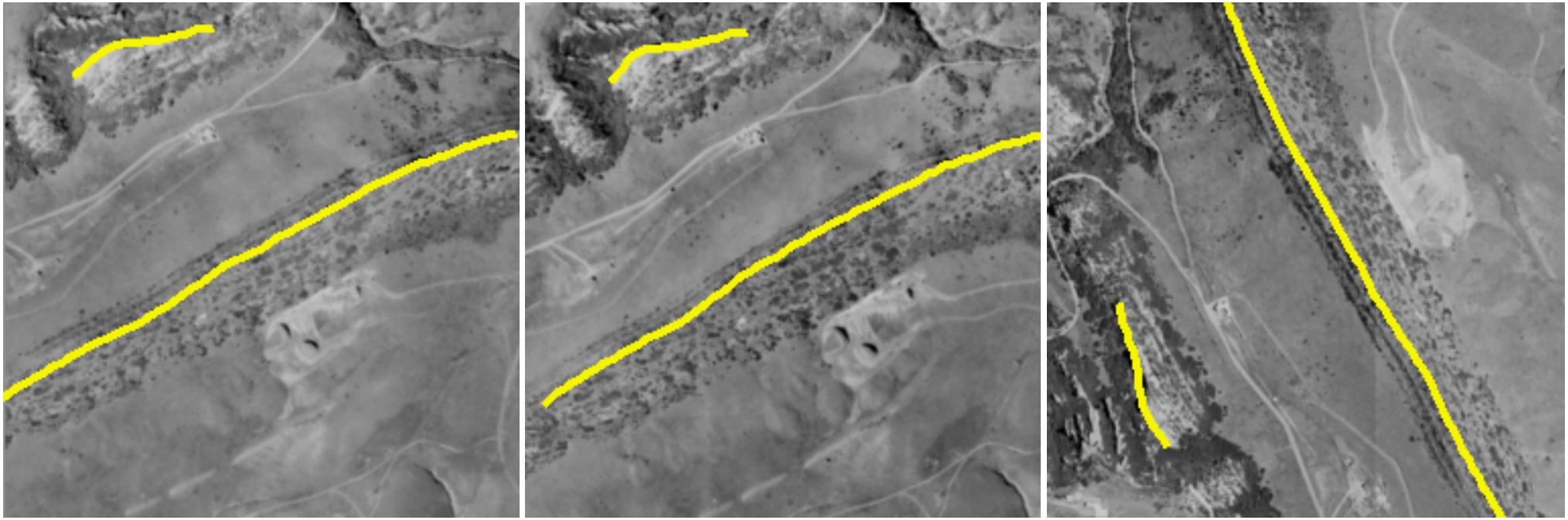




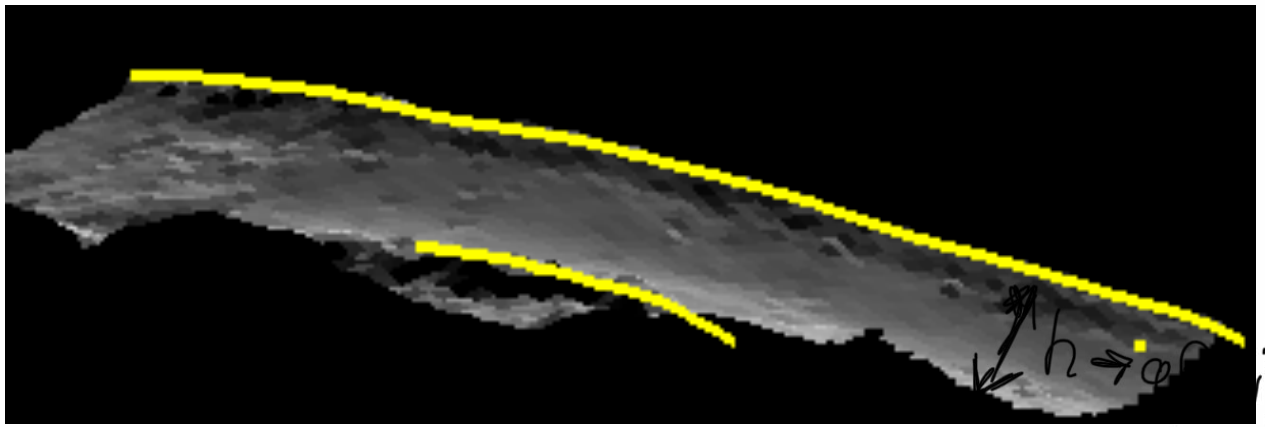


It takes far fewer clicks to trace the roads using semi-automated tools than doing entirely by hand.

Modeling a Ridge Line in 3D



Three different views of the same landscape



Synthetic side view.

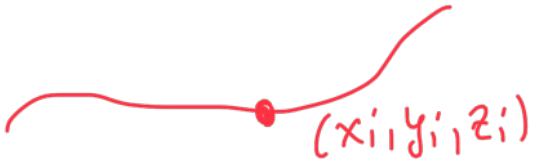
Modeling a Building in 3D

Sketch roughly the contours

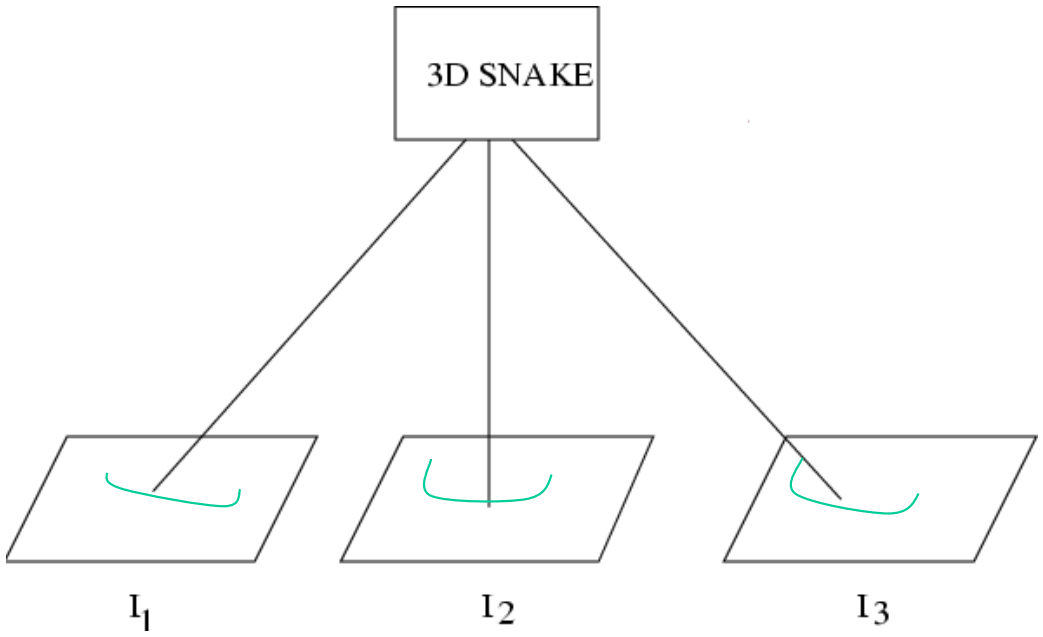
Ribbon Snakes optimizing it

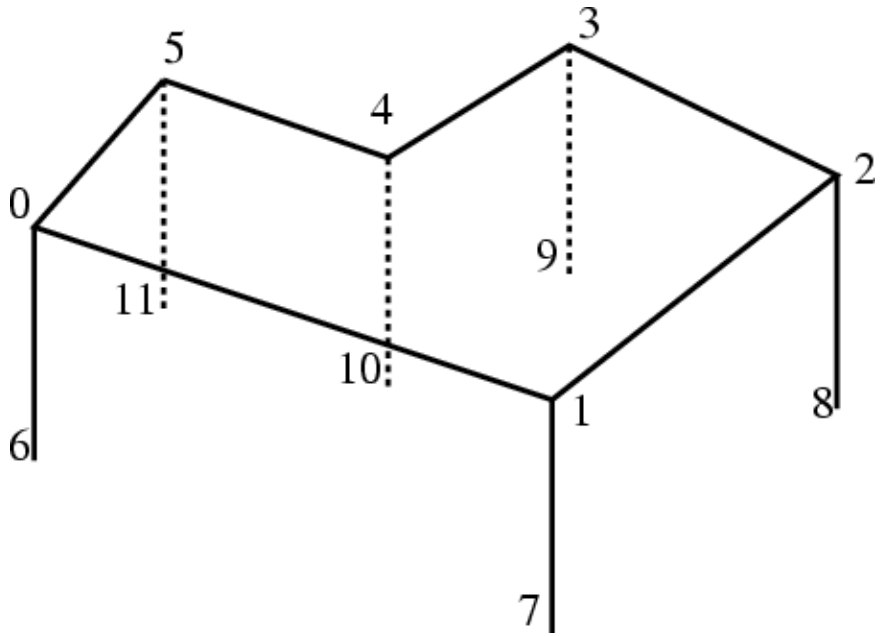
⇒ finding exact heights of the building

3D Snakes



For rigid line





Smooth 3—D snake

Rectilinear 3—D snake

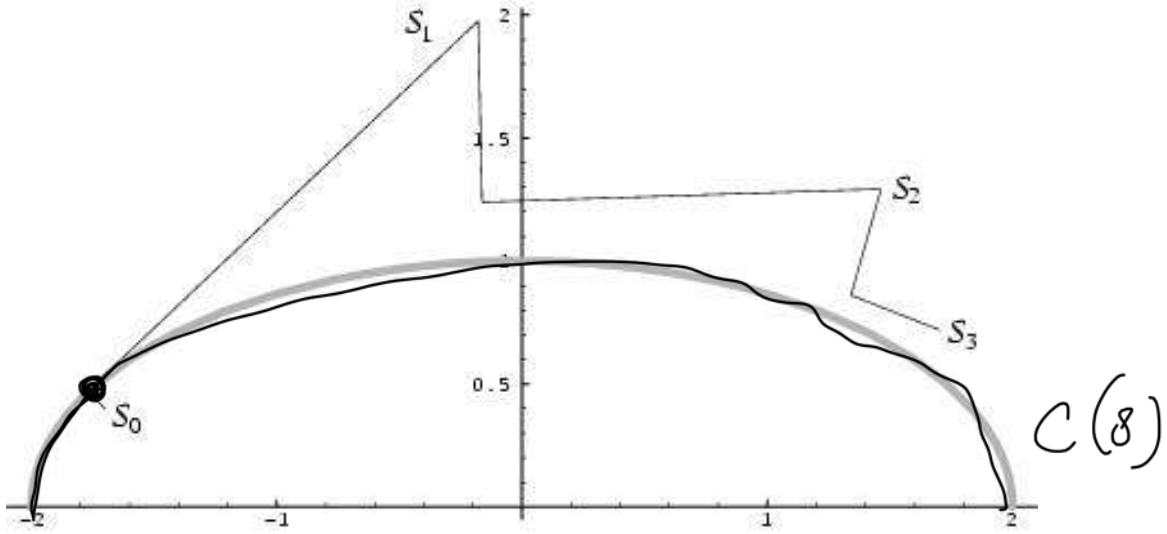
Using the camera models, you project into 2D

Dynamics Equations

$$
\begin{aligned}
(K + \alpha I) X_t &= \alpha X_{t-1} - \frac{\delta E_G}{\delta X} \\
(K + \alpha I) Y_t &= \alpha Y_{t-1} - \frac{\delta E_G}{\delta Y} \\
(K + \alpha I) Z_t &= \alpha Z_{t-1} - \frac{\delta E_G}{\delta Z}
\end{aligned}
$$

→ Solve three linear equations at each iteration.

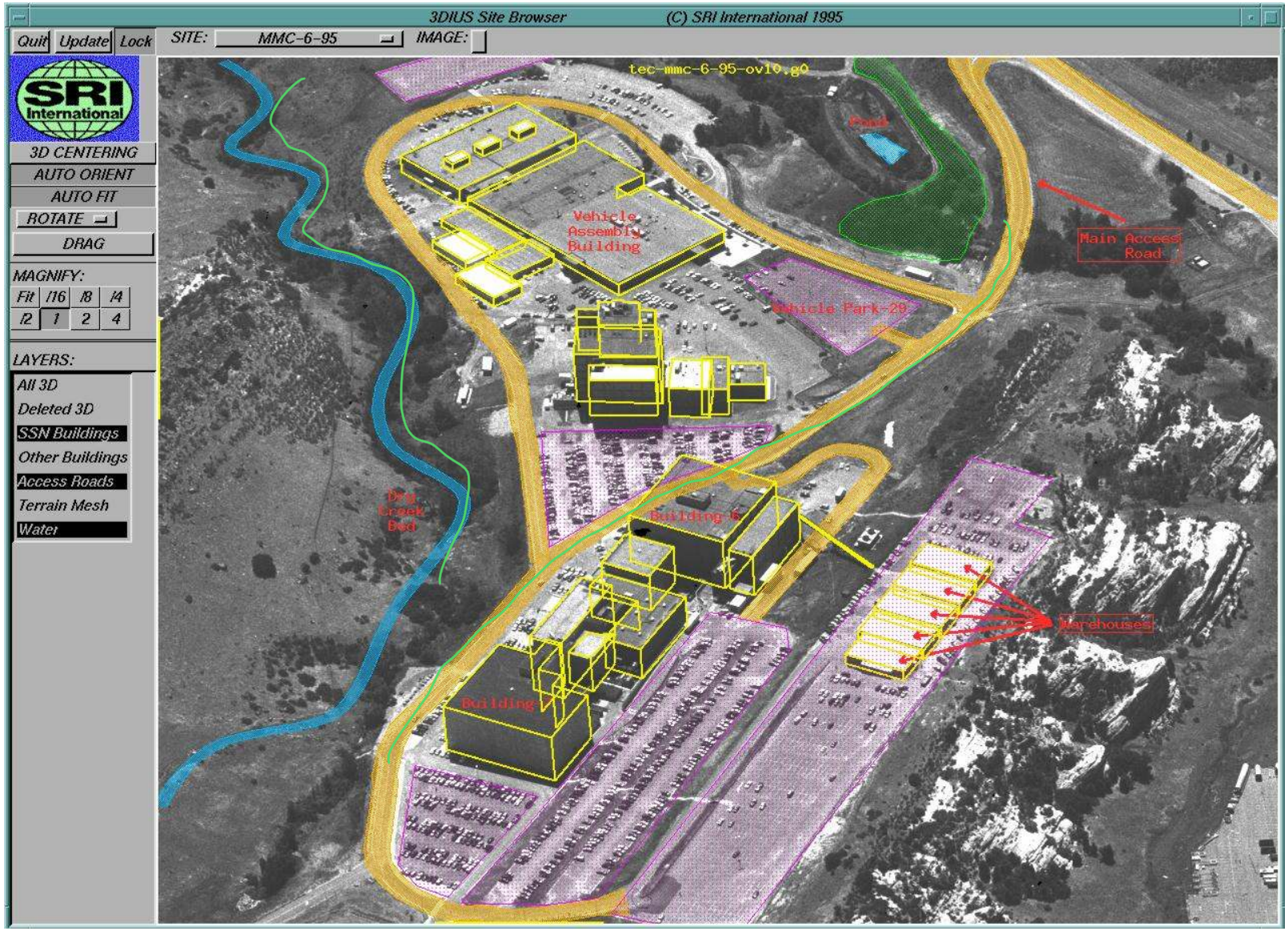
Constrained Optimization



- Minimize F(S) subject to C(S) = 0

Site Modeling (1996)

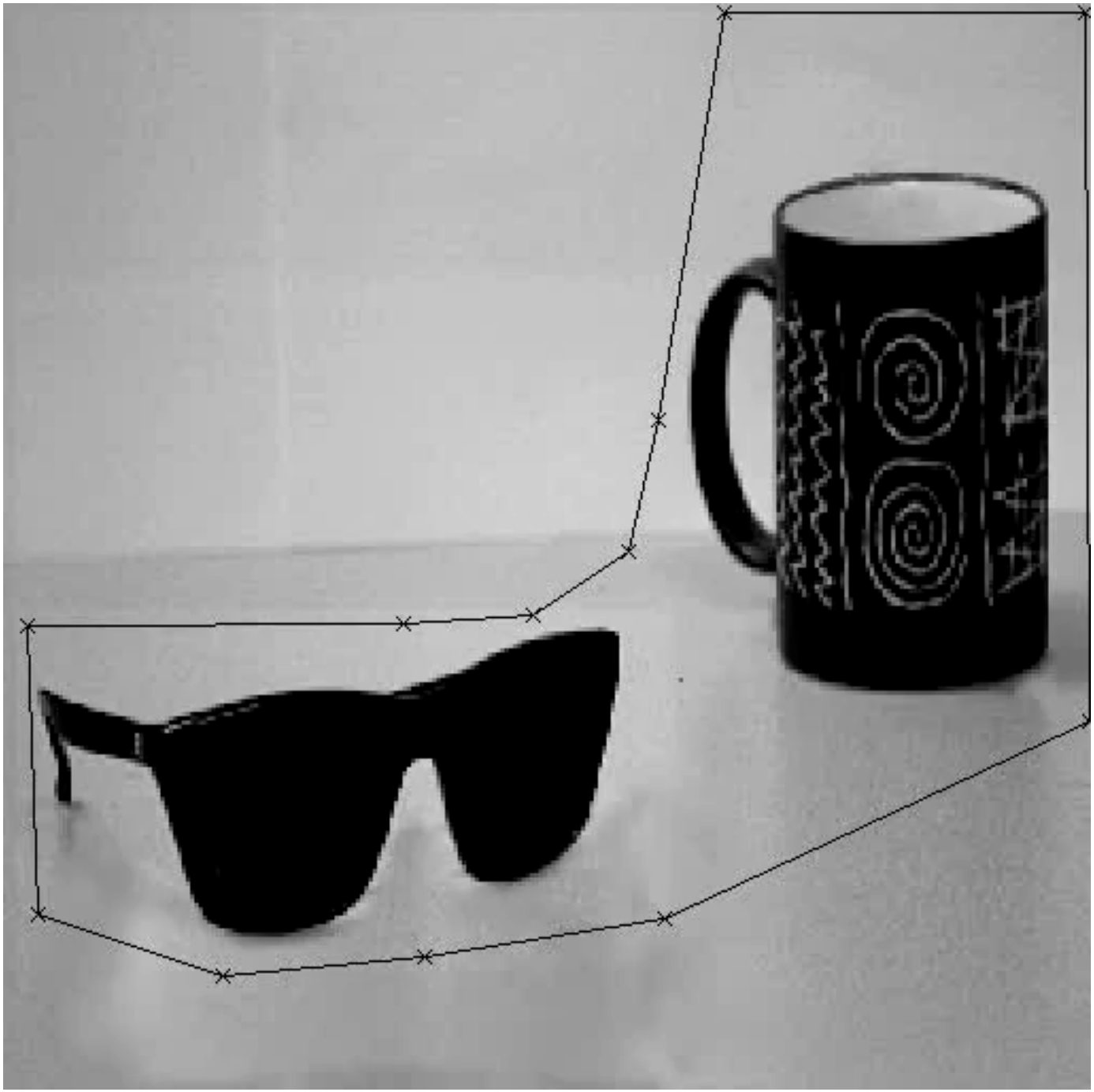
Constraints => Roads on the terrain, roads flow downhill, and consistent



Site Modeling (2019)

Reconstructed images/shapes

Level Sets



Implicit vs Explicit

Curve Evolution

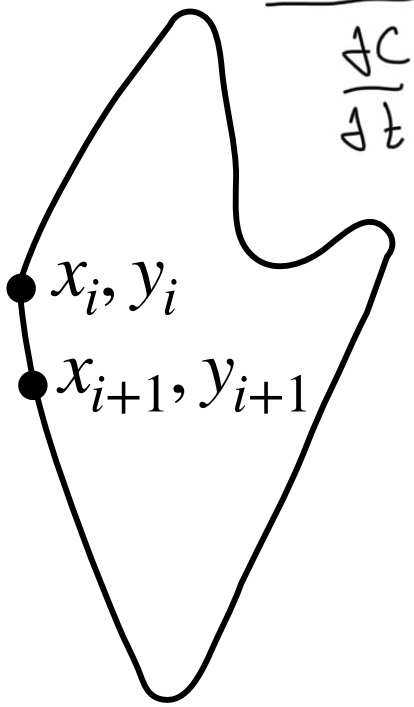
Generic formulation:

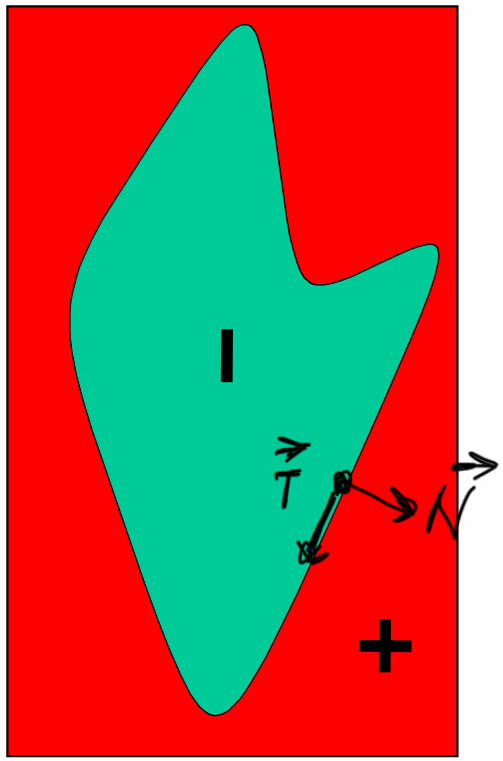
$$\frac{\partial C}{\partial t} = \alpha(s,t)\vec{T} + \beta(s,t)\vec{N}$$

Reparameterization:

$$\frac{\partial C}{\partial t} = \beta(s,t)\vec{N}$$

displacement along the normal





Issues: 1) Involves curve sampling and resampling

2) Curvature estimates are numerically unstable

Special cases: $\beta = \pm 1$ ← curve will grow/shrink

$\beta = \beta_0 - \beta_1 K$ ← curvature

$$z = \Phi(x, y)\ ,$$

$$z > 0 \text{ outside,}$$

$$z < 0 \text{ inside,}$$

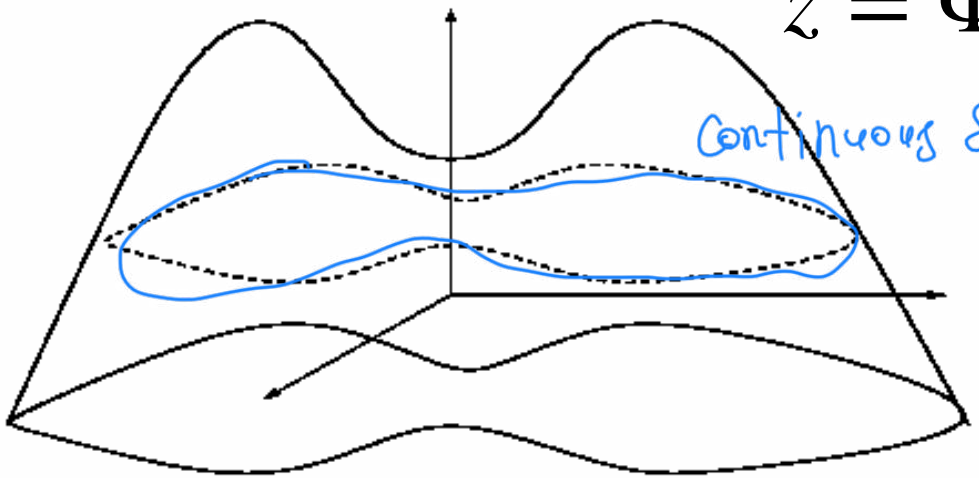
—> Consider the curve as the zero level set of a surface.

Topology Changes are Possible

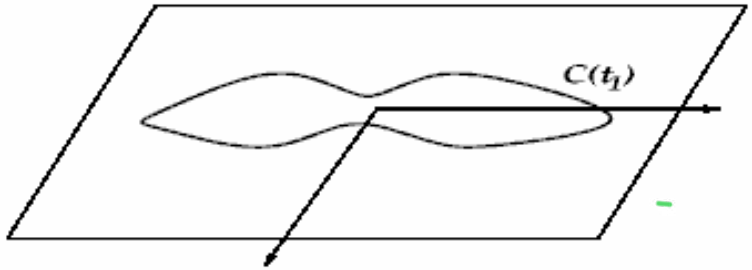
Surface is where $\Phi(x,y)=0$

$$z = \Phi(x, y, t_1)$$

Continuous surface

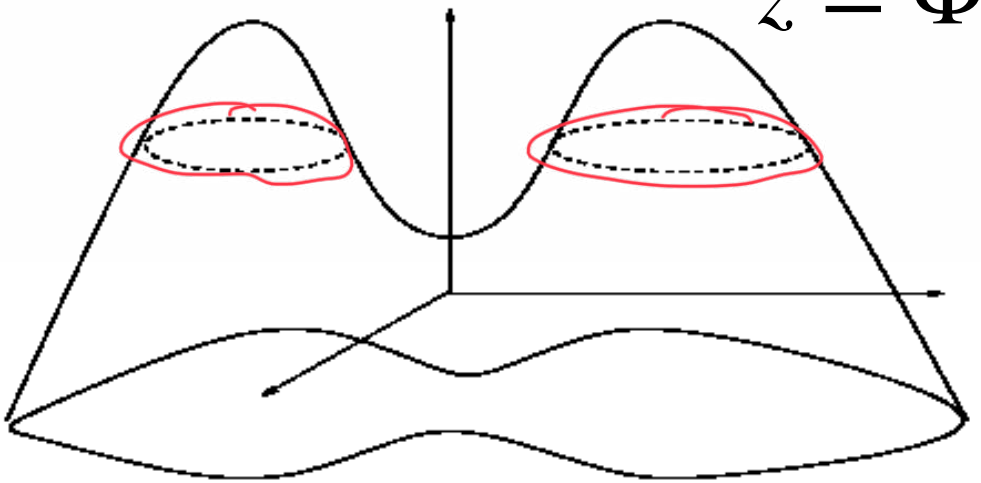


Both of them are zero-crossings of the implicit field ✓

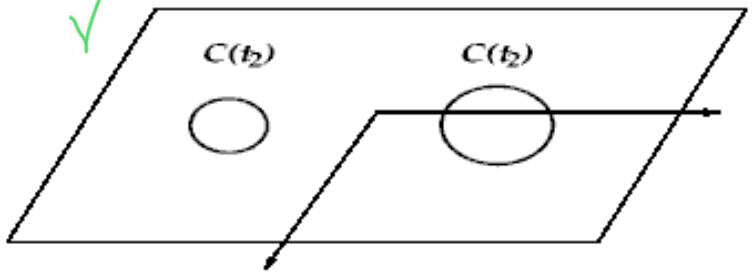


$$z = \Phi(x, y, t_2)$$

When Φ is deformed (with some field Φ)



✓



Curve Evolution

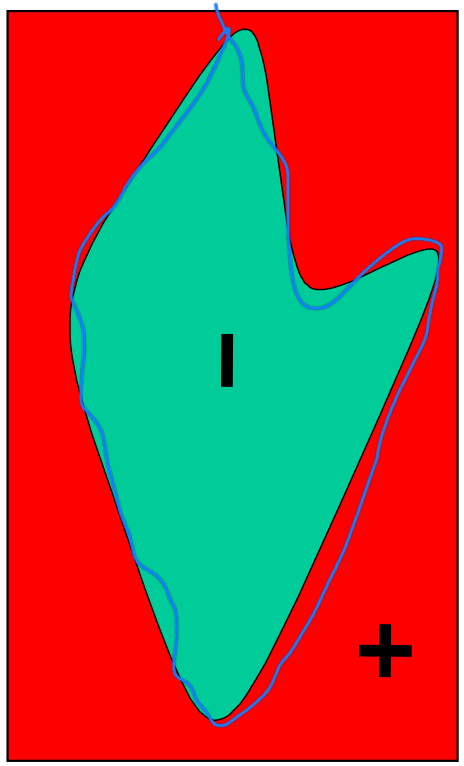
Level Sets

$\Phi(x, y) = 0$

Consider the curve as the zero level set of the surface:

field

$$z = \Phi(x, y, t)$$



Evolution equation:

$$0 = \Phi_t + \beta(\kappa)\,|\nabla\Phi|$$

$$\text{where } \kappa = \frac{\Phi_{xx}\Phi_y^2 - 2\Phi_{xy}\Phi_x\Phi_y + \Phi_{yy}\Phi_x^2}{\Phi_x^2 + \Phi_y^2}$$

curvature

$\beta(\kappa)$ is the speed at which the surface deforms.

Level Set Smoothing

Smoothing occurs when $\beta(\kappa) = -\kappa$

Desirable properties:

- Converges towards circles.
- Total curvature decreases.
- Number of curvature extrema and zeros of curvature decreases.

Relationship with Gaussian smoothing:

- Analogous to Gaussian smoothing of boundary over the short run, but does not cause self-intersections or overemphasize elongated parts.
- Can be implemented by Gaussian smoothing the characteristic function of a region.

Shape Recovery

If you want snake-like behaviour

Evolution equation: $$0 = \Phi_t + \beta(\kappa)\,|\nabla\Phi|$$

where: $$\beta(\kappa) = k_I(1-\epsilon\kappa)$$

$$k_I = \frac{1}{1+\nabla I}$$

If there are a lot of gradients $\Rightarrow$ k_I-small $\Rightarrow$ $\beta(\kappa)$-small $\Rightarrow$ curve will almost not move (Φ_t)

no gradients $\Rightarrow$ curve will move more

gradient of the image

→ Expansion stops at the boundaries.

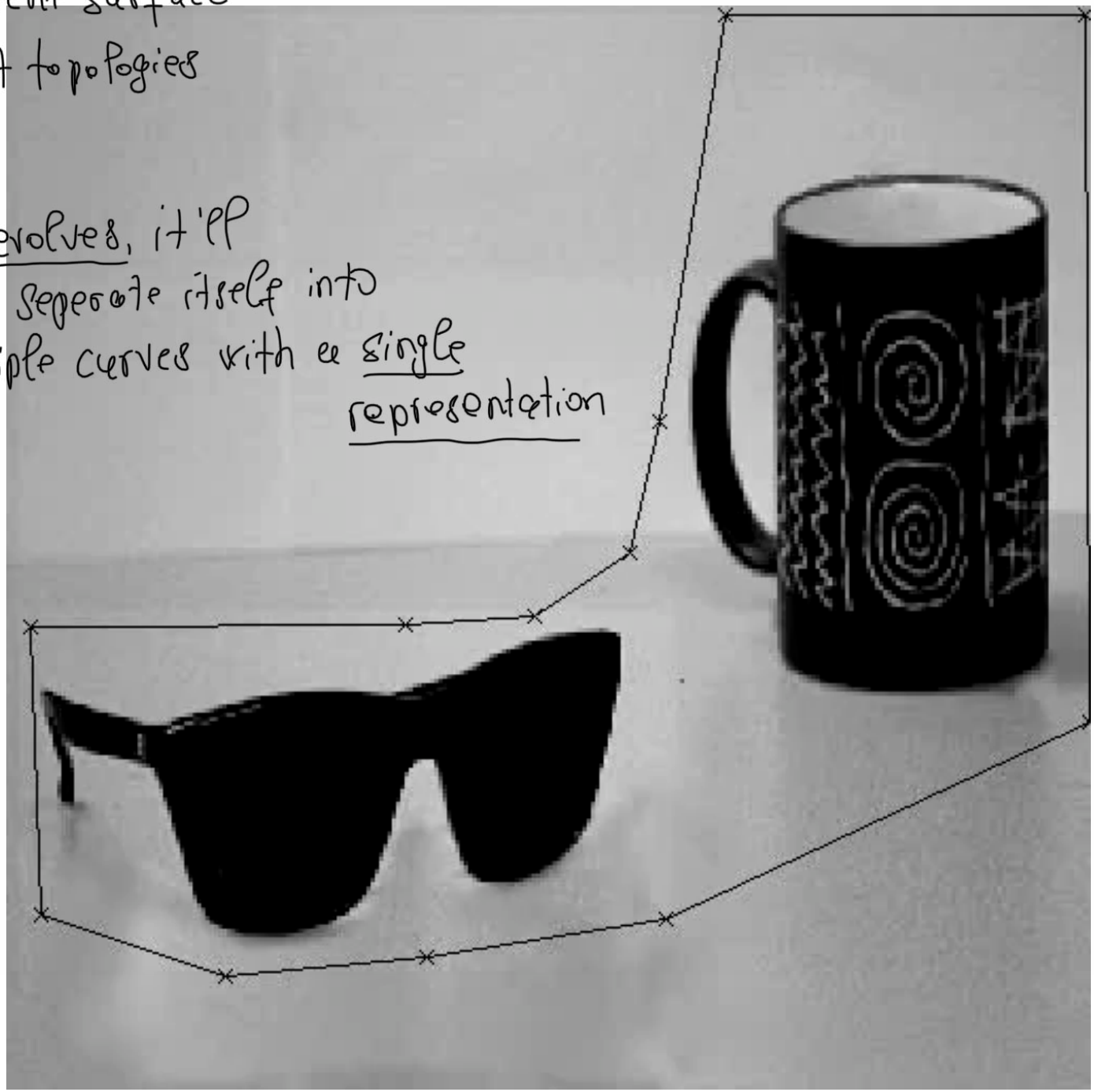
Level Sets

Stops by the boundaries

Level Sets

with implicit surface represent. you can represent surface with different topologies

As the field evolves, it'll seperate itself into multiple curves with a single representation



Techniques

Semi-Automated Techniques:

- Dynamic programming
- Deformable Models

Fully Automated Techniques:

- Hough transform
- Graph Based Approaches

User not intervened

Finding Lines

Input:

- Canny edge points.
- Gradient magnitude and orientation.

Output:

Find

- All straight lines in image.

Hough Transform

simple

Given a parametric model of a curve:

(line defined by 2 parameters)

- Map each contour point onto the set of parameter values for which the curves passes through it.

voting scheme

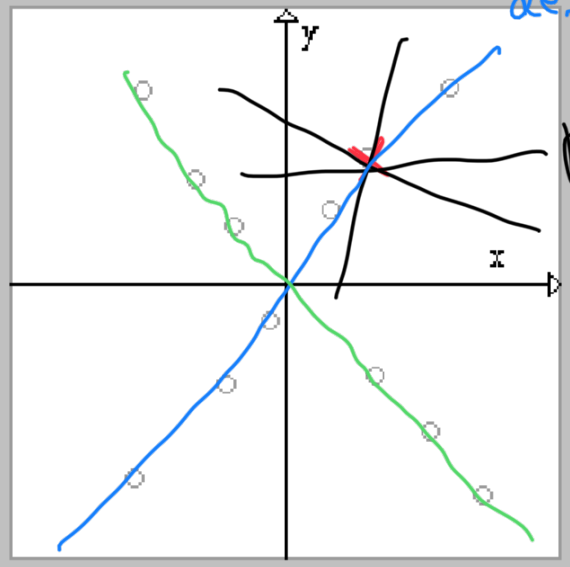
voting for curves

- Find the intersection for all parameter sets thus mapped.

Accumulate votes

Voting Scheme

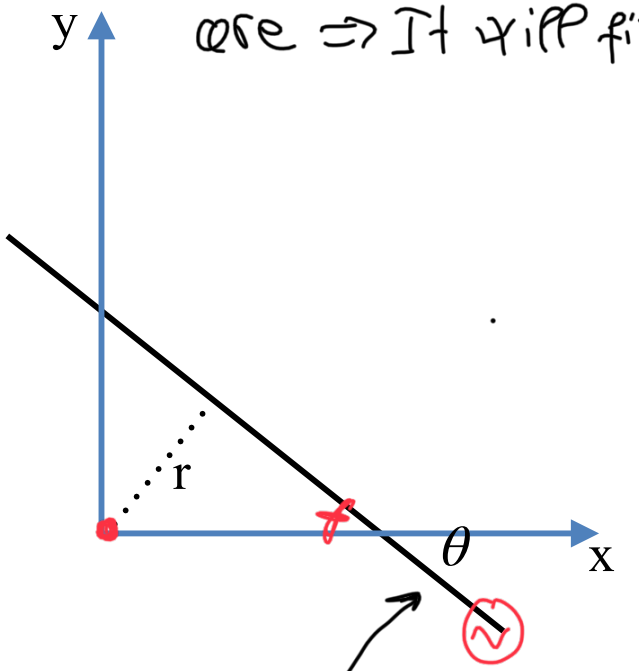
define this line



pencil of lines

Accumulative array

Strength: You don't need to specify how many lines there are ⇒ It will find by itself

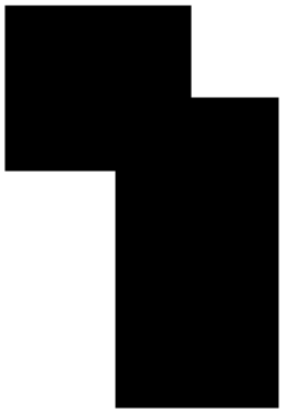


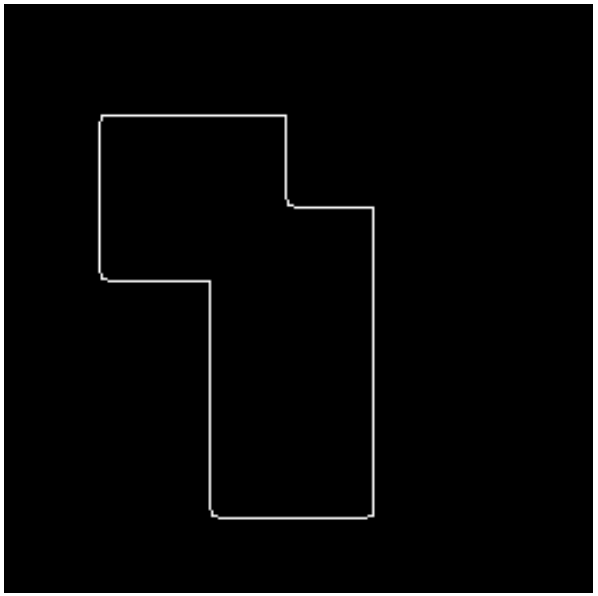
(x,y) points are all the pts that lie on that line parametrized by r, θ

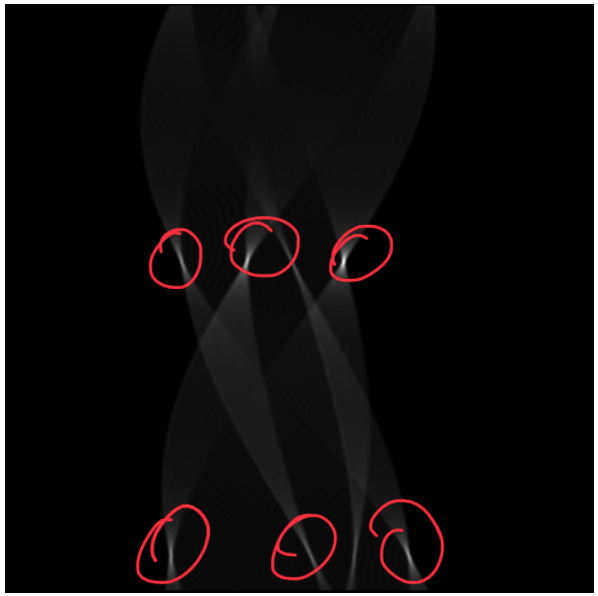
Reverse: Given (x,y) with pencil of lines, draw figure on the left to find all (r, θ)

$$x\cos(\theta) + y\sin(\theta) = r \;,\; 0 \leq \theta \leq \pi$$

Synthetic Lines







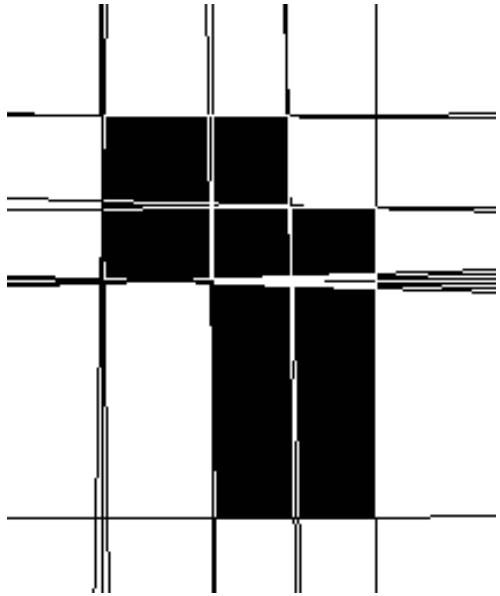
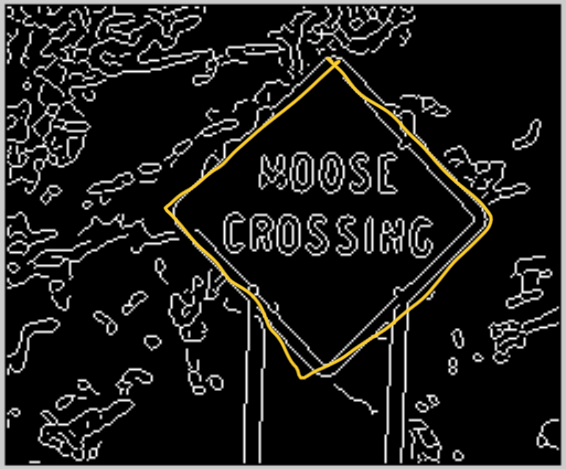


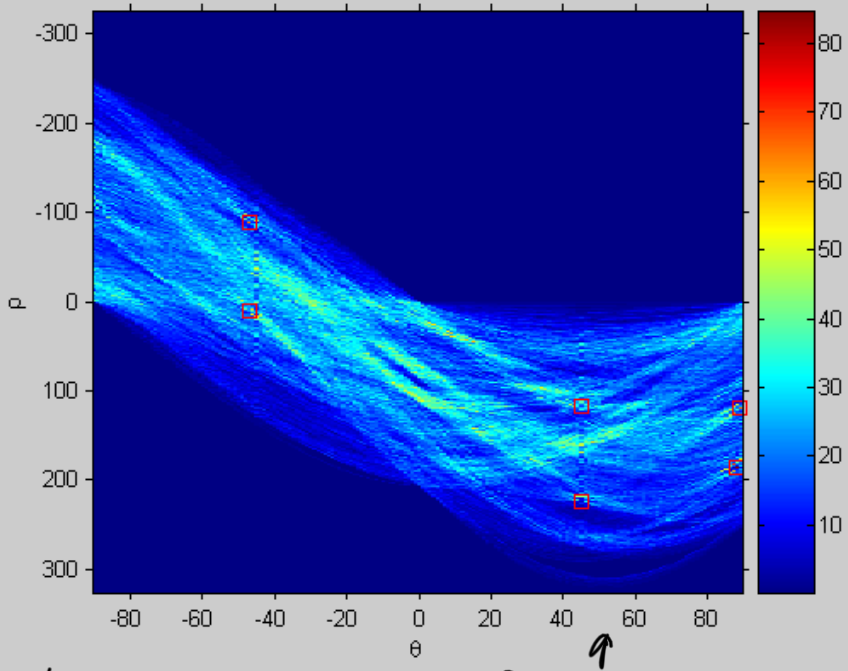
Image Contours Accumulator Lines

Once the contour points are associated to individual lines, you can perform least squares fitting.

Real Lines

Line segments





Votes in the accumulator array

find contours and cut it off

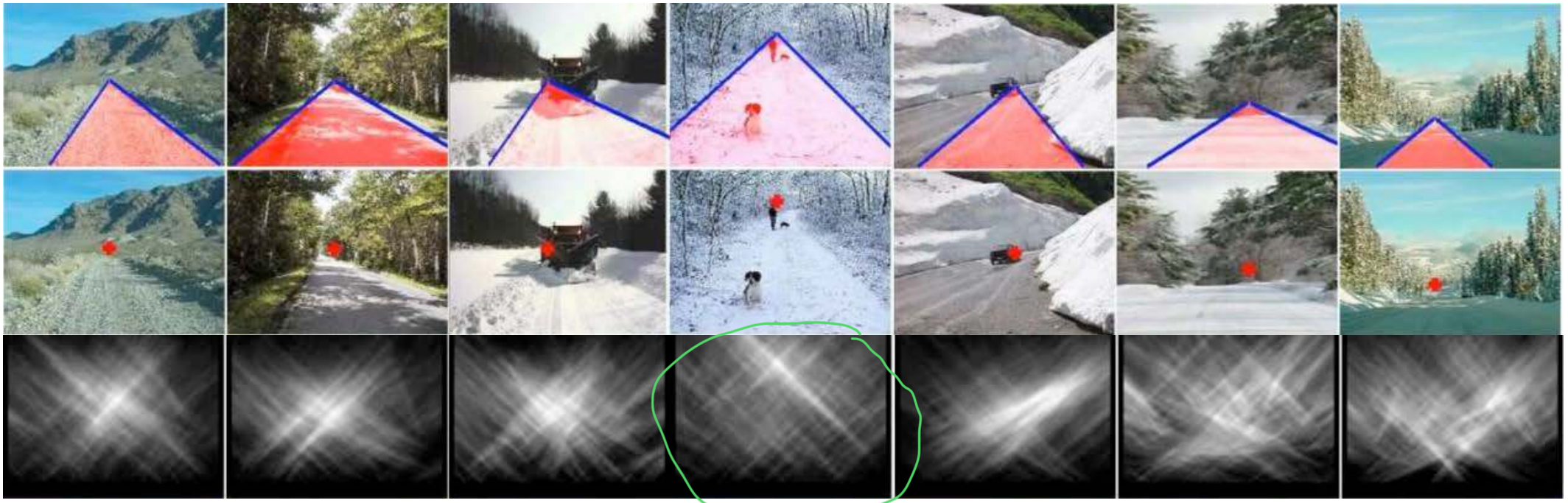
→ ∀ maxima associates with a straight line (infinite length)

(they vote for all possible lines that go through themselves)

Road Lines


Road markings
(voted for the same line)
→ not necessarily
continuous
a lane
in the road

Road Edges ⇒ you will look for a maxima



? contour votes ⇒ reveals edges of the road

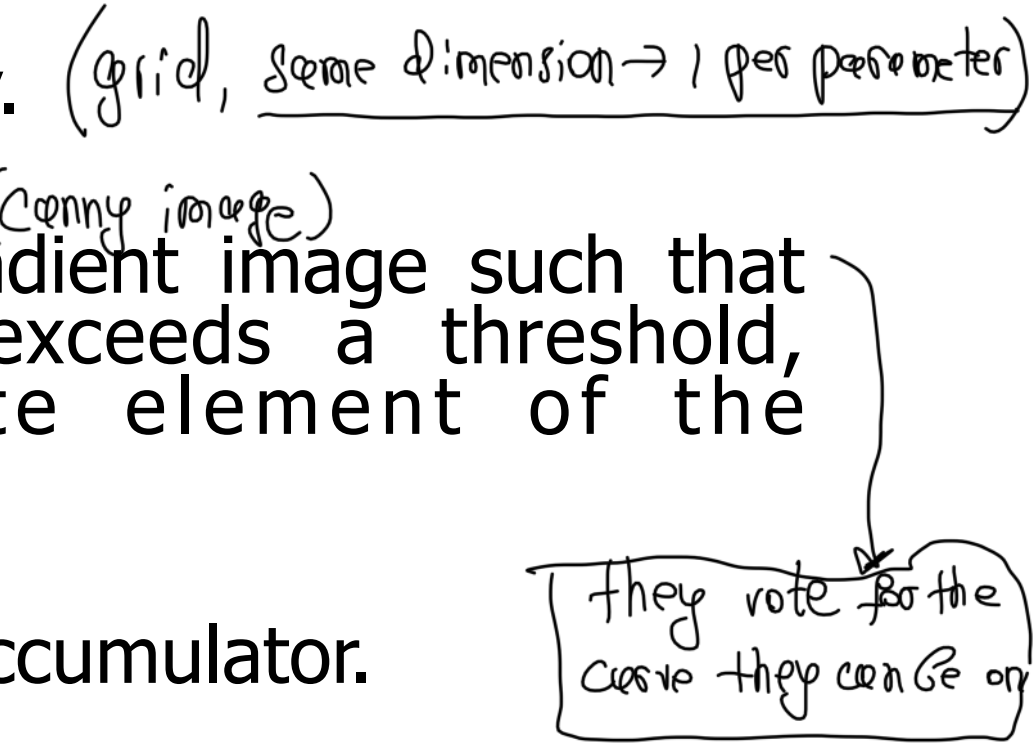
accumulator array on the scene

↳ we are looking for the peak which corresponds to (r, θ)

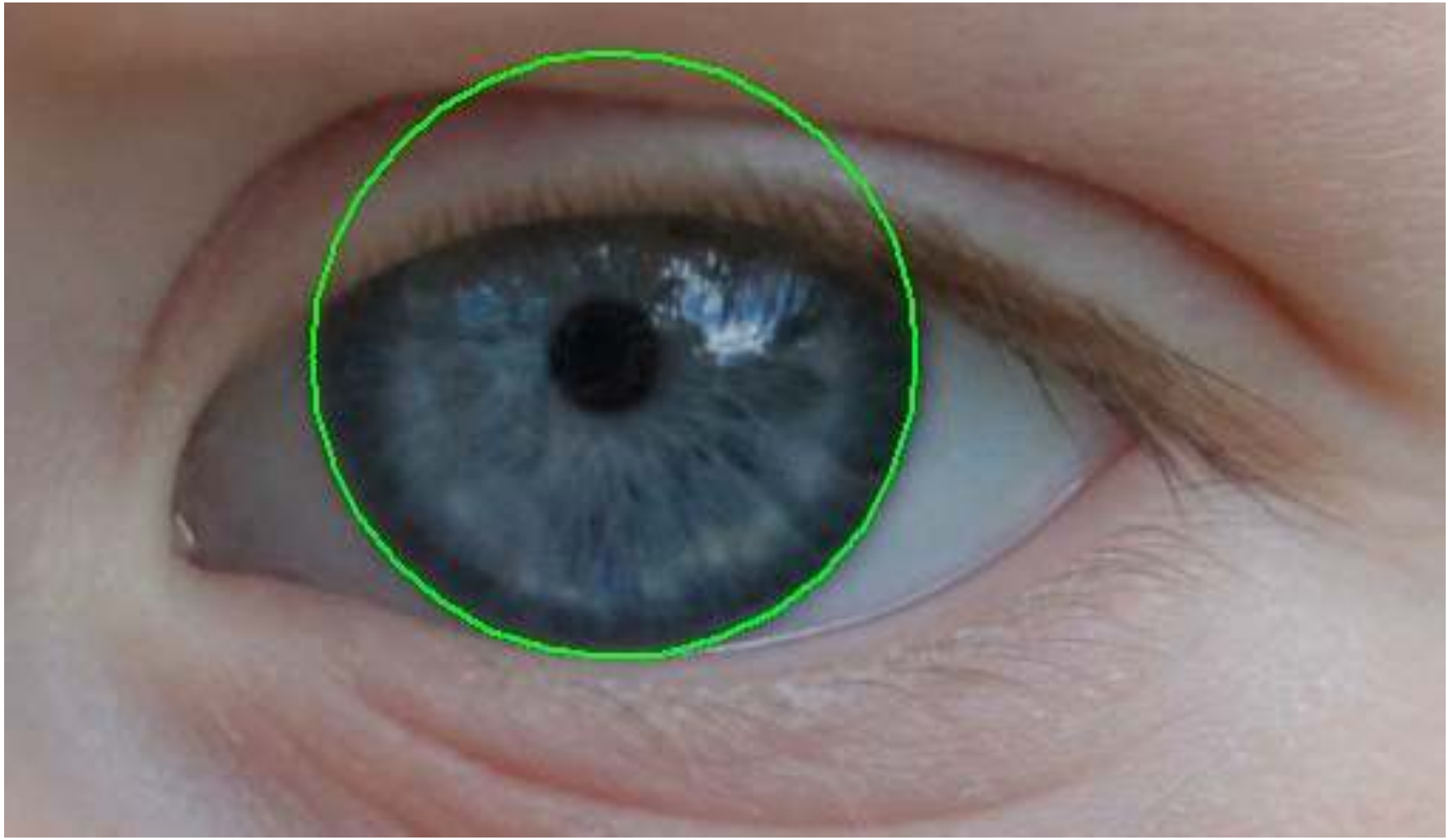
In projection, they meet at vanishing pt

Generic Algorithm

- Quantize parameter space with 1 dimension per parameter.
- Form an accumulator array.
- For each point in the gradient image such that the gradient strength exceeds a threshold, increment appropriate element of the accumulator.
- Find local maxima in the accumulator.

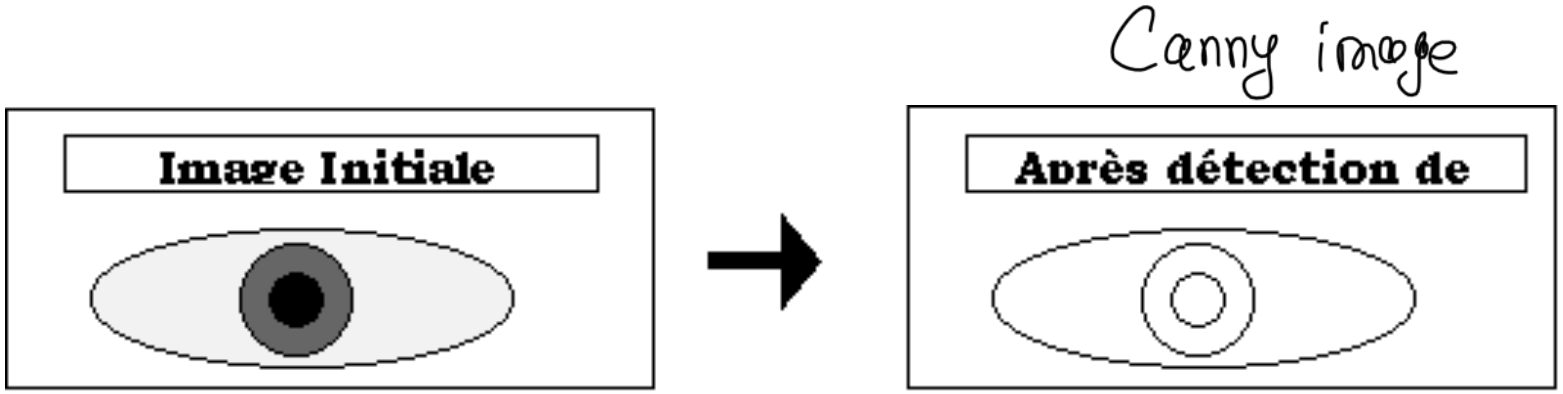


Iris Detection

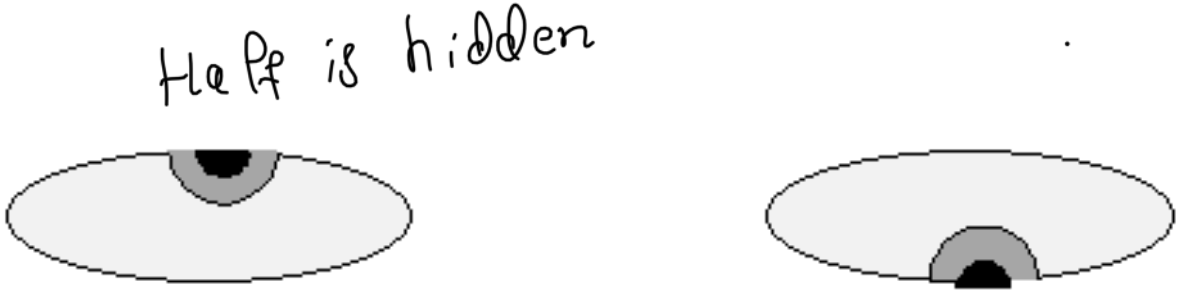


Occlusions

In theory:



In practice:



Circle Detection

$\forall \theta$ with $0 < \theta < 2\pi$

Circle of equation:

$$\begin{aligned} x &= x_0 + r\cos(\theta) \\ y &= y_0 + r\sin(\theta) \end{aligned}$$

Therefore: If you know point (x,y) (belonging to an edge)

$$\begin{aligned} x_0 &= x - r\cos(\theta) \\ y_0 &= y - r\sin(\theta) \end{aligned}$$

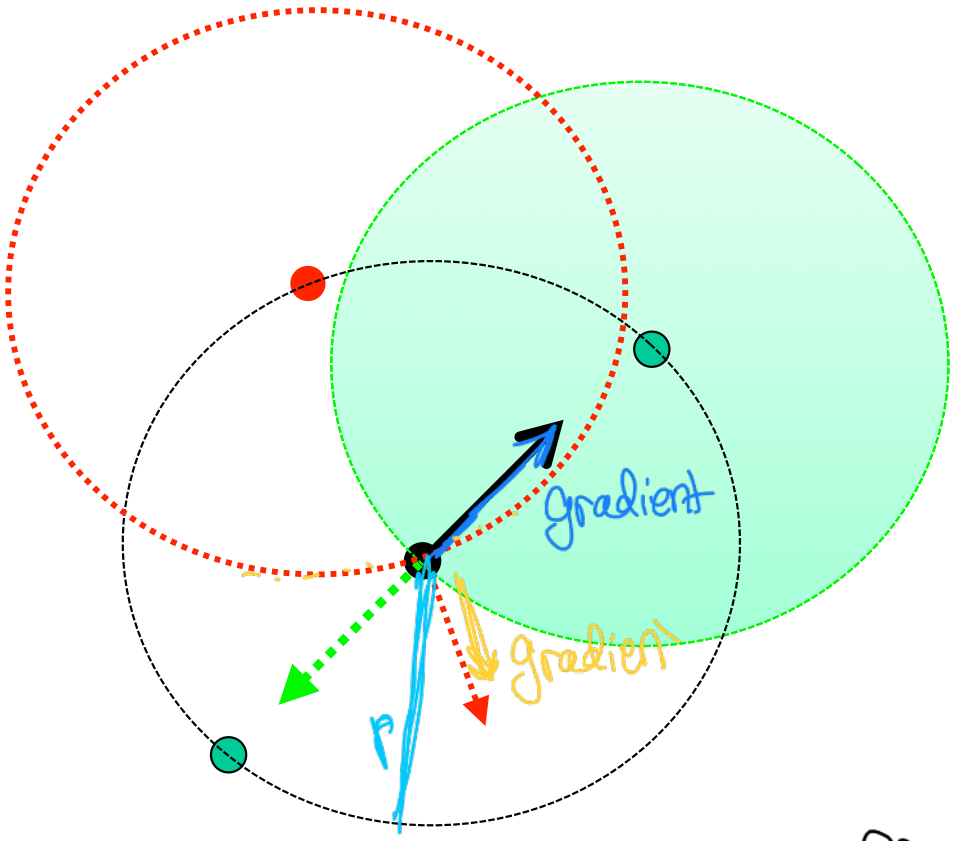
center of circle

for fixed r

for different r's, you get different centers

$\Rightarrow (x,y)$ votes on the circle as well

Gradient Orientation



In practice, we don't know where circle is => only gradient

Black-dotted

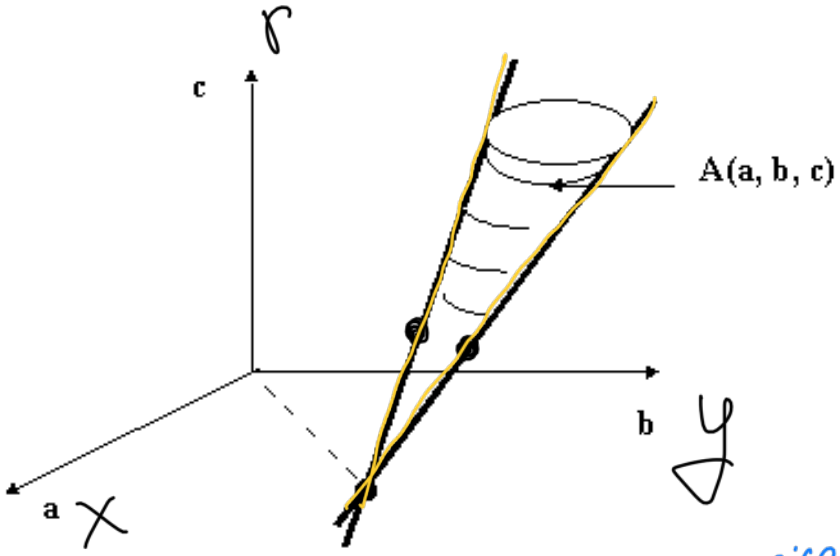
Can vote either along the entire circle or only at two points per value of the radius.

direction

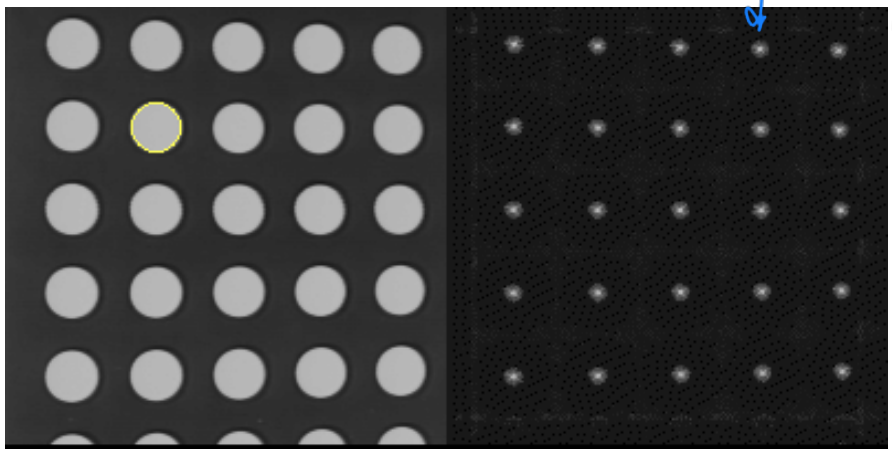
Because you take into consideration the gradient

Simple Image

Voting scheme:



Result:

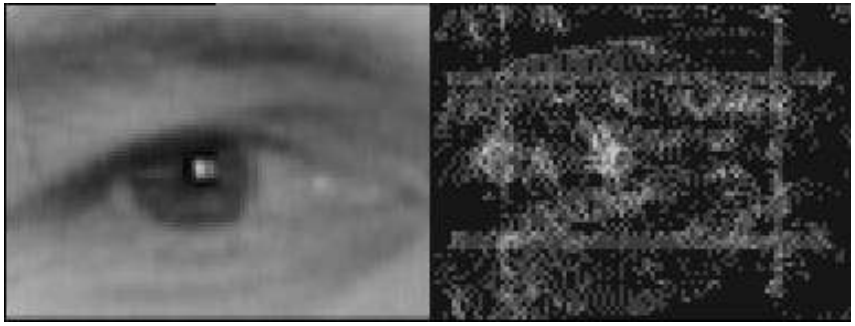


center of circles
nice maxima for the
Accumulator
array

point with vote in 3D-parameter space

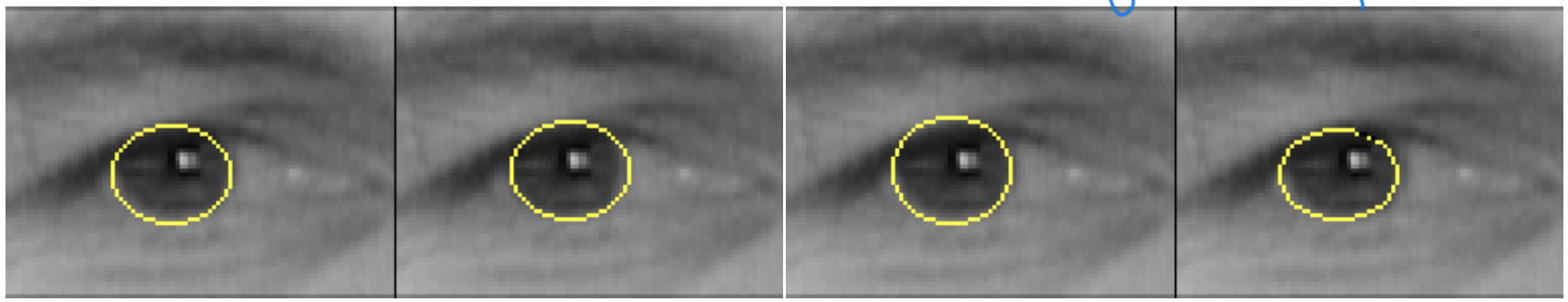
Eye Image

Image and accumulator:



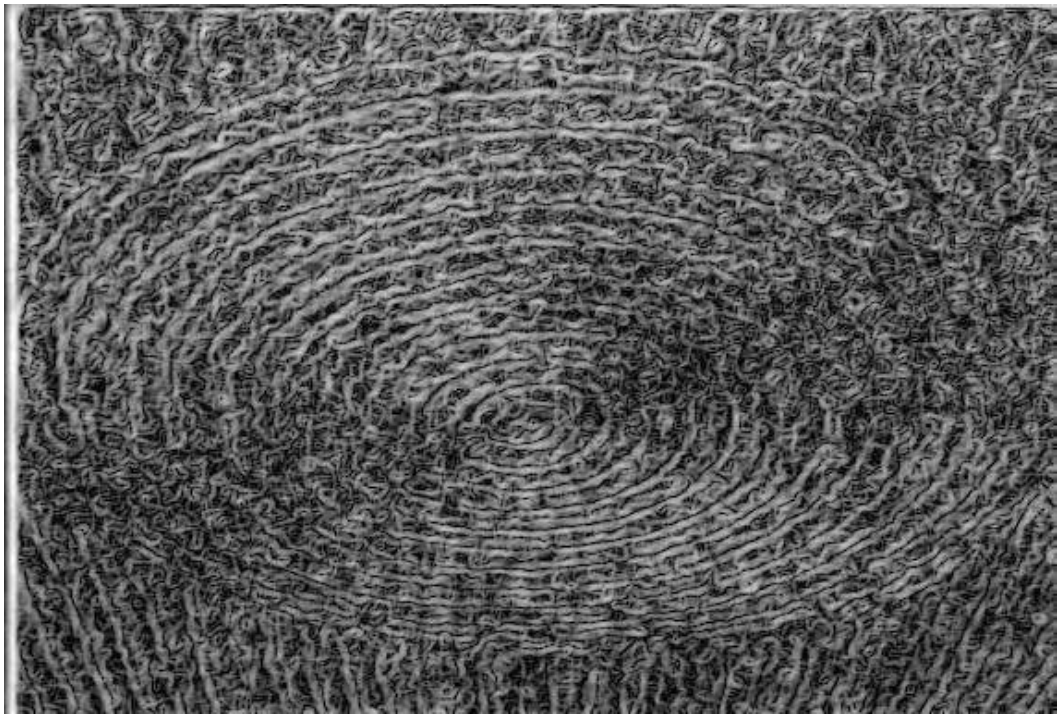
Best four candidates:

Finding maxima from accumulator array



Ellipses

Gradient of the image → useful



Delineation does not look right

Ellipse Detection

Ellipse of equation:

$$\begin{aligned} x &= x_0 + a\cos(\theta) \\ y &= y_0 + b\sin(\theta) \end{aligned}$$

Therefore:

$$\begin{aligned} x_0 &= x - a\cos(\theta) \\ y_0 &= y - b\sin(\theta) \end{aligned}$$

If you have a point (x, y), for a given (a, b) $\Rightarrow$ it will vote for the center (x_0, y_0)

2D - accumulator array

Gradient Orientation

For each ellipse point:

$$\frac{dx}{d\theta} = -a\sin(\theta)$$

$$\frac{dy}{d\theta} = \quad b\cos(\theta)$$

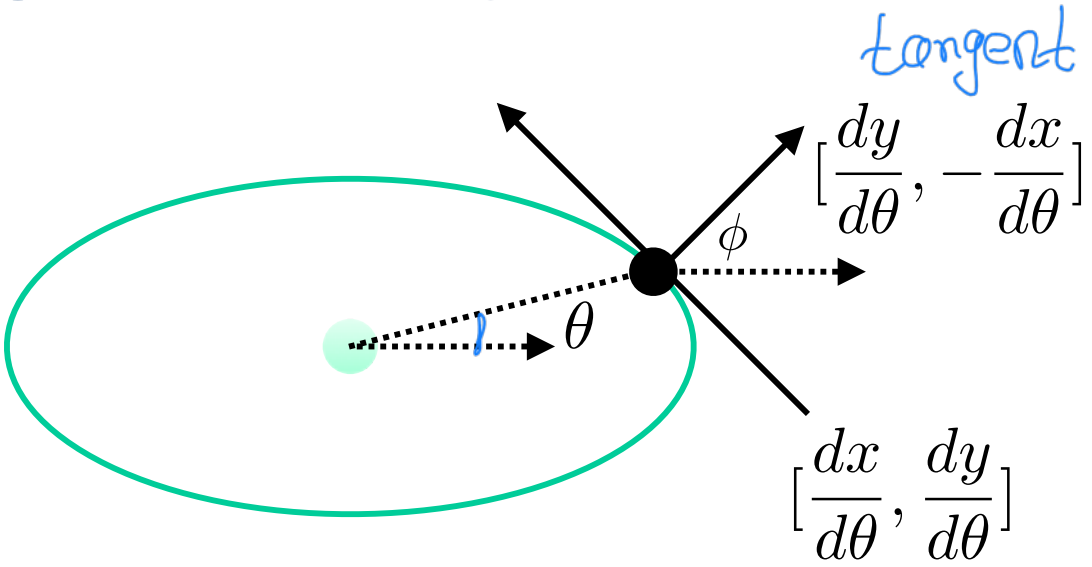
$$\phi = \text{atan}(-\frac{dx}{d\theta}, \frac{dy}{d\theta})$$

$$= \text{atan}(a\sin(\theta), b\cos(\theta))$$

$$= \text{atan}(\frac{a}{b}\tan(\theta))$$

$$\tan(\phi) = \frac{a}{b}\tan(\theta)$$

$$\Rightarrow \theta = \text{atan}(\frac{b}{a}\tan(\phi))$$

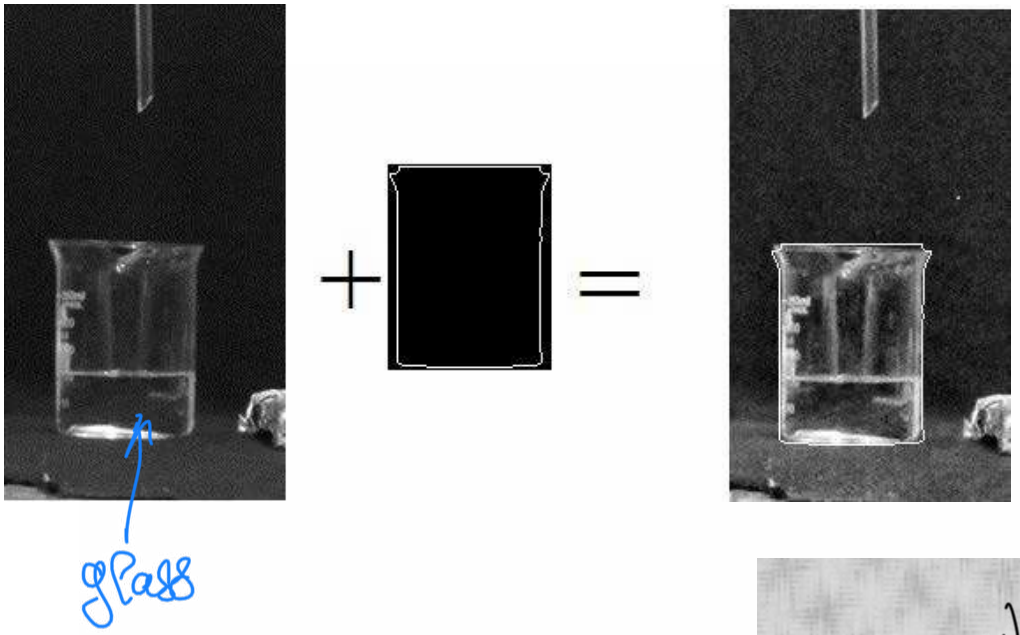


direction
vote for the center where tangent will be compatible

If ellipse can be rotated => 5D-accumulator array

The accumulator need only be incremented for this θ.

Generalized Hough

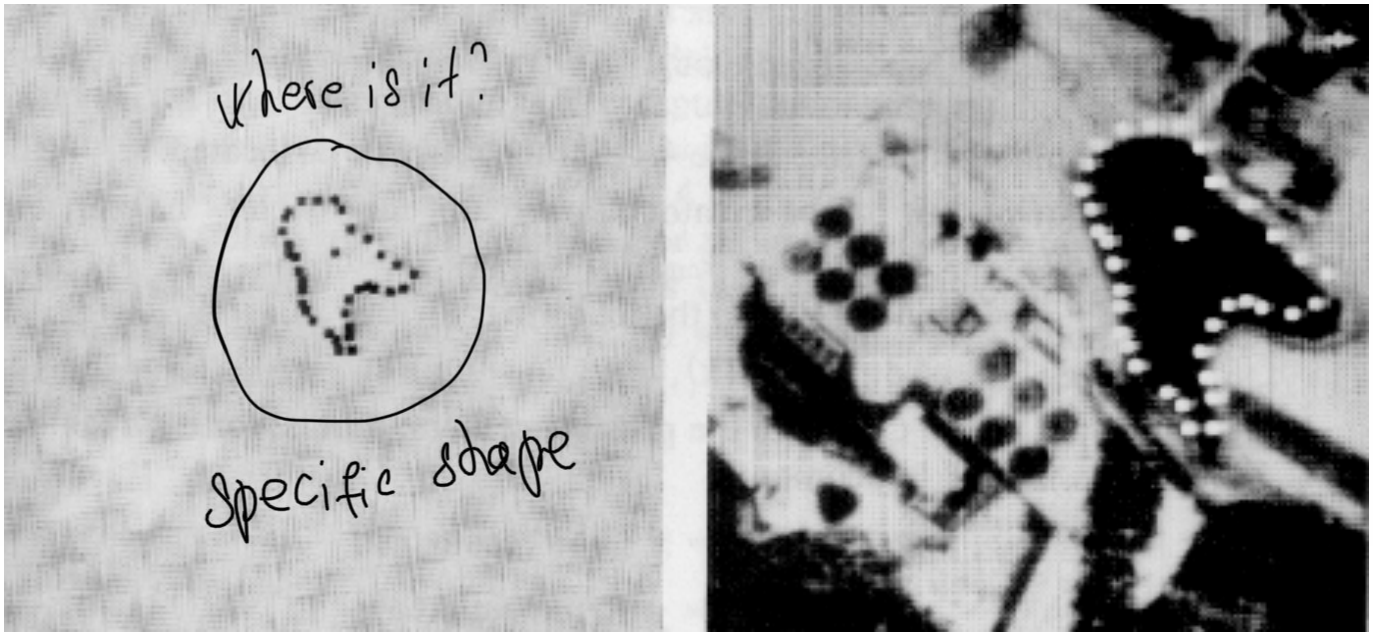


Draw contours of a becher and then

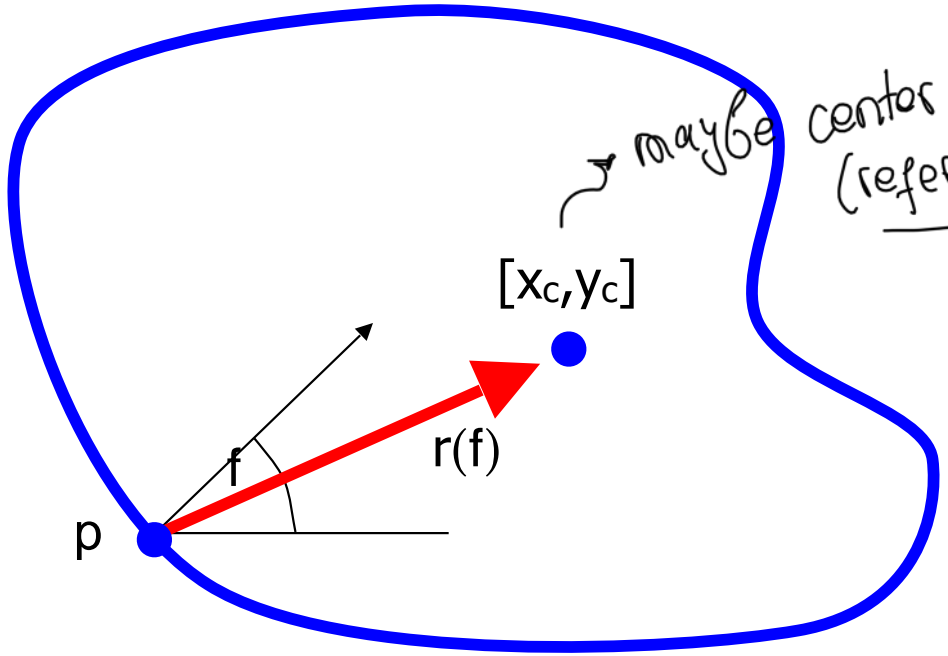
Finding a becher ...

Locate it

... or a lake.



Generalized Hough



maybe center of gravity (reference)

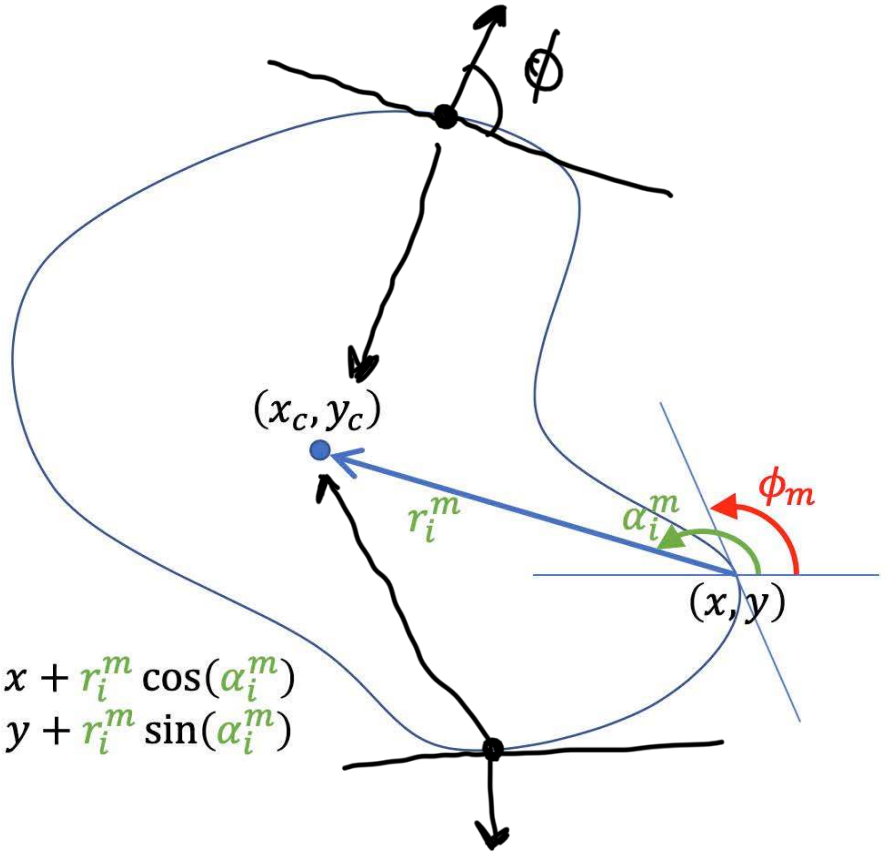
⇒ where is reference pt in our image?

- We want to find a shape defined by its boundary points in terms of the location of a reference point $[x_c,y_c]$.
- For every boundary point p, we can compute the displacement vector $r = [x_c,y_c] – p$ as a function of local gradient orientation f.

Ballard. , PR'81

R-Table

ϕ	$R(\phi_i)$
ϕ_1	$(r_1^1, \alpha_1^1), (r_2^1, \alpha_2^1), \ldots\ldots (r_{n1}^1, \alpha_{n1}^1)$
ϕ_2	$(r_1^2, \alpha_1^2), (r_2^2, \alpha_2^2), \ldots\ldots (r_{n2}^2, \alpha_{n2}^2)$
..	.....
ϕ_m	$(r_1^m, \alpha_1^m), (r_2^m, \alpha_2^m), \ldots\ldots (r_i^m, \alpha_i^m) \ldots\ldots (r_{nm}^m, \alpha_{nm}^m)$
..	.....
ϕ_M	$(r_1^M, \alpha_1^M), (r_2^M, \alpha_2^M), \ldots\ldots (r_{nM}^M, \alpha_{nM}^M)$



$$x_c = x + r_i^m \cos(\alpha_i^m)$$
$$y_c = y + r_i^m \sin(\alpha_i^m)$$

(x,y) which has gradient
⇒ it'll vote for "center" (x_c, y_c)
for all elements in R-table

not 1-to-1

Set of potential displacement vectors r,a given the boundary orientation f. ϕ_i's

--> Generalized template matching.

Algorithm

1. Make an R-table for the shape to be located.

2. Form an accumulator array of possible reference points initialized to zero.

3. For each edge point,

   - Compute the possible centers, that is, for each table entry, compute

$$x = x_e + r_\phi \cos(\theta(\phi))$$
$$y = y_e + r_\phi \sin(\theta(\phi))$$

   - Increment the accumulator array

Voting scheme ⇒ ∀ edge point votes for all places the center could be wrt to it given its gradient direction

Robust, reliable, fast

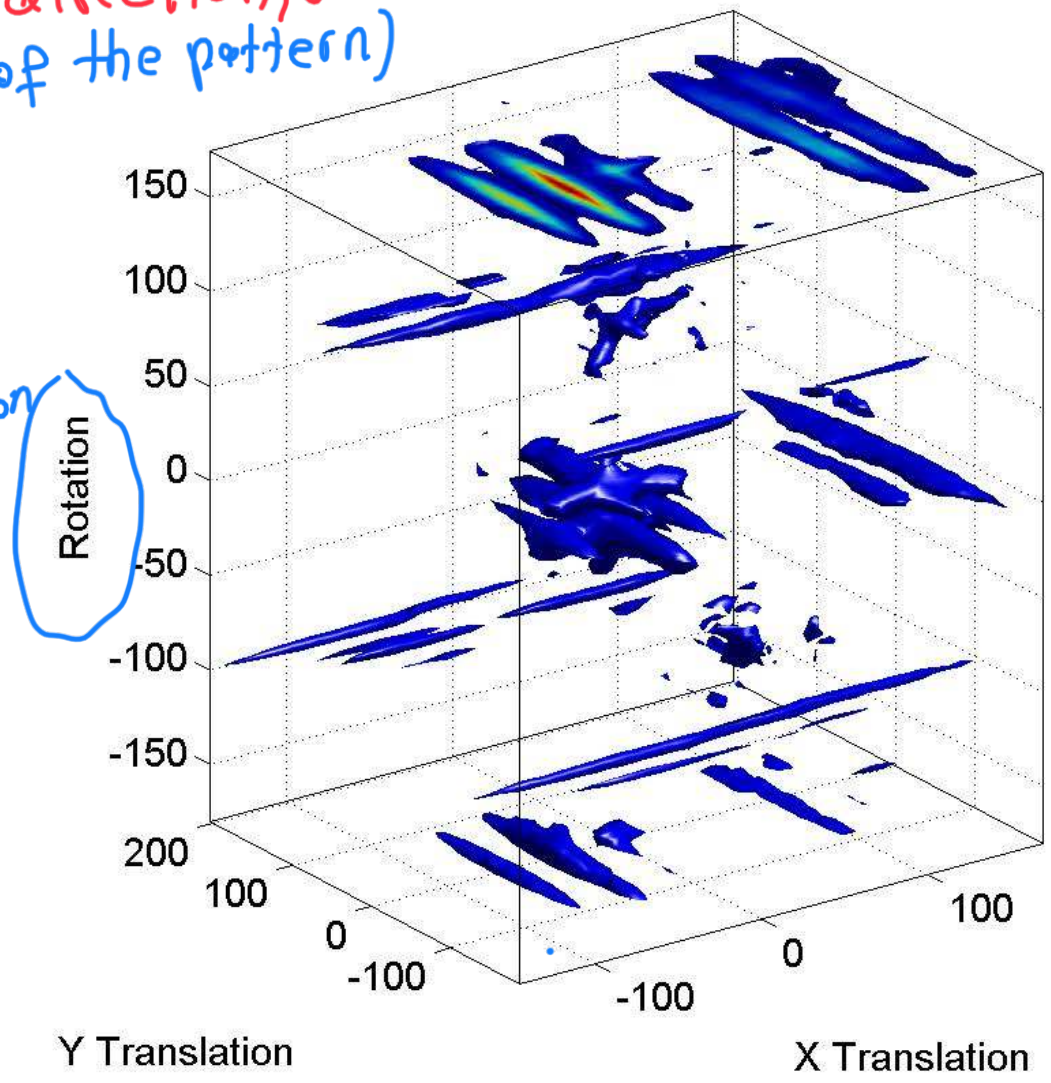
Real-Time Hough

Accumulator

3D-accumulator

Previously, we were assuming to know gradient direction, but in practice → no (orientation of the pattern)

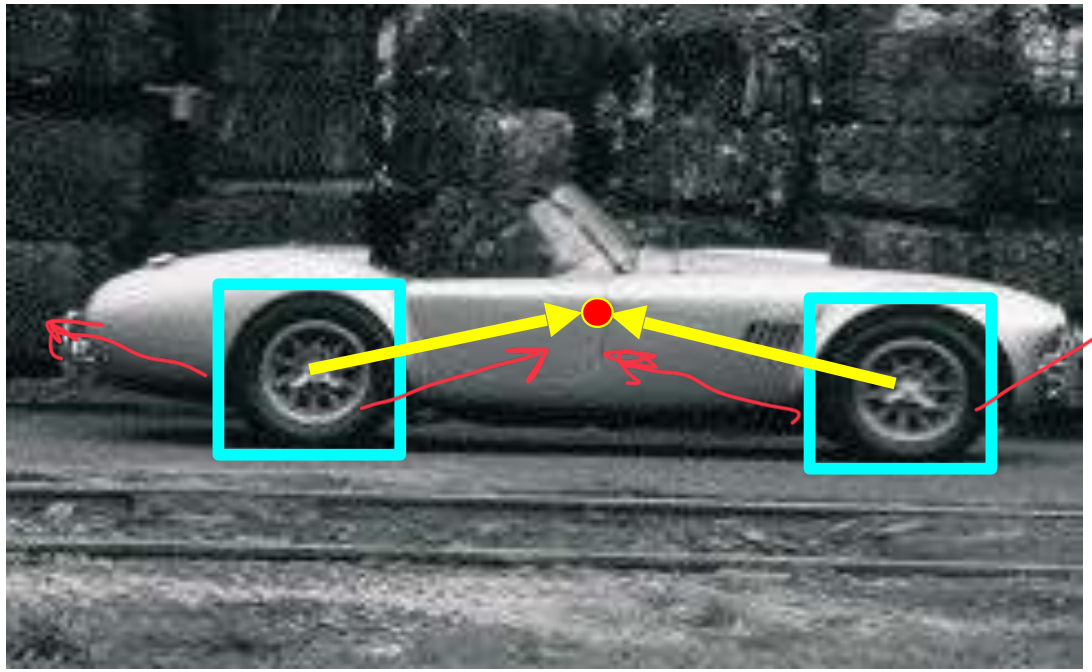
∀ possible orientation of the pattern



From Delineation To Detection

Detecting cars ⇒ detecting wheels

they can vote for the position of the car and get estimate for the center of the car



Visual codeword with displacement vectors

Training image

Hough (from generalized)

Instead of indexing displacements by gradient orientation, index by "visual codeword".

Limitations

Fairly, effectively detects curves → including complicated ones
↳ defined relatively by small # of parameters
↳ knowing shape priori, only looking for orientations

Computational cost grows exponentially with the number of model parameters:

→Only works for objects whose shape can be defined by a small number of parameters.

→Approach is robust but lacks flexibility.

when assumptions met

From Delineation To Detection

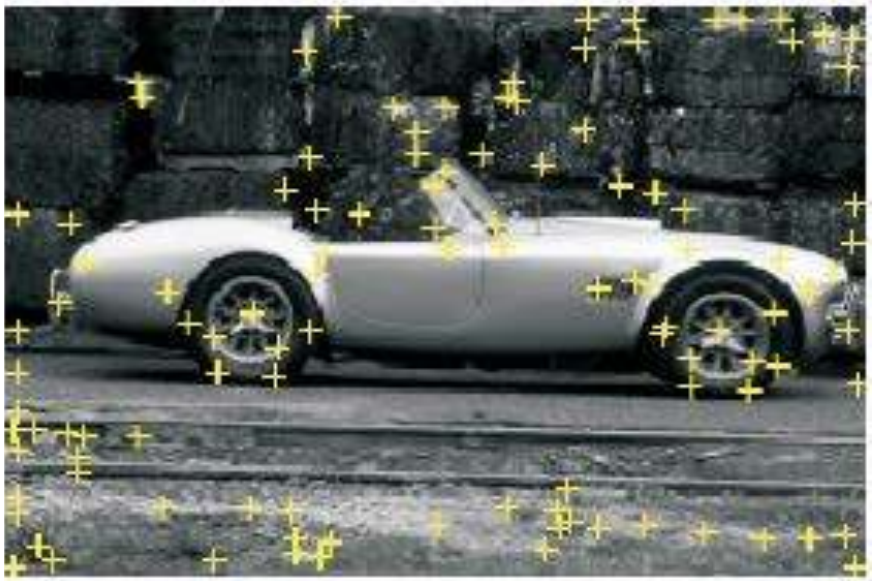
Test image

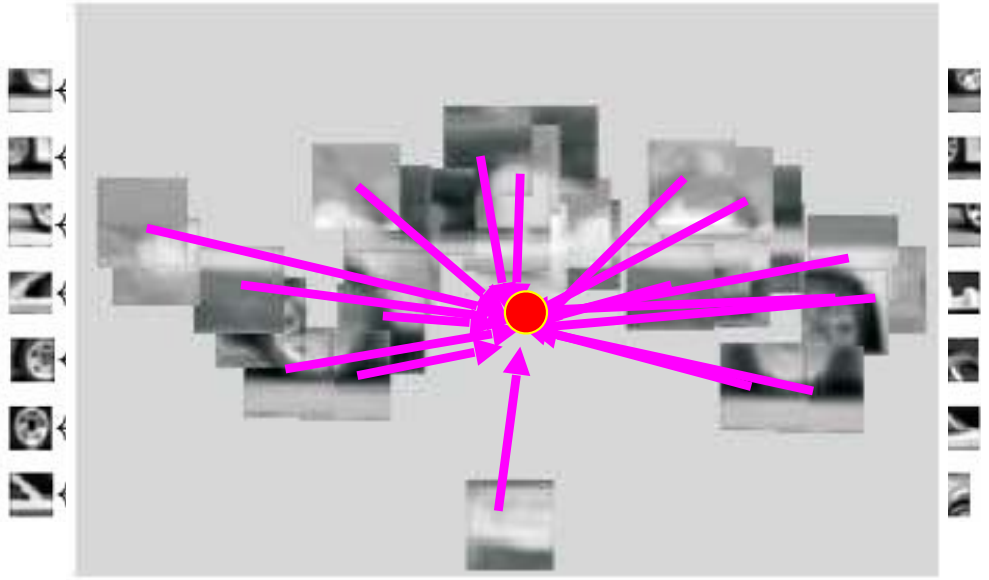
Instead of indexing displacements by gradient orientation, index by "visual codeword".

Leibe & al. , ECCV'04

Optional: Training

(ML, not DL)

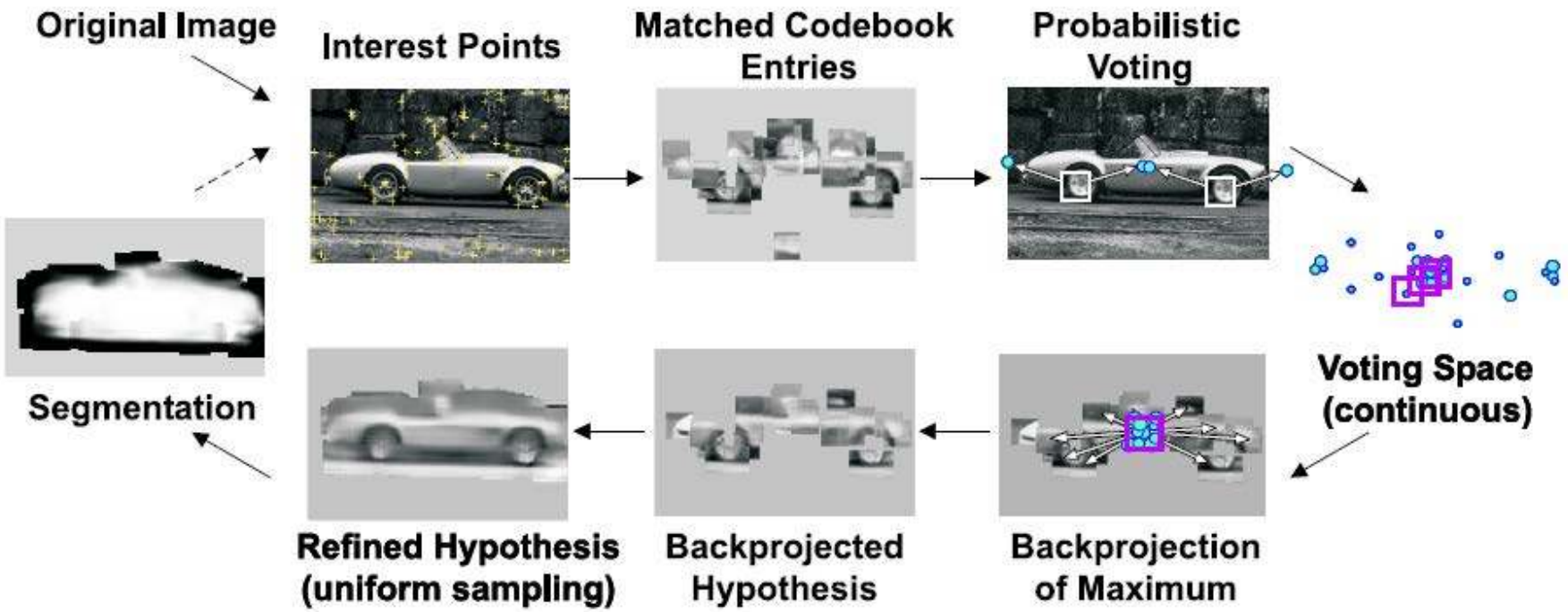




1. Use clustering to build codebook of patches around extracted interest points.
2. Map the patch around each interest point to closest codebook entry.
3. For each codebook entry, store all positions it was found, relative to object center.

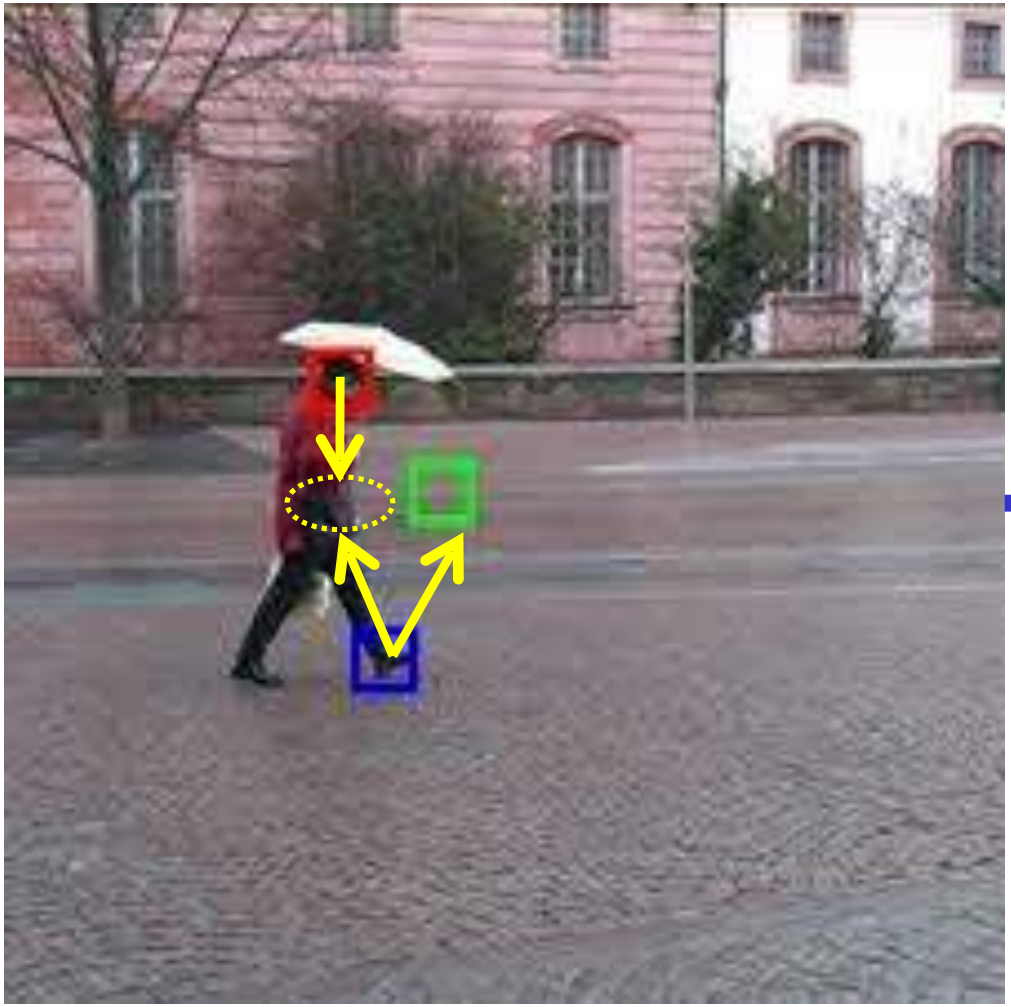
--> Build an R table.

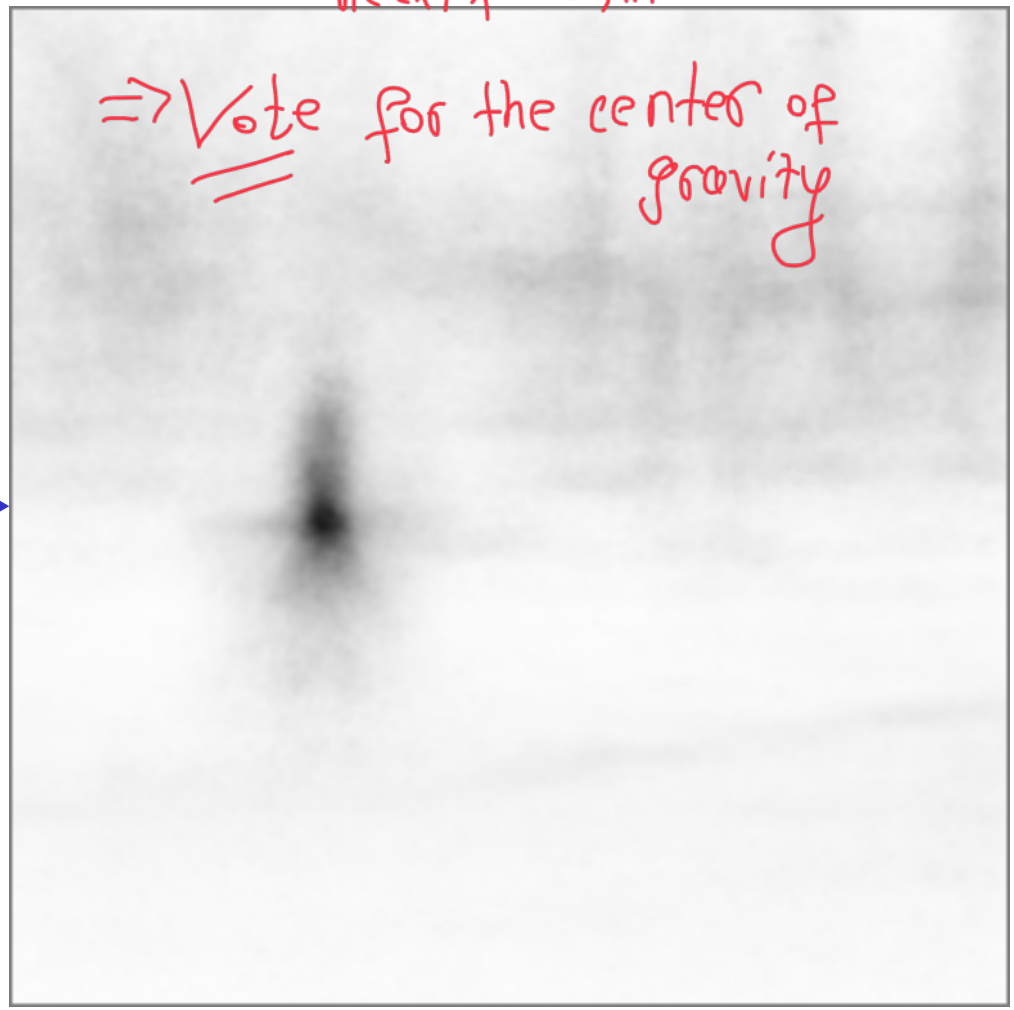
Optional: Testing



1. Given test image, extract patches, match to codebook entry.
2. Cast votes for possible positions of object center.
3. Search for maxima in voting space.
4. Extract weighted segmentation mask based on stored masks for the codebook occurrences.

Pedestrian Detection

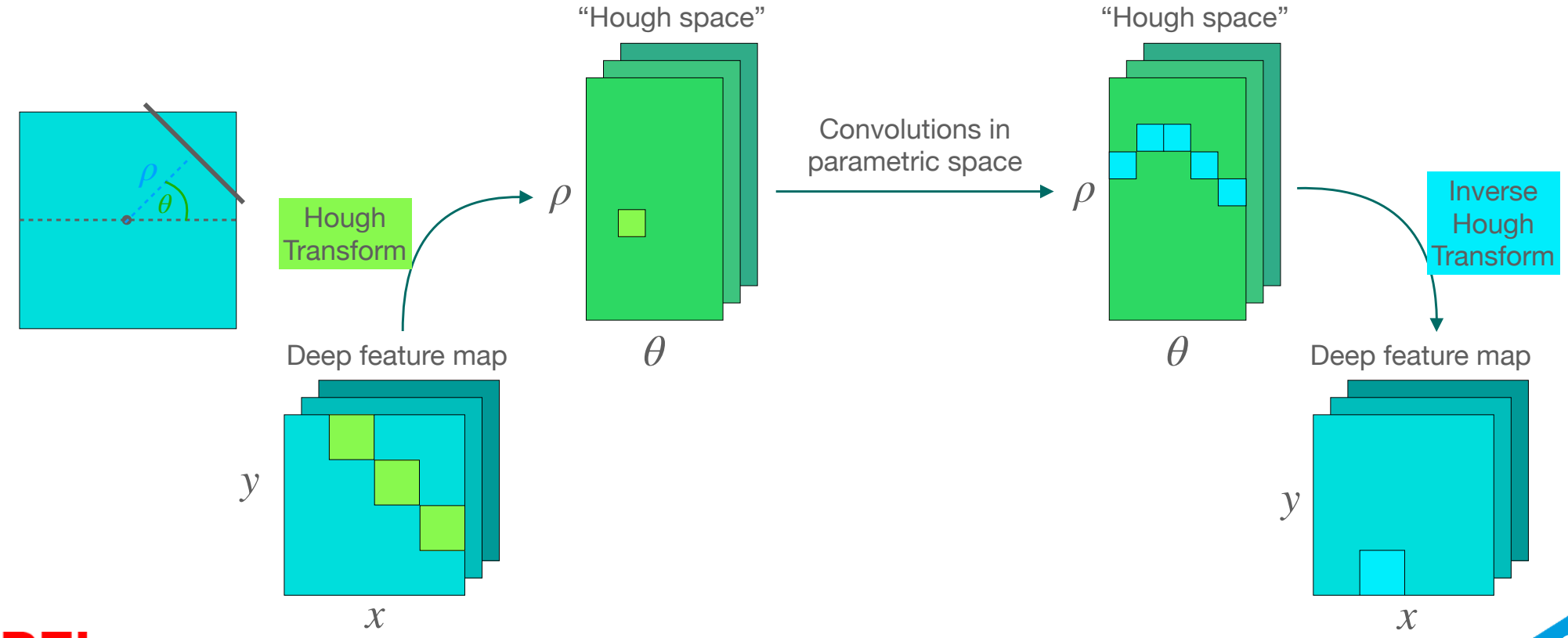




Gall & al. , PAMI'11

Occlusion Handling

Deep Hough Transform



Techniques

Semi-Automated Techniques:

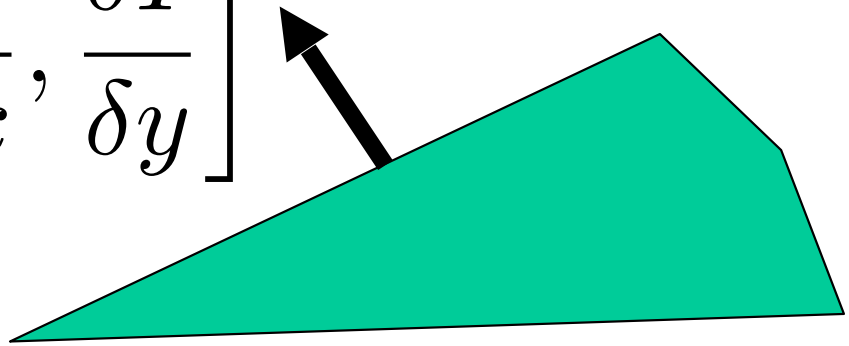
- Dynamic programming
- Deformable Models

Fully Automated Techniques:

- Hough transform
- Graph Based Approaches

Magnitude and Orientation

$$\left[\frac{\delta I}{\delta x}, \frac{\delta I}{\delta y}\right]$$



$$\text{Contrast: } G = \sqrt{\frac{\delta I}{\delta x}^2 + \frac{\delta I}{\delta y}^2}$$

$$\text{Orientation: } \Theta = \arctan\left(\frac{\delta I}{\delta y}, \frac{\delta I}{\delta x}\right)$$

Minimum Spanning Tree

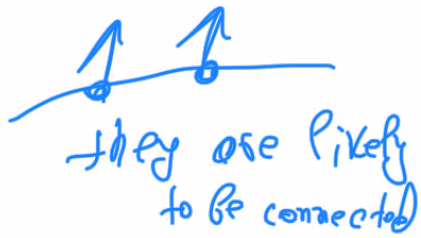
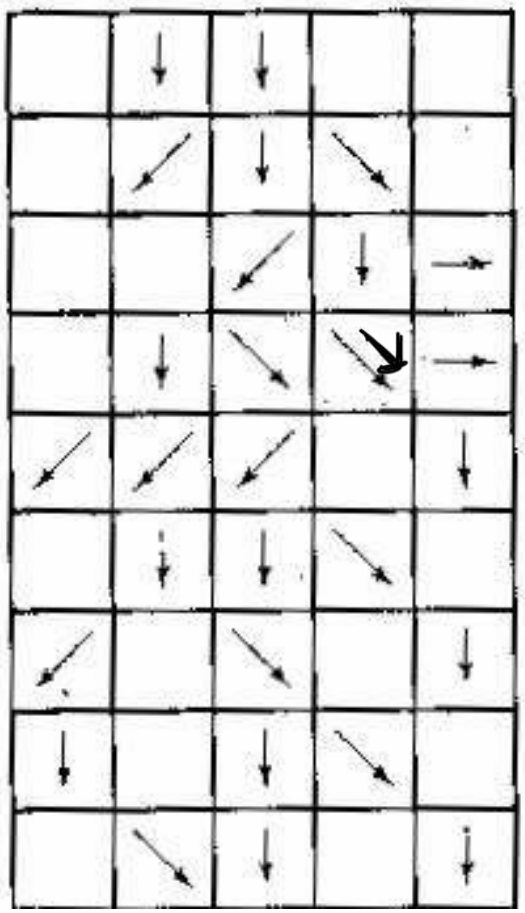


Image modeled as a graph:

Image represented as a grid



gradient

grid cells

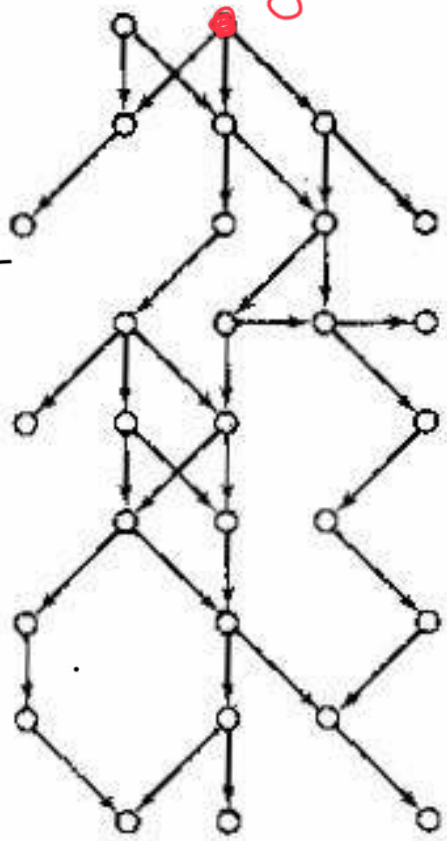


Image as a graph

-> Generate minimal distance graph O(N log(N) algorithm)

Delineation 1998

Detecting contours
Detect road centerlines

Pong, not too furry
Build ST and find paths
Find generic paths

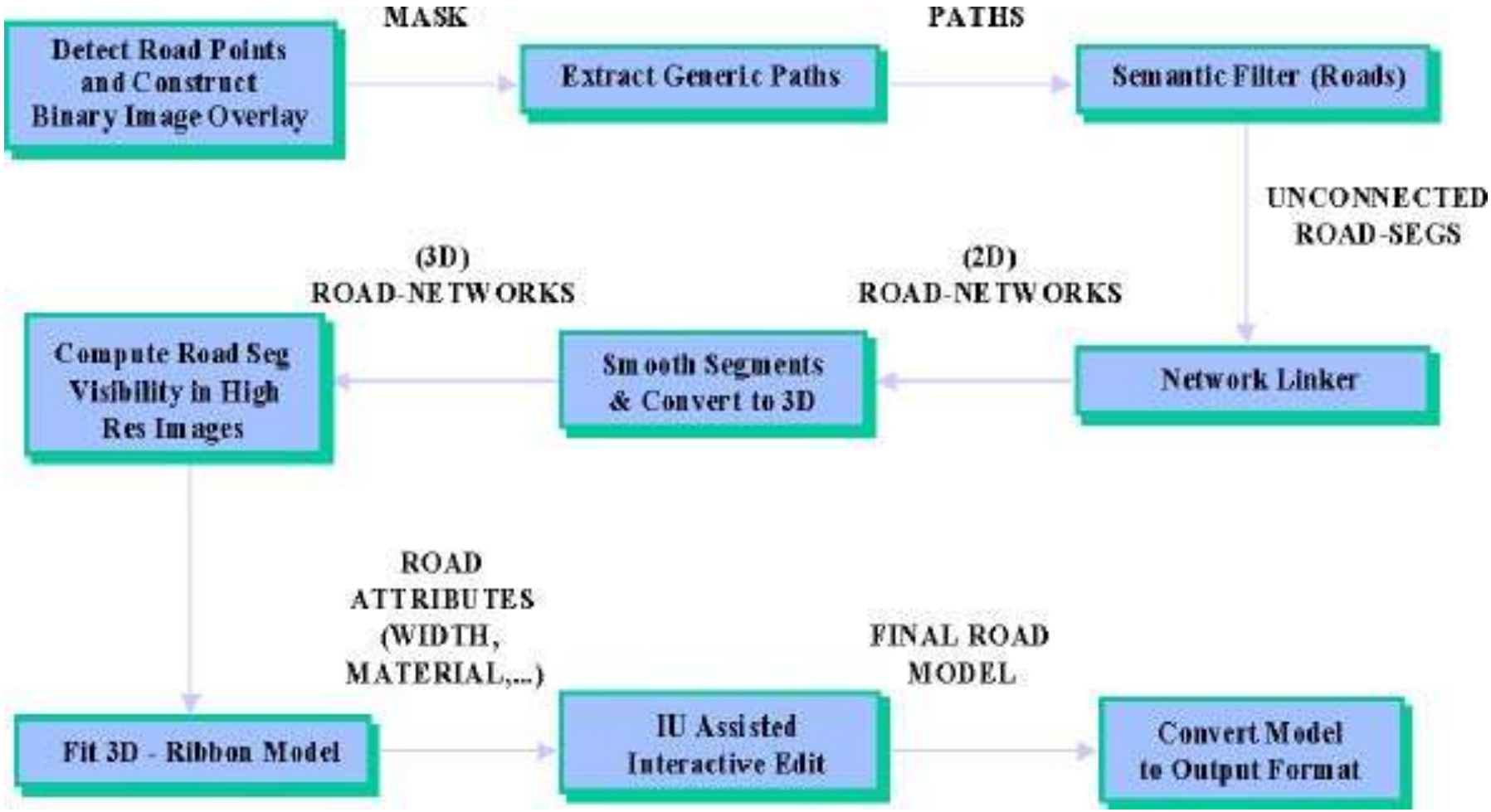
Get rid of short, wiggly, not informative
Apply semantic filter

Find road widths

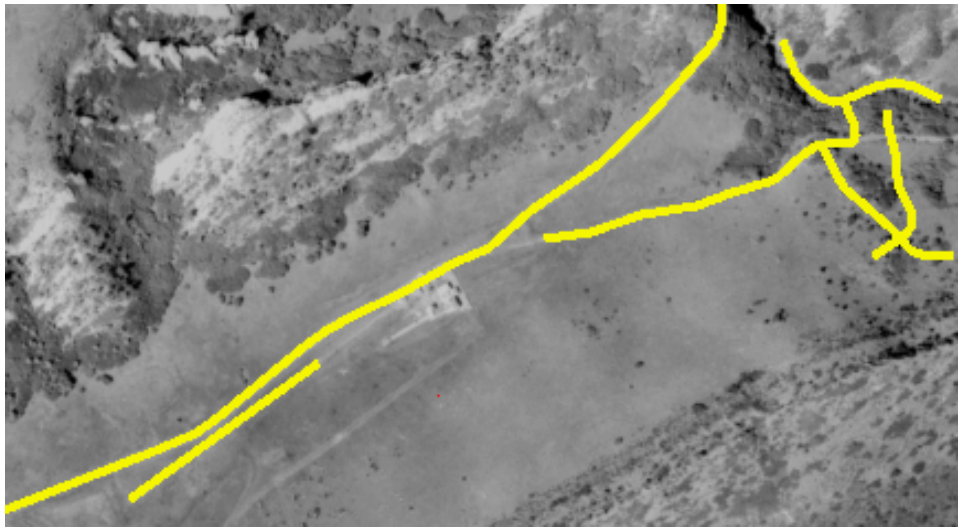
Fischler & Heller, 1998.

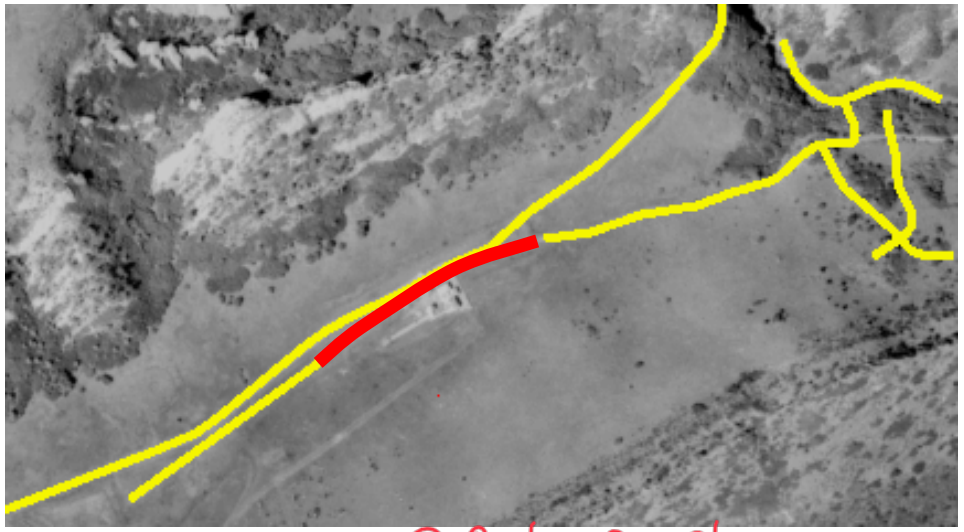
From Image To Roads

A lot of params need to be tuned by human

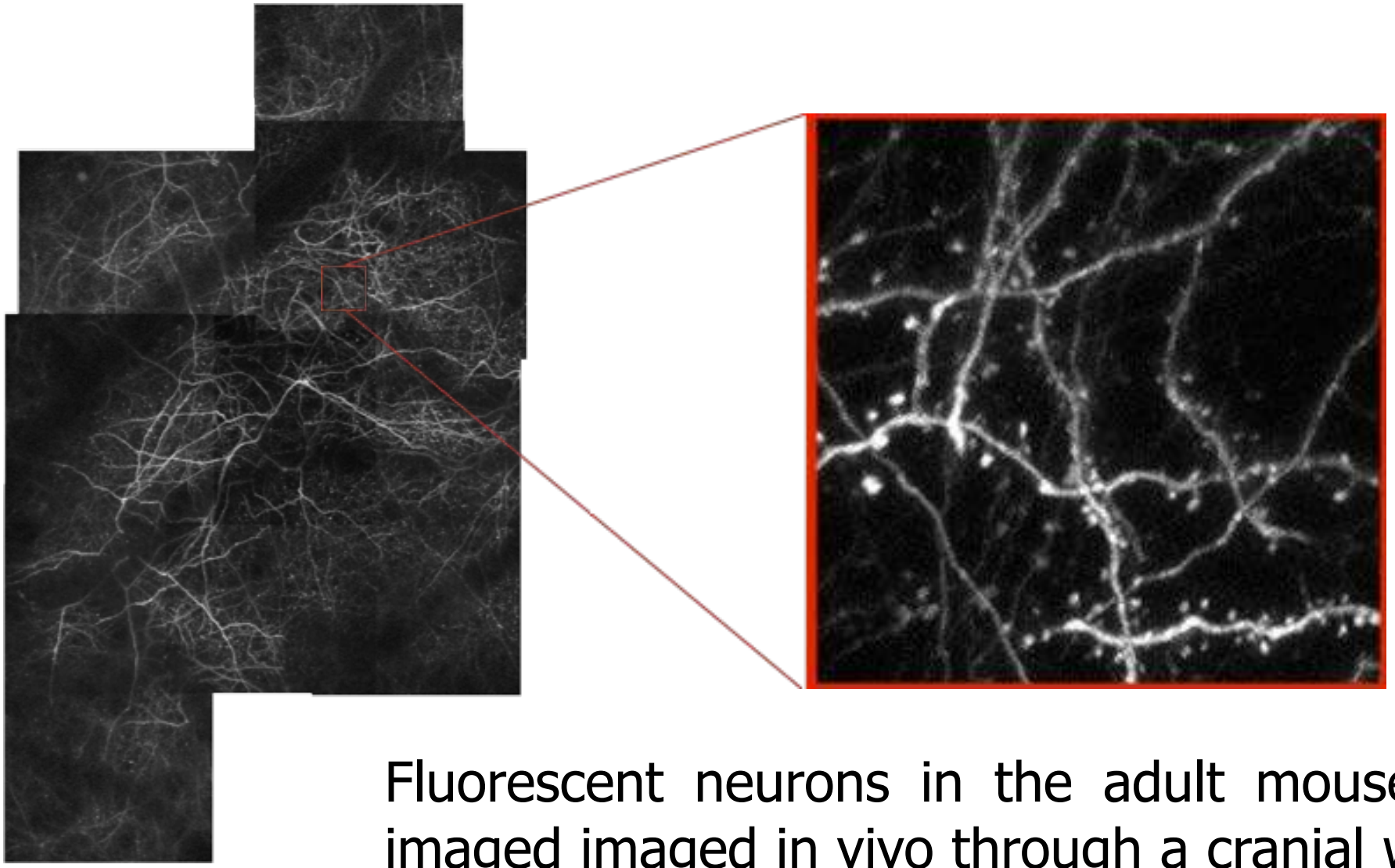


Road Editing



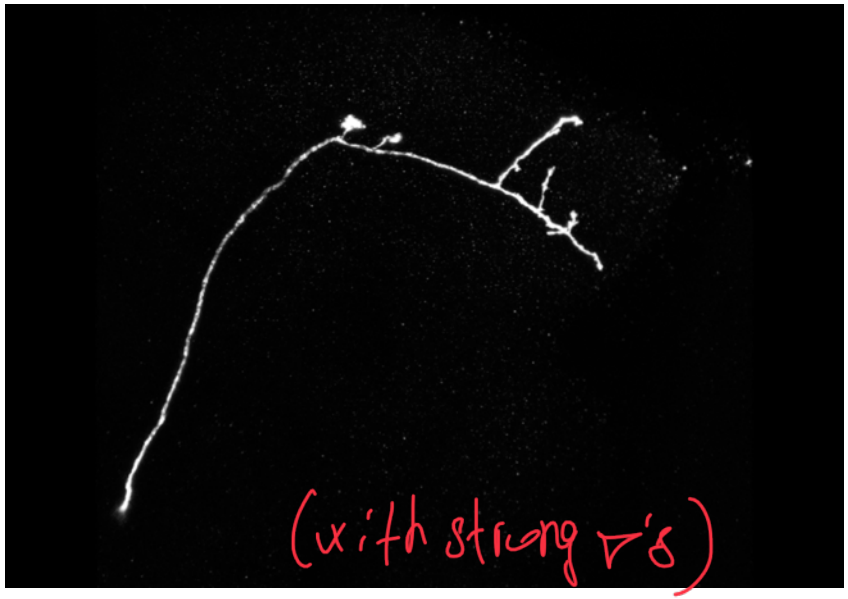


Dendrites And Axons

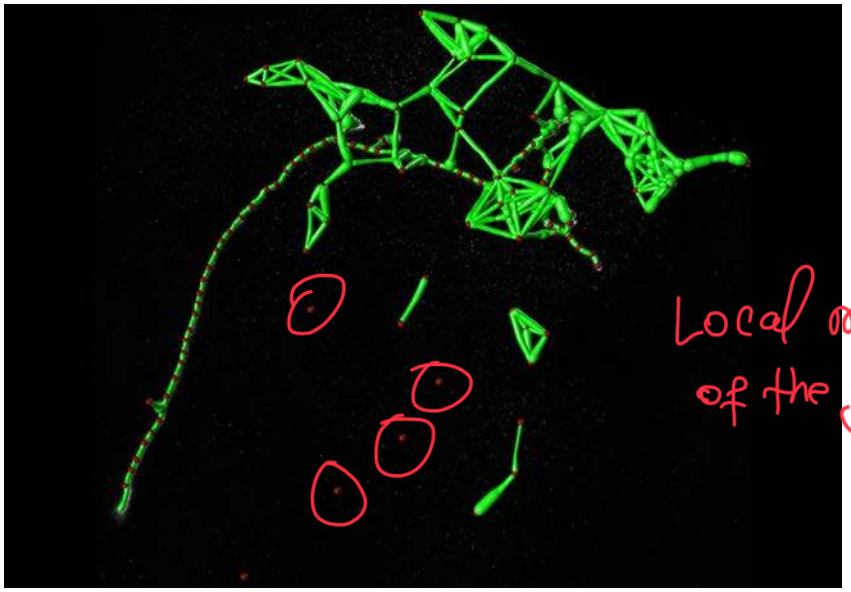


Fluorescent neurons in the adult mouse brain imaged imaged in vivo through a cranial window using a 2-photon microscope.

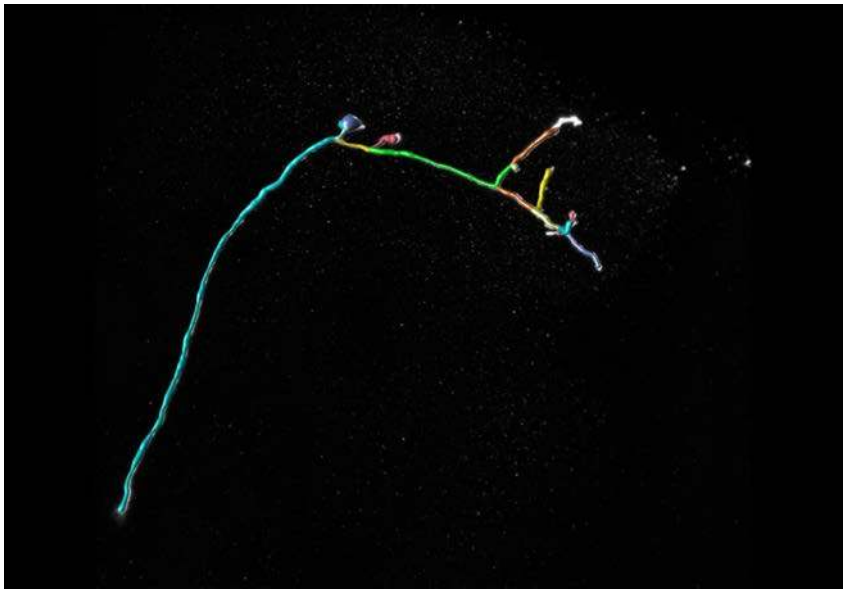
Delineation 2012: Neurites …



Filtered Image



Graph



Maximum Likelihood Subtree

Keep paths that're most probable and give final delienation of the axon

.. and Roads

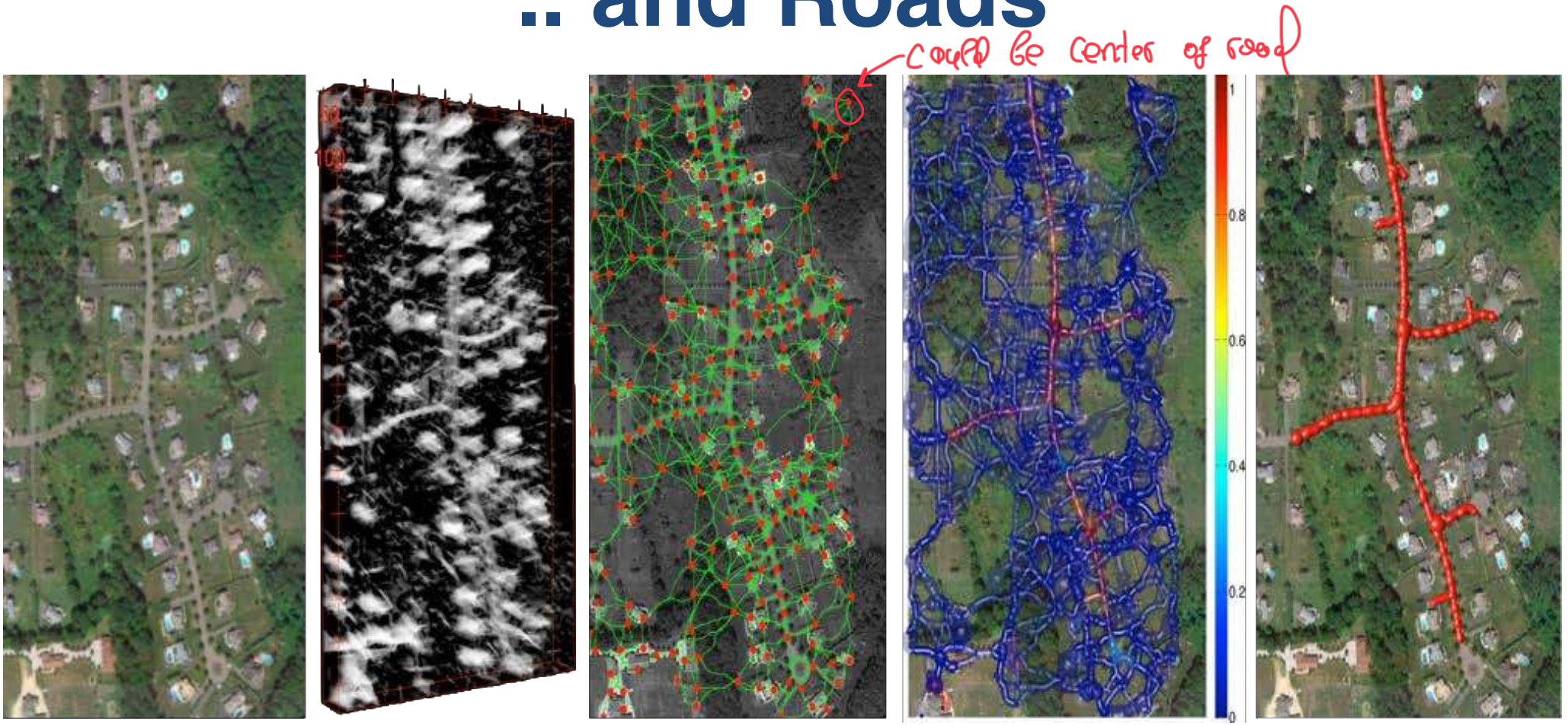


Image | Filtered image | Graph | Weighted graph | Subtree

2D at several scales

Local maxima

(ML ⇒ likelihood of being part of road)

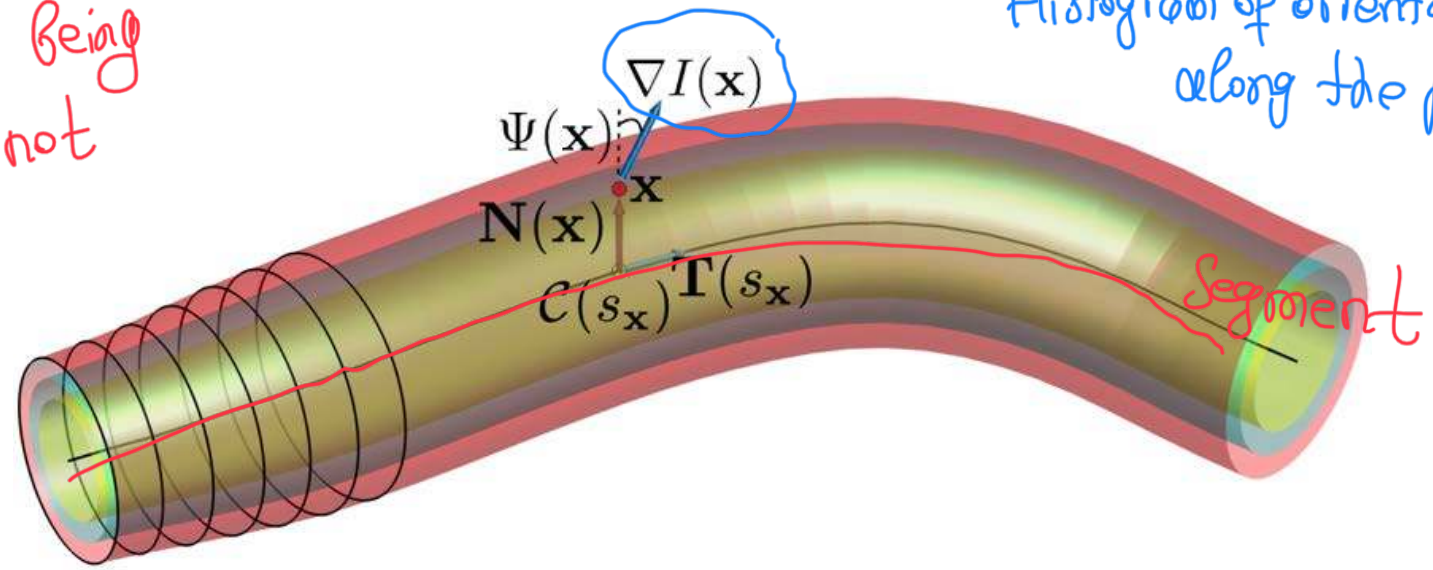
—> Machine plays a crucial role to ensure that the **same algorithm works in different situations.**

Turetken et al, CVPR'12.

Histogram of Gradient Deviations

Classify segment being a road line or not

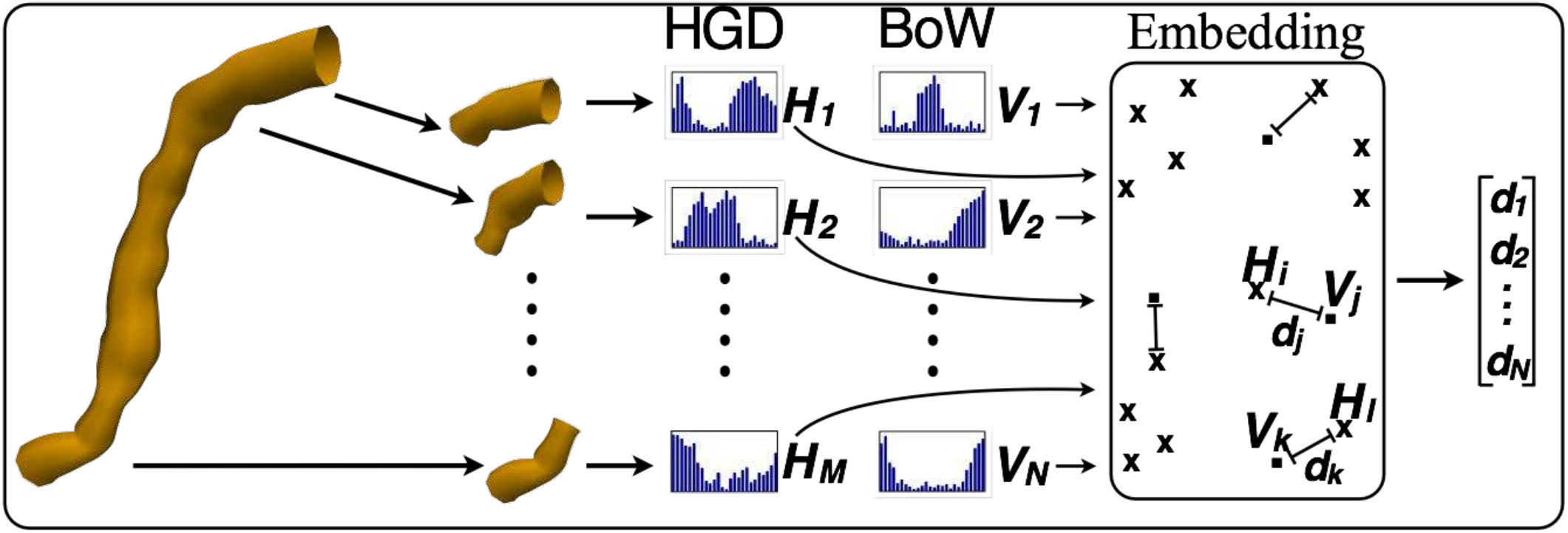
Histogram of orientations of ∇ along the path



$$\Psi(\mathbf{x}) = \begin{cases} \text{angle}(\nabla I(\mathbf{x}), \mathbf{N}(\mathbf{x})) \text{ , if } \|\mathbf{x} - \mathcal{C}(s_\mathbf{x})\| > \varepsilon \\ \text{angle}(\nabla I(\mathbf{x}), \mathbf{\Pi}(\mathbf{x})) \text{ , otherwise,} \end{cases}$$

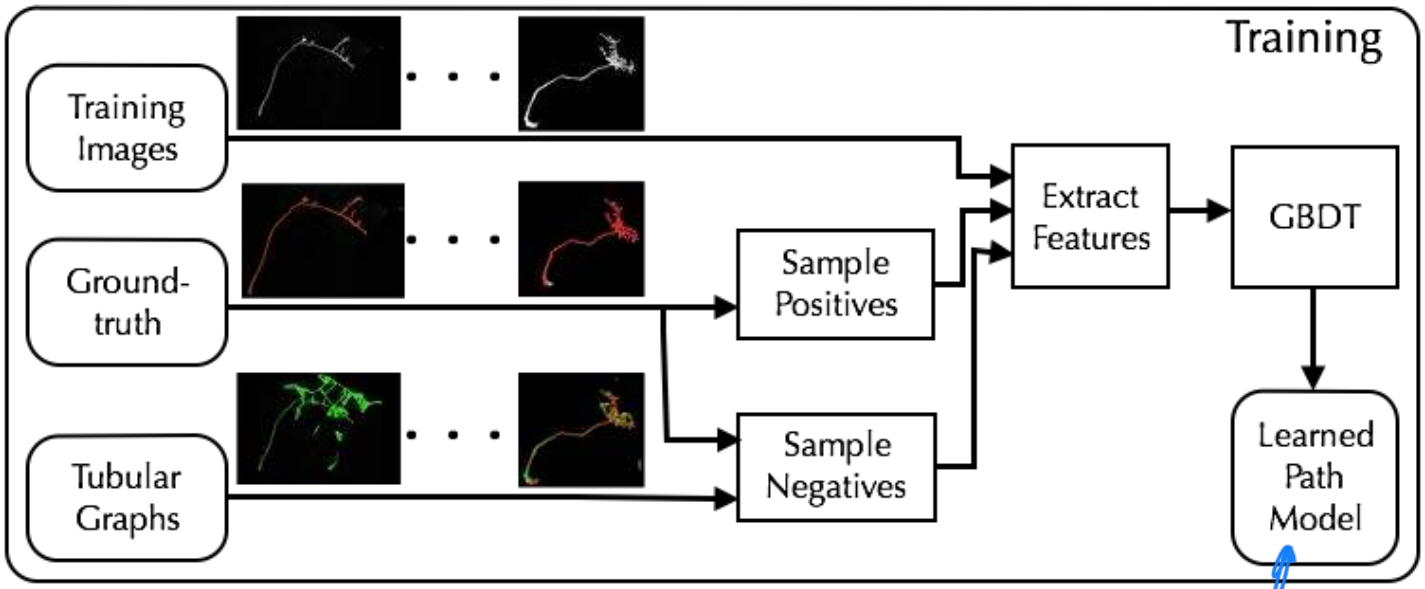
—> One histogram per radius interval plus four geometric features (curvature, tortuosity, ….).

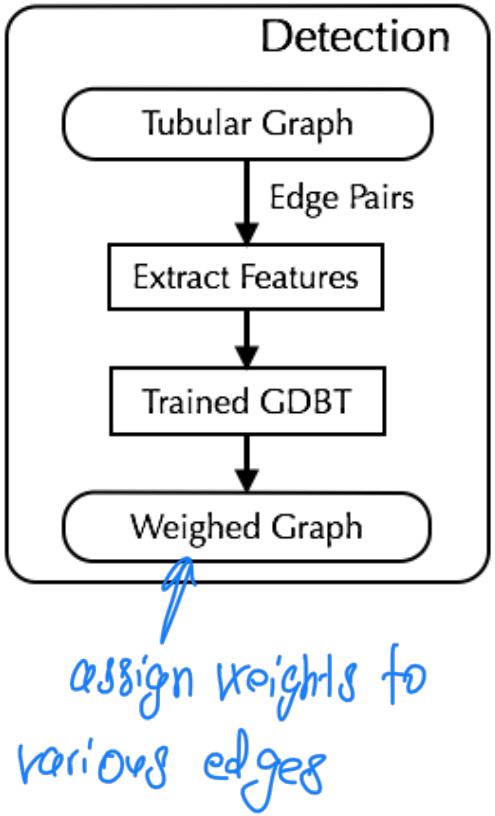
Optional: Embedding



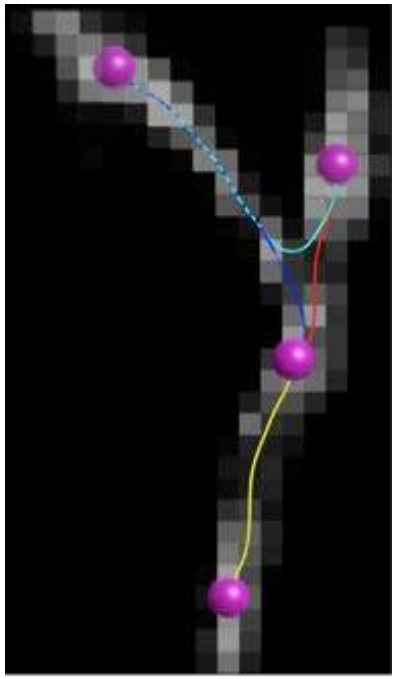
—> Same length feature vectors whatever the actual length of the path.

Optional: Path Classification

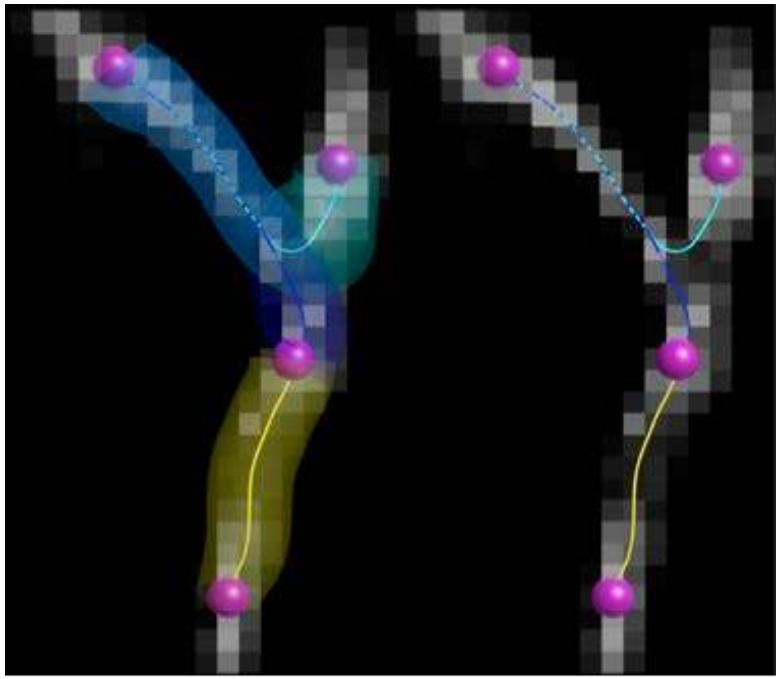




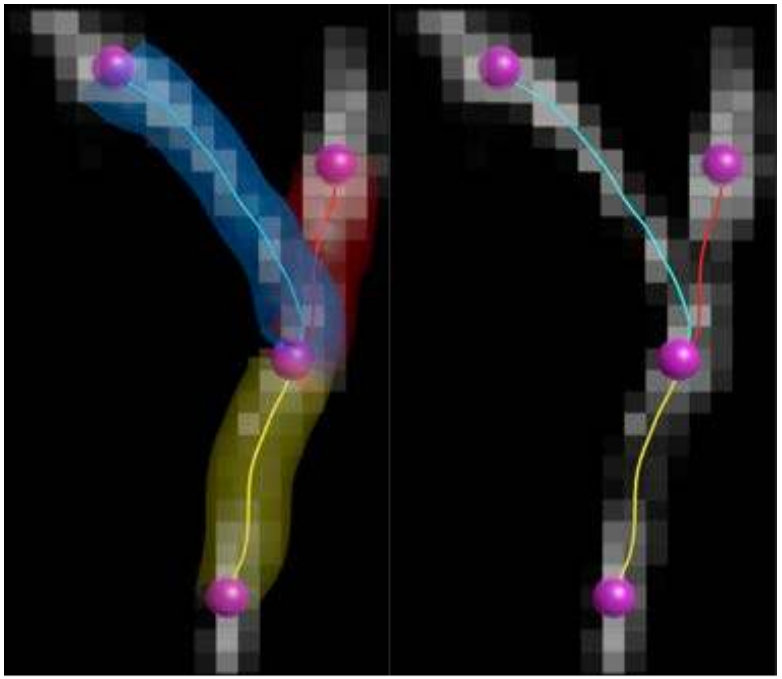
Optional: Finding the Best Tree



Tubularity graph



Without edge pair term



With edge pair term

$$
\begin{aligned}
\mathbf{t}^* &= \underset{\mathbf{t}\in\mathcal{T}(G)}{\operatorname{argmax}} P(\mathbf{T}=\mathbf{t}|I) \ , \\
&= \underset{\mathbf{t}\in\mathcal{T}(G)}{\operatorname{argmax}} P(I|\mathbf{T}=\mathbf{t})P(\mathbf{T}=\mathbf{t}) \ , \\
&= \underset{\mathbf{t}\in\mathcal{T}(G)}{\operatorname{argmin}} \sum_{e_{ij}\in G} c_{ij}^d t_{ij}
\end{aligned}
$$

maximize objective function

Optional: QMIP Formulation

$$\min \sum_{e_{ij} \in E, e_{jk} \in E} c_{ijk}\ t_{ij}\ t_{jk}$$

$$\text{s.t.} \sum_{v_j \in V \setminus \{v_r\}} y^l_{rj} \leq 1, \qquad \forall v_l \in V \setminus \{v_r\},$$

$$\sum_{v_j \in V \setminus \{v_k\}} y^l_{jk} \leq 1, \qquad \forall v_l \in V \setminus \{v_r\},$$

$$\sum_{v_j \in V \setminus \{v_i, v_r\}} y^l_{ij} - \sum_{v_j \in V \setminus \{v_i, v_l\}} y^l_{ji} = 1, \quad \begin{matrix} \forall v_k \in V \setminus \{v_r\}, \\ \forall v_i \in V \setminus \{v_r, v_k\}, \end{matrix}$$

$$y^l_{ij} \leq t_{ij}, \qquad \forall e_{ij} \in E,\ v_l \in V \setminus \{v_r, v_i, v_j\},$$

$$y^l_{il} = t_{il}, \qquad \forall e_{il} \in E,$$

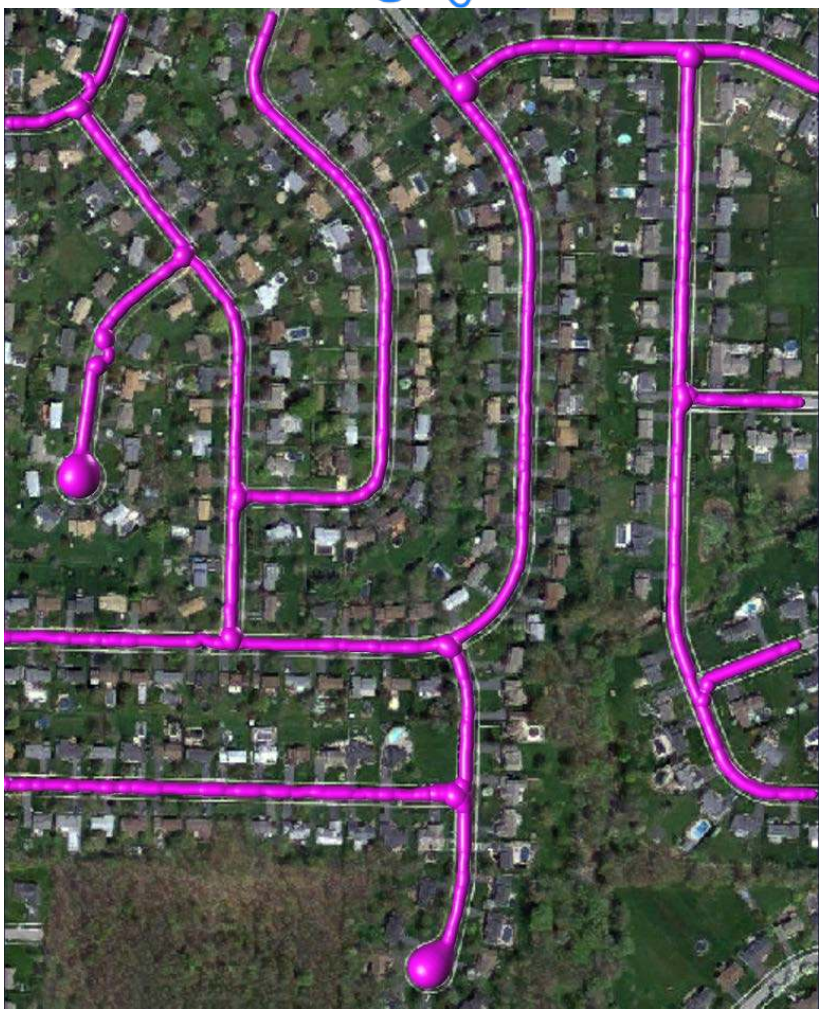
$$y^l_{ij} \geq 0, \qquad \forall e_{ij} \in E,\ v_l \in V \setminus \{v_r, v_i\},$$

$$t_{ij} \in \{0, 1\}, \quad \forall e_{ij} \in E.$$

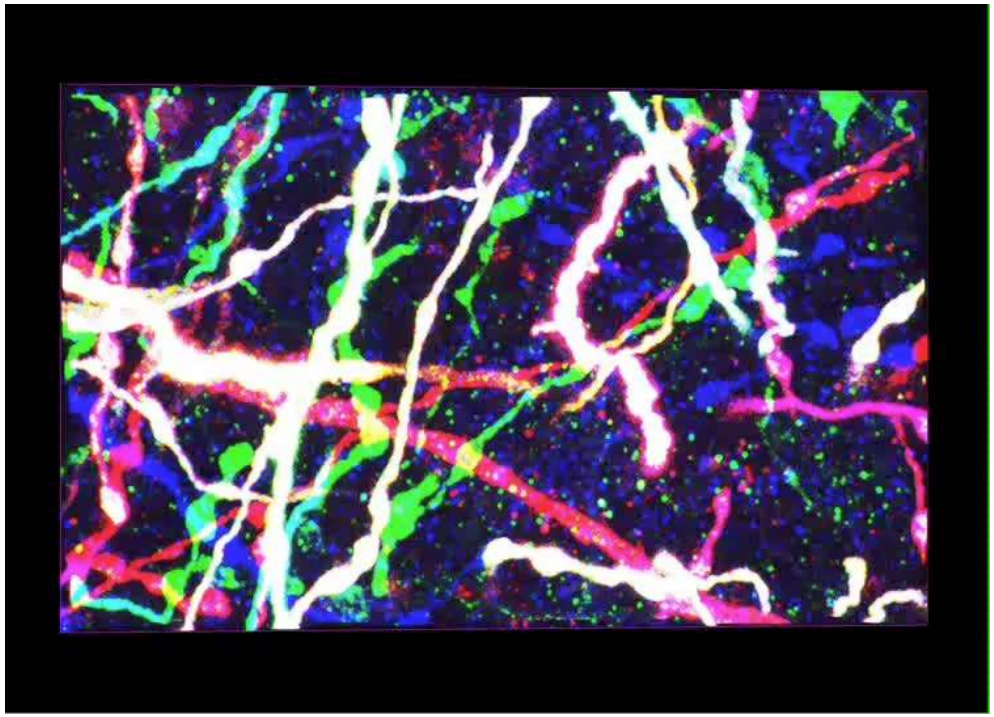
given the root note v_r.

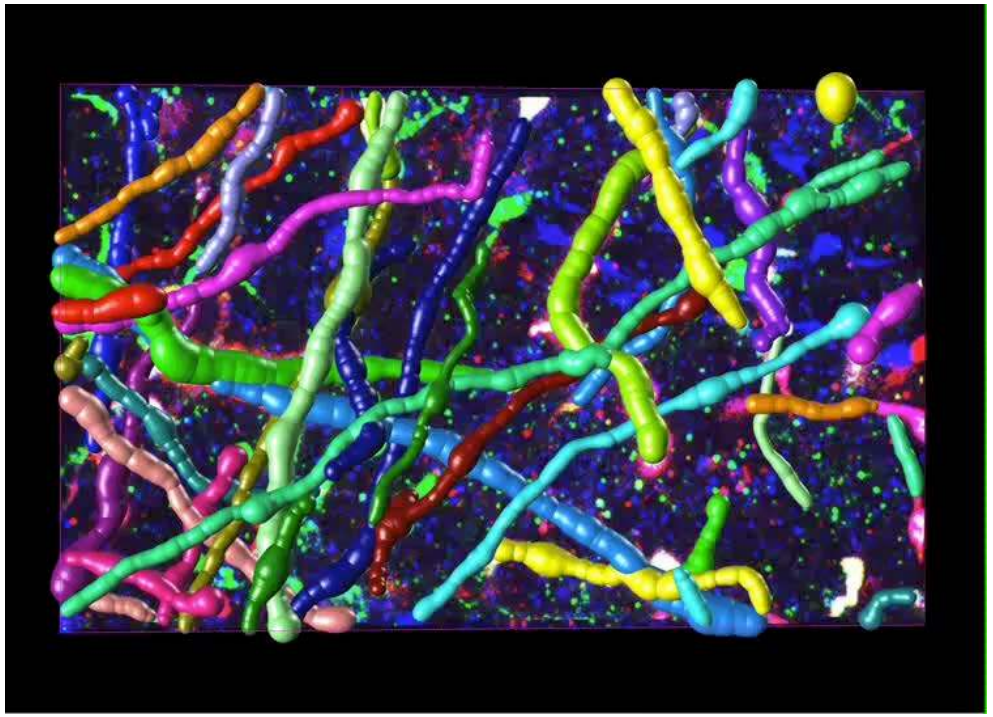
Roads

Very good delienation



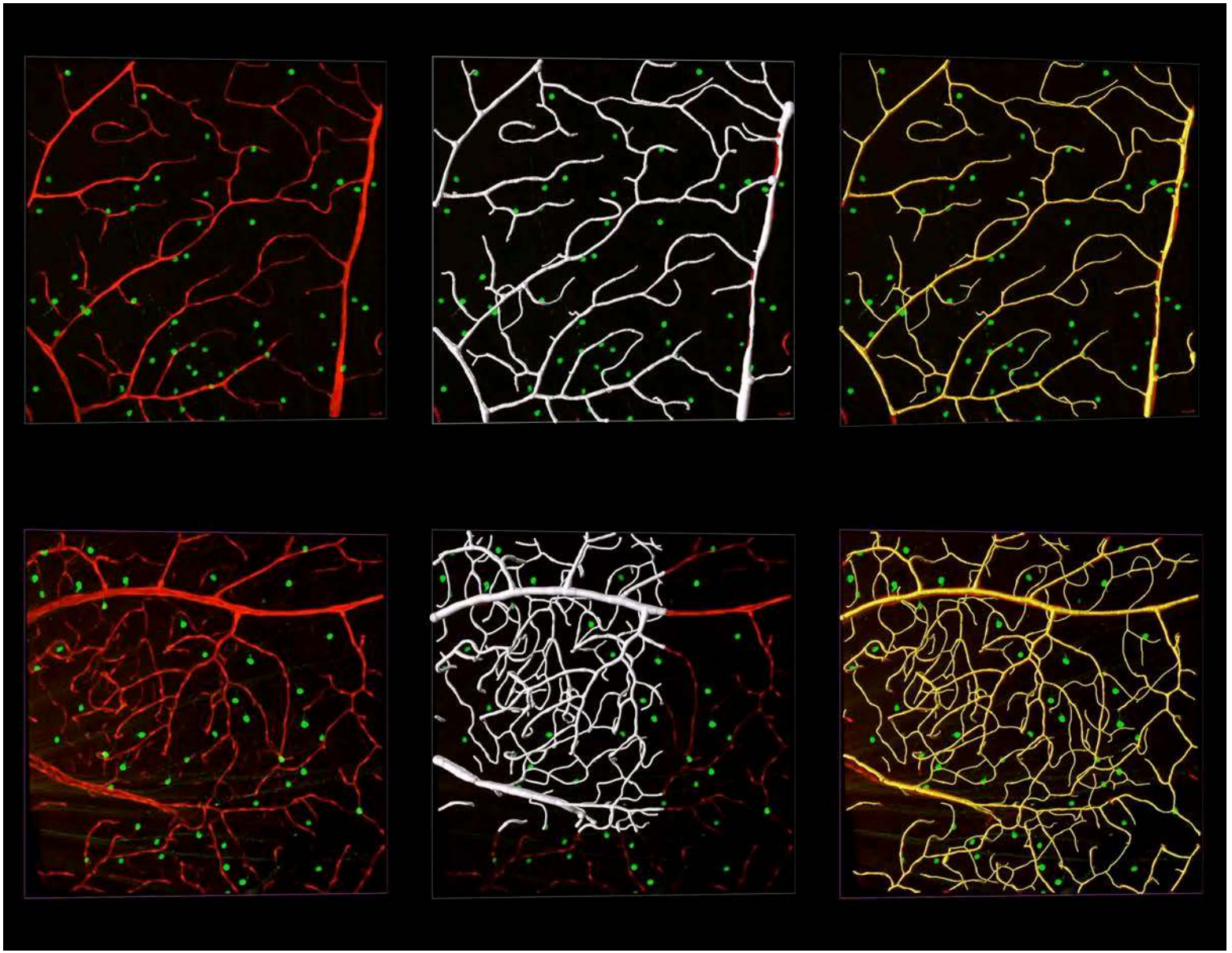
Brainbow Images





Blood Vessels

Only changing classifier ⇒ which paths are good/bad?



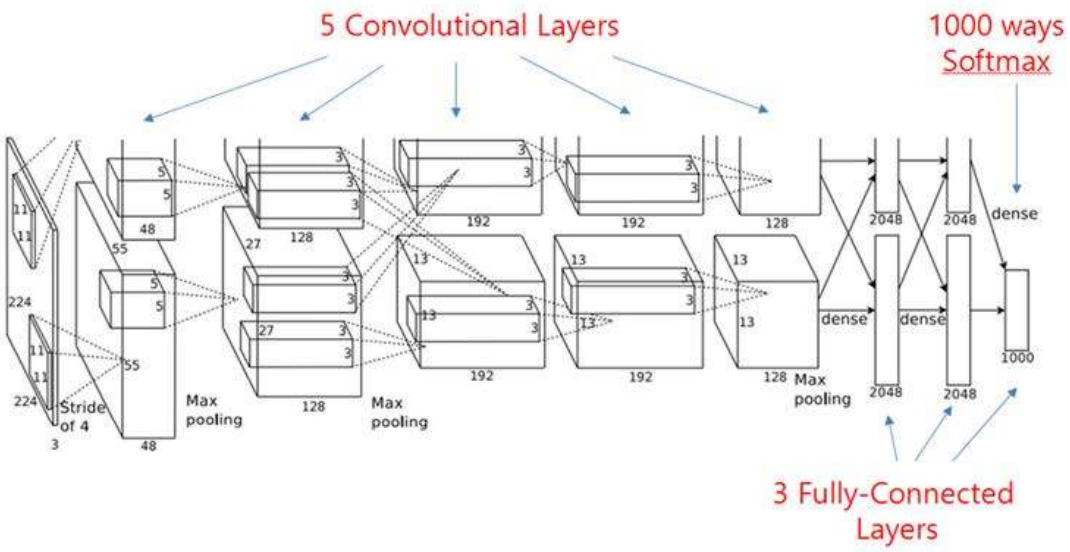
Deep Tsunami

The New York Times

Turing Award Won by Three Pioneers in Artificial Intelligence

From left, Yann LeCun, Geoffrey Hinton and Yoshua Bengio. The researchers worked on key developments for neural networks, which are reshaping how computer systems are built.

Reminder: AlexNet (2012)



Task: Image classification

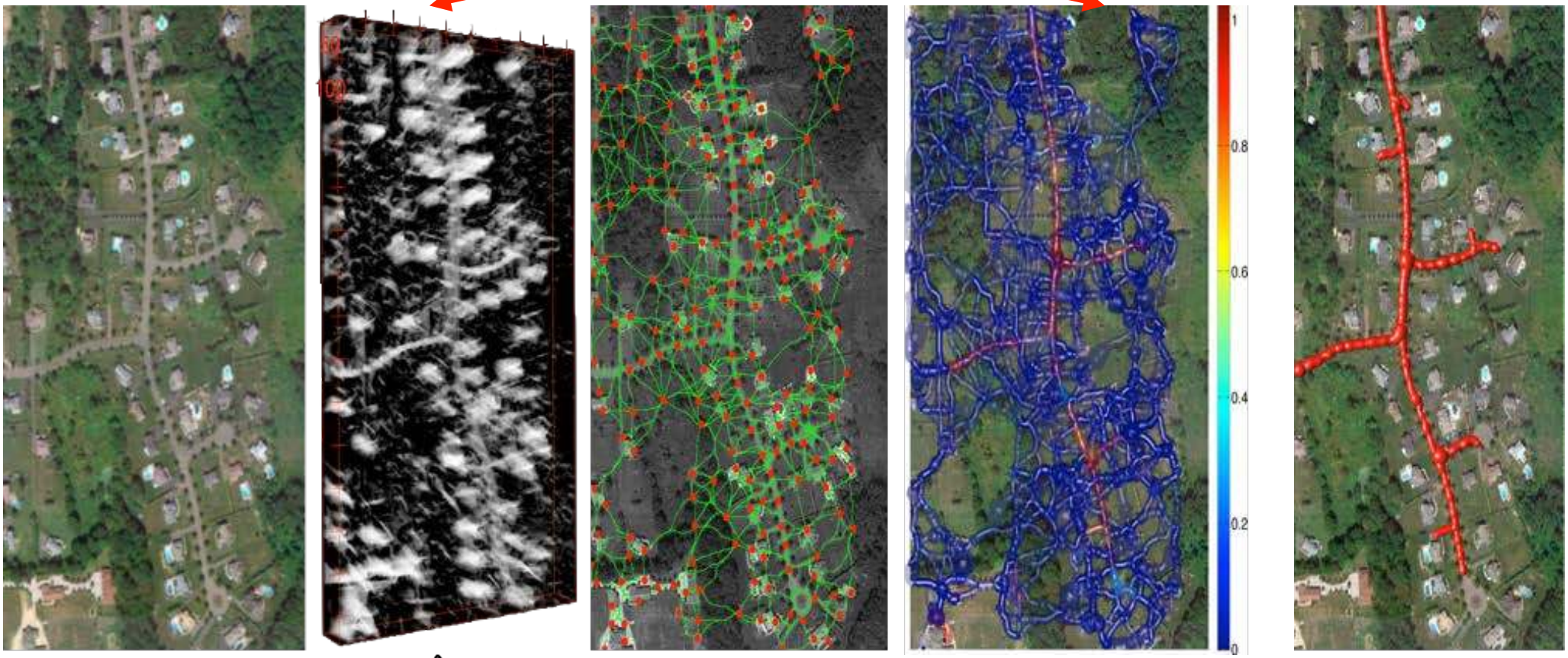
Training images: Large Scale Visual Recognition Challenge 2010

Training time: 2 weeks on 2 GPUs

Major Breakthrough: Training large networks has now been shown to be practical!!

Delineation 2012

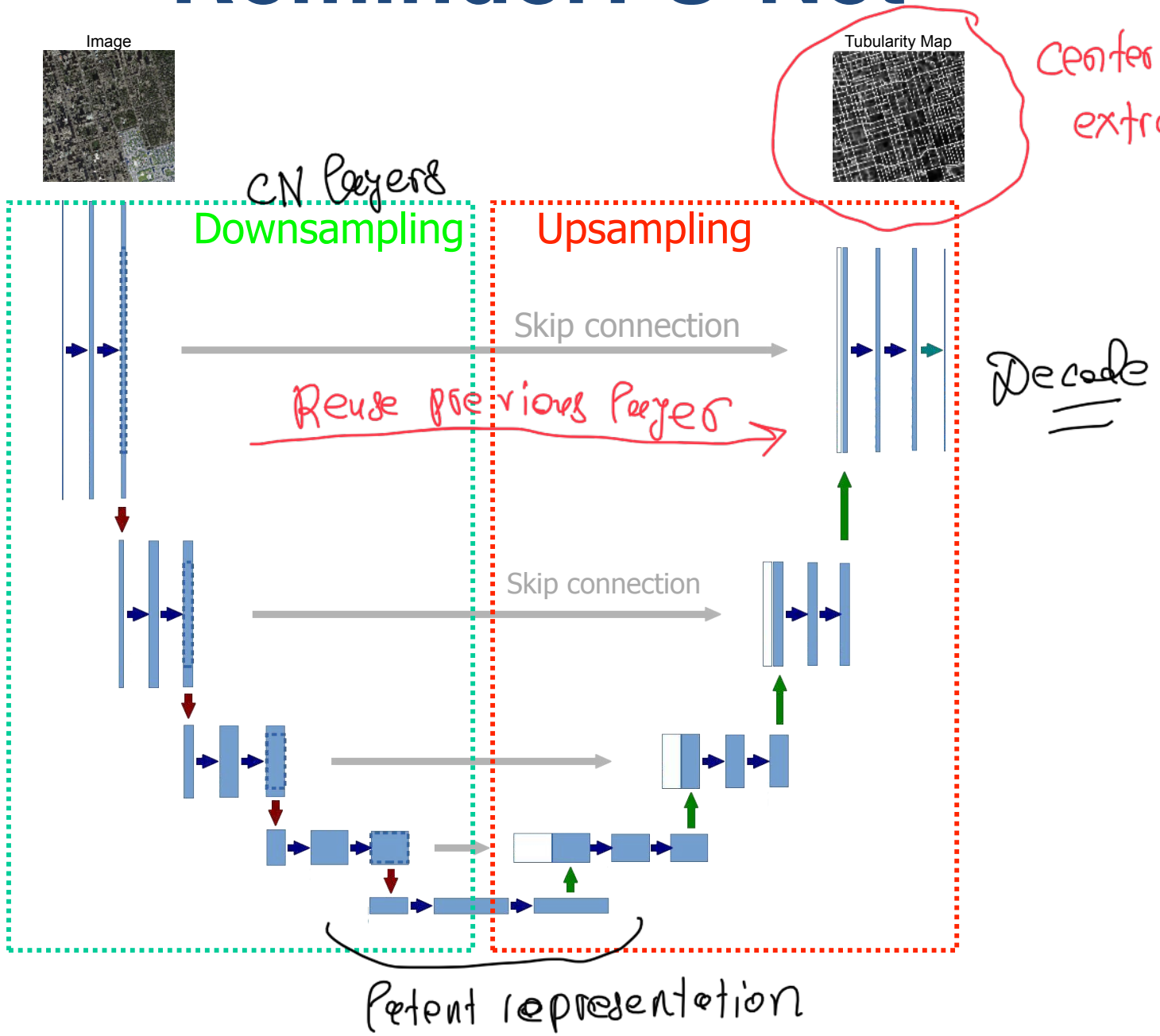
These two steps are closely related!



Detecting centerlines

Testing whether path is road/not

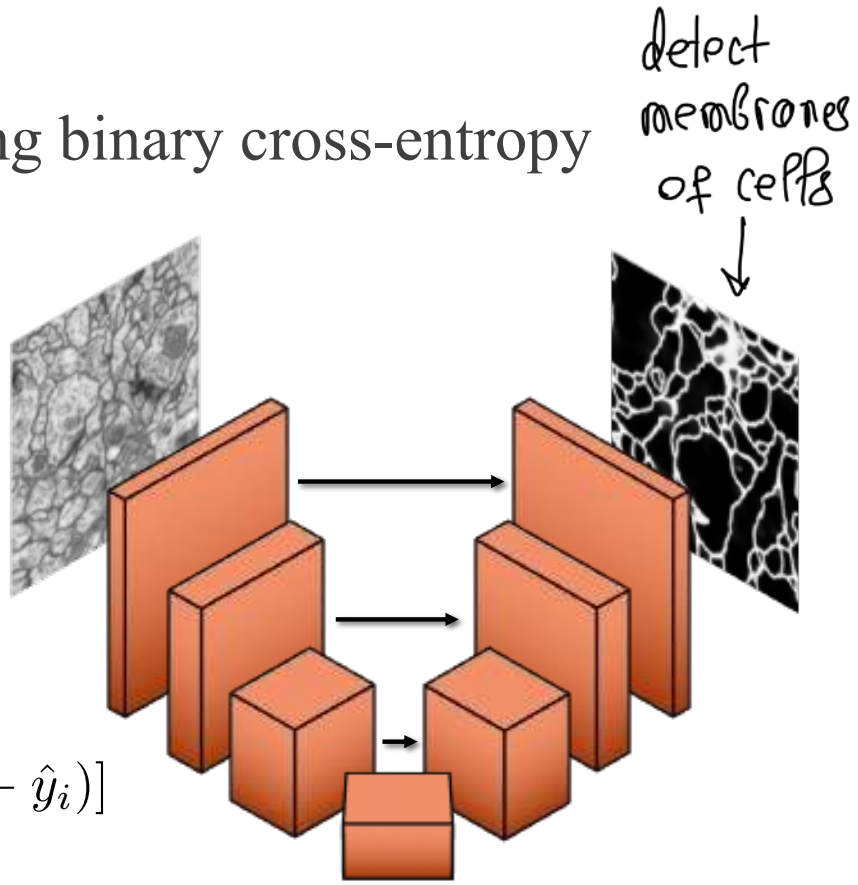
Reminder: U-Net



—> Train a U-Net to output a tubularity map.

Training a U-Net

Train Encoder-decoder U-Net architecture using binary cross-entropy



Minimize

$$L_{bce}(\mathbf{x}, \mathbf{y}; \mathbf{w}) = -\frac{1}{i}\sum_{1}^{P}[y_{\,\mathrm{i}}\log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$$

where

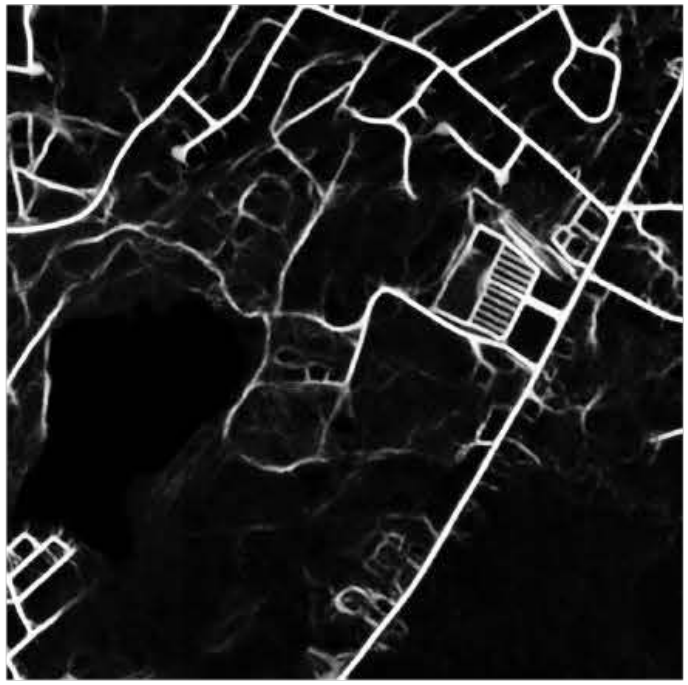
If you have centerline → 1
(not) → 0

- $\hat{\mathbf{y}} = f_{\mathbf{w}}(\mathbf{x})$,
- $\mathbf{x}$ in an input image,
- $\mathbf{y}$ the corresponding ground truth.

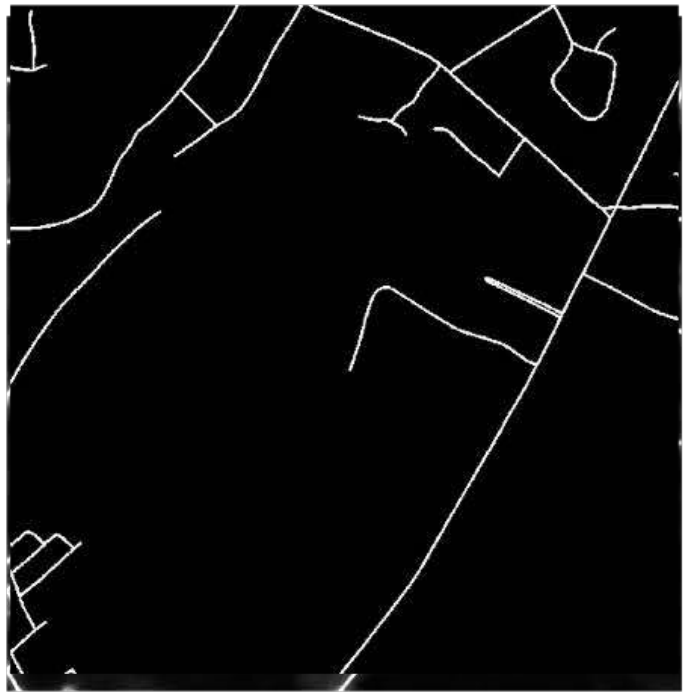
Mosinska et al, CVPR'18.

Network Output

Image



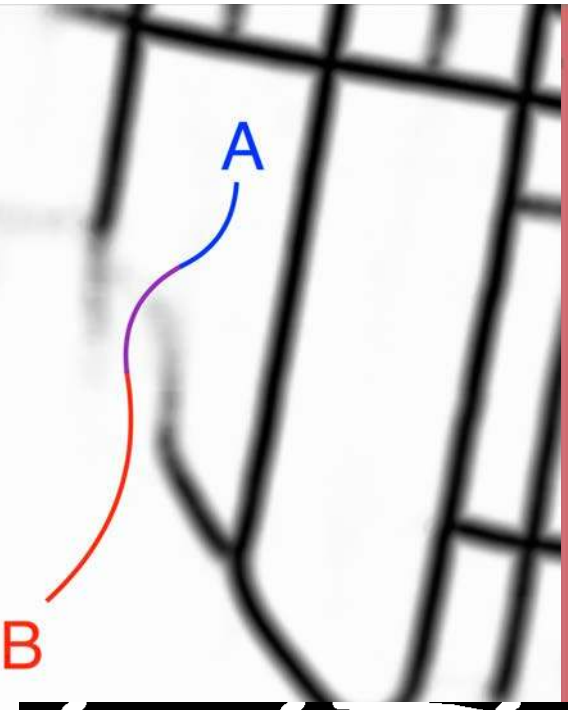
BCE Loss

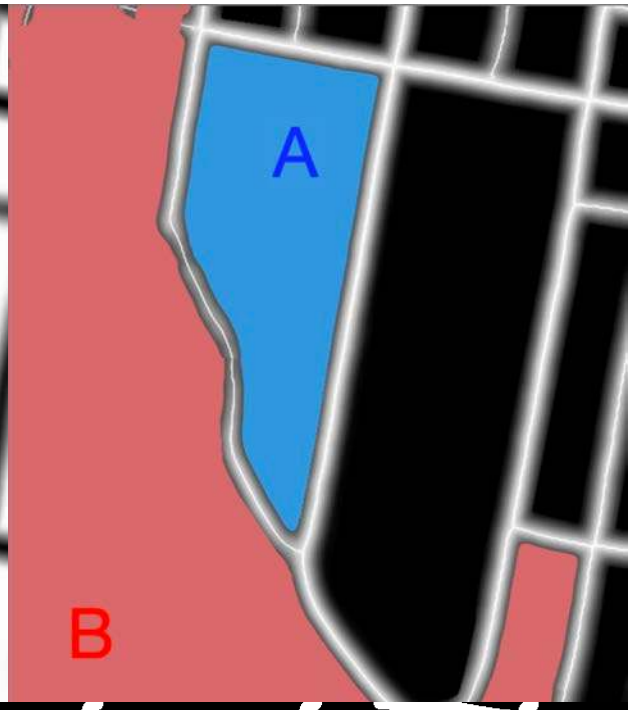


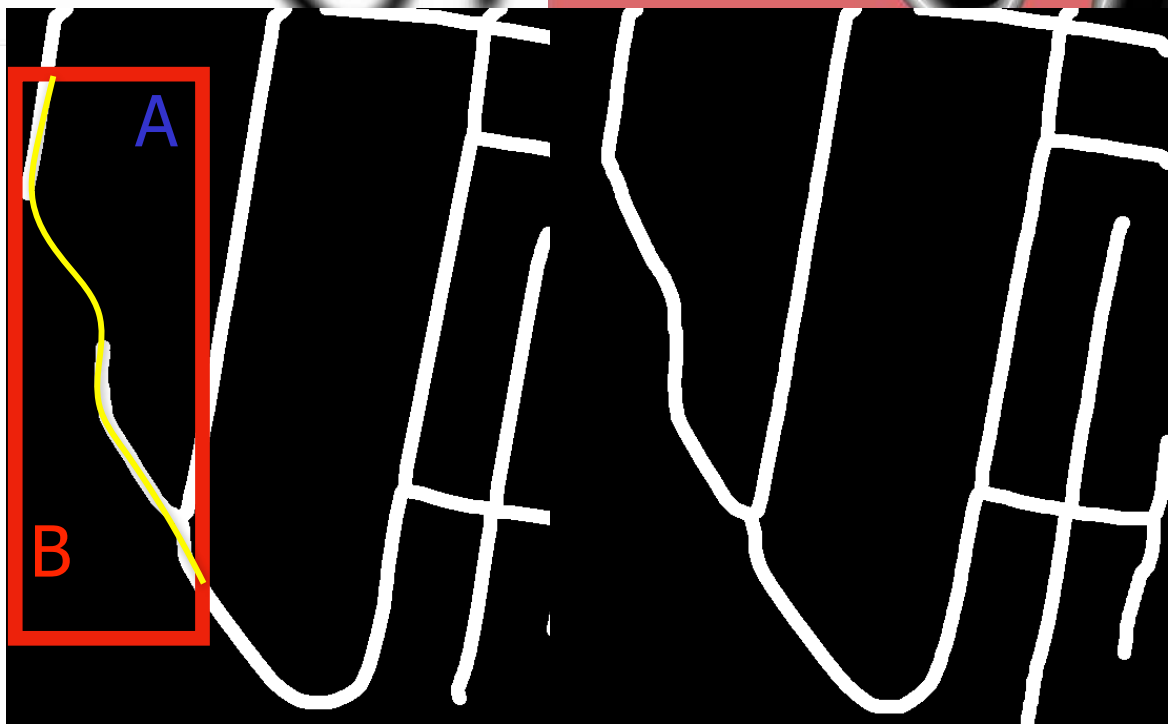
Ground truth

Accounting for Topology

→ Looking for specific lines that have some form of //







Onur et al., PAMI'21

- The yellow road is partially hidden by trees.
- A standard U-Net misses the hidden portion.
- We add to the loss function used to train the network a term that encourages points such as A and B to be separated.
- The re-trained U-Net now finds the complete road.

BCE and Topology Loss

Iterative Refinement

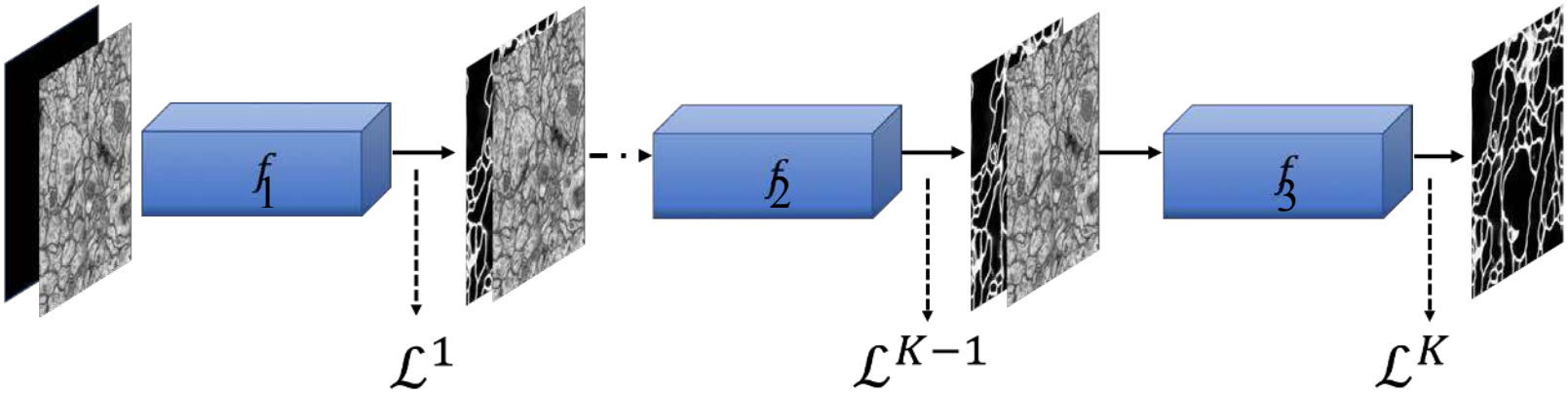
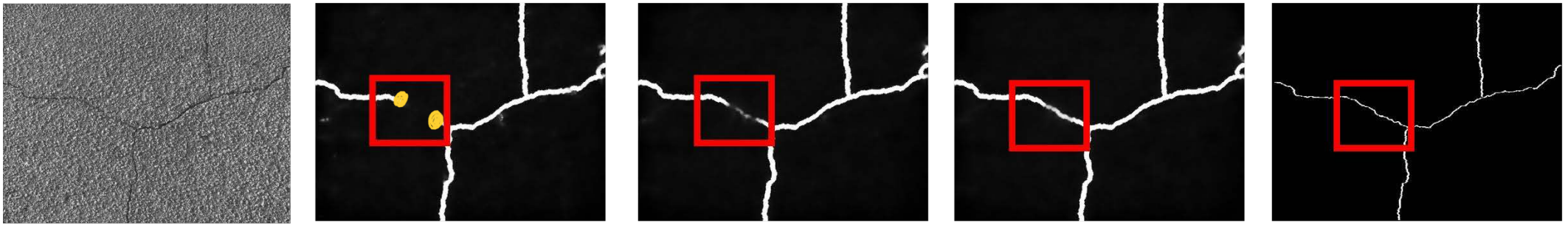


Image Iter 1 Iter 2 Iter 3 Ground truth

progressively better



not sure but likely

Use the same network to progressively refine the results keeping the number of parameters constant

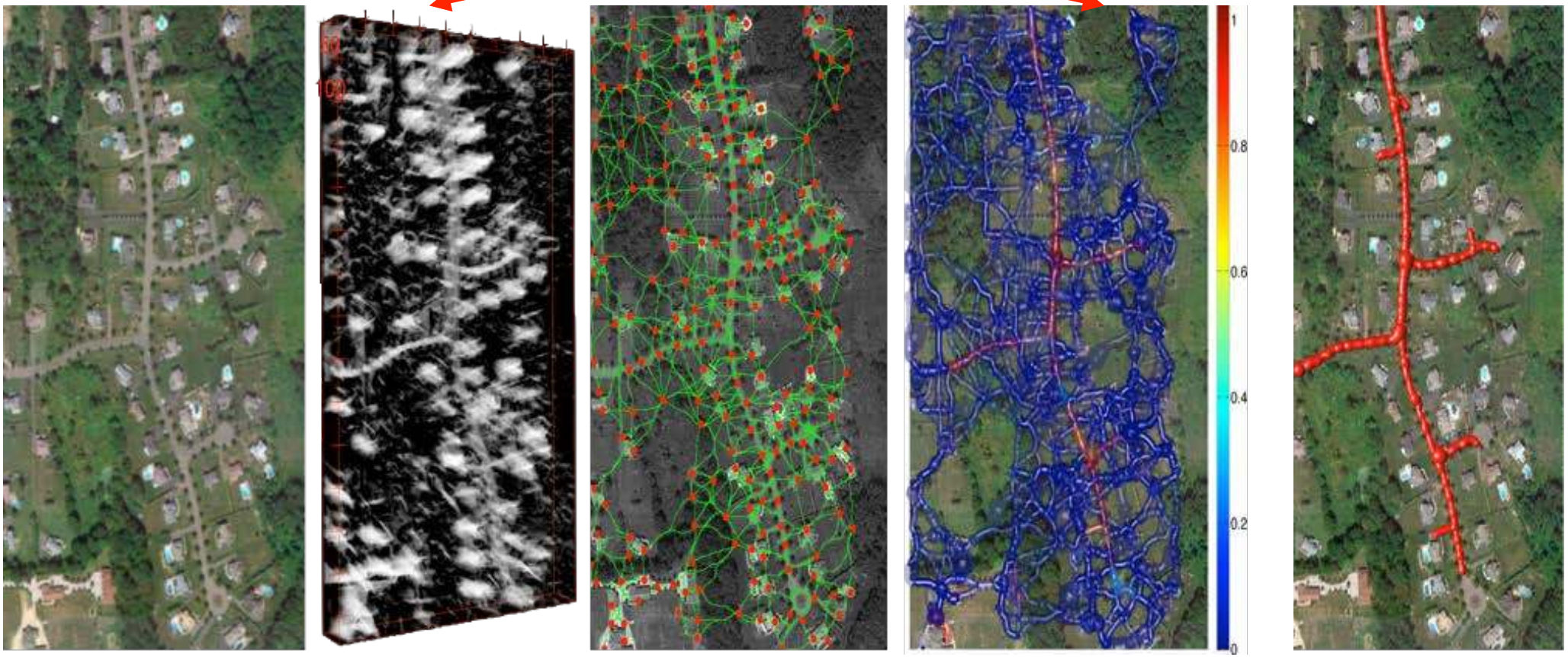
Delineation Steps

1. Compute a probability map.
2. Sample and connect the samples. graph
3. Assign a weight to the paths.
4. Retain the best paths. DP

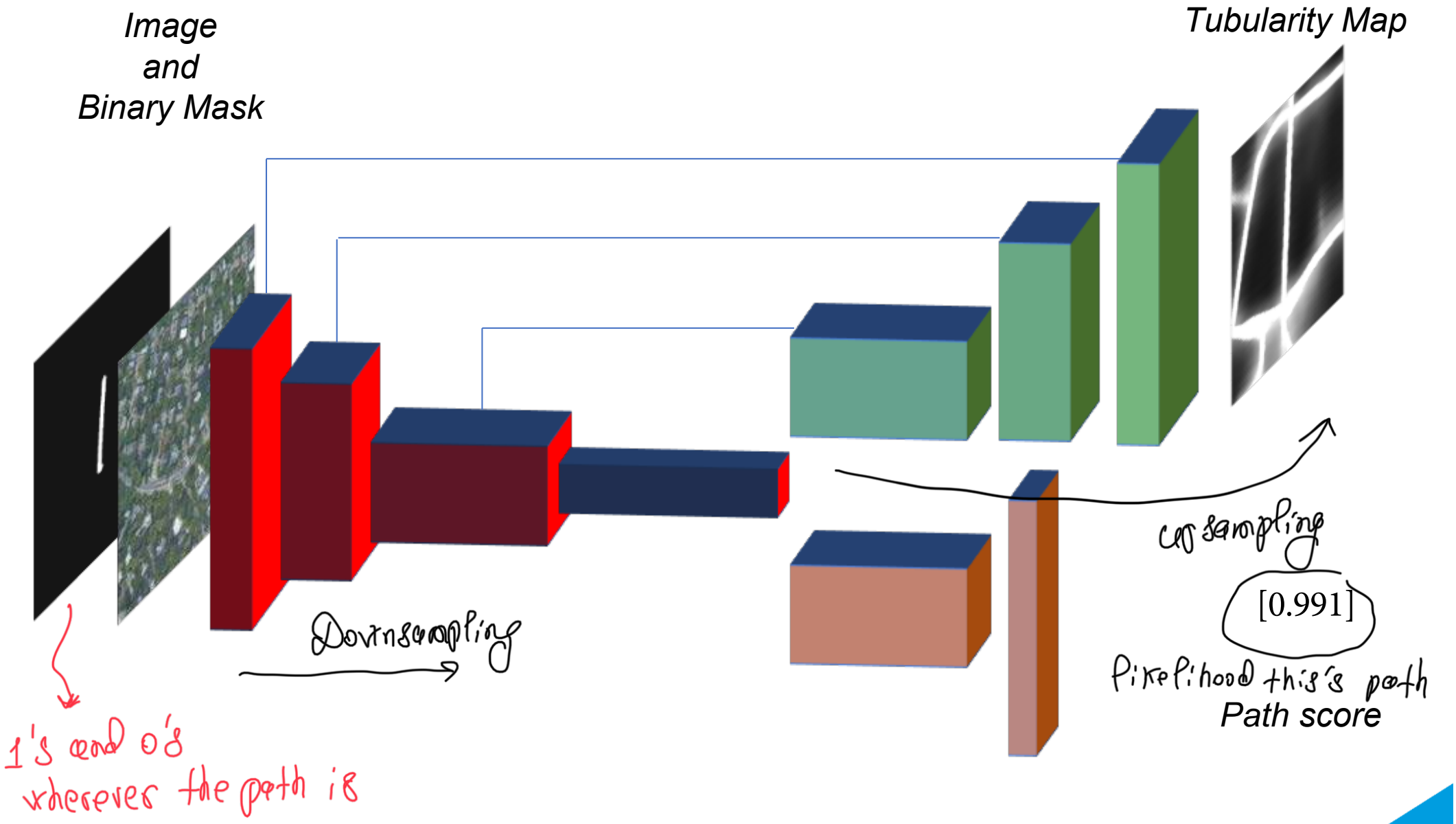
Iterative refinement:
→ Done both at training and testing

Delineation 2019

These two steps are performed by the same network



Dual Use U-Net



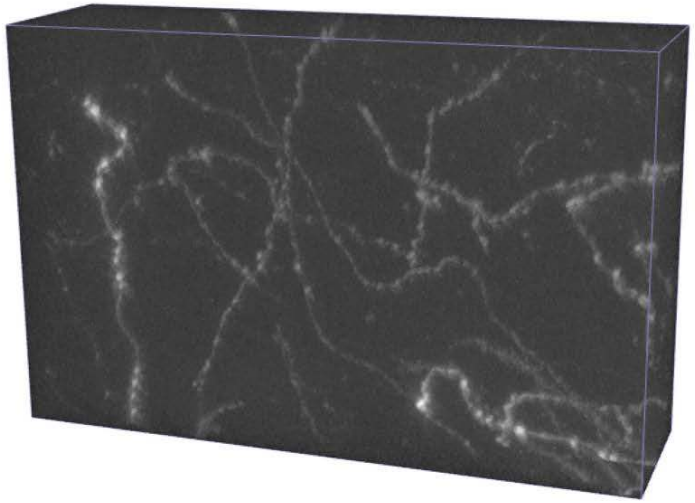
Streets Of Toronto

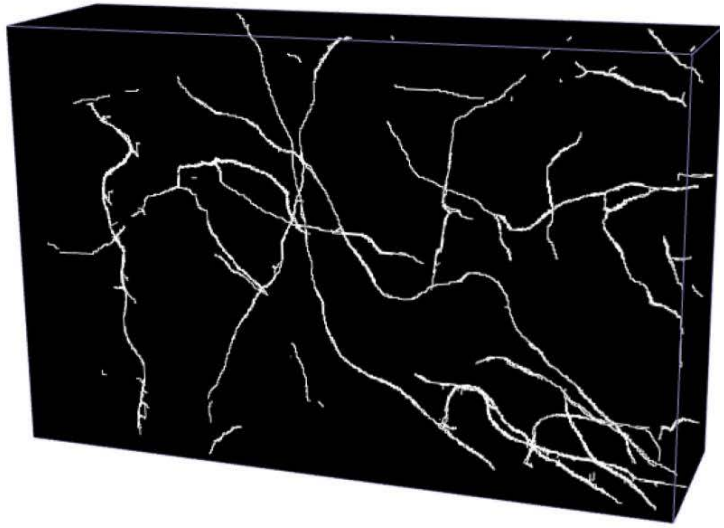
— False negatives
— False positives

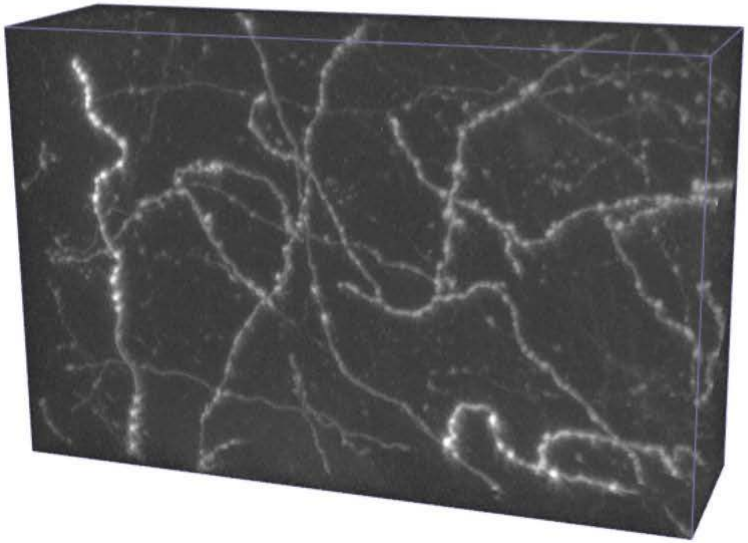
Green → good answers

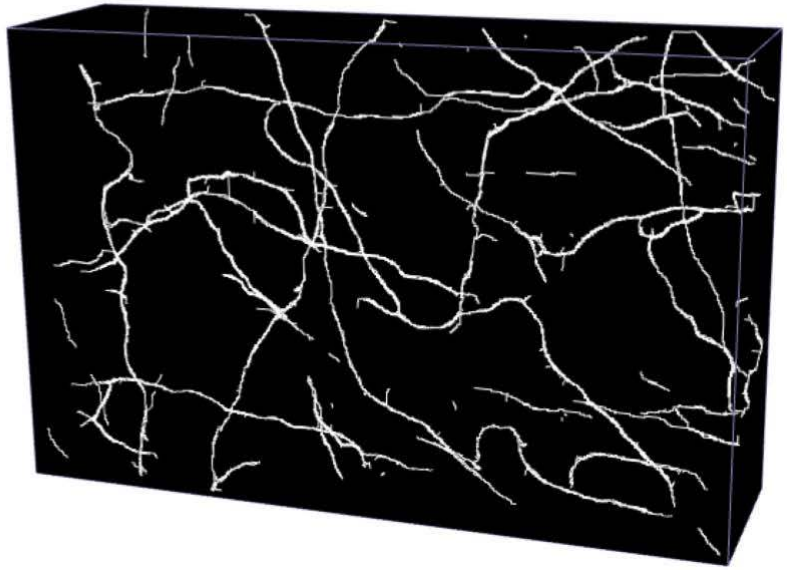
Dendrites And Axons

Generic



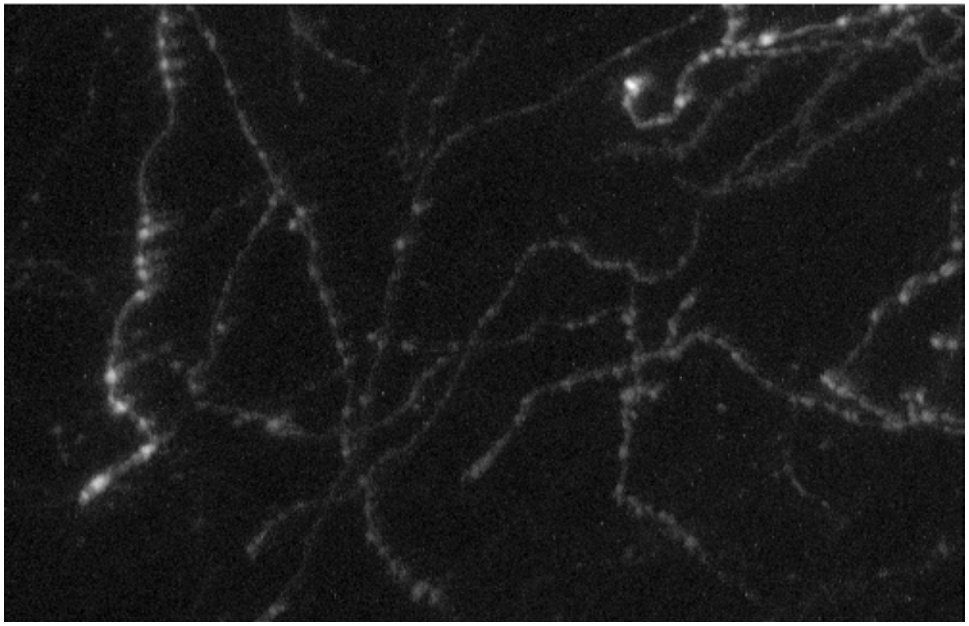




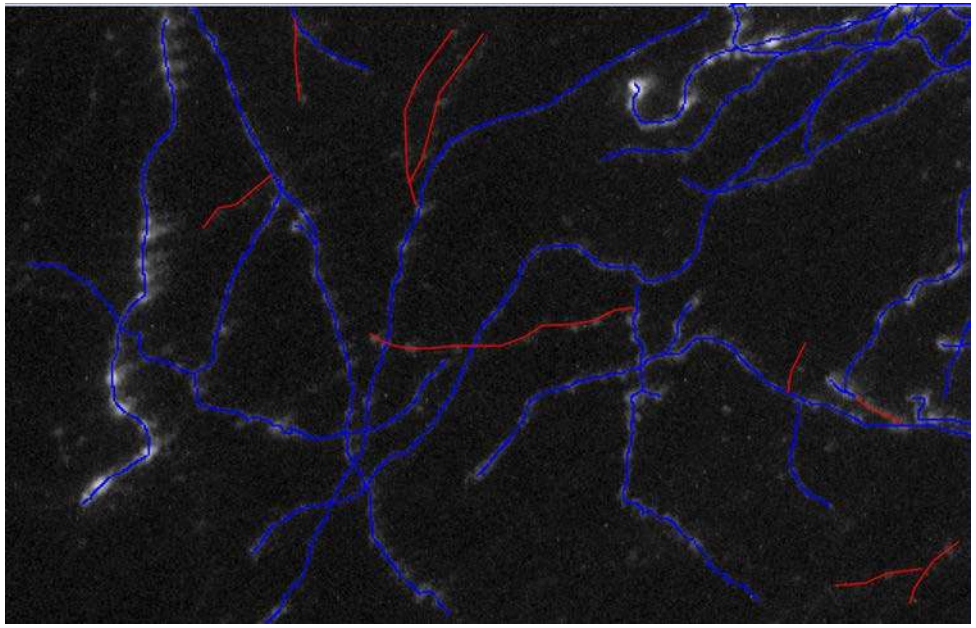


Just train differently

Typical Annotations



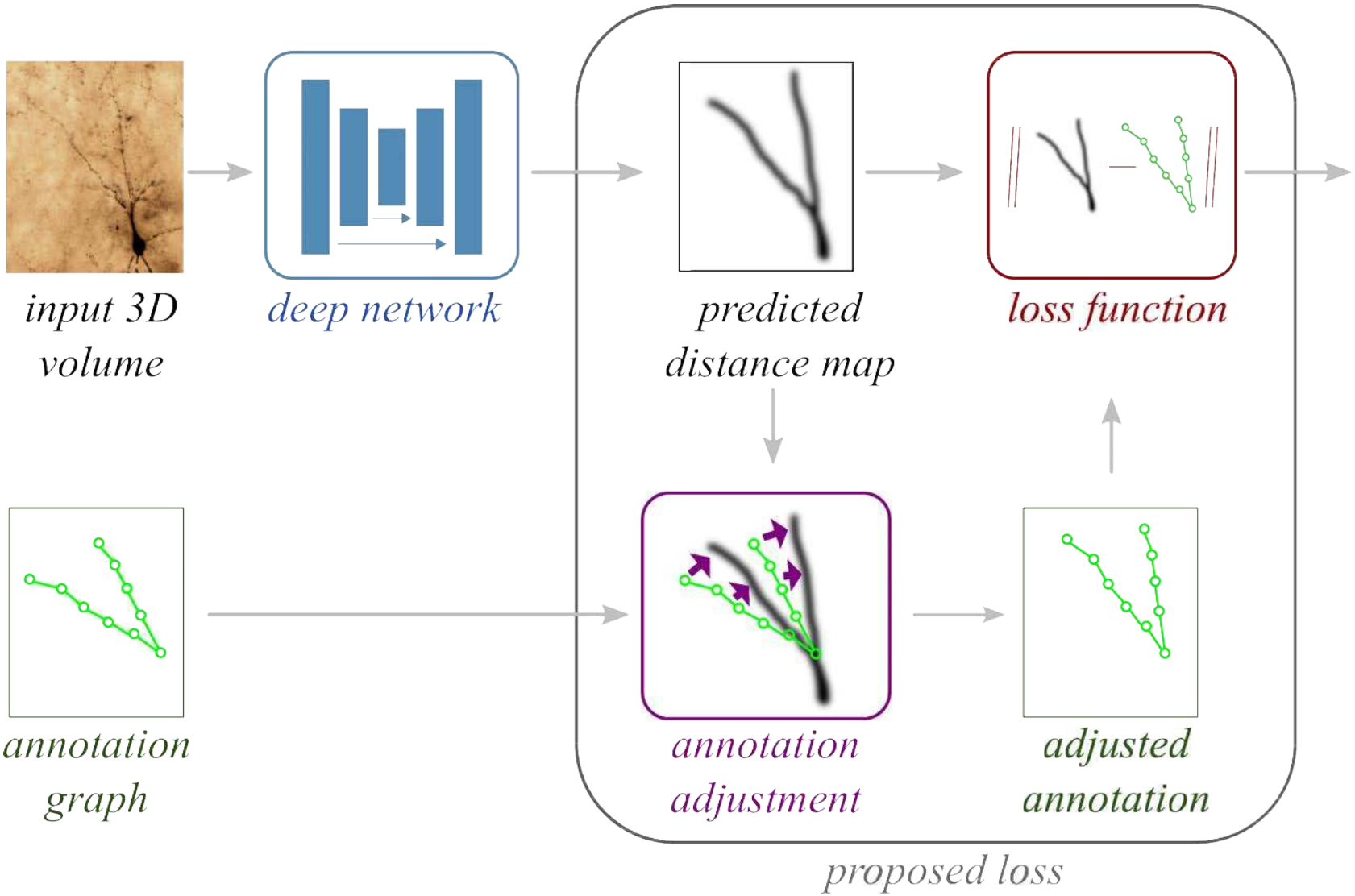
Original Image



"Ground truth" + Mistakes

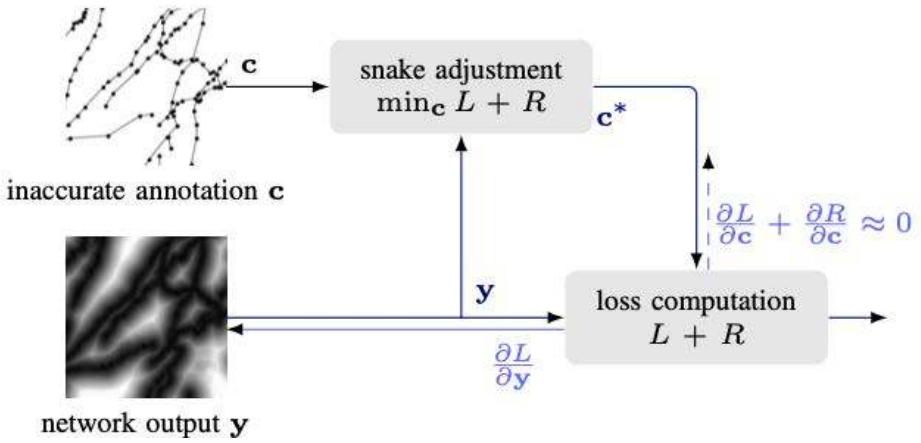
—> Human annotations are often imprecise.

Correcting the Annotations



To account for annotation inaccuracies during training, we jointly train the network and adjust the annotations while preserving their topology.

Annotations as Network Snakes



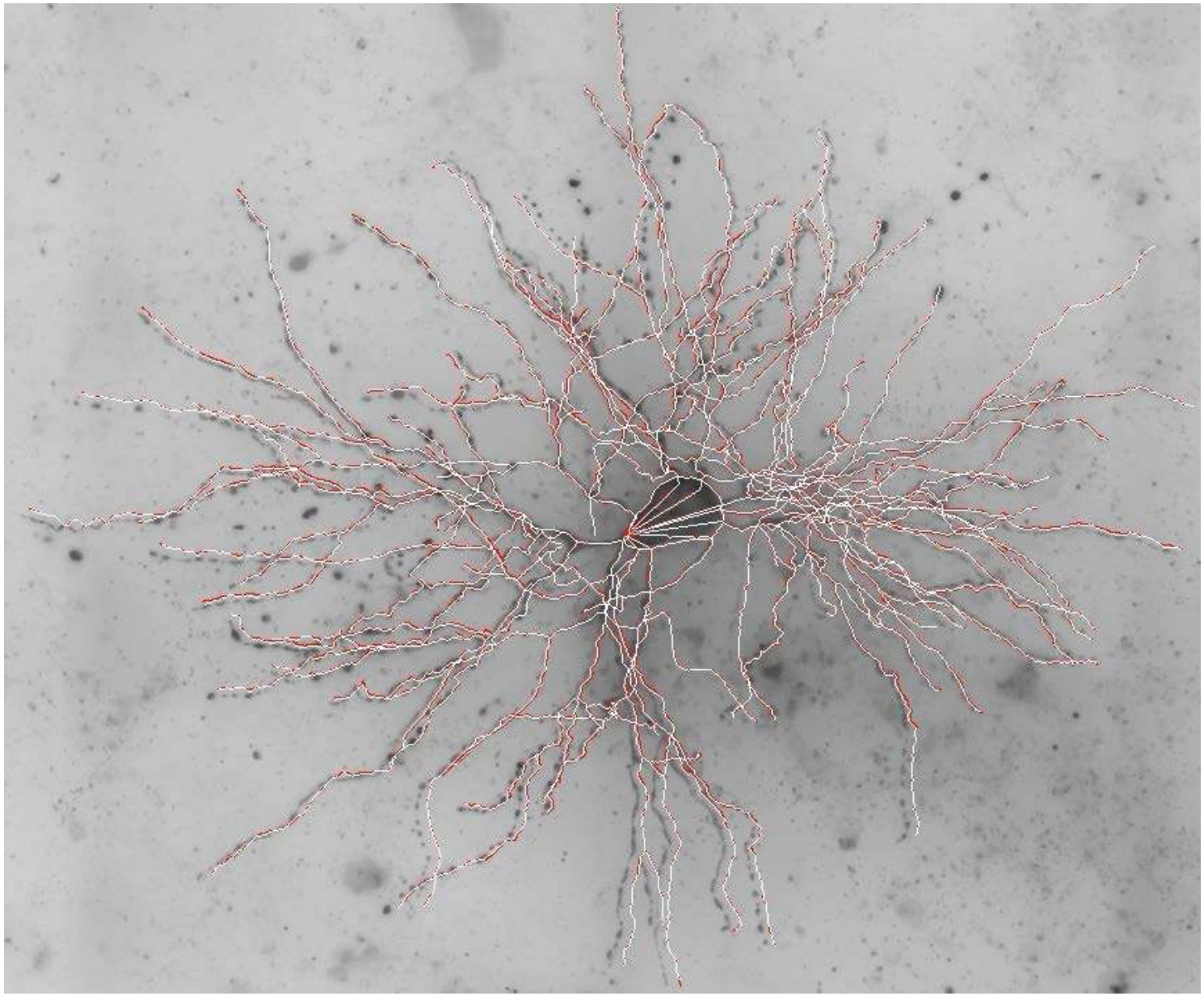
$$\Theta^*, \mathbf{C}^* = \text{argmin}_{\Theta,\mathbf{C}} \sum_{i=1}^{N} \mathscr{L}\left(D(\mathbf{c}_i), \mathbf{y}_i\right) + R(\mathbf{c}_i)$$

Distance between network output and annotations.

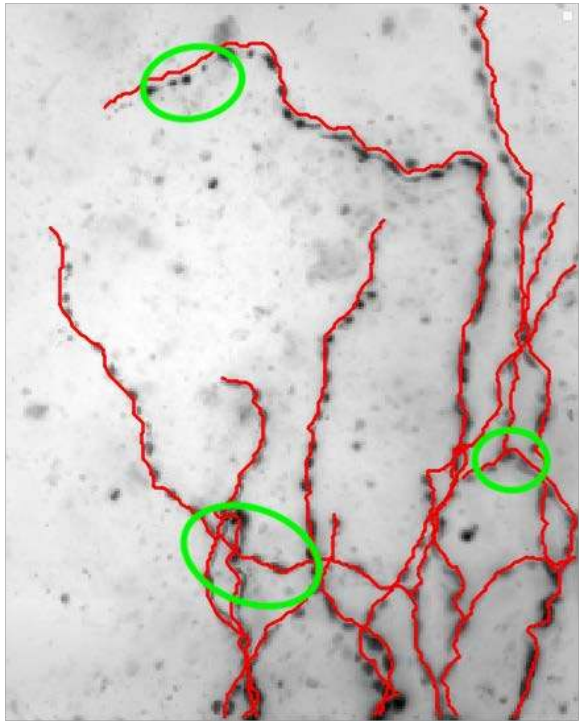
where

- The y_i are the network outputs;
- The $\mathbf{c}_i$ are the annotation vertices;
- $\mathbf{C}$ is the vector obtained by concatenating all the $\mathbf{c}_i$;
- D is a distance transform ;
- $\mathscr{L}$ is the MSE loss;
- R is a regularization term.

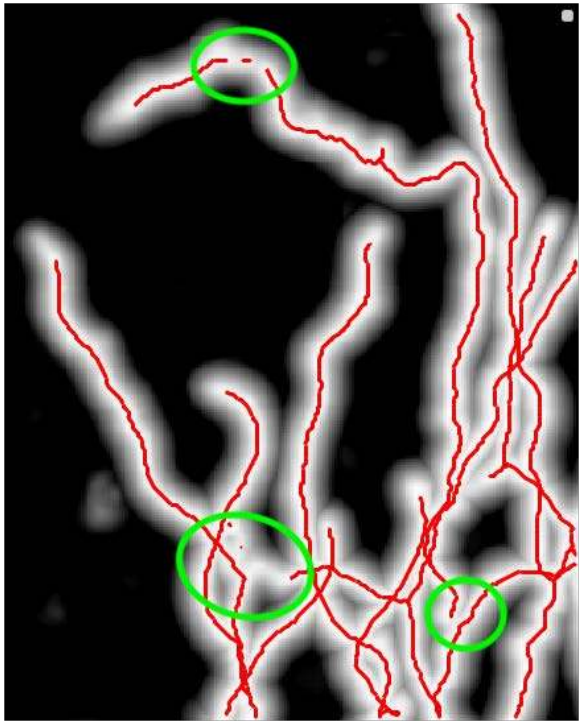
Snake Optimization



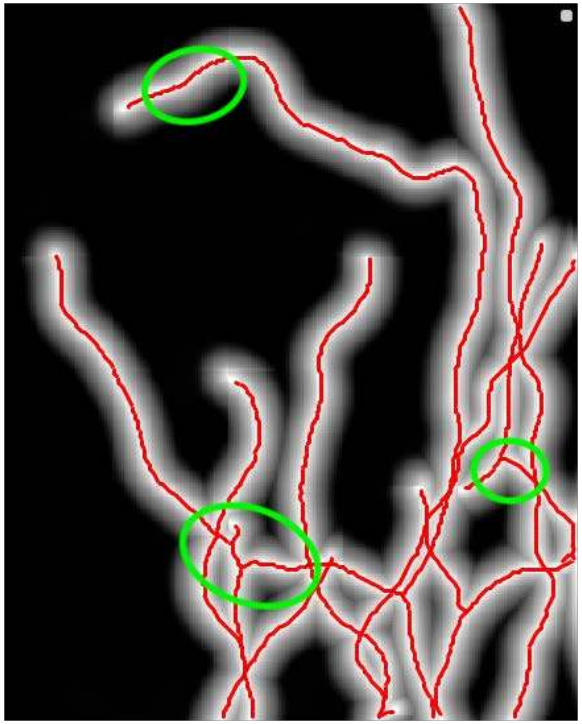
Improved Results



Annotated image



Vanilla U-Net



Network snakes

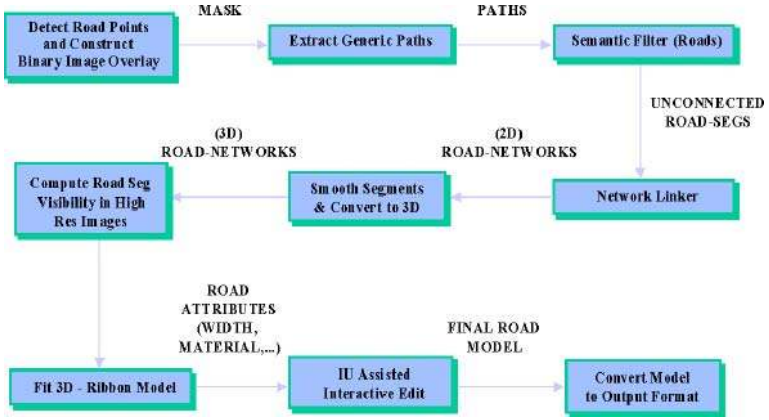
On the Job

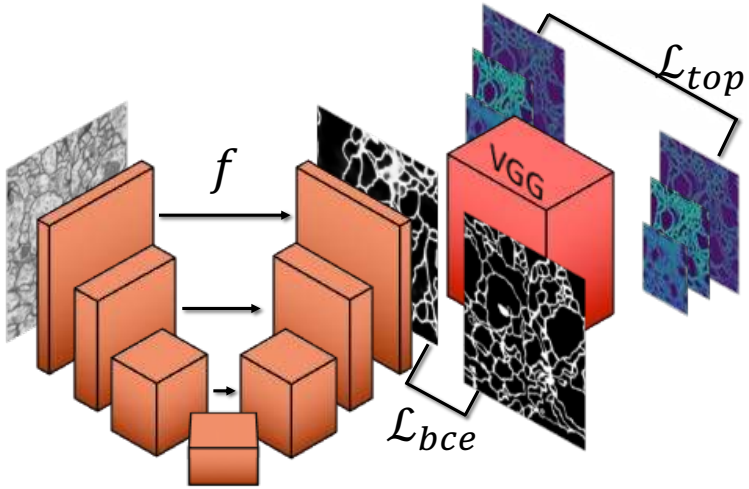
The new job is good, we do of course a lot of Deep Learning, but **also some good old-school computer vision** e.g. registration 😉 So the material from Computer Vision class is definitely helpful and I wouldn't change it to another all-Deep Learning class (even in the light of today's Turing Award).

Best,

Agata

1998 - 2038





1998 2018 2038

It is difficult to make predictions, especially about the future.
Sometimes attributed to Niels Bohr.

In Short

- Edge and image information is noisy.
- Models are required to make sense out of it.

→ An appropriate combination of graph-based techniques, machine learning, and semi-automated tools is required.