

# Algorithms for Image Analysis

---

## ***2D Segmentation (part II)***

### ***Deformable Models***

Acknowledgements: many slides from the University of Manchester,  
demos from Visual Dynamics Group (University of Oxford),

# Deformable Models in 2D

---

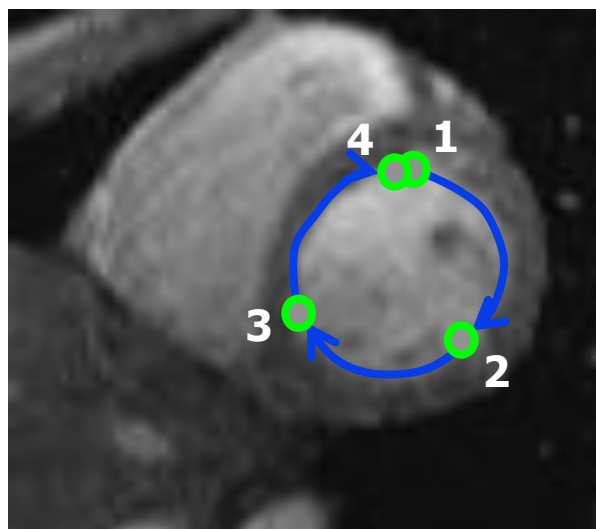
- Active Contours or “snakes”
  - “snakes” vs. “livewire”
  - (discrete) energy formulations for snakes
  - relation to Hidden Markov Models (HMM)
- Optimization (discrete case)
  - Gradient Descent
  - Dynamic Programming (DP), Viterbi algorithm
  - DP versus Dijkstra

Extra Reading: Sonka et.al 5.2.5 and 8.2  
*Active Contours* by Blake and Isard

# “Live-wire” vs. “Snakes”

---

- intelligent scissors [Mortensen, Barrett 1995]
- live-wire [Falcao, Udupa, Samarasekera, Sharma 1998]

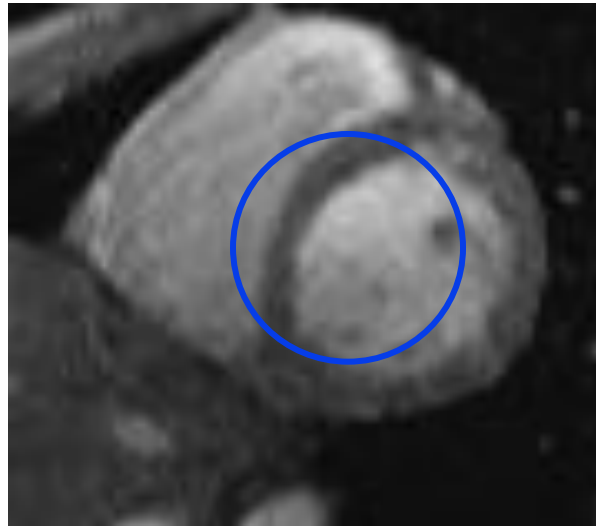


*Shortest paths* on image-based graph connect seeds placed on object boundary

# “Live-wire” vs. “Snakes”

---

- Snakes, active contours [Kass, Witkin, Terzopoulos 1987]
- In general, *deformable models* are widely used

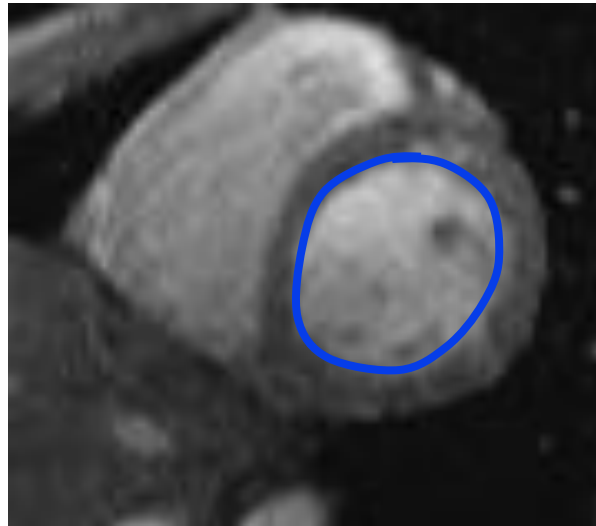


Given: initial contour (model) near desirable object

# “Live-wire” vs. “Snakes”

---

- Snakes, active contours [Kass, Witkin, Terzopoulos 1987]
- In general, *deformable models* are widely used



Given: initial contour (model) near desirable object  
Goal: evolve the contour to fit exact object boundary

# Tracking via deformable models

---

1. Use final contour/model extracted at frame  $t$  as an initial solution for frame  $t+1$
2. Evolve initial contour to fit exact object boundary at frame  $t+1$
3. Repeat steps 1 and 2 for  $t' = t+1$

# Tracking via deformable models

---

Acknowledgements: [Visual Dynamics Group](#), Dept. Engineering Science, University of Oxford.



Applications:

- Traffic monitoring
- Human-computer interaction
- Animation
- Surveillance
- Computer Assisted Diagnosis in medical imaging

# Tracking via deformable models

---



Tracking Heart Ventricles

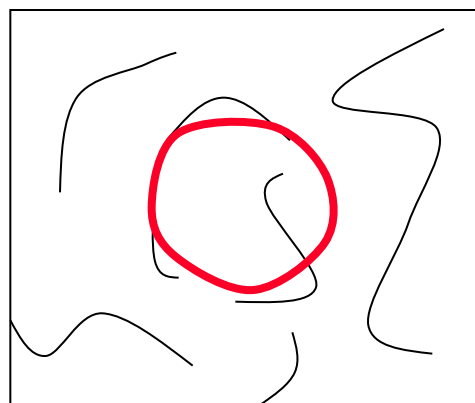


gradient descent w.r.t. some  
function describing snake's quality

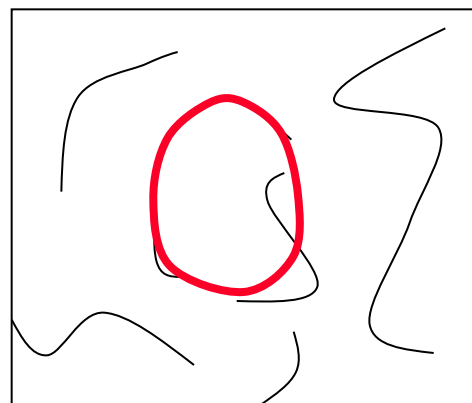
# “Snakes”

---

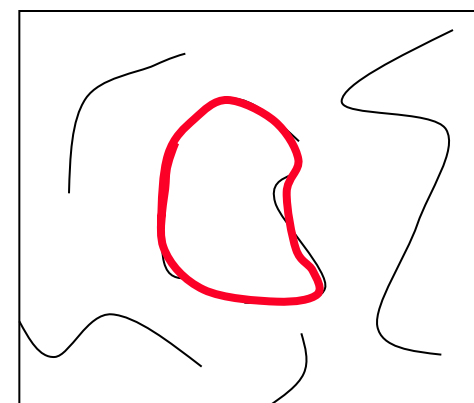
- A smooth 2D curve which matches to image data
- Initialized near target, iteratively refined
- Can restore missing data



initial



intermediate

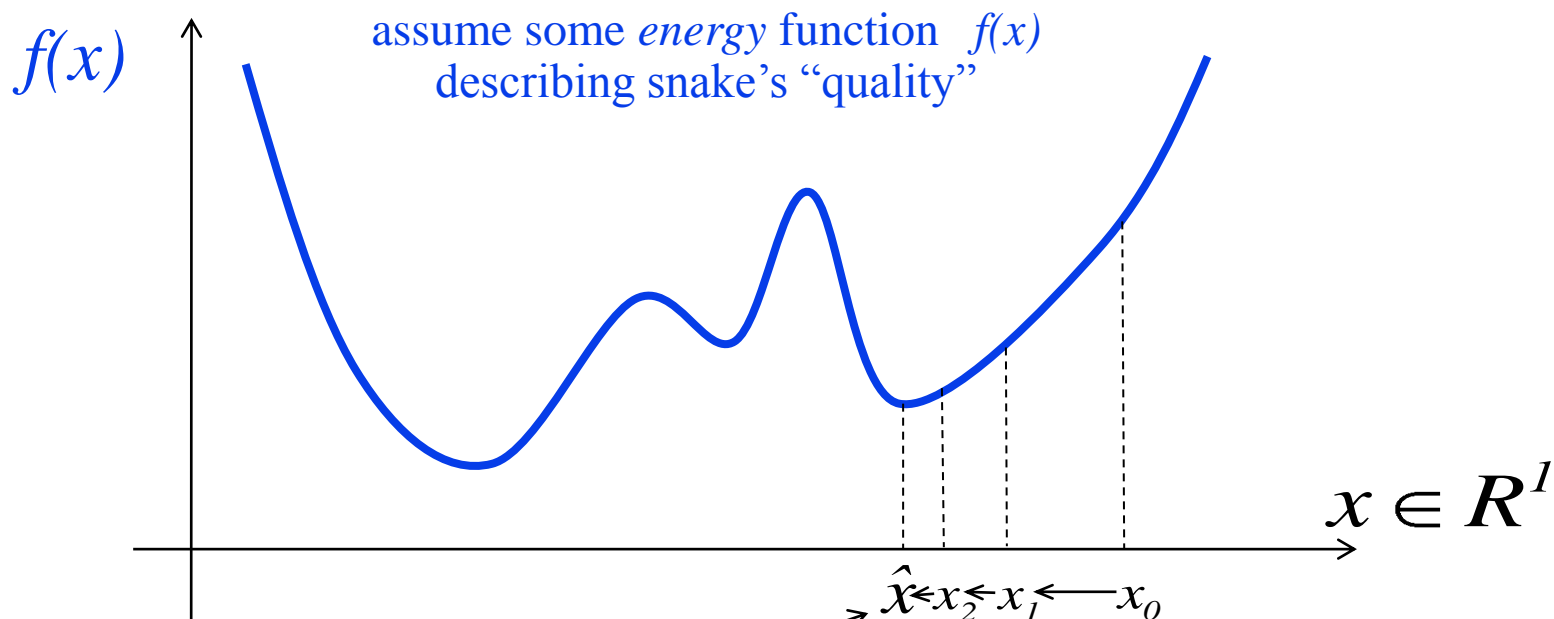


final

Q: How does that work? ....

# Preview

for simplicity, assume that  
"snake" is a vector (or point) in  $R^1$



local minima  
for  $f(x)$

gradient descent for 1D functions

$$x_{i+1} = x_i - \Delta t \cdot f'(x_i)$$

**Q: Is snake (contour) a point in some space? ... Yes**

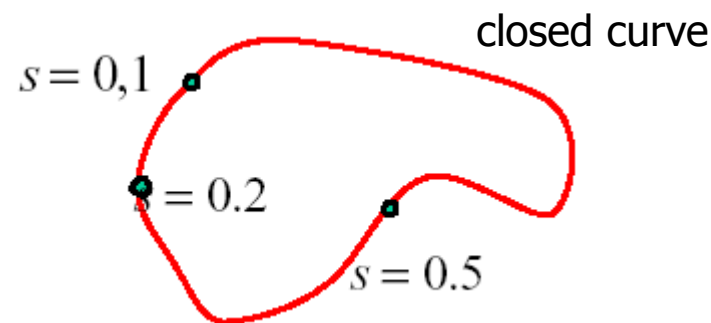
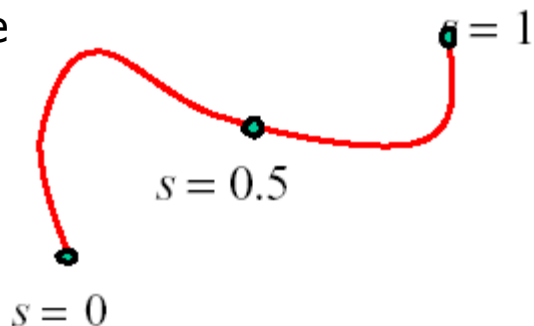
# Parametric Curve Representation

(continuous case)

- A curve can be represented by 2 functions

$$\mathbf{v}(s) = (x(s), y(s)) \quad \begin{matrix} \text{parameter} \\ 0 \leq s \leq 1 \end{matrix}$$

open curve



Note: in computer vision and medical imaging the term “snake” is commonly associated with such *parametric representation of contours*. (Other representations will be discussed later!)

Here, contour is a point in  $R^\infty$  (space of functions)

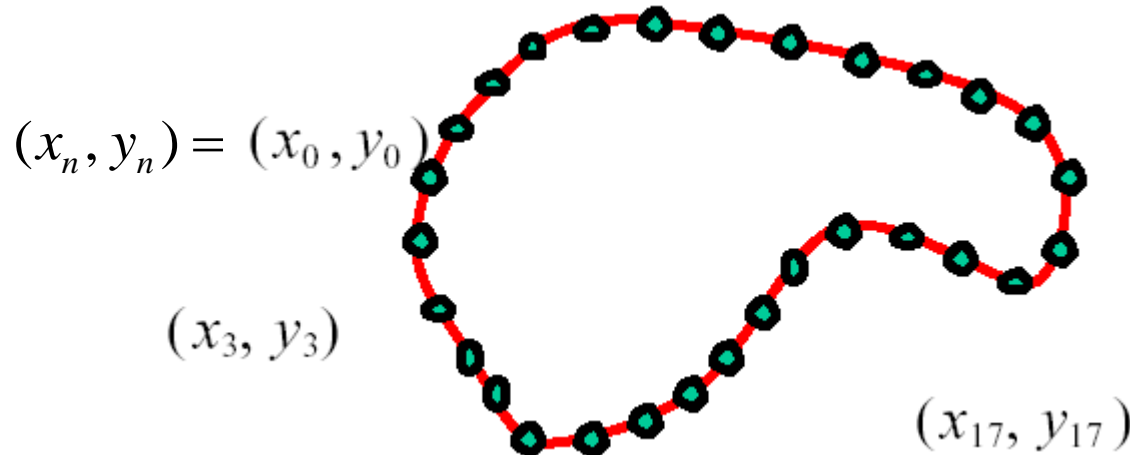
$$\mathbf{C} = \{ \mathbf{v}(s) / s \in [0,1] \}$$

# Parametric Curve Representation

(discrete case)

- A curve can be represented by a set of 2D points

$$\mathbf{v}_i = (x_i, y_i) \quad 0 \leq i < n \quad \text{parameter}$$



Here, contour is a point in  $R^{2n}$

$$\mathbf{C} = (\mathbf{v}_i / 0 \leq i < n) = (x_0, y_0, x_1, y_1, \dots, x_{n-1}, y_{n-1})$$

# Measuring snake's quality: Energy function

Contours can be seen as points  $\mathbf{C}$  in  $R^{2n}$  (or in  $R^\infty$ )

$$\mathbf{C} = (\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{n-1}) = (x_0, y_0, x_1, y_1, \dots, x_{n-1}, y_{n-1})$$

We can define some *energy function*  $E(\mathbf{C})$  that assigns some number (quality measure) to all possible snakes

$$\begin{array}{ccc}
 R^{2n} & \xrightarrow{E(C)} & R \\
 \text{(contours } \mathbf{C}) & & \text{(scalars)}
 \end{array}$$

WHY?: Somewhat philosophical question, but specifying a quality function  $E(C)$  is an objective way to define what "good" means for contours  $C$ .

**Moreover, one can find "the best" contour (segmentation) by optimizing energy  $E(C)$ .**

Q: Did we use any function (energy) to measure quality of segmentation results in

- |                        |                          |
|------------------------|--------------------------|
| 1) image thresholding? | NO                       |
| 2) region growing?     | NO                       |
| 3) live-wire?          | YES (will compare later) |

# Energy function

---

Usually, the total energy of snake is a combination of *internal* and *external* energies

$$E = E_{in} + E_{ex}$$

Internal energy encourages smoothness or any particular shape

Internal energy incorporates prior knowledge about object boundary allowing to extract boundary even if some image data is missing

External energy encourages curve onto image structures (e.g. image edges)

# Internal Energy

(continuous case)

- The smoothness energy at contour point  $\mathbf{v}(s)$  could be evaluated as

$$E_{in}(\mathbf{v}(s)) = \alpha(s) \left| \frac{d\mathbf{v}}{ds} \right|^2 + \beta(s) \left| \frac{d^2\mathbf{v}}{d^2s} \right|^2$$

Elasticity/stretching
Stiffness/bending

Then, the interior energy (smoothness) of the whole snake  $\mathbf{C} = \{\mathbf{v}(s) / s \in [0,1]\}$  is

$$E_{in} = \int_0^1 E_{in}(\mathbf{v}(s)) ds$$

# Internal Energy

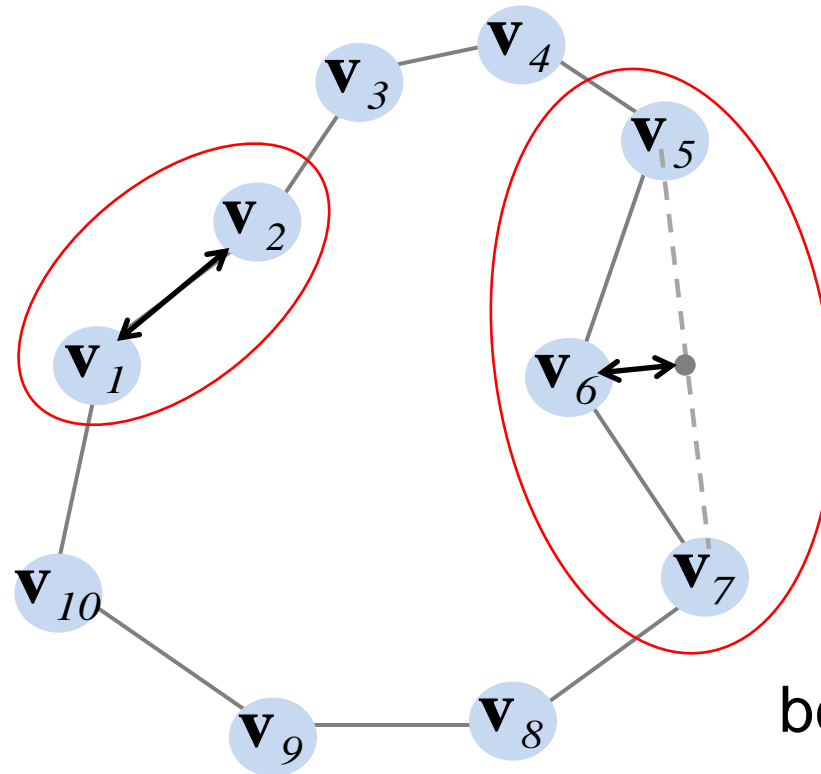
(discrete case)

$$\mathbf{C} = (\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{n-1}) \in \mathbb{R}^{2n}$$

$$\mathbf{v}_i = (x_i, y_i)$$

elastic energy  
(elasticity)

$$\frac{d\mathbf{v}}{ds} \approx \mathbf{v}_{i+1} - \mathbf{v}_i$$



bending energy  
(stiffness)

$$\frac{d^2\mathbf{v}}{ds^2} \approx (\mathbf{v}_{i+1} - \mathbf{v}_i) - (\mathbf{v}_i - \mathbf{v}_{i-1}) = \mathbf{v}_{i+1} - 2\mathbf{v}_i + \mathbf{v}_{i-1}$$



# Internal Energy

(discrete case)

---

$$\mathbf{C} = (\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{n-1}) \in \mathfrak{R}^{2n}$$

$$\frac{d\mathbf{v}}{ds} \approx \mathbf{v}_{i+1} - \mathbf{v}_i$$

$$\mathbf{v}_i = (x_i, y_i)$$

$$\frac{d^2\mathbf{v}}{ds^2} \approx (\mathbf{v}_{i+1} - \mathbf{v}_i) - (\mathbf{v}_i - \mathbf{v}_{i-1}) = \mathbf{v}_{i+1} - 2\mathbf{v}_i + \mathbf{v}_{i-1}$$

$$E_{in} = \sum_{i=0}^{n-1} \boxed{\alpha |\mathbf{v}_{i+1} - \mathbf{v}_i|^2} + \boxed{\beta |\mathbf{v}_{i+1} - 2\mathbf{v}_i + \mathbf{v}_{i-1}|^2}$$

Elasticity

Stiffness

# External energy

---

- The external energy describes how well the curve matches the image data locally
- Numerous forms can be used, attracting the curve toward different image features

# External energy

---

- Suppose we have an image  $I(x,y)$
- Can compute image gradient  $\nabla I = (I_x, I_y)$  at any point
- Edge strength at pixel  $(x,y)$  is  $|\nabla I(x,y)|$
- *External energy* of a contour point  $\mathbf{v}=(x,y)$  could be

$$E_{ex}(\mathbf{v}) = -|\nabla I(\mathbf{v})|^2 = -|\nabla I(x,y)|^2$$

- *External energy* term for the whole snake is

$$E_{ex} = \int_0^1 E_{ex}(\mathbf{v}(s)) ds$$

continuous case  
 $\mathbf{C} = \{\mathbf{v}(s) | s \in [0,1]\}$

$$E_{ex} = \sum_{i=0}^{n-1} E_{ex}(\mathbf{v}_i)$$

discrete case  
 $\mathbf{C} = \{\mathbf{v}_i | 0 \leq i < n\}$

# Basic Elastic Snake

- The total energy of a basic elastic snake is

$$E = \alpha \cdot \int_0^1 \left| \frac{dv}{ds} \right|^2 ds - \int_0^1 \left| \nabla I(v(s)) \right|^2 ds$$

continuous case  
 $C = \{v(s) / s \in [0,1]\}$

$$E = \alpha \cdot \sum_{i=0}^{n-1} |v_{i+1} - v_i|^2 - \sum_{i=0}^{n-1} \left| \nabla I(v_i) \right|^2$$

discrete case  
 $C = \{v_i / 0 \leq i < n\}$

elastic smoothness term  
(interior energy)

image data term  
(exterior energy)

# Basic Elastic Snake

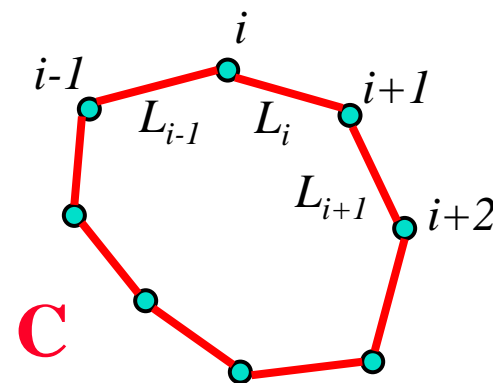
(discrete case)

$$\mathbf{C} = (\mathbf{v}_i / 0 \leq i < n) = (x_0, y_0, x_1, y_1, \dots, x_{n-1}, y_{n-1})$$

$$E_{in} = \alpha \cdot \sum_{i=0}^{n-1} L_i^2$$

This can make a curve shrink  
(to a point)

$$= \alpha \cdot \sum_{i=0}^{n-1} (x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2$$



$$E_{ex} = - \sum_{i=0}^{n-1} |\nabla I(x_i, y_i)|^2$$

$$= - \sum_{i=0}^{n-1} |I_x(x_i, y_i)|^2 + |I_y(x_i, y_i)|^2$$

# Basic Elastic Snake

(discrete case)

---

- The problem is to find contour  $\mathbf{C} = (x_0, \dots, x_{n-1}, y_0, \dots, y_{n-1}) \in R^{2n}$  that minimizes

$$E(\mathbf{C}) = \alpha \cdot \sum_{i=0}^{n-1} (x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2 - \sum_{i=0}^{n-1} |I_x(x_i, y_i)|^2 + |I_y(x_i, y_i)|^2$$

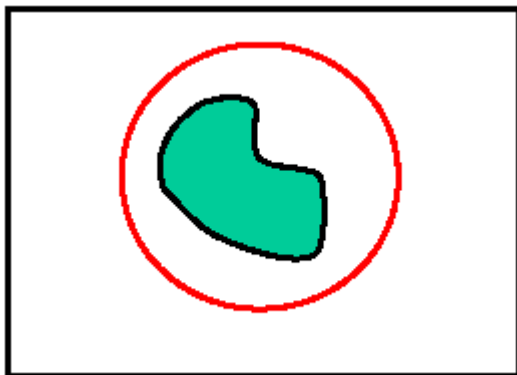
- Optimization problem for function of  $2n$  variables
  - can compute local minima via **gradient descent** (coming soon)
  - potentially more robust option: **dynamic programming** (later)

# Basic Elastic Snake

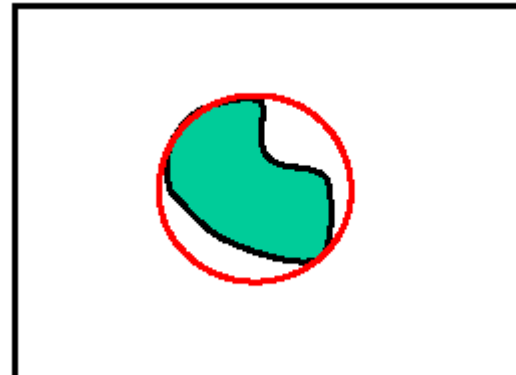
## Synthetic example

---

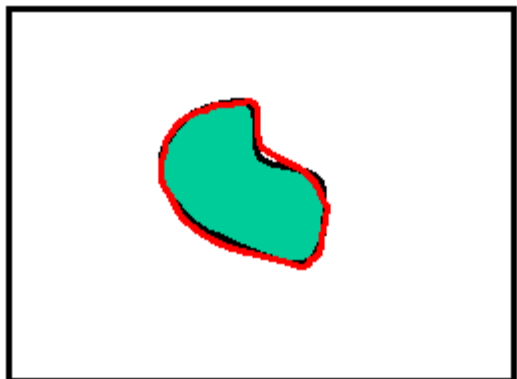
(1)



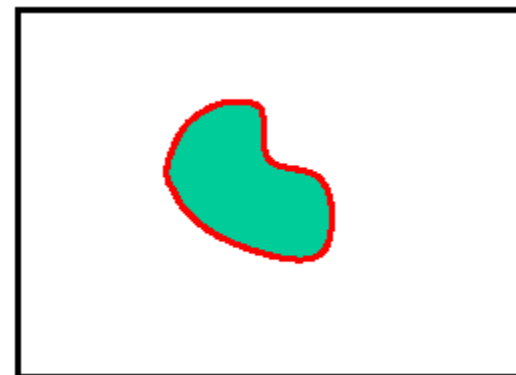
(2)



(3)



(4)

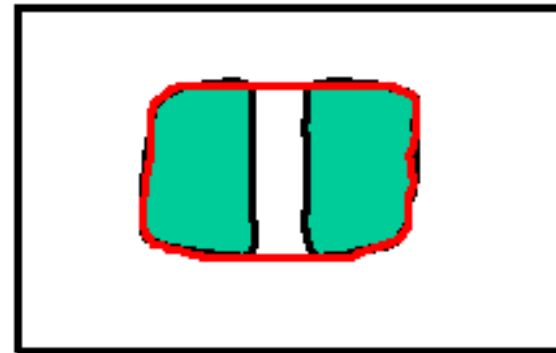
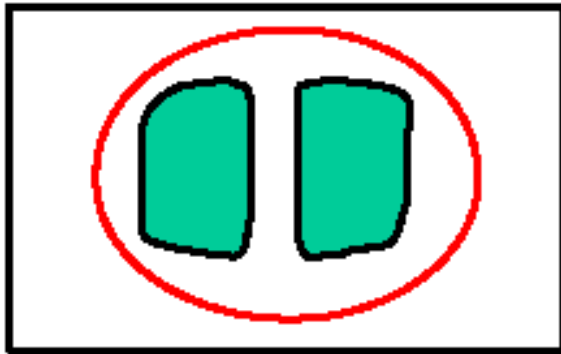


# Basic Elastic Snake

## Dealing with missing data

---

- The smoothness constraint can deal with missing data:





# Basic Elastic Snake

## Relative weighting

---

- Notice that the strength of the internal elastic component can be controlled by a parameter,  $\alpha$

$$E_{in} = \alpha \cdot \sum_{i=0}^{n-1} L_i^2$$

- Increasing this increases stiffness of curve

large  $\alpha$ medium  $\alpha$ small  $\alpha$

# Encouraging point spacing

---

- To stop the curve from shrinking to a point

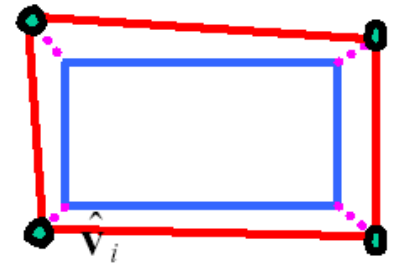
$$E_{in} = \alpha \cdot \sum_{i=0}^{n-1} (L_i - \hat{L}_i)^2$$

- encourages particular point separation

# Simple shape prior

- If object is some smooth variation on a known shape, use

$$E_{in} = \alpha \cdot \sum_{i=0}^{n-1} (v_i - \hat{v}_i)^2$$



– where  $\{ \hat{v}_i \}$  give points of the basic shape

- May use a statistical (Gaussian) shape model

$$E_{in} = -\ln N(v | \hat{v}) \propto (v - \hat{v})^T \cdot C \cdot (v - \hat{v})$$

# Alternative External Energies

---

## ■ Directed gradient measures

$$E_{ex} = - \sum_{i=0}^{n-1} \left| u_{x,i} \cdot I_x(\mathbf{v}_i) + u_{y,i} \cdot I_y(\mathbf{v}_i) \right|$$

- Where  $\mathbf{u}_i = (u_{x,i}, u_{y,i})$  is the unit normal to the boundary at contour point  $v_i$
- This gives a good response when the boundary has the same direction as the edge, but weaker responses when it does not

# Additional Constraints

---

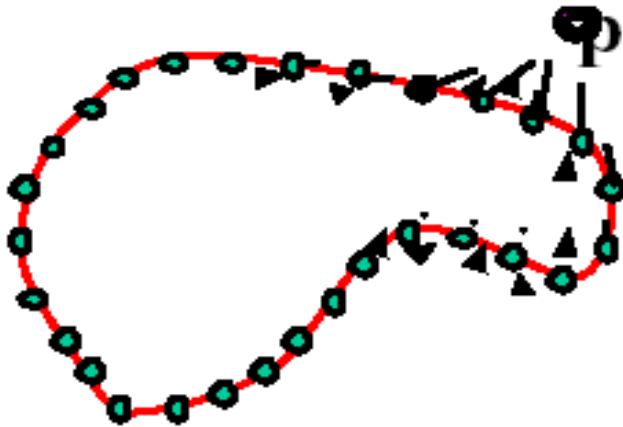
- Snakes originally developed for interactive image segmentation
- Initial snake result can be nudged where it goes wrong
- Simply add extra external energy terms to
  - Pull nearby points toward cursor, or
  - Push nearby points away from cursor

# Interactive (external) forces

---

- Pull points towards cursor:

$$E_{pull} = - \sum_{i=0}^{n-1} \frac{r^2}{|v_i - p|^2}$$



Nearby points get  
pulled hardest

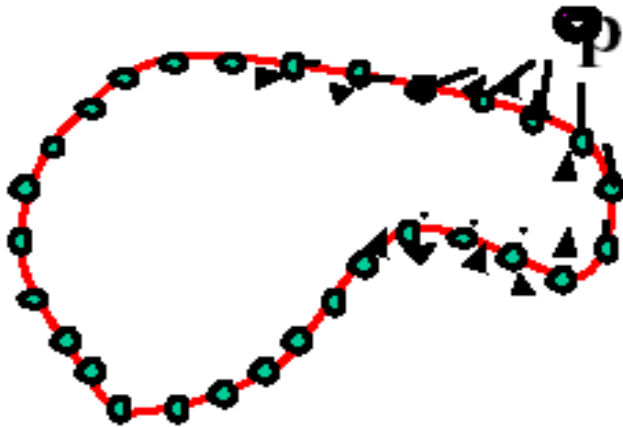
Negative sign gives  
better energy for  
positions near  $p$

# Interactive (external) forces

---

- Push points from cursor:

$$E_{push} = + \sum_{i=0}^{n-1} \frac{r^2}{|v_i - p|^2}$$



Nearby points get  
pushed hardest

Positive sign gives  
better energy for  
positions far from p

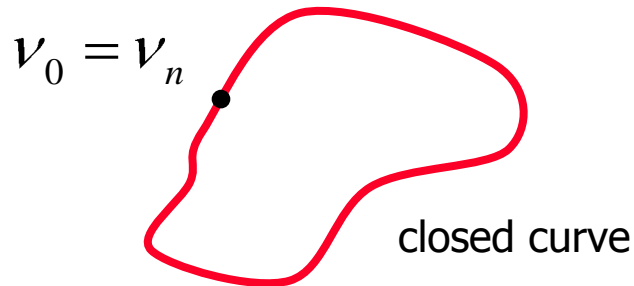
# Dynamic snakes

---

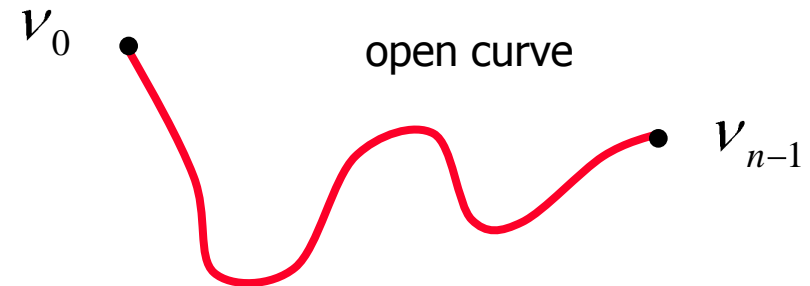
- Adding motion parameters as variables (for each snake node)
- Introduce energy terms for motion consistency
- primarily useful for tracking (nodes represent real tissue elements with mass and kinematic energy)



# Open and Closed Curves



$$E_{in} = \sum_{i=0}^{n-1} (v_{i+1} - v_i)^2$$



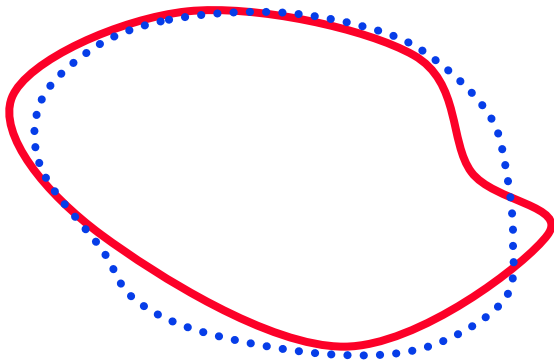
$$E_{in} = \sum_{i=0}^{n-2} (v_{i+1} - v_i)^2$$

- When using an open curve we can impose constraints on the end points (e.g. end points may have fixed position)
  - **Q:** What are similarities and differences with the live-wire if the end points of an open snake are fixed?

# Discrete Snakes Optimization

---

- At each iteration we compute a **new** snake position within proximity to the **previous** snake
- **New** snake energy should be smaller than the **previous** one
- Stop when the energy can not be decreased within local neighborhood of the snake (local energy minima)

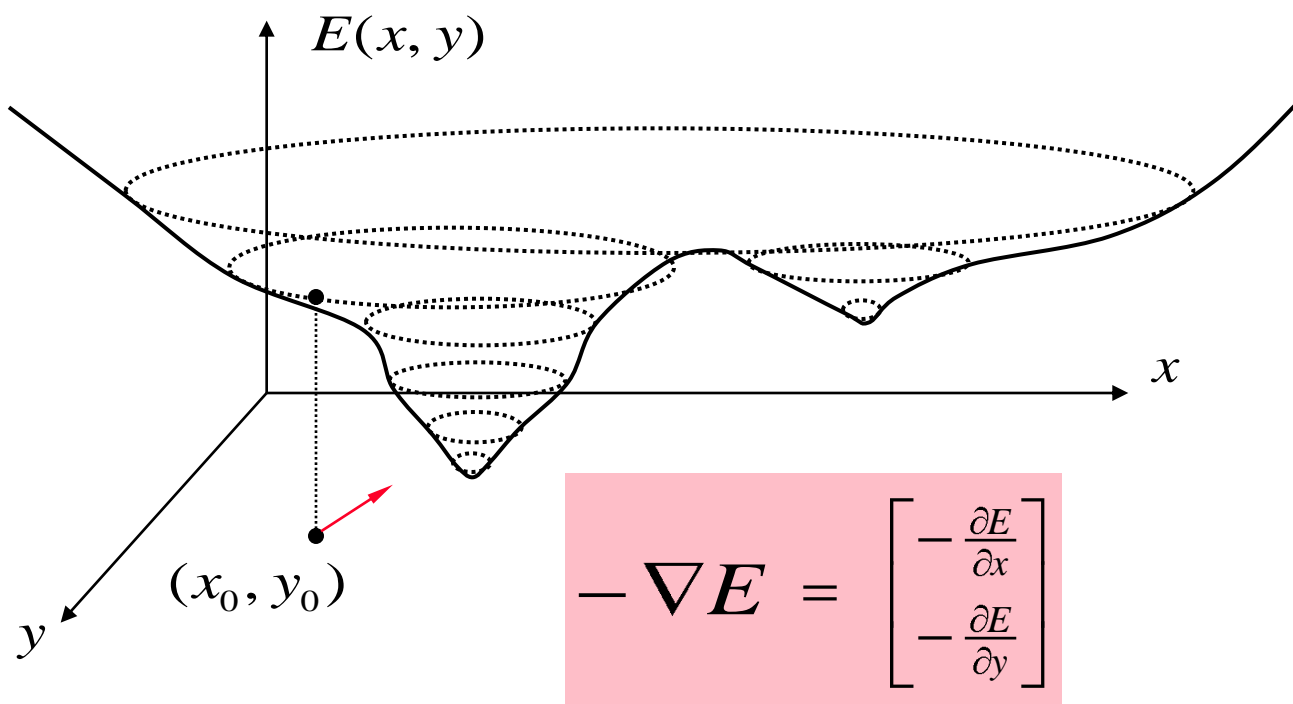


## Optimization Methods

1. *Gradient Descent*
2. *Dynamic Programming*

# Gradient Descent

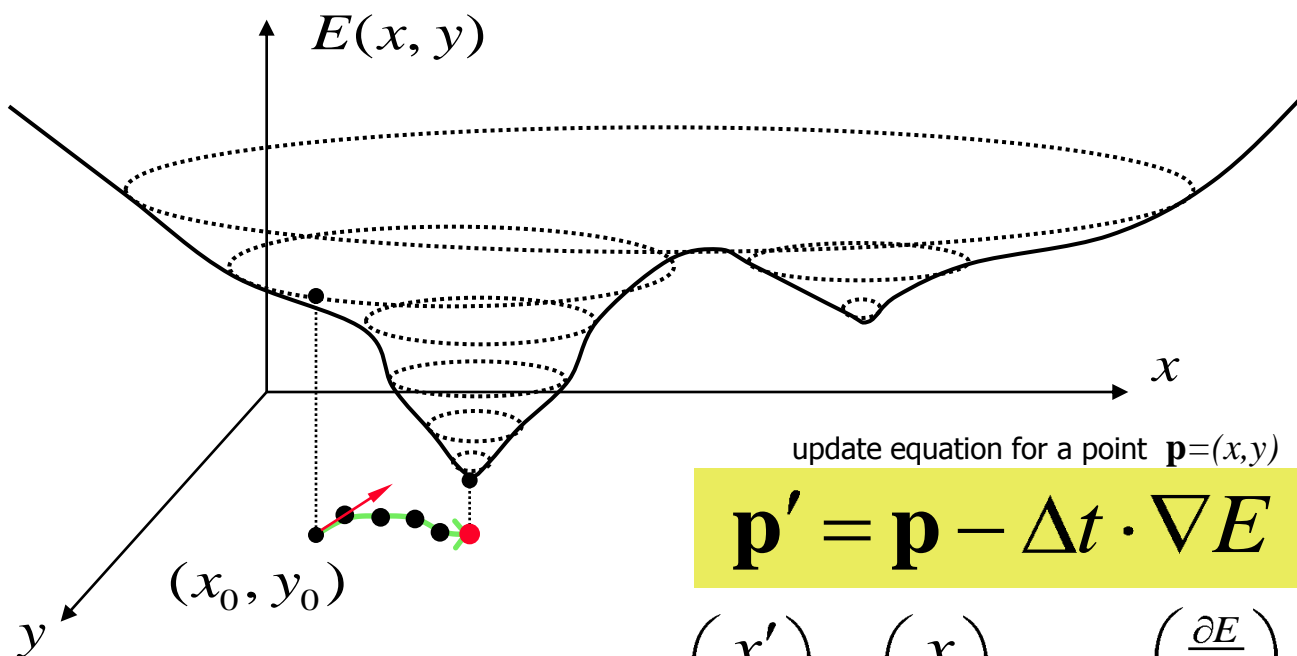
- Example: minimization of functions of 2 variables



**negative gradient** at point  $(x, y)$  gives direction of the steepest descent towards lower values of function  $E$

# Gradient Descent

- Example: minimization of functions of 2 variables



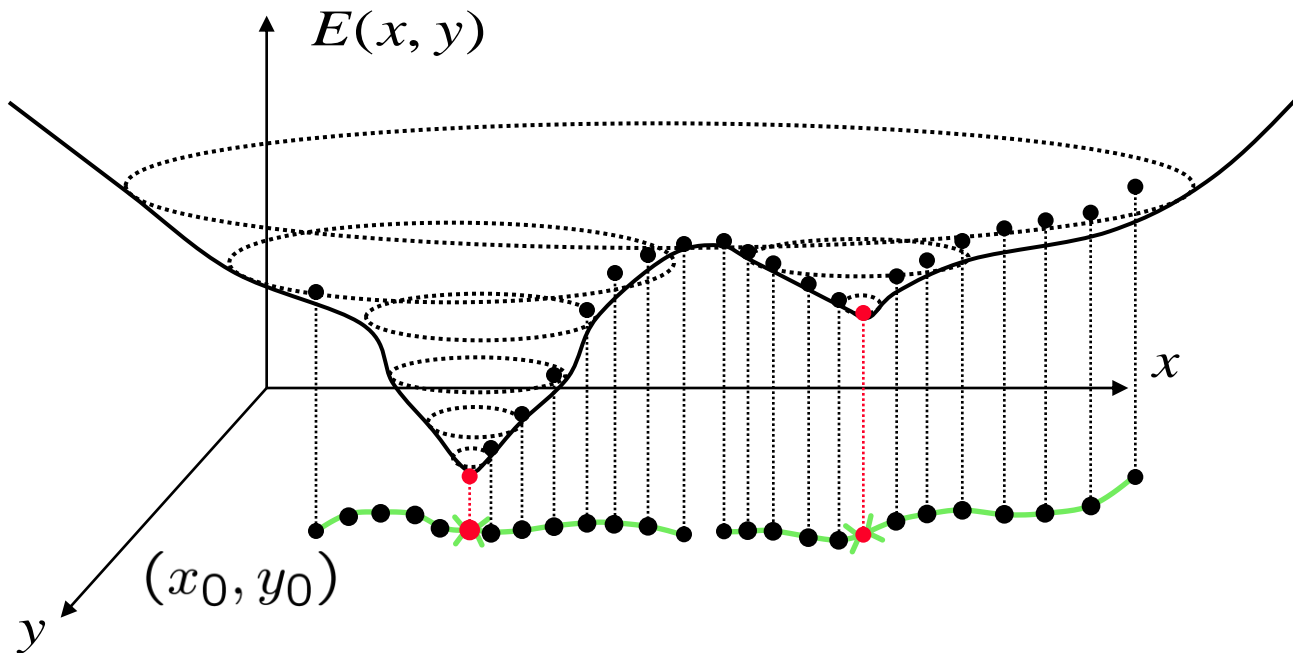
$$\mathbf{p}' = \mathbf{p} - \Delta t \cdot \nabla E$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} - \Delta t \cdot \begin{pmatrix} \frac{\partial E}{\partial x} \\ \frac{\partial E}{\partial y} \end{pmatrix}$$

Stop at a **local minima** where  $\nabla E = \vec{0}$

# Gradient Descent

- Example: minimization of functions of 2 variables



High sensitivity wrt. the initialisation !!

# Gradient Descent for Snakes

$$E(\overbrace{x_0, \dots, x_{n-1}, y_0, \dots, y_{n-1}}^{\mathbf{C}}) = - \sum_{i=0}^{n-1} |I_x(x_i, y_i)|^2 + |I_y(x_i, y_i)|^2$$

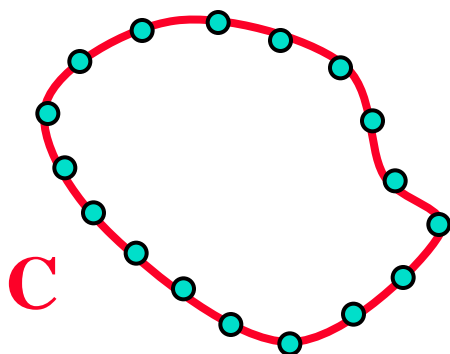
simple elastic snake energy

here, *energy* is a function of 2n variables

$$+ \alpha \cdot \sum_{i=0}^{n-1} (x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2$$

update equation for the whole snake

$$\mathbf{C}' = \mathbf{C} - \nabla E \cdot \Delta t$$



$$\begin{pmatrix} x'_0 \\ y'_0 \\ \dots \\ x'_{n-1} \\ y'_{n-1} \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \\ \dots \\ x_{n-1} \\ y_{n-1} \end{pmatrix} - \begin{pmatrix} \frac{\partial E}{\partial x_0} \\ \frac{\partial E}{\partial y_0} \\ \dots \\ \frac{\partial E}{\partial x_{n-1}} \\ \frac{\partial E}{\partial y_{n-1}} \end{pmatrix} \cdot \Delta t$$

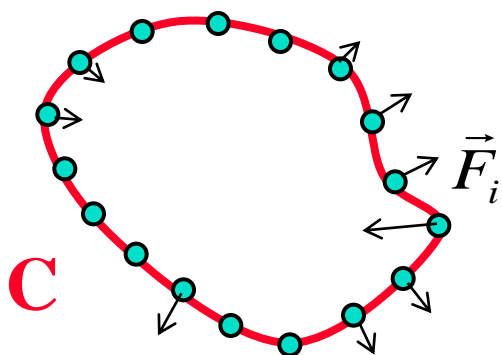
# Gradient Descent for Snakes

$$E(\overbrace{x_0, \dots, x_{n-1}, y_0, \dots, y_{n-1}}^{\mathbf{C}}) = - \sum_{i=0}^{n-1} |I_x(x_i, y_i)|^2 + |I_y(x_i, y_i)|^2$$

here, *energy* is a function of 2n variables

$$+ \alpha \cdot \sum_{i=0}^{n-1} (x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2$$

simple elastic snake energy



update equation for each node

$$\mathbf{v}'_i = \mathbf{v}_i + \vec{F}_i \cdot \Delta t$$

$$\vec{F}_i = - \begin{bmatrix} \frac{\partial E}{\partial x_i} \\ \frac{\partial E}{\partial y_i} \end{bmatrix}$$

$$\begin{pmatrix} x'_0 \\ y'_0 \\ \dots \\ x'_{n-1} \\ y'_{n-1} \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \\ \dots \\ x_{n-1} \\ y_{n-1} \end{pmatrix} - \begin{pmatrix} \frac{\partial E}{\partial x_0} \\ \frac{\partial E}{\partial y_0} \\ \dots \\ \frac{\partial E}{\partial x_{n-1}} \\ \frac{\partial E}{\partial y_{n-1}} \end{pmatrix} \cdot \Delta t$$

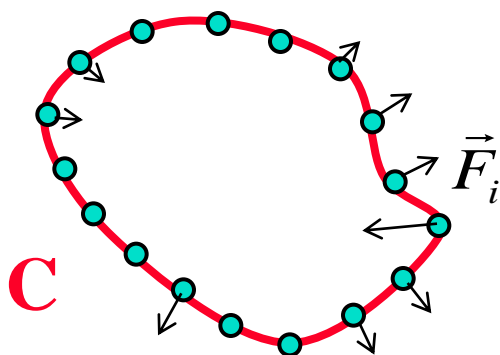
# Gradient Descent for Snakes

$$E(\overbrace{x_0, \dots, x_{n-1}, y_0, \dots, y_{n-1}}^{\mathbf{C}}) = - \sum_{i=0}^{n-1} |I_x(x_i, y_i)|^2 + |I_y(x_i, y_i)|^2$$

here, *energy* is a function of 2n variables

$$+ \alpha \cdot \sum_{i=0}^{n-1} (x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2$$

simple elastic snake energy



update equation for each node

$$\mathbf{v}'_i = \mathbf{v}_i + \vec{F}_i \cdot \Delta t$$

$$\vec{F}_i = - \begin{bmatrix} \frac{\partial E}{\partial x_i} \\ \frac{\partial E}{\partial y_i} \end{bmatrix} = ?$$

$$\frac{\partial E}{\partial x_i} = -2 \cdot I_x \cdot I_{xx} - 2 \cdot I_y \cdot I_{yx} - \alpha \cdot 2 \cdot (x_{i+1} - x_i) + \alpha \cdot 2 \cdot (x_i - x_{i-1})$$

$$\frac{\partial E}{\partial y_i} = -2 \cdot I_x \cdot I_{xy} - 2 \cdot I_y \cdot I_{yy} - \alpha \cdot 2 \cdot (y_{i+1} - y_i) + \alpha \cdot 2 \cdot (y_i - y_{i-1})$$

**Q: Do points move independently?**

**NO, motion of point  $i$  depends on positions of neighboring points** 5-40



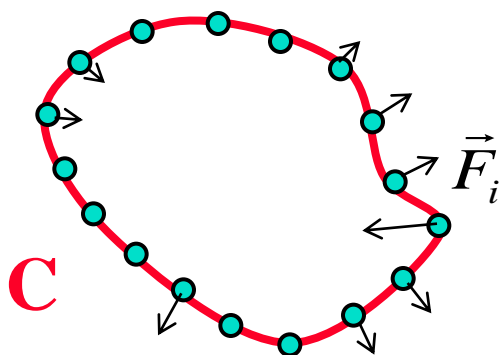
# Gradient Descent for Snakes

$$E(\overbrace{x_0, \dots, x_{n-1}, y_0, \dots, y_{n-1}}^{\mathbf{C}}) = - \sum_{i=0}^{n-1} |I_x(x_i, y_i)|^2 + |I_y(x_i, y_i)|^2$$

here, *energy* is a function of 2n variables

$$+ \alpha \cdot \sum_{i=0}^{n-1} (x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2$$

simple elastic snake energy



update equation for each node

$$\mathbf{v}'_i = \mathbf{v}_i + \vec{F}_i \cdot \Delta t$$

$$\vec{F}_i = - \begin{bmatrix} \frac{\partial E}{\partial x_i} \\ \frac{\partial E}{\partial y_i} \end{bmatrix} = ?$$

$$\frac{\partial E}{\partial x_i} = \underbrace{-2 \cdot I_x \cdot I_{xx} - 2 \cdot I_y \cdot I_{yx}}_{\text{from exterior (image) energy}} - \alpha \cdot 2 \cdot (x_{i+1} - x_i) + \alpha \cdot 2 \cdot (x_i - x_{i-1})$$

$$\frac{\partial E}{\partial y_i} = \underbrace{-2 \cdot I_x \cdot I_{xy} - 2 \cdot I_y \cdot I_{yy}}_{\text{from exterior (image) energy}} - \alpha \cdot 2 \cdot (y_{i+1} - y_i) + \alpha \cdot 2 \cdot (y_i - y_{i-1})$$

from exterior  
(image) energy

from interior  
(smoothness) energy

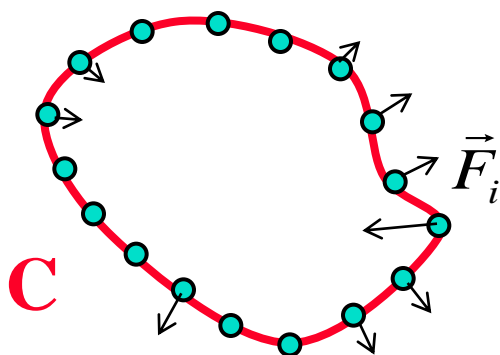
# Gradient Descent for Snakes

simple elastic snake energy

$$E(\overbrace{x_0, \dots, x_{n-1}, y_0, \dots, y_{n-1}}^{\mathbf{C}}) = - \sum_{i=0}^{n-1} |I_x(x_i, y_i)|^2 + |I_y(x_i, y_i)|^2$$

here, *energy* is a function of 2n variables

$$+ \alpha \cdot \sum_{i=0}^{n-1} (x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2$$



update equation for each node

$$\mathbf{v}'_i = \mathbf{v}_i + \vec{F}_i \cdot \Delta t$$

$$\vec{F}_i = - \begin{bmatrix} \frac{\partial E}{\partial x_i} \\ \frac{\partial E}{\partial y_i} \end{bmatrix} = ?$$

$$\vec{F}_i = \nabla (|\nabla I|^2) \Big|_{(x_i, y_i)}$$

motion of  $\mathbf{v}_i$  towards higher magnitude of image gradients

$$+ 2\alpha \cdot \frac{d^2 \mathbf{v}}{d^2 s}$$

This term for  $\mathbf{v}_i$  depends on neighbors  $\mathbf{v}_{i-1}$  and  $\mathbf{v}_{i+1}$

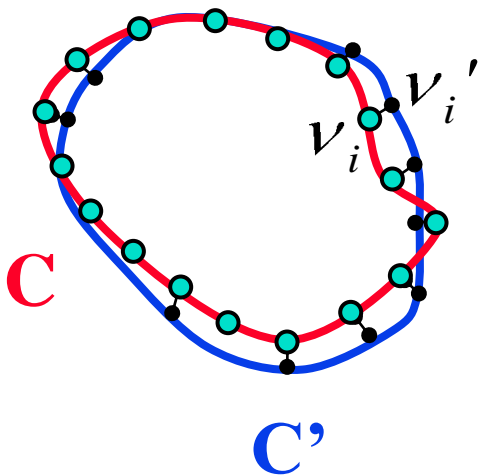
motion of  $\mathbf{v}_i$  reducing contour's bending

# Discrete Snakes:

## “*Gradient Flow*” evolution

$$dC = -dt \cdot \nabla E$$

Contour evolution via  
“Gradient flow”



update equation for each node

$$\mathbf{v}'_i = \mathbf{v}_i + \vec{F}_i \cdot \Delta t$$

$$i = 0, \dots, n-1$$

Stopping criteria:

$$\vec{F}_i = 0 \text{ for all } i \quad \Leftrightarrow \quad \nabla E = 0$$

local minima of energy  $E$

# Difficulties with Gradient Descent

---

- Very difficult to obtain accurate estimates of high-order derivatives on images (discretization errors)
  - E.g., estimating  $\nabla E_{ex}$  requires computation of second image derivatives  $I_{xx}$ ,  $I_{xy}$ ,  $I_{yy}$
- Gradient descent is not trivial even for one-dimensional functions. Robust numerical performance for 2n-dimensional function could be problematic.
  - Choice of parameter  $\Delta t$  is non-trivial
    - Small  $\Delta t$ , the algorithm may be too slow
    - Large  $\Delta t$ , the algorithm may never converge
  - Even if “converged” to a good local minima, the snake is likely to oscillate near it

# Alternative solution for 2D snakes: Dynamic Programming

---

- In many cases, snake energy can be written as a sum of *pair-wise interaction* potentials

$$E_{total}(v_0, \dots, v_{n-1}) = \sum_{i=0}^{n-1} E_i(v_i, v_{i+1})$$

- More generally, it can be written as a sum of *higher-order interaction* potentials (e.g. triple interactions).

$$E_{total}(v_0, \dots, v_{n-1}) = \sum_{i=0}^{n-1} E_i(v_{i-1}, v_i, v_{i+1})$$

# Snake energy: pair-wise interactions

Example: simple elastic snake energy

$$E_{total}(x_0, \dots, x_{n-1}, y_0, \dots, y_{n-1}) = - \sum_{i=0}^{n-1} |I_x(x_i, y_i)|^2 + |I_y(x_i, y_i)|^2 + \alpha \cdot \sum_{i=0}^{n-1} (x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2$$

$$E_{total}(v_0, \dots, v_{n-1}) = - \sum_{i=0}^{n-1} \|\nabla I(v_i)\|^2 + \alpha \cdot \sum_{i=0}^{n-1} \|v_{i+1} - v_i\|^2$$

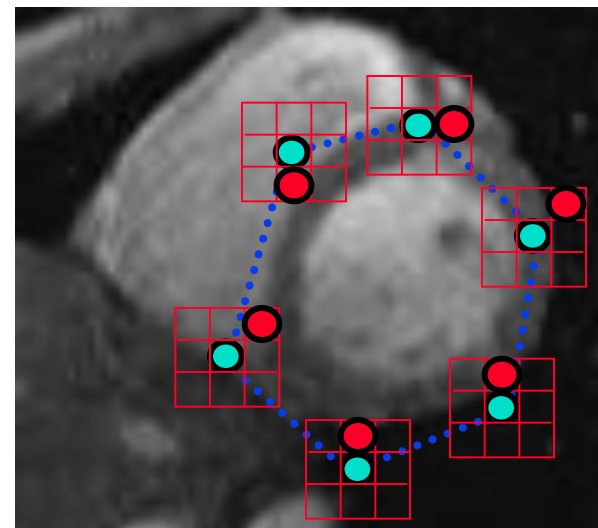
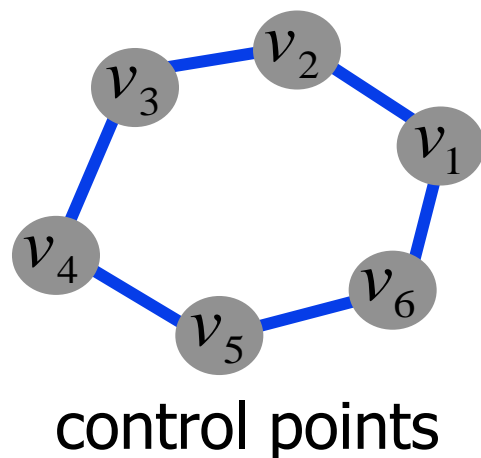
$$E_{total}(v_0, \dots, v_{n-1}) = \sum_{i=0}^{n-1} E_i(v_i, v_{i+1})$$

where  $E_i(v_i, v_{i+1}) = -\|\nabla I(v_i)\|^2 + \alpha \|v_i - v_{i+1}\|^2$

**Q: give an example of snake with triple-interaction potentials?**

# DP Snakes

[Amini, Weymouth, Jain, 1990]

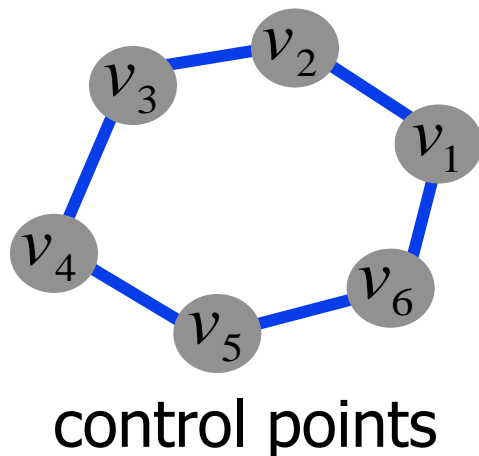


$$E(v_1, v_2, \dots, v_n) = \overset{\text{First-order interactions}}{E_1(v_1, v_2) + E_2(v_2, v_3) + \dots + E_{n-1}(v_{n-1}, v_n)}$$

Energy  $E$  is minimized via Dynamic Programming

# DP Snakes

[Amini, Weymouth, Jain, 1990]



$$E(v_1, v_2, \dots, v_n) = \overset{\text{First-order interactions}}{E_1(v_1, v_2) + E_2(v_2, v_3) + \dots + E_{n-1}(v_{n-1}, v_n)}$$

**Energy  $E$  is minimized via Dynamic Programming**

Iterate until optimal position for each point is the center of the box,  
i.e. the **snake is optimal in the local search space** constrained by boxes

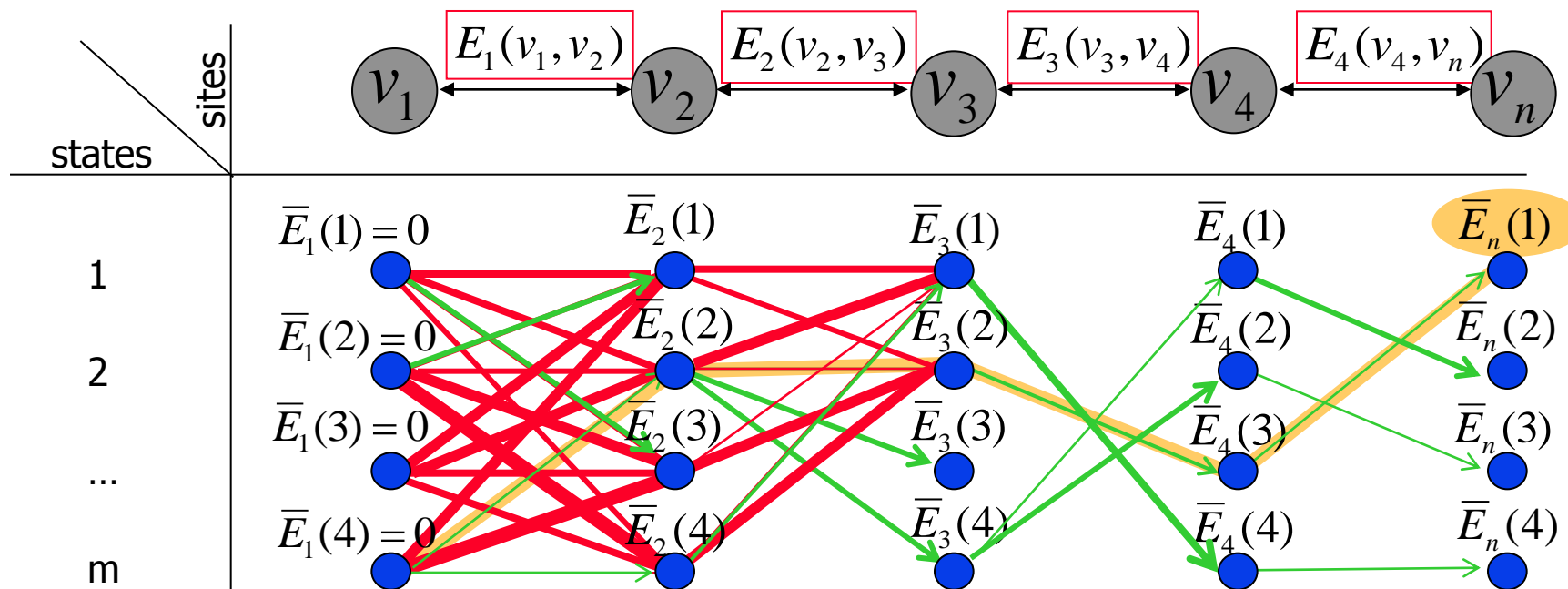


# Dynamic Programming (DP)

## Viterbi Algorithm

Here we will concentrate on **first-order interactions**

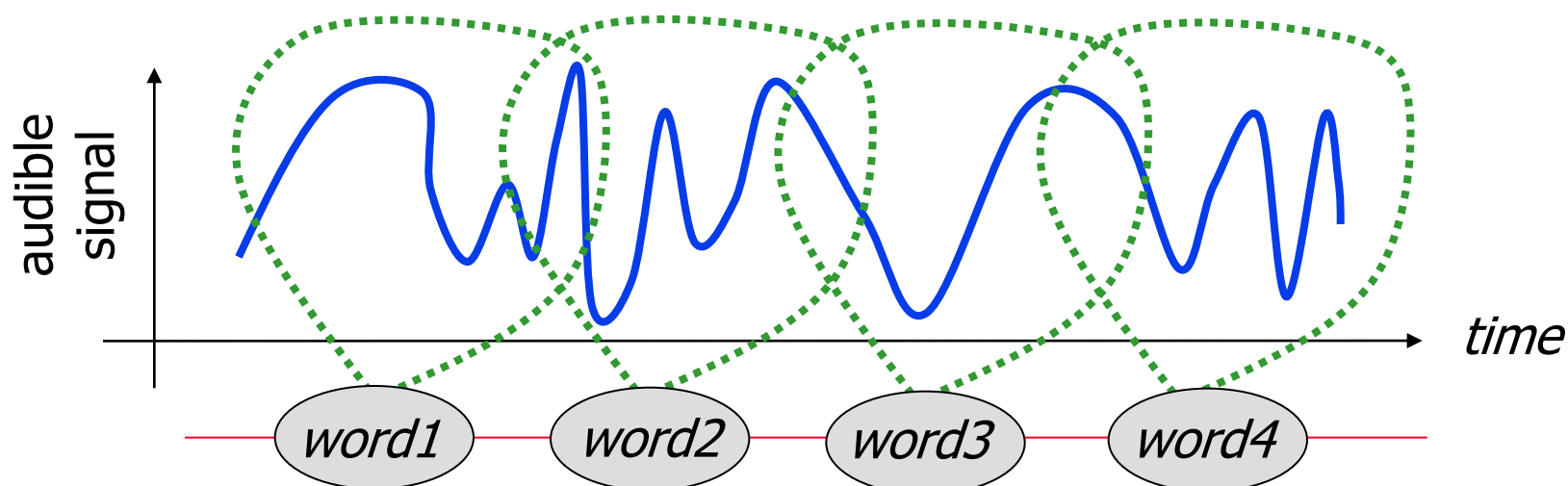
$$E_1(v_1, v_2) + E_2(v_2, v_3) + \dots + E_{n-1}(v_{n-1}, v_n)$$



Complexity:  $O(nm^2)$ , Worst case = Best Case

# Dynamic Programming and Hidden Markov Models (HMM)

- DP is widely used in speech recognition



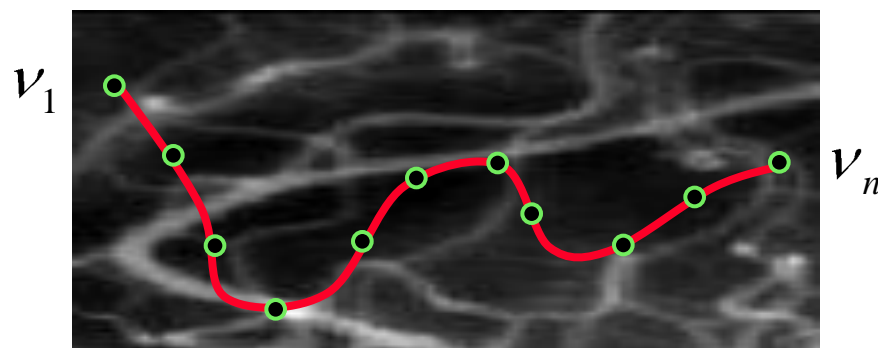
ordered (in time) hidden variables (words) to be estimated from **observed signal**

$$E_1(v_0, v_1) + \dots + E_i(v_{i-1}, v_i) + \dots + E_n(v_{n-1}, v_n)$$

$$-\ln\{\Pr(\text{signal}(t_i) | \text{word}_i)\} - \ln\{\Pr(\text{word}_i | \text{word}_{i-1})\}$$

# Snakes can also be seen as Hidden Markov Models (HMM)

- Positions of snake nodes are hidden variables
- Timely order is replaced with spatial order
- Observed audible signal is replaced with image



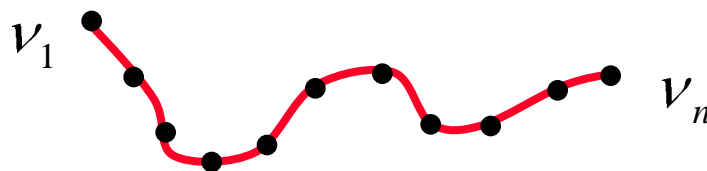
$$E_1(v_1, v_2) + \dots + E_i(v_i, v_{i+1}) + \dots + E_{n-1}(v_{n-1}, v_n)$$

$$-\|\nabla I(v_i)\| + E_{elastic}(v_i, v_{i+1})$$

# Dynamic Programming for a closed snake?

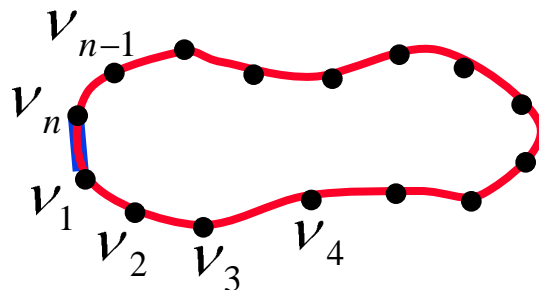
Clearly, DP can be applied to optimize an open ended snake

$$E_1(v_1, v_2) + E_2(v_2, v_3) + \dots + E_{n-1}(v_{n-1}, v_n)$$



Can we use DP for a “looped” energy in case of a closed snake?

$$E_1(v_1, v_2) + E_2(v_2, v_3) + \dots + E_{n-1}(v_{n-1}, v_n) + E_n(v_n, v_1)$$



# Dynamic Programming for a closed snake

---

$$E_1(v_1, v_2) + E_2(v_2, v_3) + \dots + E_{n-1}(v_{n-1}, v_n) + E_n(v_n, v_1)$$

1. Can use Viterbi to optimize snake energy in case  $v_1 = \mathbf{c}$  is fixed. (in this case the energy above effectively has no loop)
2. Use Viterbi to optimize snake for all possible values of  $\mathbf{c}$  and choose the best of the obtained  $m$  solutions.

**for exact solution  
complexity  
increases to  
 $O(nm^3)$**

# Dynamic Programming for a closed snake

---

$$E_1(v_1, v_2) + E_2(v_2, v_3) + \dots + E_{n-1}(v_{n-1}, v_n) + E_n(v_n, v_1)$$

DP has problems with “**loops**” (even one loop increases complexity).  
However, some approximation tricks can be used in practice...

1. Use DP to optimize snake energy with fixed  $v_1$  (according to a given initial snake position).
2. Use DP to optimize snake energy again. This time fix position of an intermediate node  $v_{n/2} = \hat{v}_{n/2}$  where  $\hat{v}$  is an optimal position obtained in step 1.

**This is only an approximation, but complexity is good:  $O(nm^2)$**

# Dynamic Programming

## for snakes with higher order interactions

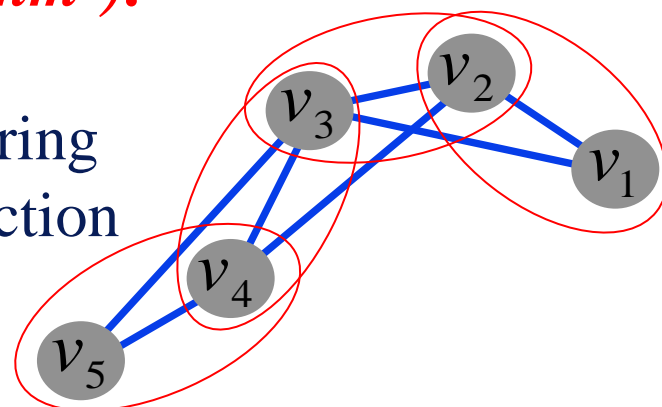
---

$$E_1(v_1, v_2, v_3) + E_2(v_2, v_3, v_4) + \dots + E_{n-2}(v_{n-2}, v_{n-1}, v_n)$$

(e.g. if bending energy is added into the “model” of the snake)

**Viterbi algorithm can be generalized to 3-clique case but its complexity increases to  $O(nm^3)$ .**

*one approach:* combine each pair of neighboring nodes into one super node. Each triple interaction can be represented as a pair-wise interaction between 2 super-nodes. Viterbi algorithm will need  $m^3$  operations for each super node (why?)



# DP snakes (open case)

## Summary of Complexity

<i>energy</i>	<i>type</i> (order of interactions)	<i>complexity</i>
$\sum_{i=1}^n E_i(v_i)$	unary potentials ( $d=1$ )	$O(nm)$
$\sum_{i=1}^{n-1} E_i(v_i, v_{i+1})$	pair-wise potentials ( $d=2$ )	$O((n-1)m^2)^*$
$\sum_{i=1}^{n-2} E_i(v_i, v_{i+1}, v_{i+2})$	triple potentials ( $d=3$ )	$O((n-2)m^3)^*$
$E(v_1, v_2, \dots, v_n)$	complete connectivity ( $d=n$ )	$O(m^n)$ – exhaustive search

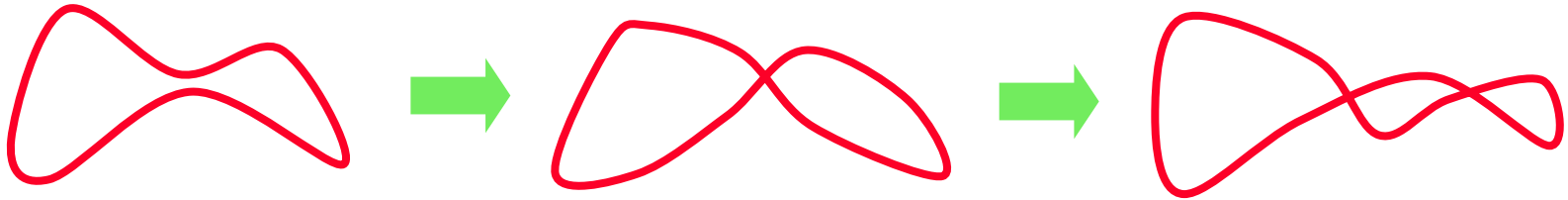
\* - adding a single loop increases complexity by factor  $m^{d-1}$



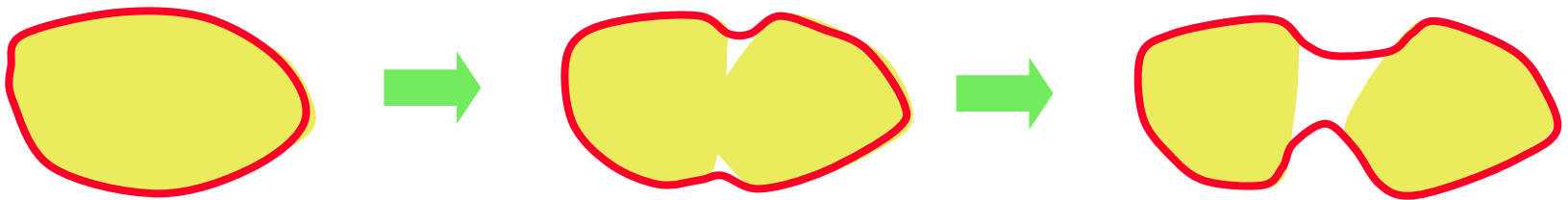
# Problems with snakes

---

- Depends on number and spacing of control points
- Snake may oversmooth the boundary
- Not trivial to prevent curve self intersecting



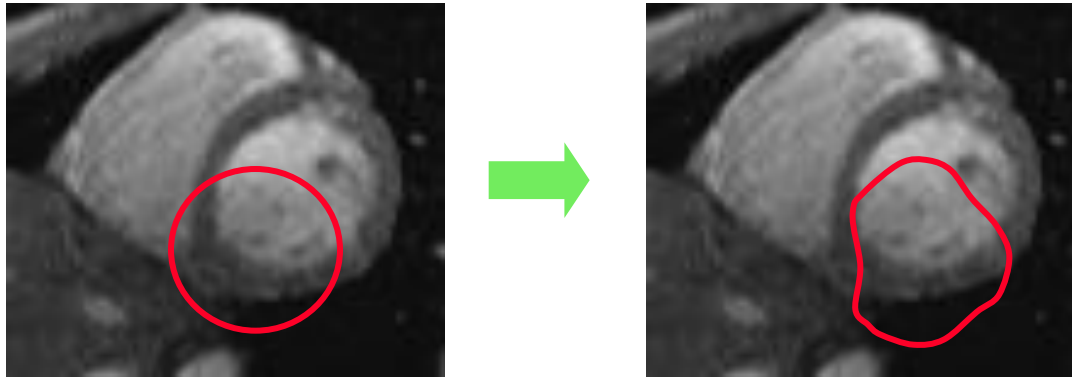
- Can not follow topological changes of objects



# Problems with snakes

---

- May be sensitive to initialization
  - may get stuck in a local energy minimum near initial contour



- Numerical stability can be an issue for **gradient descent** and **variational methods** (continuous formulation)
  - E.g. requires computing second order derivatives
- The general concept of snakes (deformable models) does generalize to 3D (deformable mesh), but many robust optimization methods suitable for 2D snakes do not apply in 3D
  - E.g.: **dynamic programming** only works for 2D snakes

# Problems with snakes

- External energy: may need to diffuse image gradients, otherwise the snake does not really “see” object boundaries in the image unless it gets very close to it.

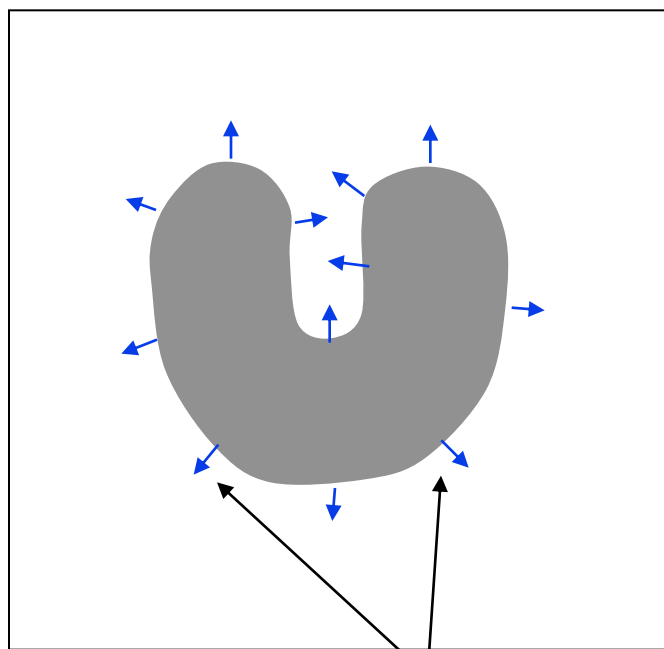
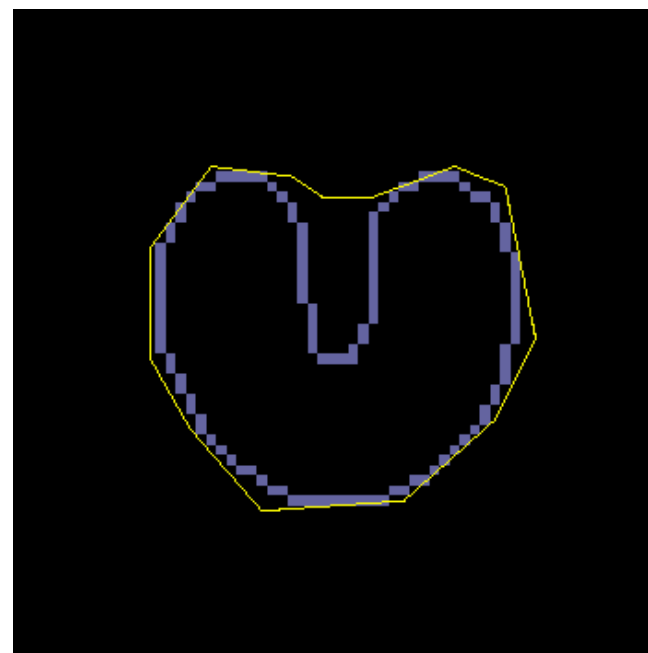
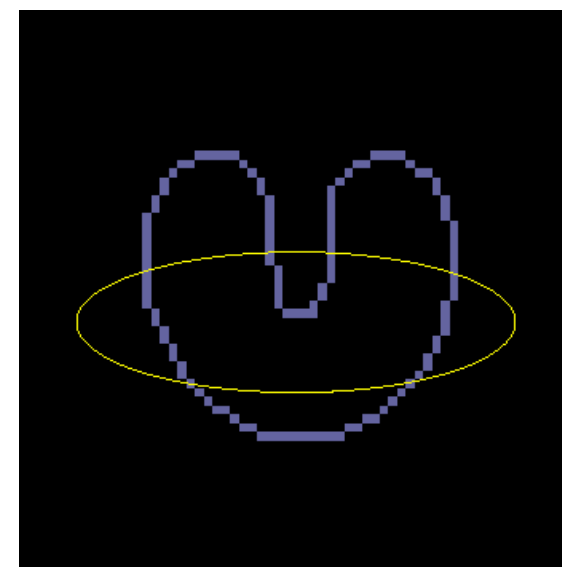
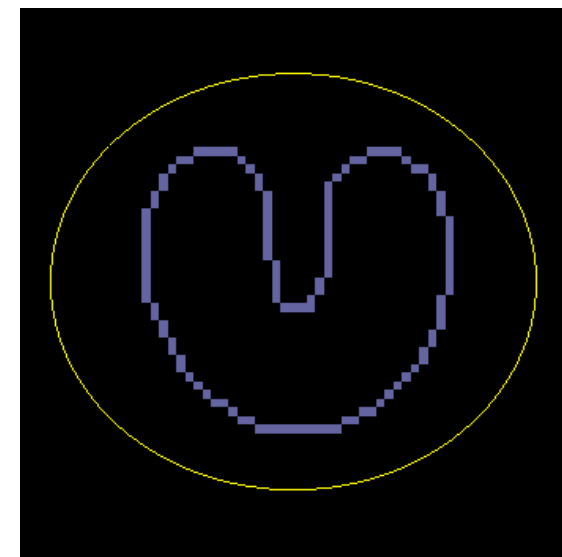


image gradients  $\nabla I$   
are large only directly on the boundary



# Diffusing Image Gradients $\nabla I$

image gradients diffused via  
*Gradient Vector Flow (GVF)*



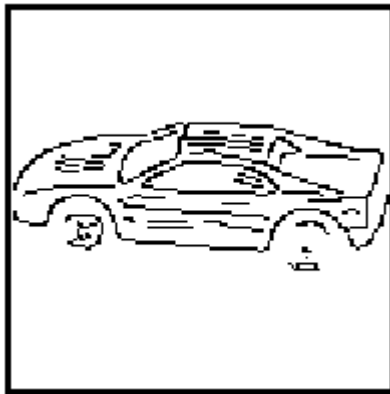
Chenyang Xu and Jerry Prince, 98

<http://iacl.ece.jhu.edu/projects/gvf/>

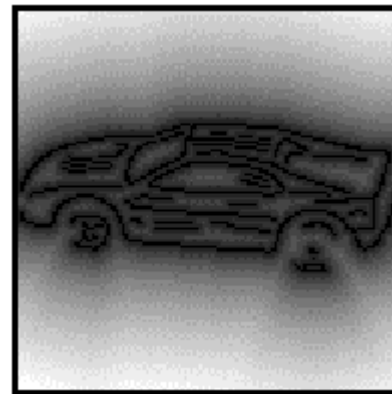
# Alternative Way to Improve External Energy

- Use  $E_{ex} = \sum_{i=0}^{n-1} D(\mathbf{v}_i)$  instead of  $E_{ex} = -\sum_{i=0}^{n-1} |\nabla I(\mathbf{v}_i)|$  where  $D()$  is
  - *Distance Transform* (for detected binary image features, e.g. edges)

Distance Transform can be visualized as a gray-scale image



binary image features  
(edges)



Distance Transform  $D(x, y)$

- *Generalized Distance Transform* (directly for image gradients)

# Distance Transform

(see p.20-21 of the text book)

*Image features (2D)*

	■						
	■						
	■						■
	■	■				■	
					■		
					■		
					■		
						■	

*Distance Transform*

1	0	1	2	3	4	3	2
1	0	1	2	3	3	2	1
1	0	1	2	3	2	1	0
1	0	0	1	2	1	0	1
2	1	1	2	1	0	1	2
3	2	2	2	1	0	1	2
4	3	3	2	1	0	1	2
5	4	4	3	2	1	0	1

**Distance Transform** is a function  $D(\cdot)$  that for each image pixel  $p$  assigns a non-negative number  $D(p)$  corresponding to distance from  $p$  to the nearest feature in the image  $I$

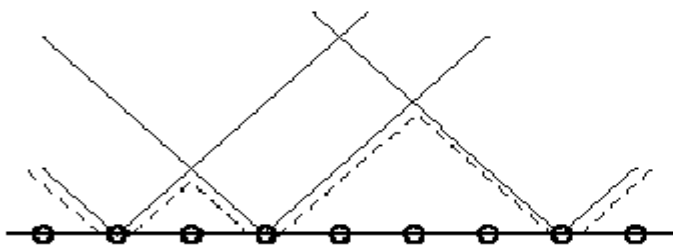
# Distance Transform

## can be very efficiently computed

- Two pass  $O(n)$  algorithm for 1D  $L_1$  norm (for simplicity just distance)
  - Initialize: For all  $j$   
 $D[j] \leftarrow 1_p[j]$
  - Forward: For  $j$  from 1 up to  $n-1$   
 $D[j] \leftarrow \min(D[j], D[j-1]+1)$
  - Backward: For  $j$  from  $n-2$  down to 0  
 $D[j] \leftarrow \min(D[j], D[j+1]+1)$

1	0
---	---

0	1
---	---



$\infty$	0	$\infty$	0	$\infty$	$\infty$	$\infty$	0	$\infty$
----------	---	----------	---	----------	----------	----------	---	----------

$\infty$	0	1	0	1	2	3	0	1
----------	---	---	---	---	---	---	---	---

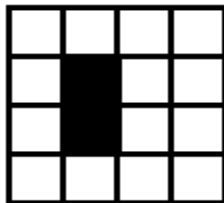
1	0	1	0	1	2	1	0	1
---	---	---	---	---	---	---	---	---

# Distance Transform

## can be very efficiently computed

- 2D case analogous to 1D
  - Initialization
  - Forward and backward pass
    - Fwd pass finds closest above and to left
    - Bwd pass finds closest below and to right
- Note nothing depends on  $0, \infty$  form of initialization
  - Can “distance transform” arbitrary array

-	1
1	0
0	1
1	-



$\infty$	$\infty$	$\infty$	$\infty$
$\infty$	0	$\infty$	$\infty$
$\infty$	0	$\infty$	$\infty$
$\infty$	$\infty$	$\infty$	$\infty$

$\infty$	$\infty$	$\infty$	$\infty$
$\infty$	0	1	$\infty$
$\infty$	0	$\infty$	$\infty$
$\infty$	$\infty$	$\infty$	$\infty$

$\infty$	$\infty$	$\infty$	$\infty$
$\infty$	0	1	2
$\infty$	0	1	2
$\infty$	1	2	3

2	1	2	3
1	0	1	2
1	0	1	2
2	1	2	3



# Distance Transform can be very efficiently computed

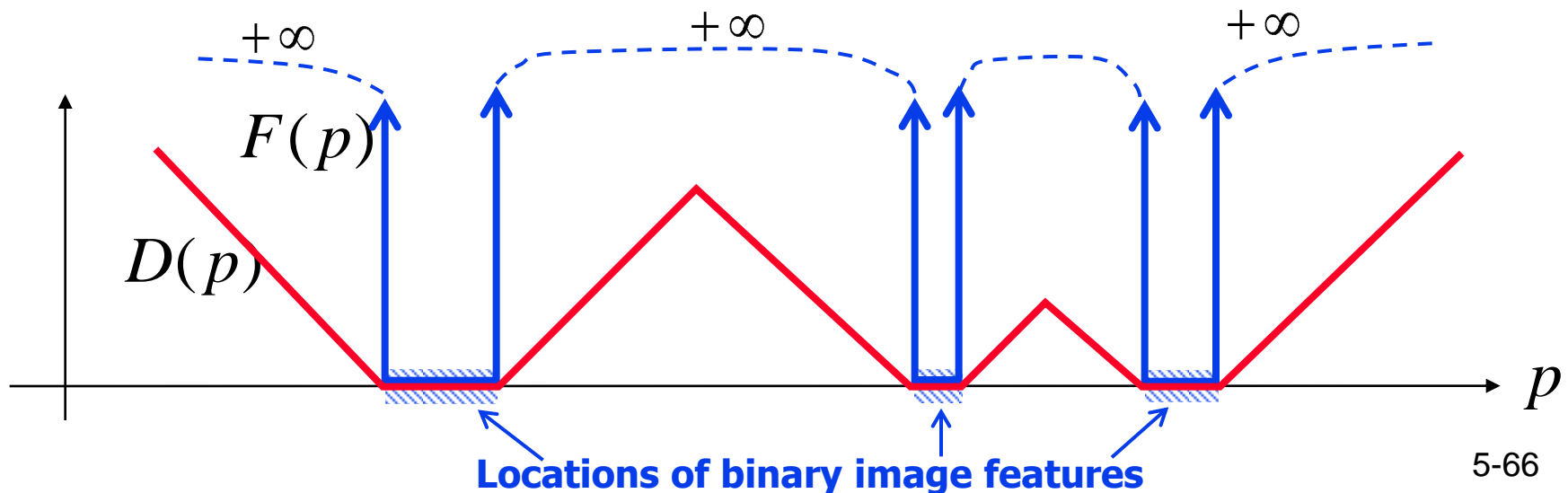
---

- Forward-Backward pass algorithm computes shortest paths in  $O(n)$  on a grid graph with regular 4-N connectivity and homogeneous edge weights 1
- Alternatively, Dijkstra's algorithm can also compute a distance map (trivial generalization for multiple sources), but it would take  $O(n \cdot \log(n))$ .
  - *Dijkstra is slower but it is a more general method applicable to arbitrary weighted graphs*

# Distance Transform: an alternative way to think about

- Assuming  $F(p) = \begin{cases} 0 & \text{if pixel } p \text{ is image feature} \\ \infty & \text{O.W.} \end{cases}$

then  $D(p) = \min_q \{ \| p - q \| + F(q) \} = \min_{q:F(q)=0} \| p - q \|$   
 is standard *Distance Transform* (of image features)



# Distance Transform vs.

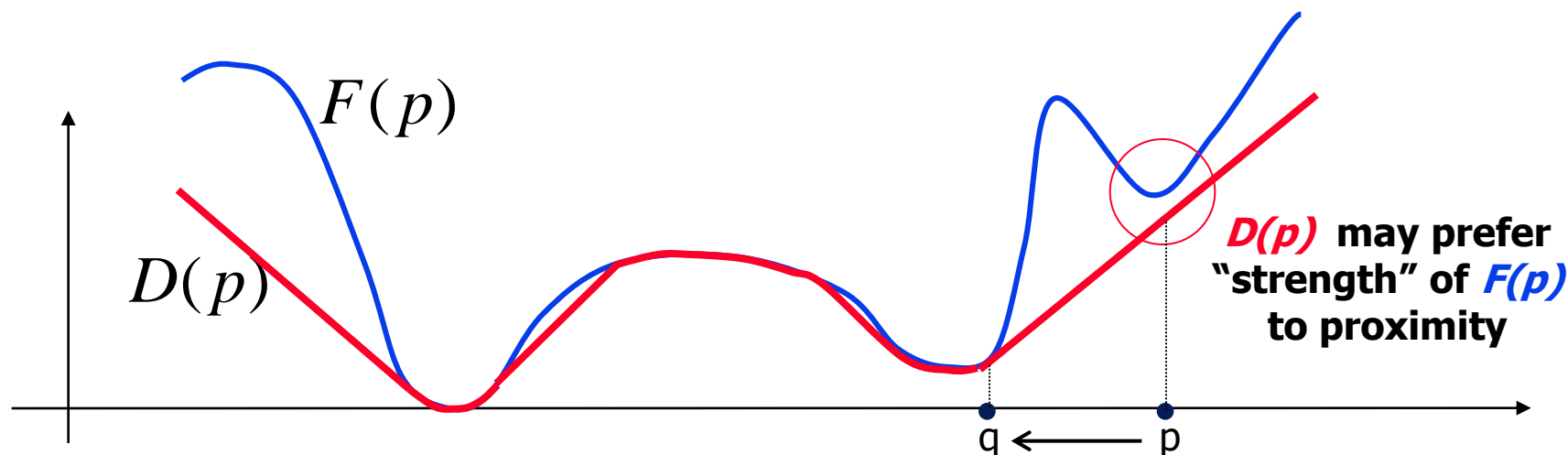
## Generalized Distance Transform

---

- For general  $F(p)$

$$D(p) = \min_q \{ \alpha \cdot \| p - q \| + F(q) \}$$

is called *Generalized Distance Transform* of  $F(p)$



$F(p)$  may represent non-binary image features (e.g. image intensity gradients)

# Generalized Distance Transforms

(see Felzenszwalb and Huttenlocher, IJCV 2005)

- The same “*Forward-Backward*” algorithm can be applied to any initial array
  - Binary (0 /  $\infty$ ) initial values are non-essential.
- If the initial array contains values of function  $F(x,y)$  then the output of the “*Forward-Backward*” algorithm is a

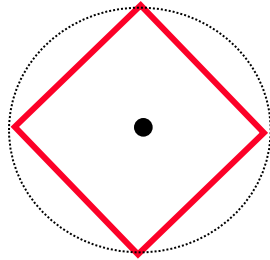
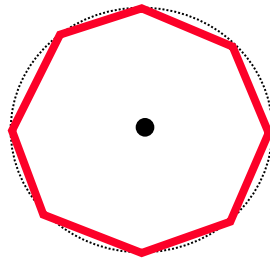
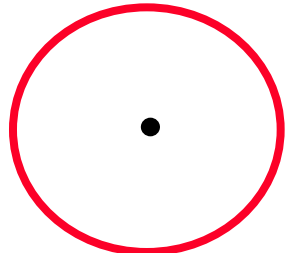
*Generalized Distance Transform*

$$D(p) = \min_{q \in I} (\alpha \cdot \|p - q\| + F(q))$$

- “Scope of attraction” of image gradients can be extended via external energy  $E_{ex} = \sum_{i=0}^{n-1} D(v_i)$  based on a generalized distance transform of

$$F(x, y) = g(|\nabla I(x, y)|)$$

# Metric properties of discrete Distance Transforms

Forward mask	Backward mask	Metric	Set of equidistant points												
<table border="1"> <tr><td>-</td><td>1</td></tr> <tr><td>1</td><td>0</td></tr> </table>	-	1	1	0	<table border="1"> <tr><td>0</td><td>1</td></tr> <tr><td>1</td><td>-</td></tr> </table>	0	1	1	-	Manhattan (L1) metric					
-	1														
1	0														
0	1														
1	-														
<table border="1"> <tr><td>1.4</td><td>1</td><td>1.4</td></tr> <tr><td>1</td><td>0</td><td></td></tr> </table>	1.4	1	1.4	1	0		<table border="1"> <tr><td></td><td>0</td><td>1</td></tr> <tr><td>1.4</td><td>1</td><td>1.4</td></tr> </table>		0	1	1.4	1	1.4	Better approximation of Euclidean metric	
1.4	1	1.4													
1	0														
	0	1													
1.4	1	1.4													
<p>In fact, "exact" Euclidean Distance transform can be computed fairly efficiently (in linear or near-linear time) without bigger masks</p> <p>1) <a href="http://www.cs.cornell.edu/~dph/matchalgs/">www.cs.cornell.edu/~dph/matchalgs/</a>            2) <i>Fast Marching Method</i> –Tsitsiklis, Sethian</p>		Euclidean (L2) metric													

# HW assignment 2

---

## ■ DP Snakes

- Use elastic snake model  $E = E_{\text{int}} + \lambda \cdot E_{\text{ext}}$  (value of  $\lambda$  is important)

- Compare  $E_{\text{int}} = \sum_{i=0}^{n-1} L_i^2$  vs.  $E_{\text{int}} = \sum_{i=0}^{n-1} |L_i - \hat{L}|^2$

- Compare  $E_{\text{int}} = \sum_{i=0}^{n-1} L_i^2$  vs.  $E_{\text{int}} = \sum_{i=0}^{n-1} |L_i|$

- Compare  $E_{\text{ext}} = \sum_{i=0}^{n-1} F(v_i)$  vs.  $E_{\text{ext}} = \sum_{i=0}^{n-1} D(v_i)$  where  $D$  is a generalized distance transform of  $F = g(|\nabla I|)$

such that  $D(p) = \min_q \{\alpha \cdot \|p - q\| + F(q)\}$  (value of  $\alpha$  is important)

- Use Viterbi algorithm for optimization
- Incorporate *edge alignment*
- Use 3x3 search box for each control point