Computer Vision

Lecture 7: Active contours, or snakes

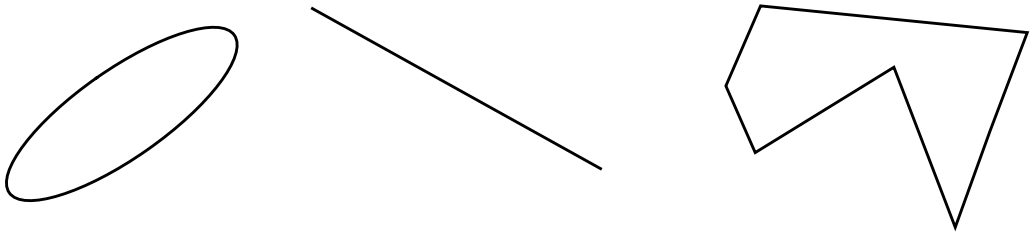
Last lecture

- The Hough transform

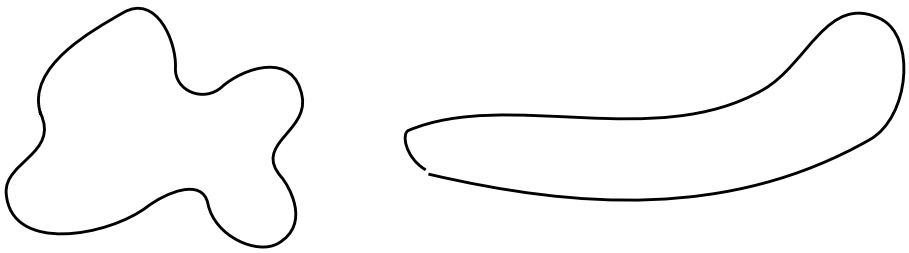
This lecture

- Active contours, or deformable contours, or *snakes.*

Hough transforms find evidence for well-defined parametrised geometrical shapes.



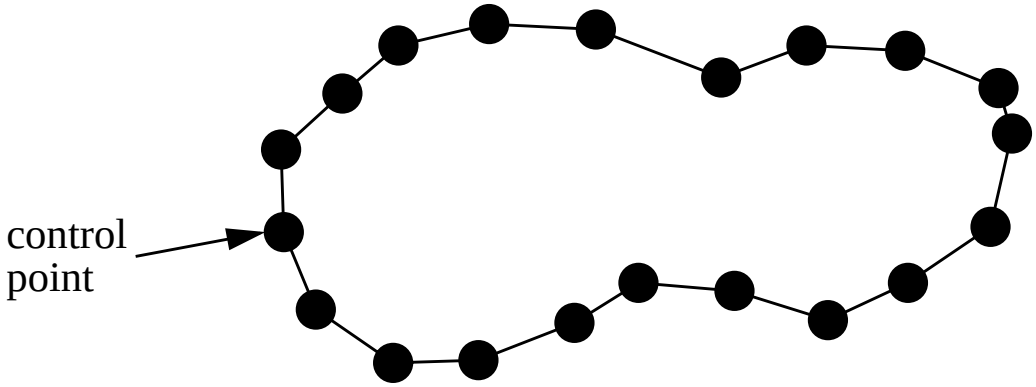
But often what is needed is to find extended shapes that do not have a concise geometrical description, but which have certain *properties* that we can specify. **Snakes** are a way of doing this.



Basic snake properties

A snake is a *contour* in the image plane, defined by a set of *control points*.

We will use closed contours, and treat the snake as being a curve that passes through the control points. (This need not always be the case.)



The snake's position and shape is made to *evolve* to satisfy

- the *intrinsic* properties that we want it to have,
- the *extrinsic* or image-related properties that we want it to have,
- and any *constraints* that we want to impose.

The algorithm is iterative: snakes are initialised (somehow!) and then some quantity is optimised in steps. Nonetheless, they are computationally very cheap.

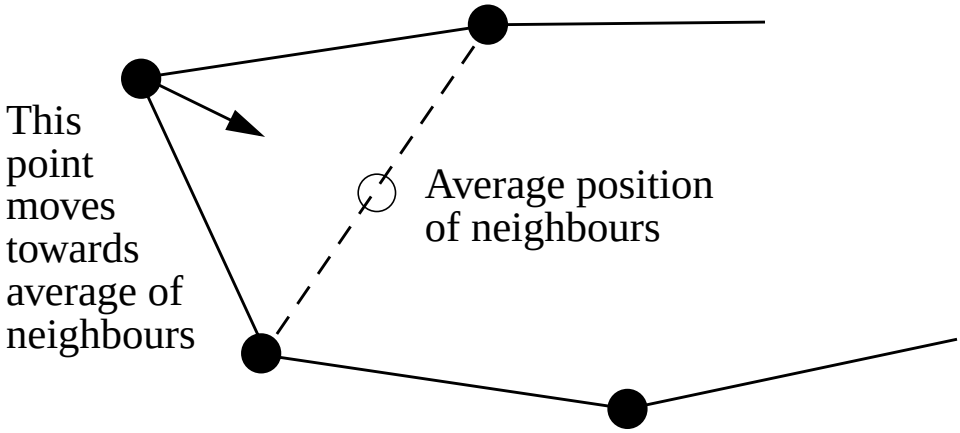
The properties we want to specify can be thought of as if they were physical properties of a physical shape. E.g. if the snake is to shrink, think of an elastic band; if to be smooth, a strip of springy metal.

We implement snakes computationally by simulating the physical model of the desired properties.

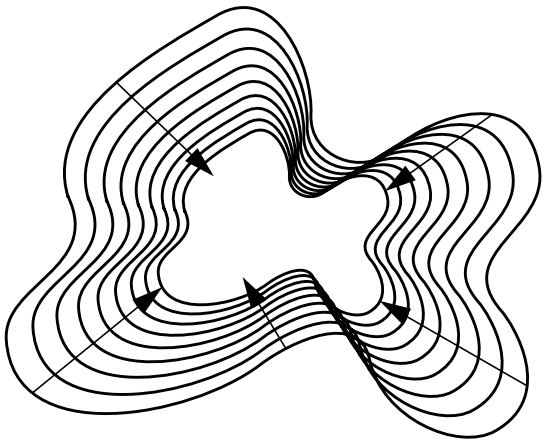
Shrinking and sticking

We can use a snake to “shrink-wrap” image structures.

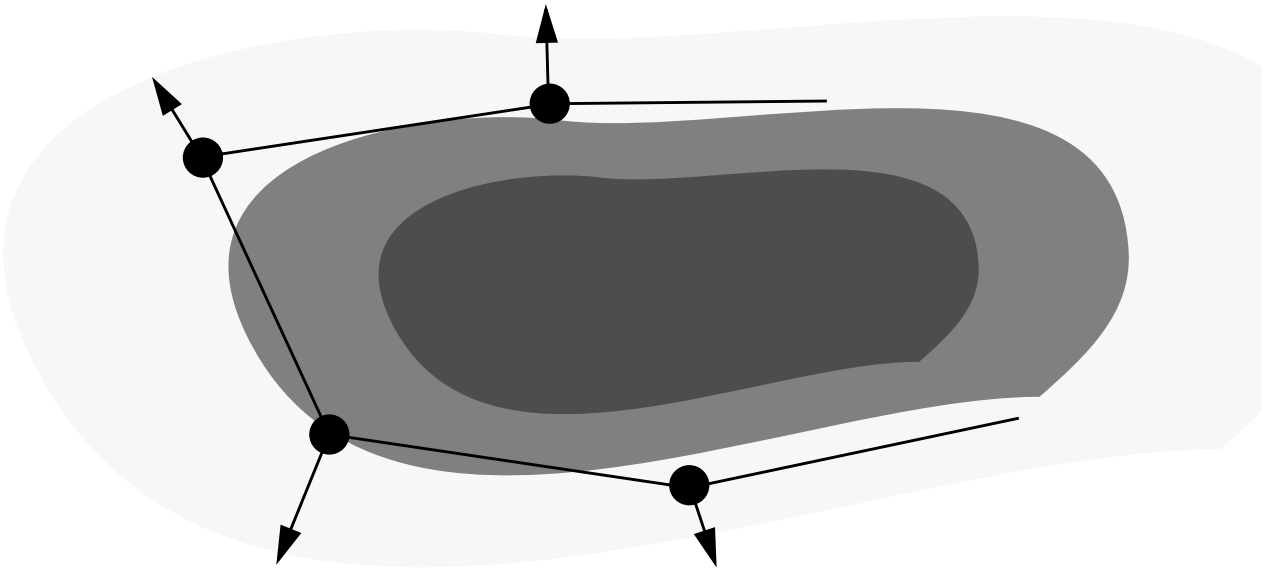
Internal property: the snake should be like a rubber band, which pulls itself together. At each time step, each control point moves closer to its neighbours:



Easy to implement: to find the coords of the average of the neighbours, just average their *x* and *y* coords separately. Move the point by some fraction of the vector from its current position to the average position. (Usually calculate all the moves before actually doing any of them.)



External property: the snake should be repelled by dark parts of the image, if we want it to shrink-wrap a dark structure.



Each control point is pushed towards brighter pixels in its neighbourhood.

We can do this by calculating the local *gradient* of image intensity, using x and y differences. Each point is moved in x by some constant times the local x gradient, and likewise for y.

Some local smoothing might be used — but it is not necessary to preprocess the whole image.

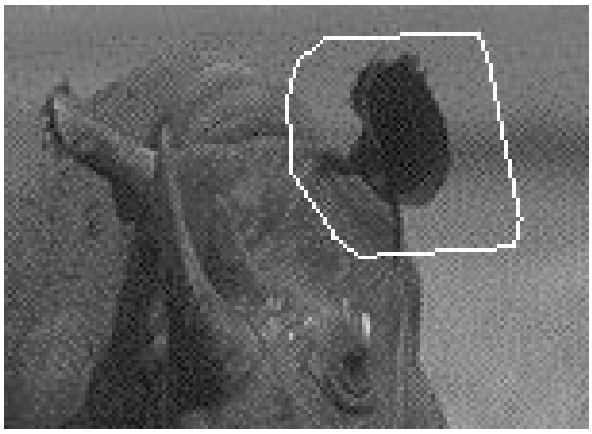
For the x coordinate of a control point, a simple formula (no smoothing) is

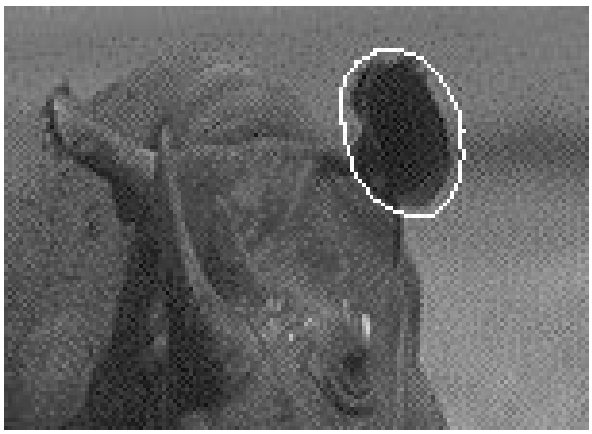
$$x_{\text{new}} = x + \alpha\left(\frac{1}{2}(x_{\text{left}} + x_{\text{right}}) - x\right) + \beta[I(x+1, y) - I(x-1, y)]$$

where x_{left} refers to the control point to the left along the contour, $I(x\text{-}1, y)$ means the grey-level one pixel to the left along the row, and α and β are constants chosen to give the right balance and amount of movement. The formula for updating the y coordinate is similar.

A snake governed by these equations is applied to the smoothed image at the top right. It shrinks until it sticks near the edge of the dark region.

The images below show the starting position of the snake, and the position after some iterations, when the elastic inward force is balanced by the outward force from the grey-level gradient.





Snake energy

We need a general way of specifying the properties we want the snake to have.

This is done using a quantity called ***energy***, by analogy with physical systems.

We *define* the energy for a snake so that states we want it to be in have low energy.

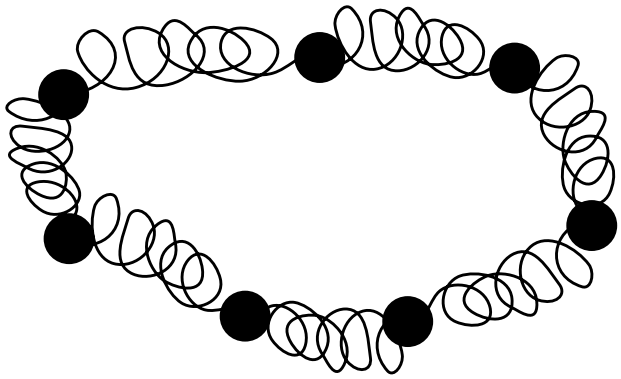
The energy is made up of internal and external parts added together:

$$E = E_{\text{internal}} + E_{\text{external}}$$

To make the snake shrink, the internal energy increases with the length of the snake. Rather than make the energy proportional to the length, we use the sum of the square of the distances between the control points:

$$E_{\text{internal}} = A \sum_{\text{control points}} (\text{distance between control point and neighbour to left})^2$$

This makes the control points tend to spread out evenly along the snake — big gaps contribute a disproportionately high energy. It also models (roughly) a physical system in which the control points are joined by springs.



To make the snake avoid dark parts of the image, we make the external energy proportional to minus the sum of the grey-levels lying under the snake. For simplicity, we might just use the grey-levels under the control points.

$$E_{\text{external}} = -B \sum_{\text{control points}} I(\text{location of control point})$$

where I is the grey-level value in the image. A and B are constants, related to α and β.

Converting energy to force

The snake must evolve so that at each step its energy decreases.

Mathematically we differentiate the energy with respect to control point position. This gives a formula showing how to move the control points to reduce the energy. This process is called *gradient descent* on the energy surface.

In the physical analogy, this amounts to calculating the *forces* acting on the control points. At each time step, each point is moved a distance proportional to the force on it. This is as if the snake was moving in a viscous fluid.

Differentiating the energy formulae above gives rise to the update rules used for the example:

- to reduce E_{internal} take a small step towards a point half-way between your neighbours;
- to reduce E_{external} take a small step up the grey-level gradient.

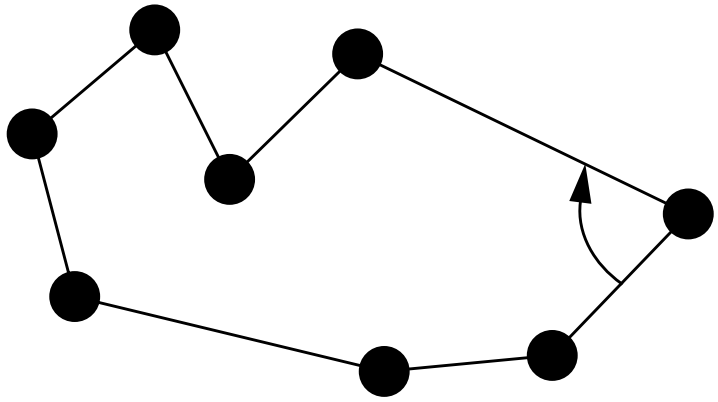
Why use an energy formulation?

Because we can state the requirements for a “good” snake directly, then generate an algorithm that implements it.

Smoothing

Suppose we want the final snake to be smooth, but not necessarily to shrink.

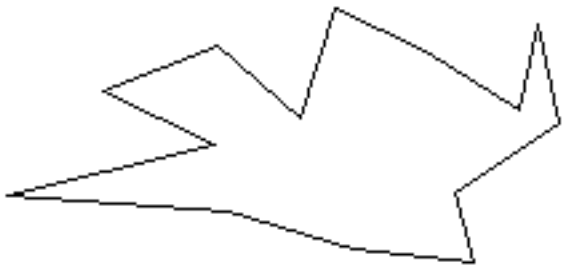
We define an internal energy that increases depending on how rough the snake is.

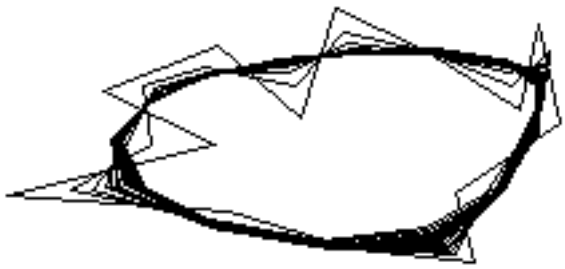


A sharp angle at a control point means a high curvature. We then use

$$E_{\text{internal}} = A \sum_{\text{control points}} (\text{curvature at control point})^2$$

which results in a force which moves each control point towards a smooth curve through its 4 nearest neighbours.





Defining the external energy

It is similarly possible to define the external energy to match particular requirements.

Here the external energy is defined so that the snake wants to lie on regions of high grey-level gradient.

The gradients are shown at the top; but they never actually have to be computed for the whole image.

The snake contracts to lie on the boundaries round the butterfly (except where the elastic energy pulls it tight) even though at some points no clear boundary is defined in the image.

Snake summary

We choose the properties — internal and external — that we want our contour to end up with. We express these as energy formulae and differentiate to get rules to move the control points. We iterate the rules so that the snake evolves to a good state.

Some further points:

- The gradient descent method described here can be improved on greatly. It requires very small steps to work properly. It is possible to take bigger steps if the properties of the internal energy are properly taken into account.
- Snakes do not have to form closed loops — open curves are possible.
- Snakes can be made to inflate — e.g. into the ventricles in ultrasound images of the heart. Many other qualities are possible — e.g. a fixed number of sharp corners.
- Snakes can be attracted to e.g. line terminations or texture changes, amongst many other image qualities.
- Snakes can be *constrained* to satisfy particular user requirements, to possess particular symmetries, to deform in particular ways only (“affine snakes”), or to form a stereo pair.
- Snakes can have *dynamic* properties that makes them suitable for tracking moving objects.
- Snakes are computationally **very cheap**.

A problem is how to initialise snakes:

- By hand — they were originally conceived as a “power assist” for human operators.
- Randomly all over.
- On the basis of some other kind of processing.