# 6 Deformable models

Often, image segmentation involves a combination of two terms: one dealing with the image data and the other describing a desirable segmentation. For example, we have used Markov random fields to impose smoothness on the segmentation. The topic of this chapter, deformable models for image segmentation, is another strategy which combines two contributions: the first originating from the image, and the second imposing smoothness.

With deformable models, image segmentation is performed by evolving (moving, displacing) a curve in an image. The curve moves under the influence of *external forces*, which are computed from the image data, and *internal forces* which have to do with the curve itself.

Deformable models are generally classified as either *parametric* (also called explicit) or *implicit* (in the context of image segmentation also called geometric), depending on the method used for representing the curve, see Figure 6.1. Despite this fundamental difference in curve representation, the underlying principles of both methods are the same [1].

In the exercise for this chapter, we use parametric curve representation, often called a *snake* [2], $C(s) = (x(s), y(s))$ where parameter $s \in [0,1]$ is an arclength. In a discrete setting, this becomes a sequence of points $(x_s, y_s)$, and parameter $s$ becomes a discrete index $s = \{1, \ldots, n\}$ indicating ordering of the points. It is convinient to represent such a curve using an $n \times 2$ array (matrix) of numbers (coordinates).

In the exercise we consider an image where the task is to separate the foreground from the background. At any time, a curve $C$ divides the image into inside and outside region. We want to deform the curve such that it, eventually, delineates the foreground. In the notation, we will use subscripts in and out for inner and the outer region of the curve.

The curve is guided by the segmentation energy $E$, which should be defined such that the desired segmentation has a minimal energy. The desired segmentation should depend on the image data, but should also be relatively smooth. To incorporate those two factors, there are

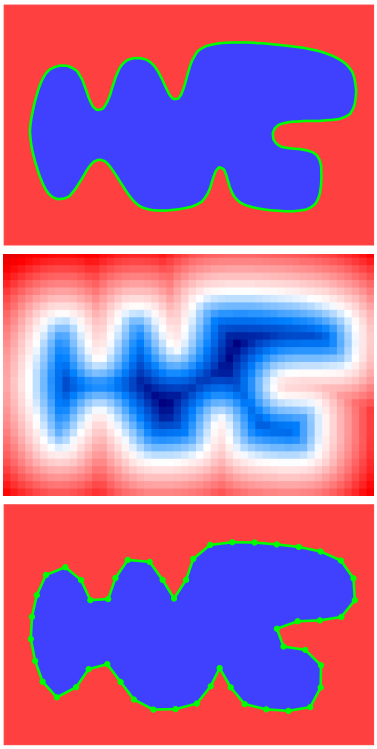

Figure 6.1: A curve (top) and its two discrete representations: implicit (middle) and parametric (bottom).

[1] Chenyang Xu, Anthony Yezzi Jr, and Jerry L Prince. On the relationship between parametric and geometric active contours. In *The Asilomar Conference on Signals, Systems,and Computers*, volume 1, pages 483–489. IEEE, 2000b

[2] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988

two energy contributions $E = E_{\text{ext}} + E_{\text{int}}$. Here, we use $E_{\text{ext}}$ to denote external energy, which is a contribution to the segmentation energy determined by image data. And we use $E_{\text{int}}$ for internal energy, which has to do with the curve itself.

A segmentation is obtained by iteratively moving the curve to minimize the energy, and the most challenging part of the approach is deriving energy-minimizing curve deformation forces $F = -\nabla E$. Since energy may be broken in two contributions, the forces also have two terms, $F = F_{\text{ext}} + F_{\text{int}}$.

To allow deformation, the curve is made dynamic (time-dependable), and its change in time, often denoted *evolution*, is given by

$$\frac{\partial C}{\partial t} = F(C)\,.$$

This exercise is inspired by the Chan-Vese algorithm [3], a deformable model for image segmentation which minimizes a piecewise-constant Mumford-Shah functional. In the original formulation, Chan-Vese uses a implicit (level-set) curve representation and a two-step optimization. We will use the solution of the Chan-Vese approach, but, instead of using level-sets, we will combine it with a parametric curve representation. For this reason, even though our model has an external energy identical to the one used by the Chan-Vese algorithm, the internal energy we use is the same as what is used for snakes.

[3] Tony F Chan and Luminita A Vese. Active contours without edges. *IEEE Transactions on image processing*, 10(2):266–277, 2001

In the following two sections, we briefly cover first the external, and then the internal energy. The detailed explanation on how external forces are derived from external energy can be found in the Chan-Vese article. How internal forces are derived is explained in Chapter 3 of the Handbook of Medical Imaging, Image Segmentation Using Deformable Models [4], subsection 3.2.1 and 3.2.4. Recall that you already implemented curve smoothing as one of the introductory exercises during the first week of the course.

[4] Chenyang Xu, Dzung L Pham, and Jerry L Prince. Image segmentation using deformable models. *Handbook of medical imaging*, 2:129–174, 2000a

## 6.1 *External energy, Chan-Vese*

An external energy closely related to the two-phase piecewise constant Mumford-Shah model is

$$E_{\text{ext}} = \int_{\Omega_{\text{in}}} (I - m_{\text{in}})^2 \, d\omega + \int_{\Omega_{\text{out}}} (I - m_{\text{out}})^2 \, d\omega$$

where $I$ is an image intensity as a function of the pixel position, while $m_{\text{in}}$ and $m_{\text{out}}$ are mean intensities of the inside and the outside region. This energy seeks the best (in a squared-error sense) piecewise constant approximation of $I$. An evolution that will deform a curve toward an energy minimum is derived as

$$F_{\text{ext}} = (m_{\text{in}} - m_{\text{out}})\,(2I - m_{\text{in}} - m_{\text{out}})\,N\,. \tag{6.1}$$

where $N$ denotes an outward unit normal of the curve.

In other words, the curve deforms in the normal direction, and for every point on the curve we only need to compute the (signed) length of the displacement. We will denote the scalar components of the force as $f_{\text{ext}} = (m_{\text{in}} - m_{\text{out}})\,(2I - m_{\text{in}} - m_{\text{out}})$. Note that this can be written as $f_{\text{ext}} = 2(m_{\text{in}} - m_{\text{out}})\left(I - \frac{1}{2}(m_{\text{in}} + m_{\text{out}})\right)$, i.e. the signed length of displacement is proportional to the difference between the image intensities and the mean of $m_{\text{in}}$ and $m_{\text{out}}$.

## 6.2 *Internal forces, snakes*

The internal energy is determined solely by the shape of the curve. In the classical snakes formulation internal forces discourage stretching and bending of the curve

$$E_{\text{int}} = \frac{1}{2}\int_0^1 \alpha \left|\frac{\partial C}{\partial s}\right|^2 + \beta \left|\frac{\partial^2 C}{\partial s^2}\right|^2 ds\,,$$

with weights $\alpha$ and $\beta$ controlling the elasticity (first-order derivative) and the rigidity (second-order derivative) term. Corresponding deformation forces are

$$F_{\text{int}} = \frac{\partial}{\partial s}\left(\alpha \frac{\partial C}{\partial s}\right) - \frac{\partial^2}{\partial s^2}\left(\beta \frac{\partial^2 C}{\partial s^2}\right)\,. \tag{6.2}$$

Those internal (regulatory) forces are the key to success of deformable models, as they provide robustness to noise.

Since our snake is discrete, the derivatives should be approximated by finite differences. Applying internal forces (regularization) now corresponds to filtering (smoothing) the curve with filters for the second and the (negative) fourth derivative, i.e. the filter $\begin{bmatrix} 1 & -2 & 1 \end{bmatrix}$ and the filter $\begin{bmatrix} -1 & 4 & -6 & 4 & -1 \end{bmatrix}$. Those contributions, weighted by parameters $\alpha$ and $\beta$ are now used to regularize (smooth) the curve. In efficient implementation this is done by a matrix multiplication, and for better stability we use a backward Euler scheme. For slightly more detail, you can revise the introductory exercise on curve smoothing 1.1.3.

## 6.3 *Final model*

For a snake consisting of $n$ points and represented using an $n \times 2$ matrix $\mathbf{C}$, a final discrete update step is, adapted from Handbook of Medical Imaging, Eq. (3.22),

$$\mathbf{C}^t = \mathbf{B}_{\text{int}}\left(\mathbf{C}^{t-1} + \tau\,\text{diag}(\mathbf{f}_{\text{ext}})\,\mathbf{N}^{t-1}\right)\,. \tag{6.3}$$

In this expression $\tau$ is the time step for displacement, while $\mathbf{B}_{\text{int}}$ is the $n \times n$ matrix used for regularizing the curve and taking the role of the internal forces. Curve normals are represented as an $n \times 2$ matrix $\mathbf{N}$, and pointwise displacement is obtained by multiplying $\mathbf{N}$ with a $n \times n$ diagonal matrix containing the displacement lengths. Note that the multiplication with matrix $\text{diag}(\mathbf{f}_{\text{ext}})$ is simpler to implement as a row-wise multiplication with a vector $\mathbf{f}_{\text{ext}}$.

## 6.4 *Exercise: Segmentation and tracking*

We will use a deformable model to segment and track a simple organism in a sequence of images. You are provided with two image sequences: crawling amoeba [5] and water bear [6]. The same code can be used for both sequences, with only a minor adjustment in a pre-processing step.

While tracking is the goal of the exercise, it is always advisable to use very simple problems while writing the first version of the code. You are therefore advised to first implement curve deformation (steps 3 to 9 in the approach sketched below) for a simple image, for example the provided `plusplus.png`. Once you can segment plus-plus, move on to segmenting and tracking image sequences.

Steps for solving the whole tracking problem are listed below, with few hints for python (and MATLAB) users. For step 8, we provide a couple of helping functions.

[5] The video of crawling amoeba is from Essential Cell Biology, 3rd Edition Alberts, Bray, Hopkin, Johnson, Lewis, Raff, Roberts, & Walter, https://www.dnatube.com/video/4163/Crawling-Amoeba

[6] The video of water bear is from Olympus microscopy resources, https://www.olympus-lifescience.com/ru/microscope-resource/moviegallery/pondscum/tardigrada/echiniscus

*Tasks*

1. Read in and inspect the movie data. In python you may use function `get_reader` from `imageio` package. In MATLAB you may use `VideoReader`.

2. Process movie frames. For our segmentation method to work, movie frames need to be transformed in grayscale images with a significant difference in intensities of the foreground and a background. For the movie showing the crawling amoeba (which is white on a dark background), it is enough to convert movie frames to grayscale images. Transforming intensities to doubles between 0 and 1 is advisable, as it might prevent issues in subsequent processing. For the movie of the echinicsus, we want to utilize the fact that foreground is yellow while background is blue. A example of suitable transformation is $(2b - (r + g) + 2)/4$, with $r$, $g$, $b$ being color channels (with values between 0 and 1) of movie frames.

3. Choose a starting frame and initialize a snake so that it roughly delineates the foreground object. You may define a circular snake with points $(x_0 + r\cos\alpha, y_0 + r\sin\alpha)$, where $(x_0, y_0)$ is a circle center,

$r$ is a radius and angular parameter $\alpha$ takes $n$ values from $[0, 2\pi\rangle$. See Figure 6.2 for example, but use approximately 100 points along the curve.

4. Compute mean intensities inside and outside the snake. In python use `polygon2mask` from package `skimage.draw` introduced in version 0.16. In MATLAB you can use `poly2mask` function.

5. Compute the magnitude of the snake displacement given by Eq. (6.1). That is, for each snake point, compute the scalar value giving the (signed) length of the deformation in the normal direction. This depends on image data under the snake and estimated mean intensities, as shown in Figure 6.3. A simple approach evaluates the image intensities under the snake by rounding the coordinates of the snake points. A more advanced approach involves interpolating the image at the positions of snake points for example using bilinear interpolation. Image interpolation is in python available in `scipy.interpolate` package, for example check the class `RegularGridInterpolator` or one of the other interpolants. You can also use the function `map_coordinates` from `scipy.ndimage`. In MATLAB you can use function `interp2`.

6. Write a function which takes snake points $\mathbf{C}$ as an input and returns snake normals $\mathbf{N}$. A normal to point $c_i$ can be approximated by a unit vector orthogonal to $c_{i+1} - c_{i-1}$. (Alternatively, and slightly better, you may average the normals of two line segments meeting at $c_i$.) Displace the snake. Estimate a reasonable value for the size of the update step by visualizing the displacement. You should later fine-tune this value so that the segmentation runs sufficiently fast, but without introducing exaggerated oscillations. This step corresponds to computing the expression in the parentheses in the Eq. (6.3).

7. Write a function which given $\alpha$, $\beta$ and $n$ constructs a regularization matrix $\mathbf{B}_{\text{int}}$. Your code from the introductory exercise could be used. Apply regularization to a snake. Estimate a reasonable values for the regularization parameters $\alpha$ and $\beta$ by visualizing the effect of regularization. You should later fine-tune these values to obtain a segmentation with the boundary which is both smooth and sufficiently detailed. This step corresponds to matrix multiplication on the right hand side of the Eq. (6.3).

8. The quality of the curve representation may deteriorate during evolution, especially if you use a large time step $\tau$ and/or weak regularization, i.e. small $\alpha$ and $\beta$. To allow faster evolution without curve deterioration, you may choose to apply a number of substeps (implemented as subfunctions) which ensure the quality of the snake:

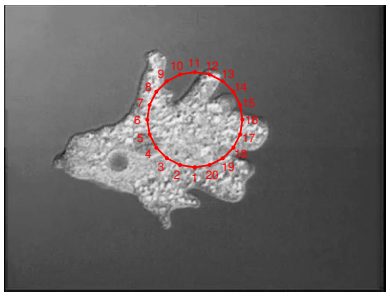

Figure 6.2: The first frame of a crawling amoeba and a circular a 20-point snake.

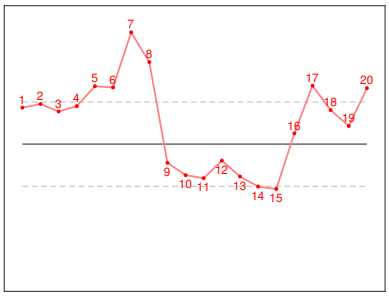

Figure 6.3: Red curve shows image intensities along the snake in Figure 6.2. Dashed gray lines indicate $m_{\text{in}}$ and $m_{\text{out}}$, while gray line indicates the mean of $m_{\text{in}}$ and $m_{\text{out}}$. Signed length of the curve displacement $f_{\text{ext}}$ is computed from the difference between the red curve and the gray line.

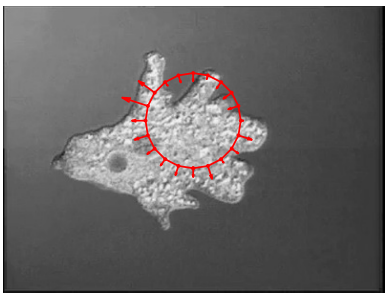

Figure 6.4: External force on the curve indicated by arrows. Displacement is in the normal direction and the length of the displacement is given by the values shown in Figure 6.3.

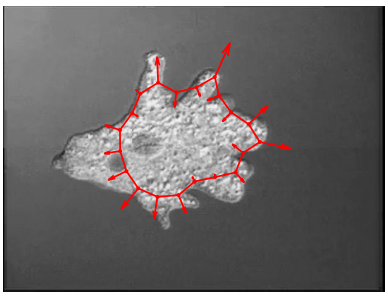

Figure 6.5: The curve and the external forces after 20 iterations.

- Constrain snake to image domain.
- Distribute points equidistantly along the snake. This can be obtained using 1D interpolation.
- Apply heuristics for removing crossings from the snake. For example, if you detect self-intersection, identify two curve segments separated by the intersection and reverse the ordering of the smallest segment.

We provide functions for distributing points and removing crossings, in python and in MATLAB. Note: the provided functions may be expecting the snake to be $2 \times n$ matrix, so make sure to check how the function is to be used.

9. Repeat steps 4–8 until a desirable segmentation is achieved. Note that the regularization matrix only depends on regularization parameters and a number of snake points. This is constant when the size of the snake and the regularization are fixed, which is a typical case. It is therefore sufficient to precompute $\mathbf{B}_{\text{int}}$ prior to looping. Figure 6.5 shows our 20-point snake during evolution.

10. Read in the next frame of the movie, and use the results of the previous frame as an initialization. Evolve the curve a few times by repeating steps 4–8.

11. Process additional frames of the image sequence.