Efficient active contour and K-means algorithms in image segmentation

J.R. Rommelse\textsuperscript{a,b}, H.X. Lin\textsuperscript{a,c} and T.F. Chan\textsuperscript{d}
\textsuperscript{a}Department of Applied Mathematical Analysis, Delft University of Technology, Delft, 2628 CD, The Netherlands
\textsuperscript{b}E-mail: J.R.Rommelse@ewi.tudelft.nl
\textsuperscript{c}E-mail: H.X.Lin@ewi.tudelft.nl
\textsuperscript{d}Department of Mathematics, University of California, Los Angeles Los Angeles, CA 90095-1555, USA
E-mail: Chan@math.ucla.edu

Abstract. In this paper we discuss a classic clustering algorithm that can be used to segment images and a recently developed active contour image segmentation model. We propose integrating aspects of the classic algorithm to improve the active contour model. For the resulting CVK and B-means segmentation algorithms we examine methods to decrease the size of the image domain.

The CVK method has been implemented to run on parallel and distributed computers. By changing the order of updating the pixels, it was possible to replace synchronous communication with asynchronous communication and subsequently the parallel efficiency is improved.

Keywords: Image segmentation, clustering algorithm, active contour model, level set functions, synchronous and asynchronous communication

1. Introduction

Looking at an image, it is usually very easy for a human to see what it represents. For a computer this is not so easy. Before computers "know" what is represented, objects must be measured, but before that, the objects must first be detected. Understanding images is very important in problems like stereo and motion estimation, part recognition or image indexing. The first step in image understanding is image segmentation. Segmentation is the problem of dividing an image into objects or distinguishing objects from a background.

This paper is based on [11], which was presented at DCABES2002 and discusses the classic K-means algorithm, the recently developed Chan – Vese (CV) active contour model and combinations of both (CVK, B-means). K-means can only handle a small subset of images and needs image enhancement for noisy or blurred images. CV and CVK are designed to handle a much larger class of images without enhancement. Unfortunately these algorithms are often slower. Without any optimizations, a typical 600×480 image will take several hours. Therefore methods that reduce the size of the image domain and parallel and distributed computers are used to speedup calculation.

Another combination of CV and K-means is the B-means algorithm. This algorithm was developed after DCABES2002 and gives segmentations with the same quality as CVK but at lower computational costs. No parallel implementation is yet available.

1.1. Image processing

An image is a representation or imitation of an object. Using this definition many images cannot be seen with the naked eye. In this paper only a small subset of images is considered: images that can be represented by a brightness quantified in every point of a physical domain. For three- or lower dimensional domains, these images can be visualized. The brightness can be quantified by a number between a minimum value (black) and a maximum value (white), three numbers...
for a red, a green and a blue channel or in general by a vector of suitable size.

Image processing produces a modified image from an original image, for example compression and restoration (noise removal, deblurring or image enhancement). Image analysis processes an original image but does not produce an image. Instead it produces a measurement. So image processing and image analysis distinguish themselves by their output, image or measurement. Image segmentation is a form of image analysis and is concerned with finding the boundaries between objects in the image. After the image is segmented and the objects are represented in mathematical form, properties of the objects can be measured in order to recognize and classify the objects. Segmentation, measurement and classification are forms of image analysis. The term image processing is sometimes used for both image processing and image analysis.

This paper concentrates on image segmentation algorithms that do not need image enhancement, since they are robust with noisy or blurred images.

1.2. Image segmentation

For every point in an image domain, \( x \in \Omega \), the intensity of a gray-valued image can be specified by a number in \([0,1]\) (0 for black, 1 for white) and the intensity of a RGB image (red, green, blue) can be specified by three numbers in \([0,1]\). In general, the intensity of a grayscale image can be specified by \( m \) numbers in \([0,1]\). Therefore images are mappings from \( \Omega \) to \([0,1]^m\) and can be written as

\[
u : \Omega \rightarrow [0,1]^m,
\]

\[
u(x) = (\mu_0(x), \ldots, \mu_{m-1}(x)), \quad x \in \Omega
\]

The image domain can be discrete (a grid of points or pixels) or continuous, has a rectangular shape and can have any number of dimensions larger than one.

For example,

\[
\Omega = [0, w_0] \times [0, w_1]
\]

for a 2D continuous image, or

\[
\Omega = \{0, \ldots, \omega_0 - 1\} \times \ldots \times \{0, \ldots, \omega_d - 1\}
\]

for a d-dimensional digital image. Here \( w_0 \) and \( w_1 \) are the width and height of the image and \( \omega_i \) is the number of grid points in the \((i+1)\)-th dimension.

The goal of image segmentation is to segment the image domain \( \Omega \) into several subdomains \( \Omega_i \), based on some appropriate criteria that involve intensity or location of colors, such that the domain is formed by the union of the subdomains and the subdomains do not overlap,

\[
\Omega = \bigcup_{i=0}^{k-1} \Omega_i, \quad \Omega_i \cap \Omega_j = \emptyset \ (i \neq j).
\]

After segmentation is completed, the subdomains are considered \( k \) separate objects, or \( k-1 \) separate objects and a background. The objects can then be measured and classified or recognized. This step towards image understanding is not covered by this paper.

2. Review of K-means algorithm

In this chapter the classic K-means clustering algorithm is introduced (section 2.1), as well as some commonly used extensions (section 2.2).

2.1. K-means clustering algorithm [6]

The K-means clustering algorithm was designed to cluster a number \((N)\) of objects \( \{O_i : i = 0, \ldots, N - 1\} \) into \( K \) clusters or classes \( \{K_i : i = 0, \ldots, K - 1\} \), based on location of the objects. The objects \( O_i \) can be assigned values \( v_i \) by using some measure \( \sigma \); \( v_i = \sigma(O_i) \), and the distance between objects can be expressed in terms of the distance between values;

\[
d(O_i, O_j) = d(v_i, v_j).\]

Based on this measure, every class \( K_i \) is assigned a virtual mean object \( \overline{O}_i \), so

\[
v_i = \sigma(\overline{O}_i) = \frac{\sum_{v_j \in K_i} v_j}{|K_i|}
\]

and the distance between an object and a class can be defined by

\[
d(O_i, K_j) = d(O_i, \overline{O}_j).
\]

The K-means segmentation is completed when all objects are assigned to classes in such a way that the distance to their mean objects is smaller than the distance to the mean objects of other classes. The algorithm works by sequentially assigning all objects to the classes that lie closest to them and then recalculating the mean objects. The objective is to assign the objects to classes so that they lie closer to the average location of their class than to the average location of other classes.

The algorithm can be used to cluster pixels of a digital image with related color if the pixels are considered objects. Location is measured in terms of color rather than the actual position of the pixels in the image; pixels...
are located in color space, not in physical space. The average location in color space of pixels in a class can be calculated by averaging the colors of all pixels in the class. The distance between a pixel \( p_i \) and the average of a class \( a_j \), can be expressed by

\[
d(p_i, a_j) = \frac{1}{m} \sum_{k=0}^{m-1} \lambda_{j,k}(p_{i,k} - a_{j,k})^2, \tag{2}
\]

(\( m = 1 \) for a gray valued image and \( m = 3 \) for a RGB image).

The parameters \( \lambda_{j,k} \) can be used to give priority to classes, indexed by \( j \), or colors, indexed by \( k \).

2.2. Extensions to K-means image segmentation [8,9]

2.2.1. On-the-fly

In the standard K-means algorithm, all pixels are updated before the new means are calculated. The means could also be updated after every pixel update. This is called on-the-fly K-means. The idea is that the calculation time per pixel update increases, but the number of iterations decreases. In practice, the means will not have to be recalculated completely after every pixel update. If the numerator, the denominator and the quotient of Eq. (1) are stored for all classes at all time, only six operations are needed to move a pixel from one class to another.

2.2.2. R-means

If the standard K-means algorithm is executed, the order in which the pixels are updated does not make a difference. However, if a pixel is updated in the on-the-fly version, it affects the means, and therefore the decisions to move other pixels. The order in which the pixels are updated influences the number of iterations, but no optimal deterministic order exists for an arbitrary input image. The R-means algorithm is a stochastic extension of the on-the-fly K-means algorithm where the order of updating the pixels is random. Pixels can be picked from a uniform distribution, or from a non-uniform distribution that models additional constraints.

2.2.3. X-means

A major disadvantage of previously discussed K-means versions is that the user must supply both the input image and the number of classes \( K \). K-means is not applicable to large sequences of input images, if the user is prompted for an extra input parameter for every image. X-means [9] is a variant of K-means where also the optimal choice of the number of classes is determined.

3. Active contour image segmentation

Image segmentation methods that work by curve evolution are called snakes or active contour models. Here a parameterized hypersurface \( C \) (or contour in 2D) moves through the image domain with respect to constraints from the image and stops on the boundaries of objects. The parameterization of the hypersurfaces must be flexible enough to deal with merging or breaking hypersurfaces. Level set functions [7] are handy tools and are explained in section 3.1. In section 3.2, an active contour model is described and some numerical aspects are discussed in section 3.3 and section 3.4.

3.1. Level set functions [7]

Level set functions can be used to track the positions of evolving fronts through time, using speed functions that can be defined in the whole space or just near the fronts. They can be applied to image segmentation if the fronts represent estimates of the boundaries between objects and evolution occurs in the directions that give better estimates. In order for the segmentation algorithm to work on any image, the fronts must be able to take on every shape. Therefore a representation must be chosen that can represent every topology and also every topological change.

On a \( d \)-dimensional domain \( \Omega \), a contour or hypersurface \( C \) is a \((d-1)\)-dimensional object, but it can be represented by a function \( \phi : \Omega \rightarrow \mathbb{R} \) on \( \Omega \):

\[
C = \{ x \in \Omega : \phi(x) = 0 \} \tag{3}
\]

\( C \) is the zero level set of \( \phi \) and

\[
n(x_0) = \frac{\nabla \phi}{|\nabla \phi|} \tag{4}
\]

is a vector normal to \( \{ x \in \Omega : \phi(x) = \phi(x_0) \} \) at position \( x_0 \) pointing in the direction of level sets with \( \phi(x) > \phi(x_0) \). To make the normal point outward, \( \phi \) is chosen to be negative inside \( C \) and positive outside \( C \) (Fig. 1).

As a contour evolves over time, it remains represented by the zero level set:

\[
C(t) = \{ x(t) \in \Omega : \phi(x(t), t) = 0 \} \tag{5}
\]

Differentiating Eq. (5) gives \( \phi_t + \nabla \phi \cdot x_t = 0 \). The speed of a contour in tangential directions is not important for tracking the contour as it evolves over time, so only the speed in normal direction is considered:

\[
\nu(x(t), t) = x_t \cdot \frac{\nabla \phi}{|\nabla \phi|}
\]
The latter can be done by setting every subdomain are constants that represent the intensity of the image on level set function 2

...,

\[ \nabla \phi | \nabla \phi | \]

\[ \phi = \frac{0}{0} \]

\[ \phi > 0 \]

\[ \phi < 0 \]

Fig. 1. Representation of a hypersurface \( C \) and the unit normal by a level set function \( \phi \) on \( \Omega \).

This results in an evolution equation for \( \phi \):

\[ \phi_t + \nu |\nabla \phi| = 0, \text{ given } \phi(x, 0) \]  

(6)

Not just the zero level set, but all the level sets move in normal direction with speed \( \nu(x, t) \).

3.2. Chan–Vese image segmentation [4, 16, 17]

In this section a short introduction to Chan-Vese image segmentation is given. A more detailed derivation is given in Appendix A1.

For vector-valued images (\( m > 1 \)), Chan & Vese base the constraints on the minimization of an energy functional

\[ F_{n,m}^{CV}(C_0, \ldots, C_{n-1}, c_0, \ldots, c_{2^n-1}) = \]

(7)

\[ \sum_{i=0}^{n-1} C_i + \frac{1}{m} \sum_{i=0}^{2^n-1} m \sum_{j=0}^{m-1} \lambda_{i,j} \int_{\Omega_i} (u_j - c_{i,j})^2 \, dx , \]

where \( C = \bigcup_{i=0}^{n-1} C_i \) is a set of \( n \) simple closed hypersurfaces \( \{C_0, \ldots, C_{n-1}\} \) that segments \( \Omega \setminus C \) in \( 2^n \) subdomains \( \{\Omega_0, \ldots, \Omega_{2^n-1}\} \), surface\( (C_i) \) is the length of a contour (in 2D) or the area of a surface or hypersurface (in 3D or higher dimensions), \( c_i \in [0, 1]^m \) are constants that represent the intensity of the image on every subdomain \( \Omega_i \), \( \mu \geq 0 \) and \( \lambda \geq 0 \) are parameters that can be set by the user.

The steps of the algorithm are to minimize the energy Eq. (7) by moving the hypersurfaces while keeping \( \{c_0, \ldots, c_{2^n-1}\} \) fixed, and then recalculate \( \{c_0, \ldots, c_{2^n-1}\} \) while keeping the hypersurfaces fixed. The latter can be done by setting \( c_i \) equal to the average color of the image on subdomain \( \Omega_i \). In case \( |\Omega_i| = 0 \), subdomain \( \Omega_i \) is empty, it is not necessary to know \( c_i \), since the subdomain does not contribute to the energy. For the former an appropriate representation for the hypersurfaces must be chosen: level set functions. Hypersurface \( C_i \) is represented by the zero level set Eq. (3) of the function \( \phi_i : \Omega \rightarrow \mathbb{R} \). If the number of hypersurfaces \( n \) is larger than one, the level set functions are combined in a vector:

\[ \Phi : \Omega \rightarrow \mathbb{R}, \quad \Phi = (\phi_0, \ldots, \phi_{n-1}). \]

\( \{c_0, \ldots, c_{2^n-1}\} \) is written as a \( 2^n \times m \) matrix \( c \) to shorten notation. The energy Eq. (7) can be rewritten in terms of the level set functions

\[ F_{n,m}^{CV}(\Phi, c) = \mu \sum_{i=0}^{n-1} \int_{\Omega} \delta(\phi_i)|\nabla \phi_i| \, dx + \]

(8)

\[ \frac{1}{m} \sum_{i=0}^{2^n-1} \sum_{j=0}^{m-1} \lambda_{i,j} \int_{\Omega(u_j - c_{i,j})^2} \chi_i(\Phi) \, dx \]

This energy can be minimized with respect to \( \Phi \) by calculating the Euler-Lagrange equations

\[ \frac{\partial \phi_i}{\partial t} = \delta(\phi_i) \left( \mu \nabla \cdot \left( \frac{\nabla \phi_i}{|\nabla \phi_i|} \right) - \frac{\text{sign}(\phi_i)}{m} \right) \]

(9)

\[ \delta(\phi_i) \frac{\nabla \phi_i}{|\nabla \phi_i|} \cdot n = 0 \]

(10)

\[ \forall i \in \{0, \ldots, n-1\} \text{ on } \partial \Omega \]

In Eq. (9), \( I(x) \) is the index of the subdomain that \( x \) belongs to and \( I_i(x) \) is the index of the subdomain that lies opposite subdomain \( I(x) \) relative to hypersurface \( i \).

Introducing penalty functions \( p_i : \Omega \rightarrow \mathbb{R} \) to express why points in \( \Omega \) should not be \( \Omega_i \),

\[ p_i(x) = \frac{1}{m} \sum_{j=0}^{m-1} (\lambda_{i,j}(u_j - c_{i,j})^2) , \]

the Euler-Lagrange equations Eq. (9) become

\[ \frac{\partial \phi_i}{\partial t} = \delta(\phi_i) \left( \mu \nabla \cdot \left( \frac{\nabla \phi_i}{|\nabla \phi_i|} \right) - \beta_{i,j}(p_I - p_i) \right) \]

Notice that these penalties are also used in the K-means algorithm Eq. (2).

To get an evolution equation for a contour moving in normal direction, \( \delta(\phi_i) \) is replaced by \( |\nabla \phi_i| \) [7]. In
that case Eq. (6) is valid with speed function
\[ \nu = \text{sign}(\phi_i(p_I - p_f)) - \mu \nabla \cdot \left( \frac{\nabla \phi_i}{|\nabla \phi_i|} \right). \]

So
\[ \frac{\partial \phi_i}{\partial t} = |\nabla \phi_i| \left( \mu \nabla \cdot \left( \frac{\nabla \phi_i}{|\nabla \phi_i|} \right) \right. \]
\[ \left. - \text{sign}(\phi_i)(p_I - p_f) \right), \quad (11) \]

and Neumann boundary conditions
\[ \nabla \phi_i \cdot n = 0 \forall i \in \{0, \ldots, n - 1\} \text{ on } \partial \Omega. \]

For example a gray-valued image with one contour
\[ \frac{\partial \phi}{\partial t} = \left| \nabla \phi \right| \left( \mu \nabla \cdot \left( \frac{\nabla \phi}{|\nabla \phi|} \right) + \lambda_0 (u - c_0)^2 \right. \]
\[ \left. - \lambda_1 (u - c_1)^2 \right), \]

or a RGB image with two contours:
\[ \frac{\partial \phi_i}{\partial t} = \mu |\nabla \phi_i| \left( \nabla \phi_i \cdot \left( \frac{\nabla \phi_i}{|\nabla \phi_i|} \right) \right. \]
\[ \left. + \frac{|\nabla \phi_i|}{3} \right) \]
\[ \begin{cases} \sum_{j=0}^{2} (\lambda_{0,i} (u_j - c_{0,j})^2 - \lambda_{1,i} (u_j - c_{1,j})^2) \\
\text{if } x \text{ lies inside } C_i \\
\sum_{j=0}^{2} (\lambda_{2,i} (u_j - c_{2,j})^2 - \lambda_{3,i} (u_j - c_{3,j})^2) \\
\text{if } x \text{ lies outside } C_i \end{cases} \]
\[ \frac{\partial \phi_i}{\partial t} = \mu |\nabla \phi_i| \left( \nabla \phi_i \cdot \left( \frac{\nabla \phi_i}{|\nabla \phi_i|} \right) \right. \]
\[ \left. + \frac{|\nabla \phi_i|}{3} \right) \]
\[ \begin{cases} \sum_{j=0}^{2} (\lambda_{0,i} (u_j - c_{0,j})^2 - \lambda_{2,i} (u_j - c_{2,j})^2) \\
\text{if } x \text{ lies inside } C_0 \\
\sum_{j=0}^{2} (\lambda_{1,i} (u_j - c_{1,j})^2 - \lambda_{3,i} (u_j - c_{3,j})^2) \\
\text{if } x \text{ lies outside } C_0 \end{cases} \]

### 3.3. Singularities

Singularities will arise if the denominator of the evolution equation Eq. (11) equals zero. This only happens in points where \( |\nabla \phi| = 0 \), or \( \nabla \phi = 0 \), so points where the normal to the hypersurface is not well defined. These singularities can be dealt with by adding \( 0 < \varepsilon \ll 1 \) to the denominator [4]
\[ \frac{\partial \phi_i}{\partial t} = |\nabla \phi_i| \left( \mu \nabla \cdot \left( \frac{\nabla \phi_i}{|\nabla \phi_i| + \varepsilon} \right) \right. \]
\[ \left. - \text{sign}(\phi_i)(p_I - p_f) \right), \quad (12) \]

or by rewriting the first term of Eq. (11) to (appendix A2)
\[ \text{trace}(F_i) - \frac{f_i^T F_i f_i}{f_i^T f_i}, \quad f_i = \nabla \phi_i, \quad F_i = \nabla \nabla^T \phi_i \]

and defining the limit for the Rayleigh quotient
\[ R_M(x) = \frac{x^T M x}{x^T x} \]

when the argument approaches the zero-vector. Since \( \lim_{x \to 0} R_M(x) \) does not exist,
\[ R_M(0) = \frac{1^T M 1}{d}, \quad 1 = (1, \ldots, 1) \in \mathbb{R}^d \]

is defined. This means that the limit operator is evaluated in the direction of the all-one-vector, or the line \( \frac{\partial \phi_i}{\partial x_1} = \ldots = \frac{\partial \phi_i}{\partial x_d} \).

The latter approach is used, so Eq. (11) becomes
\[ \frac{\partial \phi_i}{\partial t} = -\text{sign}(\phi_i)|\nabla \phi_i|(p_I - p_f) \]
\[ + \mu \left\{ \frac{\Delta \phi_i - \frac{\mu}{2} |\nabla^T \nabla \phi_i|}{|\nabla \phi_i| \cdot \left( \frac{\nabla \phi_i}{|\nabla \phi_i|} \right) \text{ if } |\nabla \phi_i| = 0 \right. \]
\[ \left. + \text{sign}(\phi_i)(p_I - p_f) \right\} (13) \]

No other singularities will occur as long as the first and second order derivatives of the level set functions exist.

### 3.4. Discretization

By discretizing a step is taken to what is called digital image processing. In digital image processing an image can be represented by a grid where every grid point is called a pixel. The color of a pixel is usually expressed by a number in \( \{0,1\} \) for a black and white image (1 bit in computer memory), a number in \( \{0, \ldots, 15\} \) or \( \{0, \ldots, 255\} \) for a gray-valued or color image (4 bits or 1 byte), and 3 numbers in for \( \{0, \ldots, 255\} \) a color image. In this article, gray values are scaled to \([0,1]\) and color values are scaled to \([0,1]^3\).

Discretizing a PDE usually means that a useful discrete representation of the PDE and the domain must be chosen. In digital image processing the domain is already represented in discrete form, a rectangular grid, \( \Omega = \{0, \omega_0 - 1\} \times \ldots \times \{0, \omega_{d-1} - 1\} \). In 2D, the origin is placed in the left upper corner. Here, to be able to work with any number of dimensions, discrete coordinates \( \eta \) are introduced such that
\[ x = (x_0, \ldots, x_{d-1}) = (h \eta_0, \ldots, h \eta_{d-1}) = h \eta, \]
with \( h \) the spatial step size,
\[ h = \min_{i \in \{0, \ldots, d-1\}} |\omega_i| - 1 \]

and
\( \eta_i \in \{0, \ldots, \omega_i - 1\} \forall i \in \{0, \ldots, d - 1\} \).

The PDE Eq. (13) is then defined on
\[
\Omega = [0, h(\omega_1 - 1)] \times \ldots \times [0, h(\omega_d - 1)]
\]
where \( \min_{i \in \{0, \ldots, d - 1\}} \{\omega_i\} = 1 \).

The time discretization is done with step size \( \Delta t \) and time index \( \tau \), so
\[
\phi_i^{\eta, \tau} = \phi_i^{\eta_0, \ldots, \eta_{d-1}, \tau} = \phi(h\eta_0, \ldots, h\eta_{d-1}, \Delta t),
\]
and central differences on a 2\(d+1\) stencil were used for the spatial derivatives. An explicit scheme was used where only the center of the stencil is taken at the new time index and all other points are taken at the old time index. For example, in 2D:
\[
\begin{align*}
\bar{d}_{xx}^{\eta, \tau} &= \phi_i^{\eta_0 + 1, \eta_1, \tau} - \phi_i^{\eta_0 - 1, \eta_1, \tau}, \\
\bar{d}_{yy}^{\eta, \tau} &= \phi_i^{\eta_0, \eta_1 + 1, \tau} - \phi_i^{\eta_0, \eta_1 - 1, \tau}, \\
\bar{d}_{xy}^{\eta, \tau} &= \frac{1}{d_{xx}^{\eta, \tau}} \cdot \left\{ \begin{array}{ll}
1 & \text{if } (\bar{d}_{xx}^{\eta, \tau})^2 + (\bar{d}_{yy}^{\eta, \tau})^2 = 0 \\
\bar{d}_{xx}^{\eta, \tau} & \text{else}
\end{array} \right., \\
\bar{d}_{yy}^{\eta, \tau} &= \frac{1}{d_{xx}^{\eta, \tau}} \cdot \left\{ \begin{array}{ll}
1 & \text{if } (\bar{d}_{xx}^{\eta, \tau})^2 + (\bar{d}_{yy}^{\eta, \tau})^2 = 0 \\
\bar{d}_{xx}^{\eta, \tau} & \text{else}
\end{array} \right.,
\end{align*}
\]
\[
\begin{align*}
\bar{d}_{xx}^{\eta, \tau} &= \phi_i^{\eta_0 + 1, \eta_1, \tau} + \phi_i^{\eta_0 - 1, \eta_1, \tau} - 2\phi_i^{\eta_0, \eta_1, \tau}, \\
\bar{d}_{yy}^{\eta, \tau} &= \phi_i^{\eta_0, \eta_1 + 1, \tau} + \phi_i^{\eta_0, \eta_1 - 1, \tau} - 2\phi_i^{\eta_0, \eta_1, \tau}, \\
\bar{d}_{xy}^{\eta, \tau} &= \phi_i^{\eta_0 + 1, \eta_1 + 1, \tau} + \phi_i^{\eta_0 - 1, \eta_1 - 1, \tau} \\
&\quad - \phi_i^{\eta_0 - 1, \eta_1 + 1, \tau}, \\
\phi_i^{\eta, \tau + 1} &= \phi_i^{\eta, \tau} + \frac{\mu \Delta t}{h^2} \\
&\quad \cdot \frac{(d_{xx}^{\eta, \tau})^2 d_{yy}^{\eta, \tau} + (d_{xy}^{\eta, \tau})^2 d_{xx}^{\eta, \tau} - \frac{1}{2} d_{xx}^{\eta, \tau} d_{yy}^{\eta, \tau} d_{xy}^{\eta, \tau}}{(d_{xx}^{\eta, \tau})^2 + (d_{yy}^{\eta, \tau})^2} \\
&\quad - \frac{\Delta t}{2h} \text{sign}(\phi_i^{\eta, \tau})((\bar{d}_{xx}^{\eta, \tau})^2 + (\bar{d}_{yy}^{\eta, \tau})^2)^{1/2} (p_t - p_{\bar{f}_i})
\end{align*}
\]

Stability of the numerical scheme can be achieved by choosing
\[
\Delta t = \frac{\alpha h^2}{2}
\]
with \( \alpha \) close to one and \( \mu \) not larger than one, which is the result of the CFL condition that states that a hypersurface may at maximum move with the distance of one pixel.

4. Extensions to CV image segmentation

Because of the CFL condition, Eq. (14), the CV image segmentation algorithm will generally require a lot more calculation time than K-means clustering. In this chapter two methods are explored that can potentially reduce the calculation time of CV image segmentation. The narrow band method (section 4.1.) updates only a small subset of all pixels per iteration, while the multiresolution method (section 4.2.) updates all pixels of a new image that is obtained by down sampling the original image.

4.1. The narrow band method

The first step in reducing the time complexity is to acknowledge that much work is done in vain. In the digital model, a hypersurface moves over a pixel if the level set function on that pixel changes sign in the iteration. Assuming that far away from the hypersurface this changing sign does not happen, it is a waste of time to first update the level set function there using the evolution equation and second reinitialize it to be the signed distance function to the hypersurface. However, this assumption might not always be justified. Instead of updating the level set functions on every grid point, a speedup will be achieved if only values on grid points near the hypersurfaces are updated. By specifying a maximum distance \( \delta \) to the contours and only updating the level set functions on grid points within this distance, a band-shaped domain is created.

Applying the narrow band method to the segmentation algorithm may cause the algorithm to fail. The narrow band method is very effective in problems with evolving fronts where the hypersurface moves with certain speed in normal direction and where the speed function is based on properties of the hypersurface. In the segmentation algorithm, the fitting term
The segmentation algorithm described above can be improved by reducing the time complexity of the segmentation algorithm. Changing the resolution of a digital image means that the same image is spread over a different amount of pixels. A multiresolution method can take advantage of this. The multiresolution method should not be confused with the standard multigrid method, in which an iterative solution and the corresponding problem are coarsened to another grid, where the problem is solved and interpolated back to the fine grid. The grids are used recursively and iteratively. The multiresolution method for the image segmentation problem uses lower resolution versions of the original image to find initial solutions for higher resolution problems instead. So where the multigrid method starts at the highest level, returns to the highest level and uses all coarse grids regularly, the multiresolution method starts at the lowest level, ends at the highest level and uses all coarse grids only once.

Because the image and the level set functions are represented by the same data structures, d-dimensional arrays (matrices if d = 2), the only addition to the previous algorithm is a mechanism for resizing a d-dimensional array. Popular mechanisms are linear (or more advanced) interpolation or nearest neighbor methods. Here they are used for the level set functions and the image respectively.

5. CV and K-means mixtures

In the CV algorithm, the color of a pixel is compared to the mean color of its subdomain and the mean colors of the subdomains that lie opposite the hypersurfaces. Therefore a pixel can stay in its subdomain or move to one of n other subdomains (if there are n hypersurfaces). However, the K-means algorithm allows pixels to move to any of the other $2^n - 1$ subdomains. Apparently in the CV algorithm, pixels might be denied the opportunity to move to the right subdomain. Or in other words, hypersurfaces might not evolve correctly in the neighborhood of other hypersurfaces (Fig. 2).

The figure shows a hypothetical situation where four classes/subdomains are separated by two hypersurfaces $\alpha$ and $\beta$. The averages in the classes are 4, 1, 2 and 3. A pixel/object with value 4 is currently assigned to the class with average 5 (A) and should be assigned to the class with average 4 (D), which means that both hypersurface $\alpha$ (B) and hypersurface $\beta$ (C) will have to move. For the obvious choice of parameters $\lambda_{i,j}$, the CV algorithm does not move the hypersurfaces, because $(4 - 5)^2 < (4 - 1)^2$ and $(4 - 5)^2 < (4 - 2)^2$. The K-means algorithm does move the hypersurfaces, because $(4 - 5)^2 > (4 - 4)^2$.

This chapter describes two attempts of mixing the two different approaches in order to find a new algo-
Fig. 2. Near intersections of hypersurfaces, the CV algorithm might fail where the K-means algorithm does not.

algorithm with better accuracy or lower calculation time. CVK (Section 5.1.) can be seen as CV with some K-means influences and B-means (Section 5.2.) can be seen as K-means with some CV influences.

5.1. CVK algorithm [10]

The CVK algorithm segments an image by evolving hypersurfaces according to the PDE

\[
\frac{\partial \phi_i(x)}{\partial t} = |\nabla \phi_i(x)| \left( \mu \nabla \cdot \left( \frac{\nabla \phi_i(x)}{|\nabla \phi_i(x)|} \right) \right) - (1 - \mu) \text{sign}(\phi_i(y))
\]

where \( \mu \in [0,1] \) and

\[
y = \arg \min_{z \in \Omega} \left\{ \frac{1}{m} \sum_{j=0}^{m-1} \lambda_i,j (u_j(z) - c_{i,j}(z))^2 \right\}
\]

is a pixel in the subdomain where the penalty function Eq. (16) is minimized, so where a pixel \( x \) should be moved to according to the K-means algorithm. Although there are many such pixels \( y \), the level set function \( \phi_i \) has the same sign on all of them.

For \( \mu = 1 \),

\[
\nu_i = \mu_i \text{curvature} = -\kappa(\phi_i)
\]

\[
= -\nabla \cdot \left( \frac{\nabla \phi_i}{|\nabla \phi_i|} \right) = -\nabla \cdot n_i,
\]

so hypersurface \( C_i \) moves in the direction opposite the normal Eq. (4), with a speed equal to its curvature. The curvature term \( v_i \text{curvature} = -\mu k(\phi_i) \) makes the hypersurfaces move in the direction that makes the curvature decrease \( (\frac{d}{dt}) |K| < 0 \). Whether that is in the direction of the normal or in opposite direction depends on the orientation of the level set functions (Fig. 3). As a result, the size of the hypersurface gets smaller. Or actually the size of the hypersurface is minimized by the energy functional and as a result, curvature gets closer to zero. By setting \( \mu \) small, small objects can be detected and larger objects are detected if \( \mu \) is chosen larger. Since noise in images is made of very small objects, the CV and CVK image segmentation algorithms can be made robust to noise or blur by setting \( \mu \) large enough.

The sign function in Eq. (15) makes sure that the level set function \( \phi_i \) on pixel \( x \) gets closer to zero or even changes sign when \( x \) is located on the wrong side of hypersurface \( C_i \). In case \( x \) is located on the right side of \( C_i \), \( \phi_i \) is updated such that \( |\partial \phi_i/\partial t| > 0 \). This is done so the color criterion can oppose the curvature criterion, to prevent the hypersurfaces from showing wiggling behavior when these criteria contradict and alternate dominance in subsequent iterations.

5.2. Narrow band K-means

The power of the CV and CVK algorithms is the robustness with respect to noise, blur or damage, caused by an active contour that keeps spatially correlated pixels connected while still honoring the color constraints. Unfortunately the speed of the active contour decreases as the resolution of the image increases Eq. (14). The narrow band K-means algorithm (B-means) attempts to mimic the active contour behavior of CVK in a K-means framework.

The origin of the B-means algorithm lies in [14] where an attempt was made to speedup the CV algorithm for one active contour, by realizing that for the CV image segmentation and many other algorithms that work by updating level set functions to minimize an energy functional, only the sign of the level set function holds significant information. The length of a contour
is approximated by
\[
\sum_{i,j} \sqrt{(H(\phi_{i+1,j}) - H(\phi_{i,j}))^2 + (H(\phi_{i,j+1}) - H(\phi_{i,j}))^2}
\]
and added to the K-means criterion Eq. (2), to segment an image into two classes. Since the sign can be represented by only one digit in binary representation, the method can be easily extended to 2\textsuperscript{K} classes or arbitrary K classes. Equation (16) is a summation of terms that can only have three values; 0, 1 or \(\sqrt{2}\), depending on the classes of the pixels \((i + 1, j)\) and \((i, j + 1)\) relative to the class of pixel \((i, j)\). In [14] K-means and on-the-fly K-means are denoted by Jacobi and Gauss-Seidel iteration schemes respectively.

In this paper the on-the-fly B-means algorithm from [14] is extended to an arbitrary number of classes. For every pixel, a value of one is added to the energy functional for all neighboring pixels that lie in different classes. So in 2D, the length term can be 0, 1, 2, 3 or 4 instead of 0, 1 or \(\sqrt{2}\). The distance of a pixel \(p_i\) to the mean \(a_j\) of class \(K_j\) is calculated by
\[
d(p_i, a_j) = \mu \sum_{k \in \text{neighbors}(i)} \delta(i, k) + \frac{1 - \mu}{m} \sum_{k=0}^{m-1} \lambda_{j,k}(p_i - a_j)\]
where
\[
\delta(i, k) = \begin{cases} 
1 & \text{if pixels } i \text{ and } k \text{ lie in different classes} \\
0 & \text{if pixels } i \text{ and } k \text{ lie in the same class}
\end{cases}
\]

The active contour is introduced by updating only the pixels that have neighbors that lie in different classes. To store the location of the band, all pixel indices of pixels that have neighbors in different classes are kept in a queue. Every iteration, an index is popped from the queue, then updated, and possibly pushed at the end of the queue. Also neighbor pixels that have become part of the band due to the last update are pushed to the queue if they were not in the queue already. Compared to the narrow band version of the CV and CVK algorithms where the level set functions have to be updated after every iteration, the narrow band of the B-means algorithm comes with practically no calculation overhead, if two conditions are satisfied: given the index of a pixel, the location of the pixel must be accessible from memory very fast, and given the location of the pixel, the location of the next pixel in the queue must be accessible fast. The data structure used consists of a grid of the same size as the digital image, where for every pixel the class index is stored along with the locations of the next and previous pixels in the queue. Also the locations of the first and last pixels in the queue are stored.

6. Parallelization

Due to the CFL condition Eq. (14), speedup methods discussed in chapter 5 are not sufficient for very large images. Therefore CV and CVK are implemented in parallel (section 6.1.). In 6.2. a different ordering of pixels is explored that allows for a more efficient use of calculation and communication resources on some parallel computers.

6.1. Parallel CV/CVK Algorithms

Reassigning pixels to classes or calculating average colors of pixels in classes can be seen as tasks of the algorithm. To reassign a pixel to a class, the color of the pixel must be compared to the average colors of some or all classes. The averages must stay constant until all comparisons are performed, so the averaging task may not start before the reassigning task has finished. Vice versa is also true, since the average colors of classes cannot be calculated correctly if pixels are shifted between classes during this calculation. In the sense of tasks that have to be performed, the segmentation algorithm is clearly sequential by nature. Therefore a data parallel model of computation is chosen. This way, several instances of the algorithm can run on different parts of the image domain. Every instance by itself runs sequentially. In general, hypersurfaces will not necessarily stay on one side of the artificial borders that arise when the image domain is decomposed. Communication must take place between the different instances of the algorithm to prevent occurrence of artificial segments in the segmented image.

The first dimension of the domain is partitioned while the other dimensions are not partitioned. So if \(\Omega = \{0, \ldots, \omega_0 - 1\} \times \ldots \times \{0, \ldots, \omega_d - 1\}\), then \(\{0, \ldots, \omega_0 - 1\}\) is partitioned into \(S\) subsets \(\{0, \ldots, \omega_0 - 1\} = \{\omega_0, \ldots, \omega_1 - 1\} \cup \ldots \cup \{\omega_{S-1}, \ldots, \omega_S - 1\}\), where
\[
\omega_t = i + \min\{i, \omega_0 \mod S\}
\]
and
\[ \mathcal{T}_i = \{\omega_0, \ldots, \omega_{i+1} - 1\} \times \{0, \ldots, \omega_1 - 1\} \times \cdots \times \{0, \ldots, \omega_{d-1} - 1\} \]

If \( d = 2 \) this is called striped partitioning; the domain is a matrix and the columns are assigned to submatrices. In 3D it could be called sliced partitioning. To let terminology stay valid for \( d > 2 \) let

\[ \{i\} \times \{0, \ldots, \omega_1 - 1\} \times \cdots \times \{0, \ldots, \omega_d - 1\} \]

be the \((i+1)\)-th column of \( \Omega \). The first \( \omega_0 \mod S \) subdomains get \( \lfloor \omega_0 / S \rfloor + 1 \) columns and the other subdomains get \( \lfloor \omega_0 / S \rfloor \) columns.

The grid points of the subdomains are assigned to \( S \) processes, a processor can run more than one process. Assigning a grid point to a process means that all data associated with that grid point is stored in the memory of the processor on which the process will run and that operations on the data will be performed by that processor. For every column that does not belong to the same process as its neighbor column an extra column is assigned; operations on this data are done by the neighbor process and data is updated during a synchronizing step between processes. In the synchronization step data is sent between processes, although sending is not the correct word if both processes are located on the same processor. This mechanism makes sure that subdomains overlap by one column, which is exactly enough for the scheme used in the discrete evolution equation.

Except after initialization, the hypersurfaces will be close to the objects to be detected. Since they are not distributed uniformly over the image, neither will be the narrow bands. Some parts of the domain may require more calculation than other parts. To distribute the work equally among the \( p \) processors, the \( S \) subdomains are mapped cyclically to the processors. Given \( p, S \) must be chosen so that \( S \mod p = 0 \). Let \( b = S/p \), then the domain is divided into \( b \) blocks and every block is distributed over \( p \) processors. So if \( p \) is given, \( b \) can be chosen small to reduce communication between processors or \( b \) can be chosen large to ensure a good load balance. In case no narrow band is used, the load is well balanced automatically, so \( b \) can be chosen equal to one.

### 6.2. Synchronous versus asynchronous communication

Before every iteration the overlapping data must be synchronized. This requires \( S-1 \) exchanges of columns between neighbor processes. Also two all-to-all broad-casts must take place to evaluate the stop criterion and to recalculate the average intensities \( c_{ij} \).

Subdomains \( \mathcal{T}_i \) are assigned to processor \( i \mod p \). If neighbor processes are not located on the same processor, which is the case for every process if \( p > 1 \), synchronization must take place. This takes either two or three steps, depending on whether \( p \) is odd or even. In every step a process communicates with either its left or its right neighbor process. Here the communication time is minimized by bundling \( b \) overlapping areas into one message. Messages do not actually have to be copied to a combined message; instead computer memory can be organized so blocks of data that have to be sent in the same communication step are always located at consecutive addresses.

#### 6.2.1. Asynchronous communication

On parallel computer systems where communication is slow, run time could be optimized if calculations and communication could be done in parallel. The advantage of communicating asynchronously is that processors do not have to wait until the other processor involved in the communication is ready. The disadvantage is that after the communication is initiated, the data involved is accessible but not yet usable. This can be dealt with by choosing the order in which grid points are updated carefully: first non-overlapping grid points must be updated, then the overlapping grid points.
7. Evaluation of the algorithms

The segmentation results of the different algorithms are compared in 7.1. In 7.2., calculation times are compared.

7.1. Qualitative comparison

For undamaged, unblurred, synthetic images, all four algorithms (K-means, CV, CVK, B-means) work well. For natural images or noisy images, the K-means algorithm cannot be used to completely segment the images [6,10], although it can still be useful to create an initial guess for other algorithms. CV, CVK and B-means are designed to handle these images as well [4,10,16]. In these algorithms the hypersurfaces are moved by two effects; a fitting term makes sure that pixels in the same object have similar color and a curvature term makes the contours move in the direction that minimizes the curvature, \( \partial_t |\kappa| \leq 0 \). Small objects have large curvature, while large objects have smaller curvature. Noise is actually made of many very small objects and have large curvature. The parameter \( \mu \) can be set by the user to specify whether large or small objects should be detected and can be used to make the algorithms robust to noise. The curvature term deals with noise and keeps detected objects from being scattered.

The improvement in quality of the CVK and B-means over CV can not be measured in terms of correct output of the algorithms, but in terms of user friendliness; both algorithms produce correct results if the right set of parameters \( \mu \) and \( \lambda_{i,j} \) are put in. However, it can be tricky to tune the parameters \( \mu \) and \( \lambda_{i,j} \) for the CV algorithm, whereas CVK and B-means work fine by just choosing \( \mu \) and setting all parameters \( \lambda_{i,j} \) equal to one [10,16]. So CVK and B-means are more robust in usage.

7.2. Quantitative comparison of the CVK versions

In the narrow band method for CV/CVK, time is saved because calculations are only performed on a small domain. On the other hand, extra administration is needed to calculate and store the location of the narrow band. In the current implementation, the narrow band does not result in speedup but some slowdown, whereas previous versions of the narrow band did result in speedup. This is not a flaw in the current implementation of the narrow band method. In previous implementation of the full domain method, the level set methods had to be reinitialized after every iteration.
This could be eliminated in the current implementation of the full domain method, but not in the implementation of the narrow band method. The narrow band in B-means slows down the K-means algorithm, but gives a significant improvement over CVK.

The multiresolution method does not only reduce the number of grid points, but also reduces the number of operations that have to be performed on every grid point. For stability reasons, $\tau = O(h^2)$, with $\tau$ the time step and $h$ the spatial step. On a coarser grid, larger time steps can be made, so lesser iterations are needed.

### 7.3. Results

The CVK algorithm was implemented (full domain, narrow band and multiresolution) in C++ using MPI functions for communication. B-means was implemented in C++ for sequential computation. For qualitative comparison, K-means was implemented in Matlab.

For undamaged, unblurred, synthetic images, K-means is be very fast. The algorithm can finish in just a few iterations, because in these kind of images there is only a small amount of different colors and pixels are assigned to classes based on color and not on location: if it is decided that a pixel with some color should be in some class, than all pixels with the same color are assigned to that class in the same iteration. The active contour algorithms CV/CVK/B-means on the other hand are slower, because only pixels near hypersurfaces are reassigned. For natural images or noisy images, the speed of the K-means algorithm is irrelevant, because the algorithm is not applicable.

Calculation time for CV is discussed in [16]. For the CVK algorithm, calculation time was measured on a Cray T3E parallel computer [19] in Delft and the DAS-2 clusters [20]. The efficiency of the algorithm run on DAS-2 on several gray-valued and color images, ranging in size from $100 \times 100$ to $600 \times 480$, using one or two level set functions, using different initializations and using different numbers of coarse grids, is shown in Fig. 7. Improvement in efficiency can be observed for most test cases on DAS-2, if asynchronous communication is used. Figure 8 shows an example of typical improvement by the asynchronous version. For large images, much less improvement can be seen, but for these images communication overhead plays a less important role relative to the increased number of calculations. Because both MPI_Send and MPI_Isend are implemented asynchronously with default setting of MPI_BUFFER_MAX on the Cray T3E, the use of MPI_Send and MPI_Isend do not influence the efficiency much.
8. Concluding remarks

In this paper we discussed two new image segmentation methods (CVK and B-means) and the parallel implementation of the former. Both methods were created by integrating a classic clustering algorithm (K-means) into a recently developed active contour model (Chan – Vese). The narrow band method and the multiresolution methods were attempts to decrease the size of the image domain and thereby the calculation time. The multiresolution method proved very useful for regular images and indispensable for large images. Because of reinitialization after every time step, the narrow band method for CVK could not compete with the full domain version. However, for B-means the narrow band is a significant improvement.

Parallelization is useful for both small and large images. Efficiency decreases when extra processors are added, but this decrease is smaller for large images than for small images. The MPI 1.1 does not support dynamic allocation of resources during the algorithm. The MPI 2.0 standard will support dynamic allocation of resources, which will make the multiresolution method more efficient; every time the algorithm moves from a coarse grid to a finer grid, more processors could be added.

Replacing synchronous communication functions with asynchronous ones made the DAS-2 [20] more efficient for most test cases. Efficiency did not change much for very large images. On the Cray T3E [19] that was used, no improvement in efficiency could be detected. Communication on the Cray is much faster than the DAS-2, but for the calculation time vice versa.

Appendix A1. Derivation of Chan – Vese image segmentation [4,16,17]

Mumford & Shaw base the constraints on the minimization of an energy functional

$$F^{MS}(\mathbf{u}, C) = \mu \cdot \text{surface}(C) + \lambda \int_\Omega |\mathbf{u} - \mathbf{u}|^2 d\mathbf{x}$$

$$+ \int_{\Omega \setminus C} |\nabla \mathbf{u}|^2 d\mathbf{x},$$

where $\mathbf{u}$ is the original image, $\mathbf{u}$ is the segmented image, $C$ is a hypersurface around detected objects,
surface($C$) is the length of the contour (in 2D) or the area of the surface or hypersurface (in 3D or higher dimensions), $\mu \geq 0$ and $\lambda \geq 0$ are parameters that can be set by the user.

Chan & Vese represent the hypersurface $C$ by a set of $n$ simple closed hypersurfaces $\{C_0, \ldots, C_{n-1}\}$.

$$C = \bigcup_{i=0}^{n-1} C_i.$$  

Every hypersurface $C_i$ segments $\Omega \setminus C$ in 2 subdomains and the set of $n$ hypersurfaces therefore segments $\Omega \setminus C$ in $2^n$ subdomains (Fig. 9).

Let

$$\gamma_i(x) = \begin{cases} 0 & \text{if } x \text{ lies inside } C_i \\ 1 & \text{if } x \text{ lies outside } C_i \end{cases},$$

be characteristic functions of the regions outside the hypersurfaces $C_i$, then every subdomain can be indexed by a binary representation (Fig. 9).

$$x \in \Omega_i \Rightarrow i = I(x) = \sum_{j=0}^{n-1} \gamma_j(x)2^j. \quad (18)$$

The index of the subdomain that lies opposite $\Omega_i$ relative to hypersurface $C_j$ is

$$\tilde{I}_j(x) = \sum_{k \in \{0, \ldots, n-1\} \setminus \{j\}} \gamma_k(x)2^k + (1 - \gamma_j(x))2^j + (1 - 2\gamma_j(x))2^j,$$

so $\tilde{I}_j(x)$ Eq. (19) can be calculated by inverting the $(j+1)$-th bit of $I(x)$ Eq. (18).

For a large subset of images the gradient is small inside objects and large on boundaries. However, objects are harder to detect if they are not defined by a gradient, like in blurred or noisy images or when boundaries are smooth. To adapt the functional so those images can be segmented too, $\mathbf{m}$ is restricted to piecewise constant functions

$$\mathbf{m}(x) = c_i \text{ if } x \in \Omega_i$$

$$\mathbf{m}(x) = \sum_{i=0}^{2^n-1} c_i \chi_i(x)$$

$\mathbf{m}$ has a constant value $c_i \in [0, 1]^m$ on every subdomain $\Omega_i$ and

$$\chi_i(x) = \begin{cases} 1 & \text{if } x \in \Omega_i \\ 0 & \text{if } x \not\in \Omega_i \end{cases}$$

is the characteristic function $\chi_i : \Omega \rightarrow \{0, 1\}$ of subdomain $\Omega_i$. 

---

Fig. 8. Efficiency improves if asynchronous communication is used on DAS-2, for some test cases.
Now the Mumford-Shaw energy functional Eq. (7) becomes

$$F_{n,m}^{MS}(C_0, \ldots, C_{n-1}, c_0, \ldots, c_{2^n-1})$$

$$= \mu \cdot \text{surface} \left( \bigcup_{i=0}^{n-1} C_i \right) + \lambda \sum_{i=0}^{2^n-1} \int_{\Omega_i} |\mathbf{u} - c_i|^2 d\mathbf{x}.$$  

Minimizing $F_{n,m}^{MS}(C_0, \ldots, C_{n-1}, c_0, \ldots, c_{2^n-1})$ with respect to $C$ and $c_i$ is called the minimal partition problem. For more generality,

$$F_{n,m}^{MS}(C_0, \ldots, C_{n-1}, c_0, \ldots, c_{2^n-1})$$

$$= \mu \cdot \text{surface} \left( \bigcup_{i=0}^{n-1} C_i \right)$$

$$+ \sum_{i=0}^{2^n-1} \lambda_i \int_{\Omega_i} |\mathbf{u} - c_i|^2 d\mathbf{x}.$$  

For vector-valued images ($m > 1$), the Chan-Vese energy functional is defined by

$$F_{n,m}^{CV}(C_0, \ldots, C_{n-1}, c_0, \ldots, c_{2^n-1})$$

$$= \mu \sum_{i=0}^{n-1} \text{surface}(C_i)$$

$$+ \frac{1}{m} \sum_{i=0}^{2^n-1} \sum_{j=0}^{m-1} \lambda_{i,j} \int_{\Omega_i} (u_j - c_{i,j})^2 d\mathbf{x},$$

or using the characteristic functions $\chi_i : \Omega \rightarrow \{0,1\}$ Eq. (20),

$$F_{n,m}^{CV}(C_0, \ldots, C_{n-1}, c_0, \ldots, c_{2^n-1})$$

$$= \mu \sum_{i=0}^{n-1} \text{surface}(C_i) + \frac{1}{m} \sum_{i=0}^{2^n-1} \sum_{j=0}^{m-1} \lambda_{i,j} \int_{\Omega_i} (u_j - c_{i,j})^2 \chi_i(\mathbf{x}) d\mathbf{x}.$$  

The characteristic functions Eq. (20) can be defined in terms of $\beta_{i,j}$ Eq. (22), $\gamma_i(\mathbf{x})$ Eq. (17) and $G_{i,j}(\mathbf{x})$ Eq. (23) (Fig. 10)

$$\beta_{i,j} = \begin{cases} 
-1 & \text{if } \Omega_j \text{ lies within } C_j \\
1 & \text{if } \Omega_j \text{ lies outside } C_j 
\end{cases}$$

$$= 2 \left[ \frac{i \mod 2^{j+1}}{2^j} \right] - 1 \quad (22)$$

$$G_{i,j}(\mathbf{x}) = \begin{cases} 
1 & \text{if } \mathbf{x} \text{ and } \Omega_j \text{ lies on the same side of } C_j \\
0 & \text{else} 
\end{cases}$$

$$= \begin{cases} 
\frac{1 - \beta_{i,j}}{2} & \text{if } \beta_{i,j} = 1 \\
1 - \gamma_j(\mathbf{x}) & \text{if } \beta_{i,j} = -1 
\end{cases} \quad (23)$$

This way the characteristic functions $\chi_i$ can be conveniently decomposed into functions $G_{i,j}$ that depend on the location of one hypersurface $C_j$ only

$$\chi_i(\mathbf{x}) = \prod_{j=0}^{n-1} G_{i,j}(\mathbf{x}), \quad (24)$$

Fig. 9. Segmentation of the image domain $\Omega$ in eight subdomains $\Omega_0, \ldots, \Omega_7$ by three hypersurfaces $C_0, C_1, C_2$. The binary representation of the index $i$ of subdomain $\Omega_i$ can be found by the relative position to the hypersurfaces. Example: $\Omega_{01}$ lies outside (1) $C_0$, inside (0) $C_1$ and inside (0) $C_2$. 

For vector-valued images ($m > 1$), the Chan-Vese energy functional is defined by
Fig. 10. Values of the characteristic functions $\chi_i$ and the functions $G_{i,j}$ expressed in black (1) and white (0). If $\Omega_i$ lies inside (outside) of $C_j$, then $G_{i,j}$ is the characteristic function of the inside (outside) of $C_j$. The characteristic functions $\chi_i$ can be constructed by multiplying the characteristic functions $G_{i,j}$.

which will prove to be very useful in the derivation of speed functions for the hypersurfaces $C_j$.

The steps of the algorithm are to minimize the energy Eq. (21) by moving the hypersurfaces while keeping $\{c_0, \ldots, c_{2^n-1}\}$ fixed, and then recalculate $\{c_0, \ldots, c_{2^n-1}\}$ while keeping the hypersurfaces fixed. The latter can be done by setting the derivatives with respect to $c_{i,j}$ to zero:

$$\frac{\partial F_{CV}}{\partial c_{i,j}} = 0 \Rightarrow c_{i,j} = \frac{\int_{\Omega_i} u_j \chi_i dx}{\int_{\Omega_i} \chi_i dx}$$

In other words, $c_i$ is the average color of the image on subdomain $\Omega_i$. In case $|\Omega_i| = 0$, subdomain $\Omega_i$ is empty, it is not necessary to know $c_i$, since the subdomain does not contribute to the energy. For the former, an appropriate representation for the hypersurfaces must be chosen: level set functions.

Hypersurface $C_i$ is represented by the zero level set of the function $\phi_i : \Omega \rightarrow \mathbb{R}$. If the number of hypersurfaces $n$ is larger than one, the level set functions are combined in a vector:

$$\Phi : \Omega \rightarrow \mathbb{R}^n, \Phi = (\phi_0, \ldots, \phi_{n-1}).$$

$\{c_0, \ldots, c_{2^n-1}\}$ is written as a $2^n \times m$ matrix $c$ to shorten notation.

With $H$ the Heaviside function Eq. (34) and $\delta$ the Dirac function Eq. (35) (Appendix A3),

$$\text{surface}(C_i) = \text{surface}(\{x \in \Omega : \phi_i(x) = 0\}) = \int_{\Omega} |\nabla H(\phi_i)| dx = \int_{\Omega} \delta(\phi_i)|\nabla \phi_i| dx,$$

the energy Eq. (21) can be rewritten in terms of the level set functions

$$F_{CV}^{n,m}(\Phi, c) = \mu \sum_{i=0}^{n-1} \int_{\Omega} \delta(\phi_i)|\nabla \phi_i| dx.$$
\[ L_S = \mu \sum_{i=0}^{n-1} \int_{\Omega} \delta(\phi_i) \frac{\nabla \phi_i \cdot \nabla \Psi_i}{|\nabla \phi_i|} + \delta'(\phi_i)|\nabla \phi_i| |\Psi_i| \, dx \]

Finally, to get an expression without \( \nabla \Psi_i \), Green’s theorem

\[ \int_{\partial \Omega} n \cdot (g \Gamma) dS = \int_{\Omega} \nabla \cdot (g \Gamma) dx \]

\[ = \int_{\Omega} f \cdot (\nabla g) dx + \int_{\Omega} (\nabla \cdot f) g dx \]

can be used with \( f = \delta(\phi_i) \frac{\nabla \phi_i}{|\nabla \phi_i|} \) and \( g = \Psi_i \) to obtain

\[ L_S = \mu \sum_{i=0}^{n-1} \int_{\partial \Omega} \delta(\phi_i) \frac{\Psi_i}{|\nabla \phi_i|} \frac{\partial \phi_i}{\partial n} dS - \mu \sum_{i=0}^{n-1} \int_{\Omega} \delta(\phi_i) \frac{\nabla \phi_i}{|\nabla \phi_i|} |\nabla \Psi_i| \Psi_i dx \]

A similar expression for \( L_F \) can be found:

\[ \chi_i(\Phi + t \Psi) - \chi_i(\Phi) \]

\[ = \prod_{k=0}^{n-1} \left( \frac{1 - \beta_{i,k}}{2} + \beta_{i,k} H(\phi_k + t \Psi_k) \right) \]

\[ + \prod_{k=0}^{n-1} \left( \frac{1 - \beta_{i,k}}{2} + \beta_{i,k} H(\phi_k) \right) \]

\[ + t \Psi_k \delta(\phi_k) + O(t^2) \]

\[ - \prod_{k=0}^{n-1} \left( \frac{1 - \beta_{i,k}}{2} + \beta_{i,k} H(\phi_k) \right) \]

\[ = \prod_{k=0}^{n-1} \left( G_{i,k}(\Phi) + t \beta_{i,k} \Psi_k \delta(\phi_k) \right) \]

\[ - \prod_{k=0}^{n-1} G_{i,k}(\Phi) + O(t^2) \]

\[ = \sum_{k=0}^{n-1} t \beta_{i,k} \Psi_k \delta(\phi_k) \prod_{l=0}^{n-1} G_{i,l}(\Phi) + O(t^2) \]

so
The Euler-Lagrange equations can now be obtained by introducing an artificial time $t$ for the curve evolution
\[
\frac{\partial \phi_i}{\partial t} = \delta(\phi_i) \left( \mu \nabla \cdot \left( \frac{\nabla \phi_i}{|\nabla \phi_i|} \right) - \frac{1}{m} \sum_{k=0}^{2^{n-1}m-1} \sum_{j=0}^{m-1} \lambda_{k,j}(u_j - c_{k,j})^2 \beta_{k,j} \prod_{l \neq k} G_{k,l}(\Phi) \right)
\]
or
\[
\frac{\partial \phi_i}{\partial t} = \delta(\phi_i) \left( \mu \nabla \cdot \left( \frac{\nabla \phi_i}{|\nabla \phi_i|} \right) - \frac{\beta_{I,j}}{m} \sum_{j=0}^{m-1} (\lambda_{I,j}(u_j - c_{I,j})^2 - \lambda_{I,j}(u_j - c_{I,j})^2) \right)
\]
Introducing penalty functions $p_I : \Omega \to \mathbb{R}$ to express why points in $\Omega$ should not be in $\Omega_I$,
\[
p_I(x) = \frac{1}{m} \sum_{j=0}^{m-1} (\lambda_{I,j}(u_j - c_{I,j})^2),
\]
the Euler-Lagrange equations Eq. (33) become
\[
\frac{\partial \phi_i}{\partial t} = \delta(\phi_i) \left( \mu \nabla \cdot \left( \frac{\nabla \phi_i}{|\nabla \phi_i|} \right) - \beta_{I,j}(p_I - P_{I,j}) \right).
\]
To get an evolution equation for a contour moving in normal direction, $\delta(\phi_i)$ is replaced by $|\nabla \phi_i|$ [7]. In that case Eq. (6) is valid with speed function
\[
u_i = \beta_{I,j}(p_I - P_{I,j}) - \mu \nabla \cdot \left( \frac{\nabla \phi_i}{|\nabla \phi_i|} \right).
\]
So
\[
\frac{\partial \phi_i}{\partial t} = \nabla \phi_i \left( \mu \nabla \cdot \left( \frac{\nabla \phi_i}{|\nabla \phi_i|} \right) - \text{sign}(\phi_i)(p_I - P_{I,j}) \right)
\]
and Neumann boundary conditions
\[
\nabla \phi_i \cdot n = 0 \quad \forall i \in \{0, \ldots, n-1\} \text{ on } \partial \Omega.
\]
Appendix A2. Alternative form of the curvature term
Let $f_i = \frac{\partial f}{\partial x_i}, f_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$, then
\[
L_F = \frac{1}{m} \sum_{i=0}^{2^{n-1}m-1} \sum_{j=0}^{m-1} \lambda_{i,j} \int_{\Omega} (u_j - c_{i,j})^2
\]
Combining Eqs (26–28) and (29) results in
\[
\mu \sum_{i=0}^{n-1} \int_{\partial \Omega} \delta(\phi_i) \frac{\Psi_i}{|\nabla \phi_i|} \frac{\partial \phi_i}{\partial n} dS
\]
\[
- \mu \sum_{i=0}^{n-1} \int_{\partial \Omega} \delta(\phi_i) \nabla \cdot \left( \frac{\nabla \phi_i}{|\nabla \phi_i|} \right) \Psi_i d\Omega
\]
\[
+ \frac{1}{m} \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} \lambda_{i,j} \int_{\Omega} (u_j - c_{i,j})^2
\]
\[
\sum_{k=0}^{n-1} \Psi_k \delta(\phi_k) \beta_{i,k} \prod_{l \neq k} G_{i,l}(\Phi) d\Omega = 0
\]
for all test functions $\Psi_i, i \in \{0, \ldots, n-1\}$. Choosing all $\Psi_i$ equal to zero except for one $i \in \{0, \ldots, n-1\}$, Eq. (30) gets decoupled:
\[
\mu \int_{\partial \Omega} \delta(\phi_i) \frac{\Psi_i}{|\nabla \phi_i|} \frac{\partial \phi_i}{\partial n} dS - \mu \int_{\Omega} \delta(\phi_i) \nabla \cdot \left( \frac{\nabla \phi_i}{|\nabla \phi_i|} \right) \Psi_i d\Omega
\]
\[
\nabla \cdot \left( \frac{\nabla \phi_i}{|\nabla \phi_i|} \right) \Psi_i d\Omega + \frac{1}{m} \sum_{k=0}^{2^{n-1}m-1} \sum_{j=0}^{m-1} \lambda_{k,j} (u_j - c_{k,j})^2 \Psi_k \delta(\phi_i) \beta_{k,j} \prod_{l \neq k} G_{k,l}(\Phi) d\Omega = 0
\]
Choosing $\Psi_i$ equal to zero on $\Omega$ and non-zero on $\partial \Omega$ gives boundary conditions for $\Phi$
\[
\delta(\phi_i) \frac{\nabla \phi_i}{|\nabla \phi_i|} \cdot n = 0
\]
\[
\forall i \in \{0, \ldots, n-1\} \text{ on } \partial \Omega.
\]
As Eq. (32) is satisfied, Eq. (31) reduces to
\[
\delta(\phi_i) \left( \mu \nabla \cdot \left( \frac{\nabla \phi_i}{|\nabla \phi_i|} \right) - \frac{1}{m} \sum_{k=0}^{2^{n-1}m-1} \sum_{j=0}^{m-1} \lambda_{k,j} (u_j - c_{k,j})^2 \beta_{k,j} \prod_{l \neq k} G_{k,l}(\Phi) \right) = 0
\]
\[ \nabla \cdot \left( \nabla f \right) = \sum_{j=0}^{d-1} \frac{\partial}{\partial x_j} \left( f_j \sum_{i=0}^{d-1} f_i^2 \right) \]
\[ = \sum_{j=0}^{d-1} \left( f_j \left( \sum_{i=0}^{d-1} f_i^2 \right) - \frac{1}{2} f_j \sum_{i=0}^{d-1} f_i^2 \right) \]
\[ 2 \sum_{i=0}^{d-1} f_i f_{ij} = \sum_{j=0}^{d-1} \left( f_j \sum_{i=0}^{d-1} f_i^2 \right) \]
\[ - \frac{1}{2} f_j \left( \sum_{i=0}^{d-1} f_i^2 \right) 2 \sum_{i=0}^{d-1} f_i f_{ij} = \left( \sum_{i=0}^{d-1} f_i^2 \right) \]
\[ \sum_{j=0}^{d-1} \sum_{i=0}^{d-1} (f_i^2 f_{jj} - f_i f_{f_{ij}}) = \left( \sum_{i=0}^{d-1} f_i^2 \right) \]
\[ \left( \sum_{i=0}^{d-1} \sum_{j \neq i} (f_i^2 f_{jj} - f_i f_{f_{ij}}) - 2 \sum_{i=0}^{d-1} \sum_{j > i} (f_i f_j f_{ij}) \right) \]

So
\[ |\nabla f| \cdot \left( \frac{\nabla f}{|\nabla f|} \right) = \frac{\sum_{i=0}^{d-1} \sum_{j \neq i} (f_i^2 f_{jj} - 2 \sum_{i=0}^{d-1} \sum_{j > i} (f_i f_j f_{ij}))}{\sum_{i=0}^{d-1} f_i^2} \]

Let \( f = \nabla f \), \( F = \nabla^T f \).

then
\[ |\nabla f| \cdot \left( \frac{\nabla f}{|\nabla f|} \right) = \text{trace}(F) - \frac{f^T F f}{f^T f} \]

Appendix A3. Special functions and regularizations

Let the Heaviside function be defined by
\[ H(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \]

Let the Dirac function be defined by
\[ \delta(x) = 0 \quad \text{if } x \neq 0 \]
\[ \int_{-\infty}^{\infty} \delta(x) dx = 1 \]

Regularizations of \( H \) and \( \delta \) that were used in [6–8]:
\[ H_\epsilon(x) = \frac{1}{2} \left[ 1 + \frac{\arctan \left( \frac{x}{\epsilon} \right)}{\pi} \right] \]
\[ \delta_\epsilon(x) = \frac{1}{\pi} \frac{\epsilon}{\epsilon^2 + x^2} \]

Appendix A4. Taylor expansion

Let \( f(t) = |\nabla (\phi + t\Psi)| \)

Expand
\[ f(t) = f(0) + \left( \frac{df}{dt} \right)_{t=0} + O(t^2) \]

Since
\[ \frac{df}{dt} \bigg|_{t=0} = \frac{1}{2} \left( \frac{\partial}{\partial x_i} (\phi + t\Psi)^2 \right)_{t=0} = 2 \sum_{i=0}^{d-1} \frac{\partial}{\partial x_i} (\phi + t\Psi)^2 \]

it follows that
\[ |\nabla (\phi + t\Psi)| = |\nabla \phi| + t \frac{\nabla \phi \cdot \nabla \Psi}{|\nabla \phi|} + O(t^2) \]

References

[1] D. Adalsteinsson and J.A. Sethian, A fast level set method for propagating interfaces, Journal of Computational Physics 118 (269), (1995).
[2] K.R. Castleman, Digital image processing, Prentice Hall, New Jersey, 1996.
[3] T.F. Chan, B.Y. Sandberg and L.A. Vese, Active contours without edges for vector-valued images, Journal of Visual Communication and Image Representation 11(2) (2000), 130–141.
[4] T.F. Chan and L.A. Vese, Active contours without edges, IEEE Transactions on Image Processing 10(2) (2001), 266–277.
[5] T.F. Chan and L.A. Vese, Variational image restoration & segmentation models and approximations, UCLA CAM report (1997), 97–47.
[6] M. Leeser, K-means algorithms for unsupervised classification, http://www.ece.neu.edu/groups/rpl/projects/kmeans/, 1999.
[7] S. Osher and R.P. Fedkiw, Level set methods and dynamic implicit surfaces, Springer, New York, 2003.
[8] D. Pelleg and A. Moore, Accelerating exact k-means algorithms with geometric reasoning, International Conference on Knowledge Discoveries and Data Mining (1999).
[9] D. Pelleg and A. Moore, X-means: Extending K-means with Efficient Estimation of the Number of Clusters, International Conference on Machine Learning, 2000.
[10] J.R. Rommelse, High performance algorithms in image segmentation, MSc thesis, Delft University of Technology, 2002.

[11] J.R. Rommelse, H.X. Lin and T.F. Chan, A robust level set algorithm for image segmentation and its parallel implementation, DCABES2002, Wuxi, China, 2002.

[12] G. Russo and P. Smereka, A remark on computing distance functions, Journal of Computational Physics 163(1) (2000), 51–67.

[13] J.A. Sethian, Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision and materials science, Cambridge University Press, Cambridge, 1999.

[14] B. Song and T.F. Chan, A fast algorithm for level set based optimization, UCLA CAM report, 2002, 02-68.

[15] R. Tsai, Rapid and accurate computation of the distance function using grids, Journal of Computational Physics 178(1) (2001), 175–195.

[16] L.A. Vese and T.F. Chan, A multiphase level set framework for image segmentation using the Mumford and Shah model, International Journal of Computer Vision 50(3) (2002), 271–293.

[17] L.A. Vese and T.F. Chan, Reduced non-convex functional approximations for image restoration & segmentation, UCLA CAM report 1997, 97–56.

[18] W.L. Wan, Scalable and multilevel iterative methods. UCLA CAM report, 1998, 98-29.

[19] High performance applied computing, http://www.hpcn.tudelft.nl/, 2000.

[20] The distributed ASCI supercomputer 2 (DAS-2), http://www.cs.vu.nl/das2/, 2002.
Submit your manuscripts at http://www.hindawi.com