Level Set Segmentation using Coupled Active Surfaces

Release 0.00

Kishore Mosaliganti[1], Benjamin Smith[2], Arnaud Gelas[1], Alexandre Gouaillard[1] and Sean Megason[1]

March 13, 2009

[1]Systems Biology, Harvard Medical School, Boston, MA-02139, USA
[2]GrUVi Lab, MIAL, School Of Computing Science, Simon Fraser University, Burnaby, V5A 1S6

Abstract

An Insight Toolkit (ITK) processing framework for simultaneous segmentation of multiple objects using active contours without edges is presented in this paper. These techniques are also popularly referred to as multiphase methods. Earlier, we had an implemented the Chan and Vese [1] algorithm that uses level-sets to accomplish region segmentation in images with poor or no gradient information. The current work extends that submission to use multiple level sets that evolve concurrently. The basic idea is to partion the image into several sets of piecewise constant intensity regions. This work is in contrast to the level-set methods currently available in ITK which necessarily require gradient information and also necessarily segment a single object-of-interest. Similar to those methods, the methods presented in this paper are also made efficient using a sparse implementation strategy that solves the contour evolution PDE at the level-set boundary. This work does not introduce any new filter but extends the earlier submitted to filters to process multiple objects. We include 2D/3D example code, parameter settings and show the results generated on a 2D cardiac image.

Contents

1 Introduction **2**

2 Description: Multiphase Level Sets **2**

3 Implementation **3**

4 Memory optimizations using kd-trees **4**

5 Usage **5**
5.1 Dense Option 5
5.2 Sparse Option 5

6 Results **7**

1 Introduction

In image analysis, we are often interested in segmenting more than a single object (of the same or different kind) from a given image. This especially happens when the objects to be segmented are adjacent to each other and the delineation of one object automatically affects the neighboring object. In such situations, it makes sense to concurrently process their segmentation in order to optimally segment the objects. As a simple example, in microscopy image analsis, there is a significant interest in segmenting nuclei or cells as shown in Figure 1. Each image is acquired at high resolution and could countain thousands of cells. These cells often cluster in regions and appear to overlap. The challenge is to split these cells into individual components. An iterative (or linear) cell extraction procedure using level-sets can cause inconsistent splits since each level-set functions does not compete with the neighboring cells. There could be an overlap of the level-set functions. Hence, in such cases, it is imperative to use multiphase methods for segmentation.

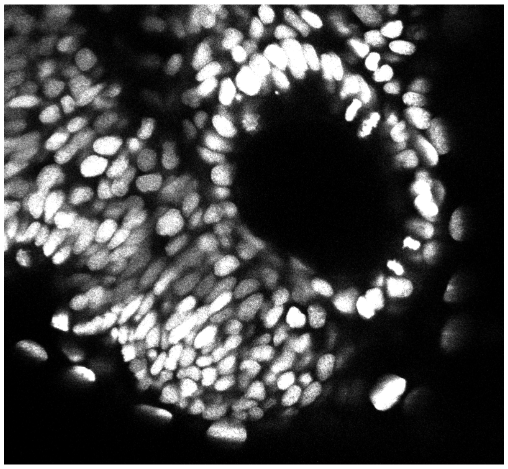


Figure 1: Multiple cells in close contact and in the same field of view.

There are several research papers in literature devoted to multiphase methods that optimize the number of level-set functions used for a generic case of *N* phases or objects. While computationally, this is optimal, it is not the most robust choice. We are largely motivated by microscopy applications where we would like to segment and track cells and place constraints on the area, volume and shapes of each individual cell. Each cell has a unique fluoroscence intensity level. Hence, in these circumstances, we deemed it best to have a unique level set function per cell. In this implementation, we extend the implementation of the earlier submission on Chan and Vese method to multiphase methods.

2 Description: Multiphase Level Sets

The basic idea of active contour models relies on detecting salient objects by evolving a curve or a surface subject to image-based constraints. Let $\Omega \subset \mathbb{R}^d$ be the image domain and $I : \Omega \to \mathbb{R}$ be a given image function. Further, let $\phi : \Omega \to \mathbb{R}$ be a signed distance function which represents the level set function that is negative in the object of interest and positive outside.

Chan and Vese [1] proposed an active contour model for segmenting images containing objects that have poor boundaries. They proposed an energy that is an piece-wise constant approximation to the Mumford and Shah functional:

$$F^{CV}(C,c_1,c_2) = \lambda_1 \int_{\Omega_{in}} |I(x)-c_1|^2 dx + \lambda_2 \int_{\Omega_{out}} |I(x)-c_2|^2 dx + \mu.Area(C) + \nu.Volume(C) \tag{1}$$

where Ω_{in} and Ω_{out} represent the region inside and outside the contour C, respectively, and c_1 and c_2 are two scalar constants that approximate the image intensities. The first two terms are often referred as global binary fitting energy terms that seek to separate an image into two regions of constant image intensities. By using a level-set formulation, the minimization problem can be converted to a level-set evolution equation.

In the multiphase case, we have N level set functions $\{\phi_1, \cdots, \phi_n\}$ and scalar intensity constants $\{c_1, \cdots, c_n\}$ respectively and c_0 represents the background intensity. The N parameters $\{\lambda_{1,1}, \cdots, \lambda_{1,n}\}$ are scalar weights of the individual object intensity fitting terms and λ_2 is the weight for the background intensity fitting term.

The formulation is easily arrived at in the following way. Consider extending the simple Chan-Vese Equation 1 to the N objects. This accounts for the summation term with the subscript i. The first term represents the i-th foreground intensity fitting with scalar constant c_i and weighted by $\lambda_{1,i}$. The second term is the background intensity fitting with scalar constant c_0 and weighted by λ_2. Note that the background is characterized by a product of the inverse Heaviside functions of the N foregrounds. The third and fourth terms represent the length and area regularization terms for the N level sets. Finally, the last term represents the overlap penalty function. This term penalizes the level set functions in regions where they overlap and γ represents the scalar penalty constant.

$$\begin{aligned} F^{multi}(\phi_1,\cdots,\phi_n,c_0,c_1,\cdots,c_n) &= \int_\Omega \sum_{i=1}^{n} \Big[\lambda_{1,i}\big(I(x)-c_i\big)^2 H(\phi_i) + \frac{\lambda_2}{n} \prod_{j\neq i} \big(1-H(\phi_j)\big)\big(I(x)-c_0\big)^2 \\ &+ \mu|\nabla\phi_i|\delta(\phi_i) + \nu H(\phi_i)\gamma \sum_{i<j} H(\phi_i)H(\phi_j) \Big] dx \end{aligned} \tag{2}$$

Using standard mathematics, we arrive at the Euler-Langrange equation for deriving the update equation for the level sets. For the i-th level set function:

$$\frac{\partial \phi_i}{\partial t} = \delta_e(\phi_i)\Big[-\lambda_{1,i}(I(x)-c_i)^2 + \lambda_2 \prod_{j\neq i} H(\phi_j)(I-c_0)^2 + \mu div(\frac{\nabla\phi_i}{|\nabla\phi_i|}) - \nu - \gamma\sum_{j\neq i} H(\phi_j)\Big] \tag{3}$$

3 Implementation

The dense and sparse filter options affect the computational performance of the method. However, they both solve the same underlying equation. Hence, the parameter settings remain the same. We now describe each of the parameters, their range and typical values. There is no typical limit that can be set on most parameters but depends on experimentation. Note that except for the first three, the remaining constitute weights to the different energy terms. Depending on their contribution to the overall energy, these weights need to be modified so that all the terms have an influence.

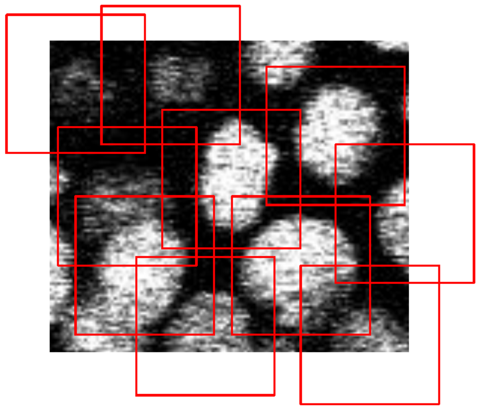
`m_Iterations` - Maximum permitted iterations of the evolution in the range $[0, \infty]$. Typical value depends on the initialization. The initialization must not be too different from the final output for best results. It is usually set by trial and error and 100 iterations are usually sufficient.
`m_MaxRMSChange` - Maximum change in the level-set function averaged over all the pixels in the range $[0, \infty]$. During convergence, a low RMS change indicates that the level-set function does not change position anymore. Usually a value between 0.1 is sufficient.
`m_Epsilon` - Defines the smoothness of the Heaviside and Delta functions defined in Equation 3. Usually in the range $[1, \infty]$. We set the value to 1 in our images and obtain good results. For very high resolution images, this value can be changed to 2 or 3.
`m_Mu` - The weight of the length regularization in the energy function and lies in the range [0,∞]. The exact specification depends on the images and the expected length of the i-th segmentation boundary. It can be as high as 10000.
`m_Nu` - The weight of the area regularization in the energy function and lies in the range [0,∞]. The exact specification depends on the images and the expected length of the i-th segmentation boundary. It can be as high as 10000.
`m_Lambda1` - Weights of the sum of squares of the zero mean intensities inside the contour i and lies in the range [0,inf]. Usually set as 1 and other parameters are decided based on this normalized value.
`m_Lambda2` - Weights of the sum of squares of the zero mean intensities outside all the contours and lies in the range [0,inf]. Usually set as 1 and other parameters are decided based on this normalized value.
`m_Gamma` - Penalty weight for overlap regions of the level set functions and lies in the range [0,∞].

4 Memory optimizations using kd-trees

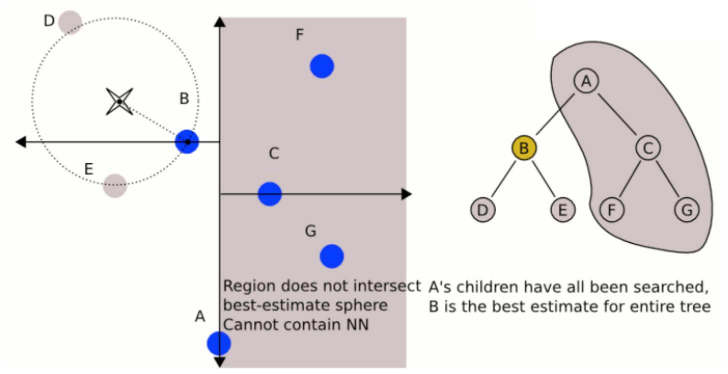
Computationally, it is memory intensive to have N level set functions defined on the image domain. For a large image with many small objects (such as cells in microscopy images), it becomes an intractable problem. Hence, we make the implementation robust by defining region-of-interest (ROI), using spatial data structures such as the kd-trees and cached lookup tables.

Each level set function (`itk::Image`) is first defined in a region-of-interest (ROI) within the image domain Ω. This is illustrated in Figure 2(a). The ROI should encompass the object to be segmented and its extent is specified by the attributes origin and size. The spacing is the same as the feature or raw intensity image. This saves us considerable computational memory space. The centroid of each ROI region is then placed in a kd-tree structure. In the update Equation 3 for each level set function, the overlaps of ROI regions are calculated by querying the kd-tree for the k-nearest neighbors ($k = 10$) as illustrated in Figure 2(b). This saves us considerable computational time.

Note that there is a cost associated with building the kd-tree that can be avoided for a small number of phases. We only instantiate the kd-tree mechanism of search when there are more than 20 phases involved.



(a)



D
B
E
F
C
G
A
Region does not intersect best-estimate sphere Cannot contain NN
A's children have all been searched, B is the best estimate for entire tree
A
B
C
D
E
F
G

(b)

Figure 2: (a) ROI defined around individual cells. (b) *k*d-tree structure constructed from ROI centroid.

5 Usage

We begin by including the appropriate header files for the solver (both dense and sparse) and function. Depending on the choice, either dense or sparse is required. We then define level-set image type and the feature image type. Note that internally, these image types are cast into float type.

```
#include "itkScalarChanAndVeseLevelSetFunction.h"
#include "itkSparseMultiphaseLevelSetImageFilter.h"

...
int main(int argc, char *argv[])
{
  ...
  unsigned int Dimension = 2;
  typedef float ScalarPixelType;
  typedef itk::Image< ScalarPixelType, Dimension > LevelSetImageType;
  typedef itk::Image< ScalarPixelType, Dimension > FeatureImageType;
```

5.1 Dense Option

If the dense option is selected, the following typedef are required.

```
typedef itk::ScalarACWOEdgesLevelSetFunction< LevelSetImageType,
  FeatureImageType > LevelSetFunctionType;
typedef itk::MultiLevelSetImageFilterDense< LevelSetImageType,
  FeatureImageType, LevelSetFunctionType, float > MultiLevelSetType;
```

5.2 Sparse Option

```
typedef itk::ScalarACWOEdgesLevelSetFunction< LevelSetImageType,
  FeatureImageType > LevelSetFunctionType;
typedef itk::MultiLevelSetImageFilter< LevelSetImageType,
  FeatureImageType, LevelSetFunctionType, float > MultiLevelSetType;
```

After initialization, it is imperative that the user specify the number of level-set functions and set the feature image and initialization for each level set function. We illustrate our example for $N = 3$.

```
  MultiLevelSetType::Pointer levelSetFilter = MultiLevelSetType::New();
  levelSetFilter->SetFunctionCount( 3 );
  levelSetFilter->SetFeatureImage( featureImage );

  levelSetFilter->SetLevelSet( 0, contourImage1 );
  levelSetFilter->SetLevelSet( 1, contourImage2 );
  levelSetFilter->SetLevelSet( 2, contourImage3 );
```

Appropriate global settings of the level set include the number of iterations, maximum permissible change in RMS values and whether to use image spacing.

```
  levelSetFilter->SetNumberOfIterations( atoi( argv[2] ) );
  levelSetFilter->SetMaximumRMSError( atof( argv[3] ) );
  levelSetFilter->SetUseImageSpacing( 0 );
```

Using a for-loop over all the level set functions, we call the *i*-th difference function (`levelSetFilter->GetTypedDifferenceFunction(i)`) and set the corresponding attributes of that level set function.

```
  for ( unsigned int i = 0; i < 3; i++ )
  {
    levelSetFilter->GetTypedDifferenceFunction(i)->SetEpsilon(
      atof( argv[4] ) );
    levelSetFilter->GetTypedDifferenceFunction(i)->SetMu( atof( argv[5] ) );
    levelSetFilter->GetTypedDifferenceFunction(i)->SetNu( atof( argv[6] ) );
    levelSetFilter->GetTypedDifferenceFunction(i)->SetLambda1(
      atof( argv[7] ) );
    levelSetFilter->GetTypedDifferenceFunction(i)->SetLambda2(
      atof( argv[8] ) );
    levelSetFilter->GetTypedDifferenceFunction(i)->SetGamma( atof( argv[9] ) );
    levelSetFilter->GetTypedDifferenceFunction(i)->SetTau( atof( argv[11] ) );
    levelSetFilter->GetTypedDifferenceFunction(i)->SetVolume( atof( argv[12] ));
  }
```

The output consists of the N level set functions that can be accessed by a for-loop.

```
  for ( unsigned int i = 0; i < 3; i++ )
  {
    WriterType::Pointer writer = WriterType::New();
    writer->SetInput( levelSetFilter->GetLevelSet( i ) );
    writer->SetFileName( argv[16+i] );

    try
    {
      writer->Update();
    }
    catch( itk::ExceptionObject & excep )
    {
      std::cerr << "Exception caught !" << std::endl;
      std::cerr << excep << std::endl;
      return -1;
    }
  }
```

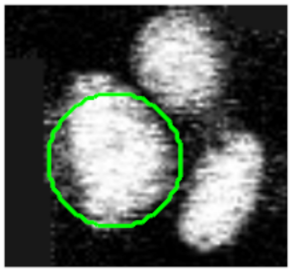
6 Results

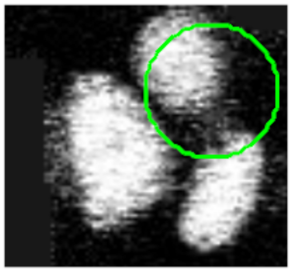
The results in this example can be obtained by using `ScalarMultiPhase2DTest.cxx` on the input image `MultiphaseCells2D.png` and initial level set images `Multiphase2Dphi1.mha`. In this example, three initial contours (circles), are evolved to segment three adjacent cells. The image can be assumed to consist of four regions of constant intensities and hence the Chan and Vese method with the multiphase extension can be applied. The circles are embedded in a distance map and evolved. The output is written out to the image `Multiphase2DSeg1.mha`, `Multiphase2DSeg2.mha` and `Multiphase2DSeg3.mha` respectively. The parameters to the filter are set at the command line to facilitate easy modification and exploration by the user. The command to run this particular executable is as follows:

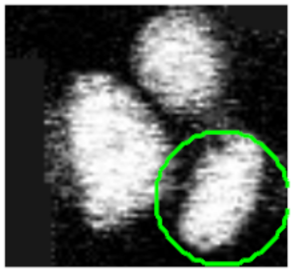
```
./ScalarSinglePhase2DTest featureImage Iterations rms epsilon mu nu Lambda1 Lambda2 contourImage1 outpu

./ScalarMultiPhase2DTest MultiphaseCells2D.png 10 0 1 0 0 1 1 4000 0 Multiphase2Dphi1.mha Multiphase2Dp
```

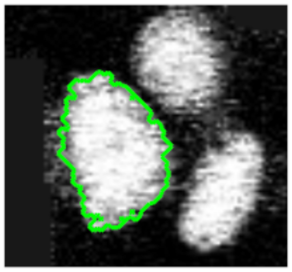
References

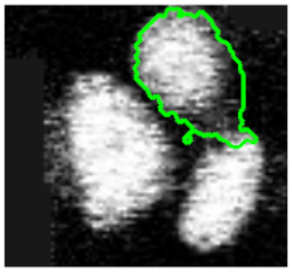
[1] T. Chan and L. Vese. An active contour model without edges. In *Scale-Space Theories in Computer Vision*, pages 141–151, 1999.

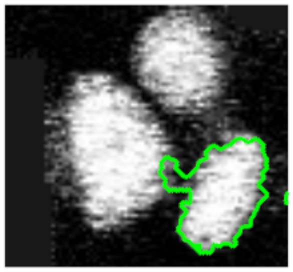

(a)

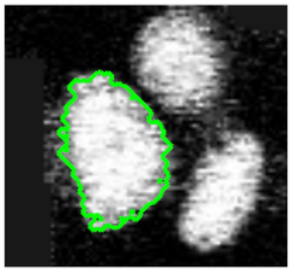

(b)

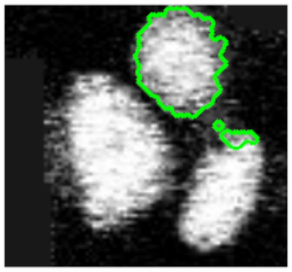

(c)


(d)


(e)


(f)


(g)


(h)

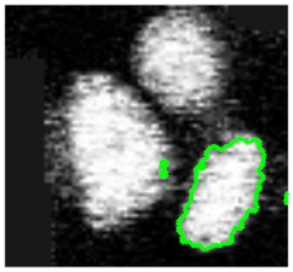

(i)

Figure 3: Parameters: $\varepsilon = 1$, $\mu = 0$, $\nu = 0$, $\lambda_{1,i} = 1$, $\lambda_2 = 1$ and $\Gamma = 4000$. The 3 phases at iterations: (a-c) 0 (d-f) 10 (g-i) 24.