Image Segmentation Using the Chan-Vese Algorithm

Robert Crandall

ECE 532 Project
Fall, 2009

1 Introduction

One of the most important and ubiquitous tasks in image analysis is segmentation. Segmentation is the process of partitioning an image into a set of distinct regions, which are different in some important qualitative or quantitative way. This is a critical intermediate step in all high level object-recognition tasks. For example, if we want to locate the face of a particular person in an image of a crowd, we must first determine which parts of the image correspond to human faces. Other common segmentation tasks include segmenting written or typset characters on a page, segmenting tumors from healthy brain tissue in an MRI image, etc.

In this paper we will introduce a method for segmenting images that was developed by Chan and Vese in [1]. This is a powerful, flexible method that can successfully segment many types of images, including some that would be difficult or impossible to segment with classical thresholding or gradient-based methods. We will outline the derivation of this method, present an algorithm for segmenting images, and examine performance on some example images.

2 Geometric Models and Level Set Methods

The Chan-Vese algorithm is an example of a geometric active contour model. Such models begin with a contour in the image plane defining an initial segmentation, and then we evolve this contour according to some evolution equation. The goal is to evolve the contour in such a way that it stops on the boundaries of the foreground region. There are various ways to define the evolution equation; for example, the contour might move with a velocity that depends on the local curvature at a given point or the image gradient at that point. The Chan-Vese algorithm evolves this contour via a level set method, discussed below.

2.1 Level Set Methods

Level set methods are a powerful tool for performing contour evolution. We define some

function $\phi(i, j, t)$ (the level-set function), where (i, j) are coordinates in the image plane and t is an artificial "time." At any given time, the level set function simultaneously defines an edge contour and a segmentation of the image. The edge contour is taken to be the zero level set $\{(i, j) \, s.t. \, \phi(i, j, t) = 0\}$, and the segmentation is given by the two regions $\{\phi \geqslant 0\}$ and $\{\phi < 0\}$. The level set function will be evolved according to some partial differential equation, and hopefully will reach a steady state $\lim_{t \to \infty} \phi$ that gives a useful segmentation of the image.

As a simple illustration of this concept, suppose we have a 100×100 image, and by some means we determine a level-set function to be $\phi(i, j) = (x - 50)^2 + (y - 50)^2 - 600$.
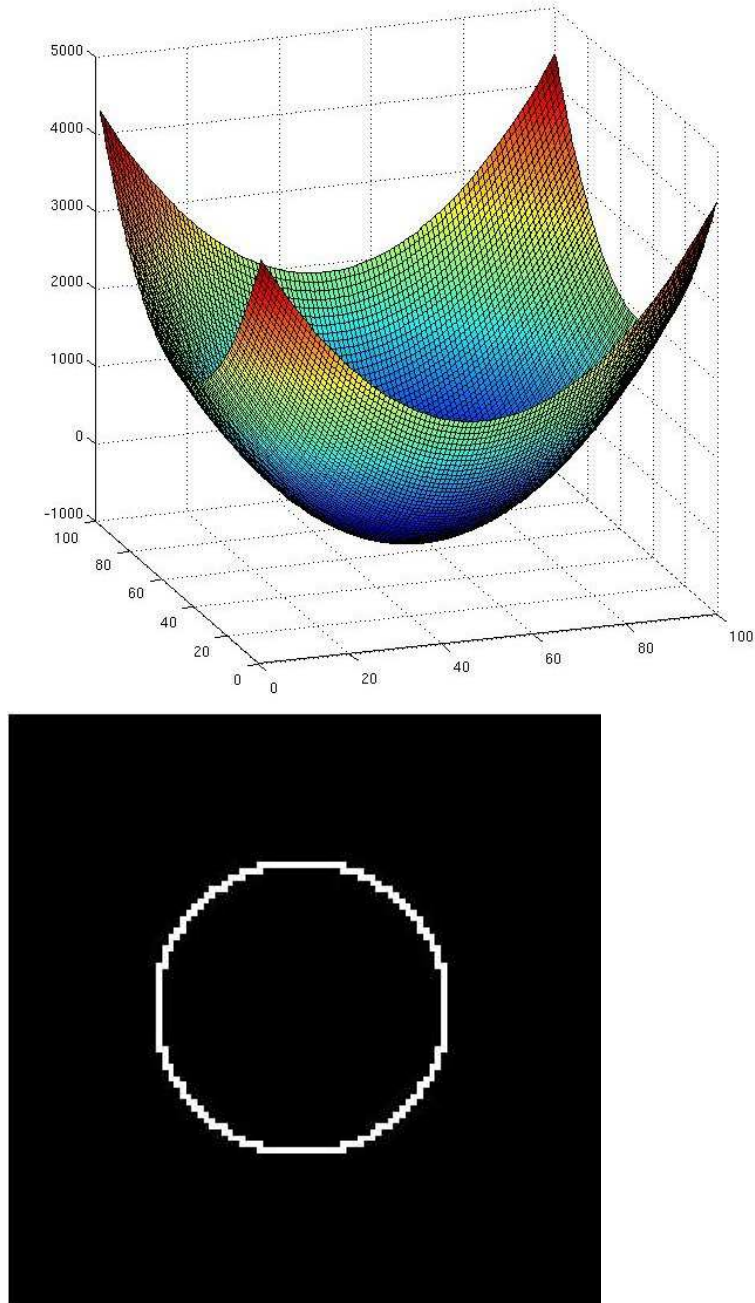

Figure 1. (top) Surface plot of an example level set function $f(x, y)$ and (bottom) its zero level contour in the image plane.

If we define the foreground to be the region where $\phi < 0$, then the foreground found by this segmentation would be the region inside the circle. Level set methods are especially useful because they can easily handle topological changes in the edge contour that would be difficult to handle with a model that directly evolves the contour. Since this contour is computed indirectly as the zero contour of a surface in a higher dimension, there are no computational issues when this contour splits from, say, a single circle into two distinct closed curves; the level set function is represented by a smooth surface in either case.

The most important (and most difficult) step is to actually determine a level set function that segments the image in a meaningful way. The simplest useful example would be to define the level set function to be the value of a gray level image at each pixel minus some threshold, i.e. set $\phi(i,j) = I(i,j) - t$. Then the level set function is positive in regions where the gray level is above the threshold, and negative in regions where the gray level is below the threshold. Of course, the level set formalism is not necessary for this simple example, but it illustrates the basic idea.

2.2 Variational Level Set Methods

The Chan-Vese algorithm uses variational calculus methods to evolve the level set function. Variational methods work by seeking a level set function that minimizes some functional. In this context, by functional we mean a mapping that takes a level set function ϕ as input, and returns a real number. The problem is then to seek a function ϕ that is a critical point (minimum or maximum) of this functional. The hope is that we can come up with a functional whose critical points are level sets that give useful segmentations for a given problem.

As a toy example, suppose we have a bi-level image I with domain Ω that takes on only the values -1 and 1. Define a functional F such that, for any function $\phi \colon \Omega \longrightarrow \mathbb{R}$,

$$F[\phi] = \int_\Omega H(\phi)\, I,$$

where H is the Heaviside function $H(x) = \begin{cases} 1, x \geqslant 0 \\ 0, x = 0 \end{cases}$. In the practical, discrete-image case the integral is interpreted as a sum over all the image pixels. The value of this functional is simply the integral of I over all the pixels where ϕ is positive. Ignoring for the moment how we might determine a minimizer for this functional computationally, it is intuitively obvious that the minimizers are precisely those functions ϕ that are positive only where $I = -1$.

Coming up with an appropriate functional, and determining a way to find a minimizer of that functional, is a difficult problem. In the rest of this paper we will focus on one such functional that works well for a large class of images, and describe how it is used in practice to perform image segmentation.

3 The Chan-Vese Segmentation Algorithm

3.1 The Fitting Energy Functional

Having introduced the necessary concepts, we come now to the main algorithm explored in this paper. We start by introducing the functional that forms the basis for the Chan-Vese algorithm, which we will refer to as the "fitting energy" functional [1]. The goal of the segmentation algorithm will be to minimize this fitting energy for a given image, and the minimizing level set function ϕ will define the segmentation. In its most general form, the fitting energy is

$$F(\phi) = \mu\left(\int_\Omega |\nabla H(\phi)|\,d\,x\right)^p + \nu \int_\Omega H(\phi)d\,x$$

$$+ \lambda_1 \int_\Omega |I - c_1|^2 H(\phi)d\,x + \lambda_2 \int_\Omega |I - c_2|^2(1 - H(\phi))\,d\,x.$$

μ, ν, λ_1, λ_2 and p are parameters selected by the user to fit a particular class of images. Note that this is a generalization of the Mumford-Shah functional introduced in [2]; the Mumford-Shah functional is obtained by setting $p = 1$, $\nu = 0$, and $\lambda_1 = \lambda_2 = 1$. Here H is the Heavyside function, I is the image to be segmented, and Ω is the domain of that image. c_1 and c_2 are the averages of the image I in the regions where $\phi \geqslant 0$ and $\phi < 0$, respectively, given by

$$c_1 = \frac{\int_\Omega I \cdot H(\phi)\,d\,x\,d\,y}{\int_\Omega H(\phi)\,d\,x\,d\,y}, c_2 = \frac{\int_\Omega I \cdot (1 - H(\phi))\,d\,x\,d\,y}{\int_\Omega (1 - H(\phi))\,d\,x\,d\,y}.$$

The first term, $\mu\left(\int_\Omega |\nabla H(\phi)|\right)^p$, can be thought of as a penalty on the total length of the edge contour for a given segmentation. If we expect a region with a smooth boundary, we might weight this term more heavily to avoid finding a complex (and therefore long) perimeter. Similarly, the term $\nu \int_\Omega H(\phi)d\,x$ is a penalty on the total area of the foreground region found by the segmentaion.

The third term, $\lambda_1 \int_\Omega |I - c_1|^2 H(\phi)d\,x$, is proportional to the variance of the image gray level in the foreground region and measures how "uniform" the region is in terms of pixel intensity. The fourth term does the same for the background region. Minimizing the sum of these two terms, then, leads to a segmentation into a foreground and background region that are each as uniform as possible. For example, in a bilevel image where pixels can take on only two values, the sum of these terms is minimized by taking a segmentation that includes all pixels of the first value in the foreground region, and no pixels of the second value. Usually, we take $\lambda_1 = \lambda_2 = 1$, but we can adjust them as necessary to weight one term more heavily. If we set $\lambda_1 = 2, \lambda_2 = 1$, then our final segmentation will have a more uniform foreground region (since the energy contributed by the "variance" in the foreground region is weighted more heavily), at the expense of allowing more variation in the background. In applications where we expect, say, an approximately black background and foreground objects with varying gray levels, we would want to set $\lambda_1 < \lambda_2$.

3.2 The Variational PDE

In [1] the Euler-Lagrange equations and the gradient-descent method are used to derive the following evolution equation for the level set function ϕ that will minimize the fitting energy $F(\phi)$ ($\phi_t := \frac{d}{d\,t}\phi$ where t is an artificial "time"):

$$\phi_t = \delta(\phi)\left[\mu\,p\left(\int_\Omega \delta(\phi)|\nabla\phi|\right)^{p-1}\mathrm{div}\left(\frac{\nabla\phi}{|\nabla\phi|}\right) - \nu - \lambda_1(I - c_1)^2 p + \lambda_2(I - c_2)^2\right].$$

In order to solve this PDE numerically, we must discretize it. Let $\phi_{i,j}^n$ denote the value of the level set function ϕ at pixel (i, j) at iteration n. As in [1], we will use the following notation for the spatial finite differences (with an implied pixel spacing of $h = 1$ unit):

$$\Delta_+^x \phi_{i,j}^n = \phi_{i+1,j}^n - \phi_{i,j}^n, \Delta_-^x \phi_{i,j}^n = \phi_{i,j}^n - \phi_{i-1,j}^n$$

$$\Delta_+^y \phi_{i,j}^n = \phi_{i,j+1}^n - \phi_{i,j}^n, \Delta_-^y \phi_{i,j}^n = \phi_{i,j}^n - \phi_{i,j-1}^n$$

For the "time" direction we will use the forward difference $\frac{\phi_{i,j}^{n+1} - \phi_{i,j}^n}{\Delta t}$.

The linearized, discretized partial differential equation that we must solve to minimize the fitting energy is given by [1] as

$$\frac{\phi_{i,j}^{n+1} - \phi_{i,j}^n}{\Delta t} =$$

$$\delta_h(\phi_{i,j}^n)\frac{\mu}{h^2}(p \cdot L(\phi^n)^{p-1})\left[\Delta_-^x\left(\frac{\Delta_+^x \phi_{i,j}^{n+1}}{\sqrt{(\Delta_+^x \phi_{i,j}^n)^2/h^2 + (\phi_{i,j+1}^n - \phi_{i,j-1}^n)^2/(2h)^2}}\right)\right.$$

$$\left.+ \Delta_-^y\left(\frac{\Delta_+^y \phi_{i,j}^{n+1}}{\sqrt{(\phi_{i+1,j}^n - \phi_{i-1,j}^n)^2/(2h)^2 + (\Delta_+^y \phi_{i,j}^n)^2/h^2}}\right)\right]$$

$$- \delta_h(\phi_{i,j}^n)\left(\nu + \lambda_1(I_{i,j} - c_1(\phi^n))^2 - \lambda_2(I_{i,j} - c_2(\phi^n))^2\right).$$

δ_h is a smoothed version of the delta function, given by

$$\delta_h(x) = \frac{1}{\pi}\frac{h}{h^2 + x^2}.$$

$L(\phi^n)$ is the perimeter length of the zero level set of ϕ^n, computed as

$$L(\phi^n) = \int_\Omega \delta_h(\phi^n)|\nabla\phi^n|\,dx\,dy.$$

We define the following constants to simplify the final expression:

$$C_1 = \frac{1}{\sqrt{(\phi_{i+1,j}^n - \phi_{i,j}^n)^2 + (\phi_{i,j+1}^n - \phi_{i,j-1}^n)^2/4}}$$

$$C_2 = \frac{1}{\sqrt{\phi_{i,j}^n - \phi_{i-1,j}^n)^2 + (\phi_{i-1,j+1}^n - \phi_{i-1,j-1}^n)^2/4}}$$

$$C_3 = \frac{1}{\sqrt{(\phi_{i+1,j}^n - \phi_{i-1,j}^n)^2/4 + (\phi_{i,j+1}^n - \phi_{i,j}^n)^2}}$$

$$C_4 = \frac{1}{\sqrt{(\phi_{i+1,j-1}^n - \phi_{i-1,j-1}^n)^2/4 + (\phi_{i,j}^n - \phi_{i,j-1}^n)^2}}.$$

Expanding out the finite difference operators and substituting in these constants, the equation becomes

$$\phi_{i,j}^{n+1}\left[1 + \frac{\Delta t}{h}\delta_h(\phi_{i,j}^n)\mu(p \cdot L(\phi^n)^{p-1})(C_1 + C_2 + C_3 + C_4)\right]$$

$$= \phi_{i,j}^n + \frac{\Delta t}{h}\delta_h(\phi_{i,j}^n)\mu\,(p \cdot L(\phi^n)^{p-1})\left[C_1\phi_{i+1,j}^{n+1} + C_2\phi_{i-1,j}^{n+1} + C_3\phi_{i,j+1}^{n+1} + C_4\phi_{i,j-1}^{n+1}\right]$$

$$- \Delta t\,\delta_h(\phi_{i,j}^n)\left[\nu + \lambda_1(I_{i,j} - c_1(\phi^n))^2 - \lambda_2(I_{i,j} - c_2(\phi^n))^2\right].$$

3.3 Solving the Variational PDE

To solve this equation for ϕ^{n+1}, we use the iterative method presented in [3]. To maintain consistency with the notation in [3], we introduce more constants and rewrite our equation as

$$\phi_{i,j}^{n+1} = F_1\phi_{i+1,j}^{n+1} + F_2\phi_{i-1,j}^{n+1} + F_3\,\phi_{i,j+1}^{n+1} + F_4\phi_{i,j-1}^{n+1} + Fp_{i,j},$$

where

$$F_i = \frac{\Delta t\delta_h(\phi_{i,j}^n)\mu(p \cdot L(\phi^n)^{p-1})\,C_i}{h + \Delta t\delta_h(\phi_{i,j}^n)\mu(p \cdot L(\phi^n)^{p-1})(C_1 + C_2 + C_3 + C_4)}\quad\text{for } i = 1, 2, 3, 4,$$

$$F = \frac{h}{h + \Delta t\,\delta_h(\phi_{i,j}^n)\mu(p \cdot L(\phi^n)^{p-1})(C_1 + C_2 + C_3 + C_4)},$$

and

$$p_{ij} = \phi_{i,j}^n - \Delta t \delta_h(\phi_{i,j}^n)\left[\nu + \lambda_1(I_{i,j} - c_1(\phi^n))^2 - \lambda_2(I_{i,j} - c_2(\phi^n))^2\right].$$

We can then compute ϕ^{n+1} as in Proposition 6.1 of [3] as follows (here M and N are the image row and column dimensions respectively):

Algorithm
```
for i = 1:M
  for j = 1:N
    Compute F_1, F_2, F_3, F_4, and F, replacing φ^n_{i-1,j}, φ^n_{i,j-1} with φ^{n+1}_{i-1,j}, φ^{n+1}_{i,j-1}
    when computing C_i
    Set φ^{n+1}_{i,j} = F_1 φ^n_{i+1,j} + F_2 φ^{n+1}_{i-1,j} + F_3 φ^n_{i,j+1} + F_4 φ^{n+1}_{i,j-1} + F p_{i,j}.
```

Note that at each step we are only using values of ϕ^{n+1} that were computed previously, so this method is explicit and can be implemented easily without need for a general PDE or ODE solver.

3.4 Reinitializing the Level Set Function

It is necessary at each iteration to rescale the level set function to keep it from becoming too flat. This is necessary because of "blurring" that occurs due to our use of the smoothed delta nction δ_h. The procedure for performing this reinitialization is outlined in [1], and described in more detail in [4]. The idea is to set $\psi(0) = \phi$, and then evolve ψ according to

$$\psi_t = \mathrm{sign}(\phi(t))(1 - |\nabla\psi|).$$

The steady state of this evolution will be the signed distance function to the zero level contour of ϕ; that is, ψ will have the same sign as ϕ at each point, and the magnitude at a point will be the distance from that point to the contour $\{\phi = 0\}$. We then reinitialize ϕ to ψ.

Another iterative scheme is used to solve this equation numerically [1]:

$$\psi_{i,j}^{n+1} = \psi_{i,j}^n - \Delta\tau\,\mathrm{sign}(\phi(t))G(\psi_{i,j}^n),$$

where

$$G(\psi_{i,j}^n) = \begin{cases} \sqrt{\max\left((a_+)^2, (b_-)^2\right) + \max\left((c_+)^2, (d_-)^2\right)} - 1 & \phi(t,i,j) > 0 \\ \sqrt{\max\left((a_-)^2, (b_+)^2 + \max\left((c_-)^2, (d_+)^2\right)\right)} - 1 & \phi(t,i,j) < 0 \\ 0 & \phi(t,i,j) = 0 \end{cases}$$

$$a = (\psi_{i,j} - \psi_{i-1,j})/h$$

$$b = (\psi_{i+1,j} - \psi_{i,j})/h$$

$$c = (\psi_{i,j} - \psi_{i,j-1})/h$$

$$d = (\psi_{i,j+1} - \psi_{i,j})/h$$

$$a_+ = \max(a, 0), a_- = \min(a, 0), \text{etc.}$$

In practice we only need to evolve this equation until the region around the zero contour of ψ is no longer changing by much; at each time step we compute

$$Q = \frac{\sum_{|\phi_{i,j}^m|}|\phi_{i,j}^{m+1} - \phi_{i,j}^m|}{\#\{|\phi_{i,j}^m| < h\}},$$
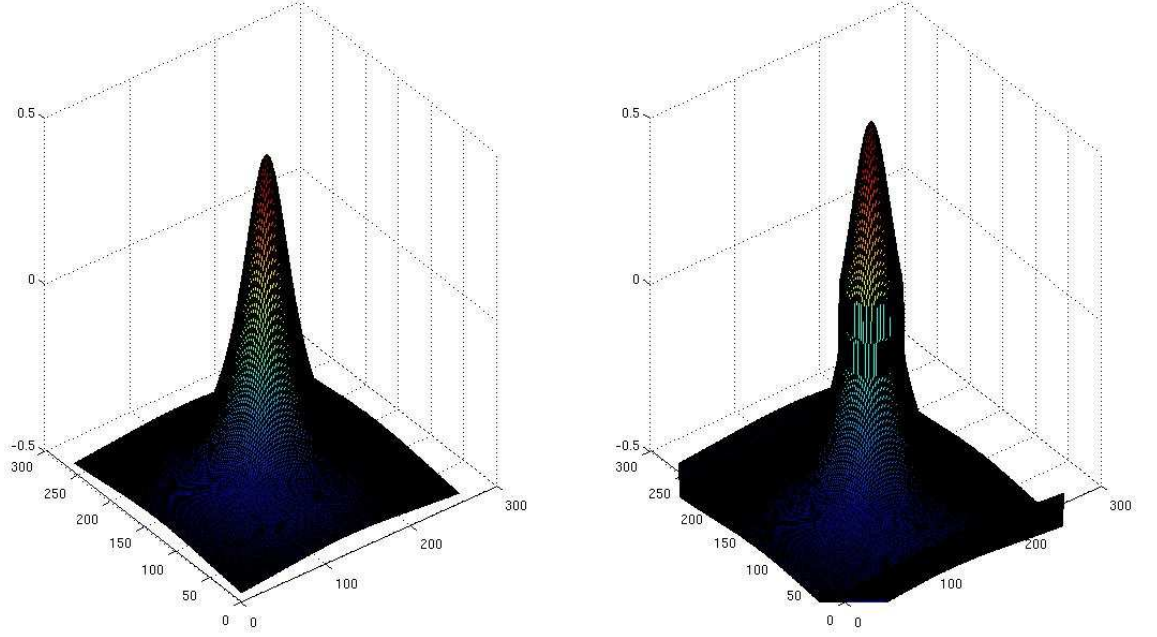
and stop the iteration when $Q < \Delta\tau \cdot h^2$.

Figure 2. (left) A "bump" level set function with a circular zero contour. (right) A "rescaled" function with the same zero level set, found by performing the reinitialization procedure described above on the left function

3.5 Summary: The Full Algorithm

We now have everything we need to perform image segmentation. Note that there is an extra loop over k; usually, we take numIterationsInner $= 1$, but for noisy images we set numIterationsInner $= 5$; that is, we run the iteration 5 times with the same values for c_1 and c_2 [1].

Algorithm
```
Given an image I, an initial level set function φ₀, and parameters μ,ν,p,λ₁,λ₂,dt,h

Set  φ = φ₀
for k = 1:numIterationsOuter
  Compute c₁ and c₂ for the current φ
  for l = 1:numIterationsInner //
    Update φ using the Chan-Vese iteration
    Reinitialize φ to the signed distance function to its zero contour
```

4 Results

We will now examine the performance of this algorithm when applied to several types of images. The following set of parameters will be taken as a baseline:

$$\lambda_1 = \lambda_2 = 1$$
$$p = 1$$
$$\mu = 0.5$$
$$\nu = 0$$
$$h = 1, dt = 0.1$$

Unless otherwise stated, we will perform one iteration of the inner loop of the algorithm; that is, we will update ϕ only once for each update of the averages c_1 and c_2.

4.1 Performance Metric

We will use a region overlap metric to evaluate performance; given a baseline foreground region R_1 (assumed to be the true foreground region) and the foreground region R_2 found by the algorithm, we set $E_1(R_1, R_2) = \frac{\#(R_1 \setminus R_2)}{\#R_1}$, $E_2(R_1, R_2) = \frac{\#(R_2 \setminus R_1)}{\#R_2}$. Thus, E_1 is a measure of the number of "false negative" pixels in the segmentation given by the algorithm and E_2 is a measure of the number of "false positives." A perfect segmentation will give $E_1 = E_2 = 0$; the worst possible segmentation (where the true foreground and background regions are reversed) will give $E_1 = E_2 = 1$. We will plot $\max(E_1, E_2)$ and $\min(E_1, E_2)$ against varying parameters.

4.2 Segmenting A Synthetic Bilevel Image

We start with a trivial example; a smooth, synthetically generated image consisting of only two distinct graylevels. The correct segmentation is immediately obvious in this case, and could be obtained easily by a thresholding algorithm. This image is segmented in one iteration with the default parameters of our algorithm.
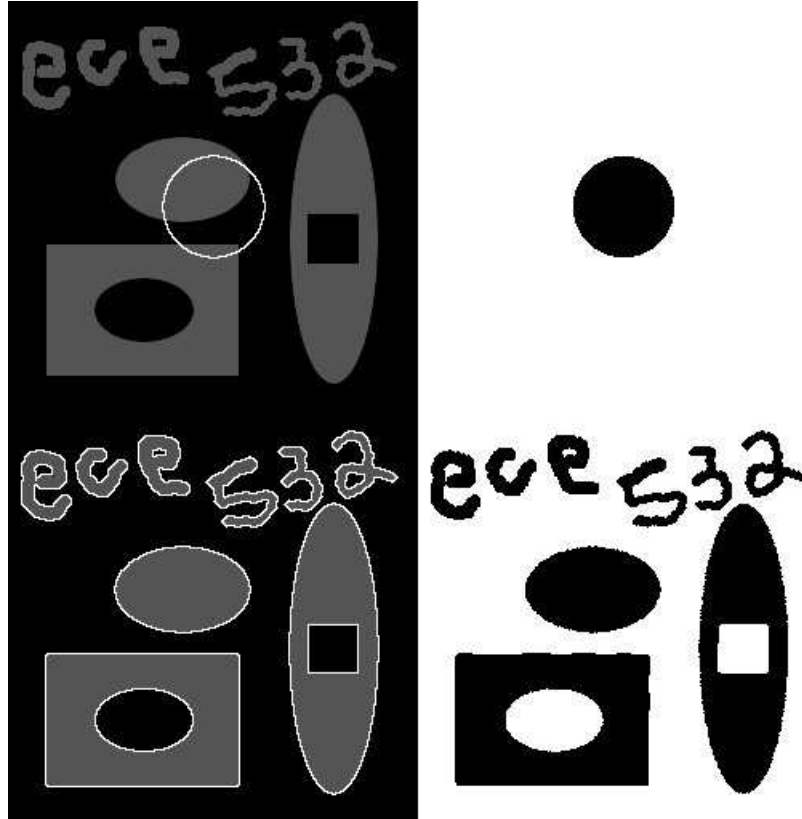
Figure 3. (left) The image with an initial contour and the edge contour after one interation. (right) the initial segmentation and the segmentation after one iteration

The segmentation is achieved in one iteration and is nearly perfect. We now vary the timestep to see how large we can make it and still segment the image well in one step.
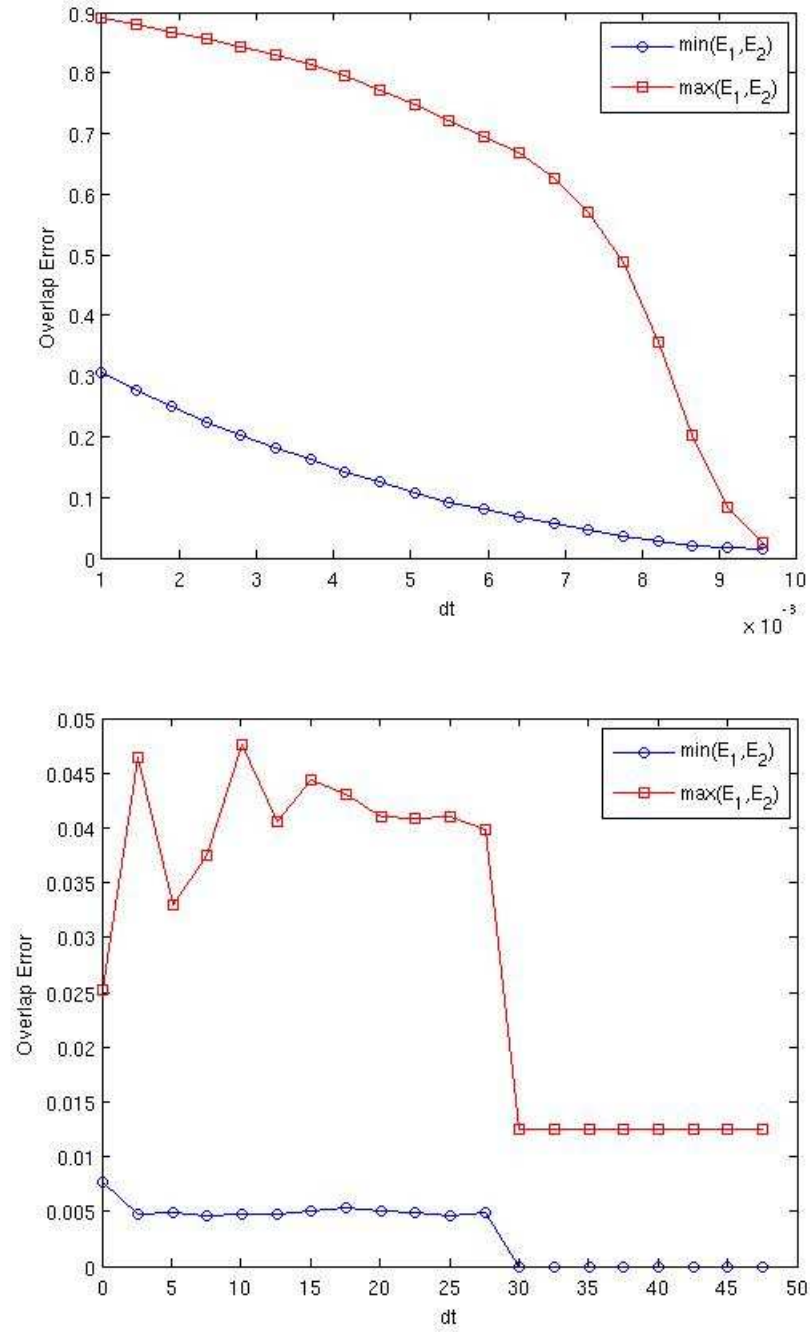

Figure 4. (top) Overlap error after one iteration for small dt, (bottom) Overlap one error after one iteration for larger dt

This image can be segmented well in one iteration for timesteps greater than about 0.001. We get a good segmentation after one iteration even for very large timesteps. Next we see what happens when we perform multiple iterations with a small timestep (dt = 0.001) or a large timestep (dt = 50.0).
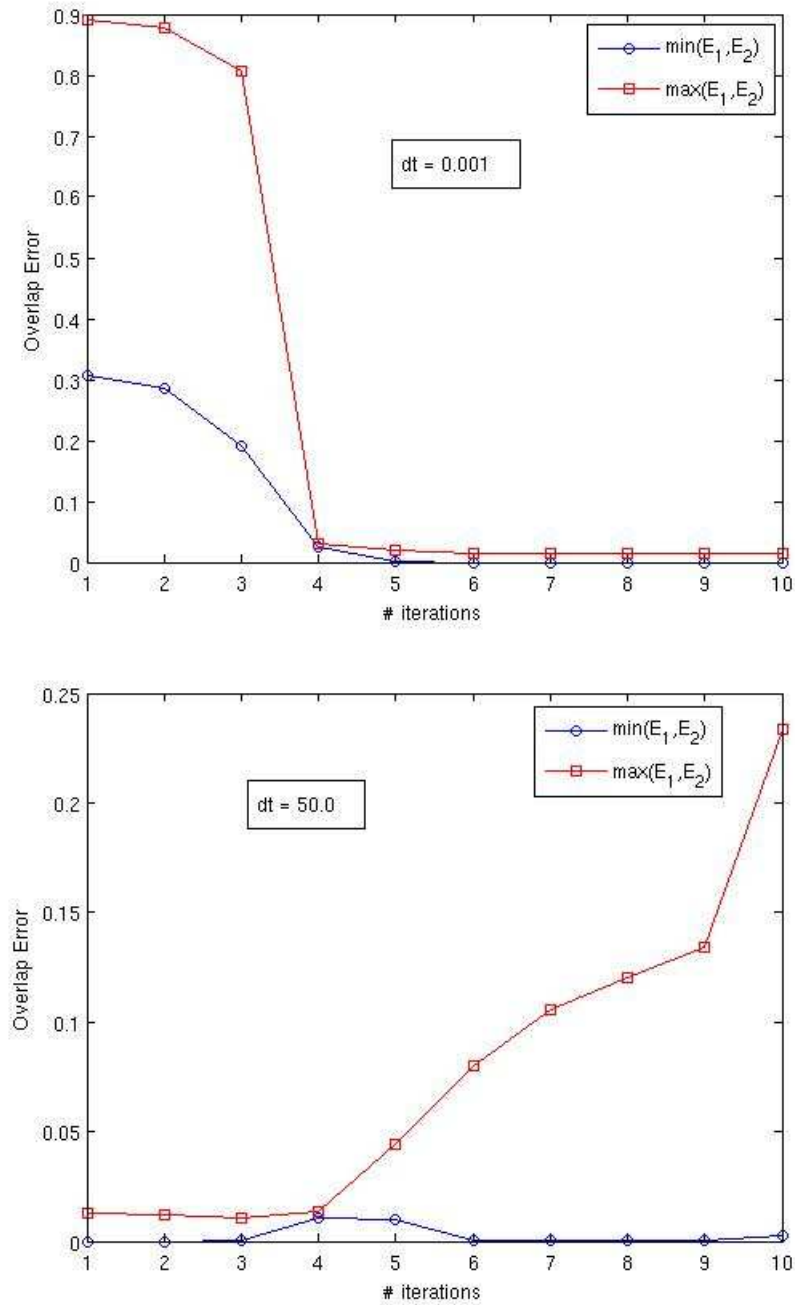
Figure 5. (top) Overlap error vs. number of iterations for a small timestep. (bottom) Overlap error vs. number of iterations for a large timestep

As expected, when the timestep is small the segmentation converges. However, if we take the timestep too large, the results diverge after a few iterations even though the segmentation is initially good. In practice, we do not know a priori how many iterations it will take to segment a given image, so we need to choose a smaller timestep; we will use $dt = 0.1$ for the rest of this paper.

4.3 A Synthetic Bilevel Image with Noise

The previous image was easily segmented by our algorithm, but that image could have been segmented just as well by a much simpler method. Now we add a significant amount of noise to get an image that is less trivial to segment correctly. There is now far too much noise to get a decent segmentation with a direct thresholding method:
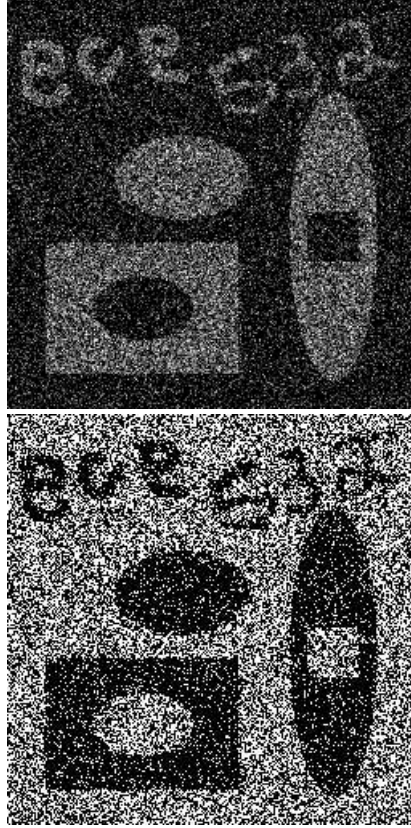
Figure 6. (top) The noisy image, and (bottom) the segmentation obtained by thresholding using the Kittler-Illingworth method. For this segmentation, max $(E_1, E_2) = 0.5625$, min $(E_1, E_2) = 0.1222$; over half the image pixels are segmented incorrectly.

An edge-based segmentation, such as a Sobel gradient method, will perform even worse since there are no well defined edges. To use such a method effectively we would need to perform denoising in a preprocessing step.

The Chan-Vese algorithm can give a good segmentation even of very noisy images. We will use the true segmentation from the noiseless image as a baseline for forming the overlap errors.

As mentioned, for this noisy image we perform five iterations of the algorithm for each fixed value of c_1, c_2. In each of the plots below, we set all the parameters to their default value, then vary one parameter at a time and plot the overlap errors against that parameter.

Note that in the plots below the two dashed green lines represent $\min(E_1, E_2)$ and $\max(E_1, E_2)$ for the Kittler-Illingworth thresholding segmentation, for comparison purposes.
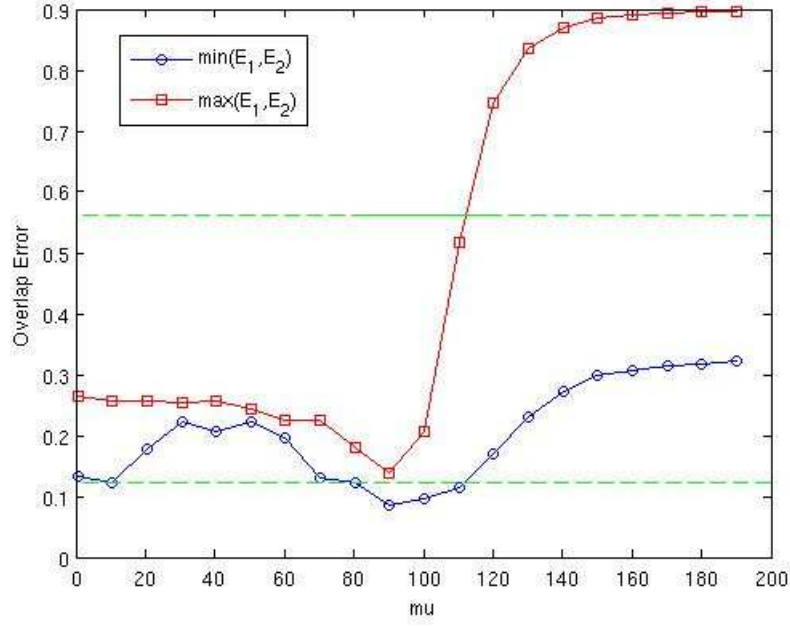

Figure 7. Overlap errors vs. length penalty μ. The best segmentation is obtained for $\mu \sim 94$.
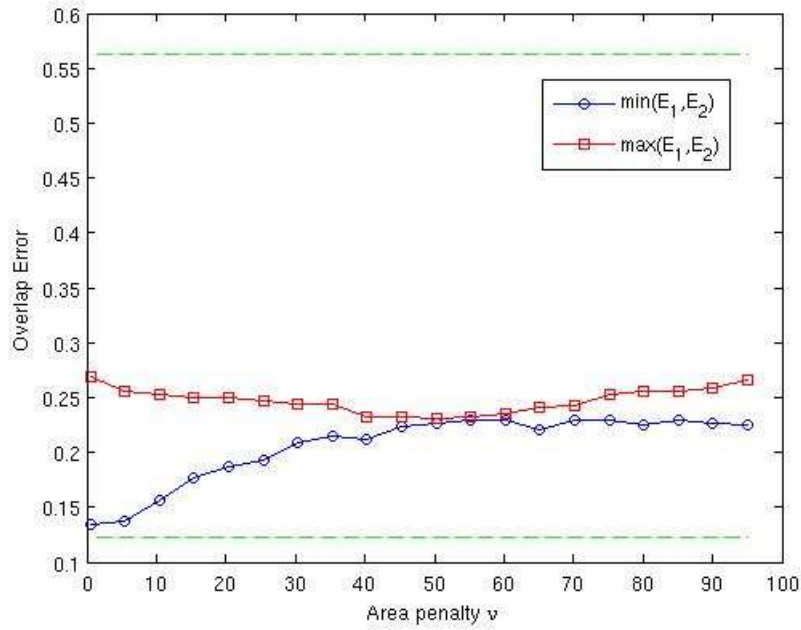
Figure 8. Overlap errors vs. ν.
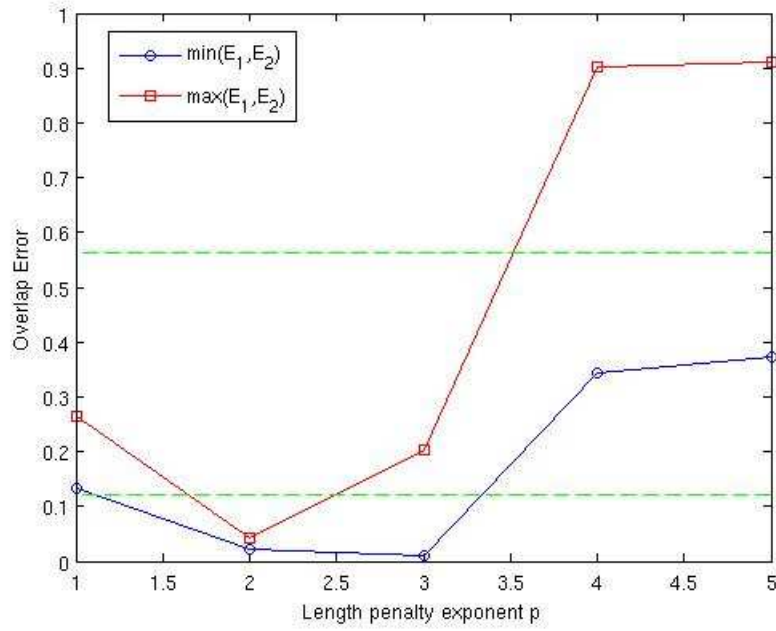
Figure 9. Overlap errors vs. p. For $p = 2$ we get a segmentation that is significantly better than the thresholding case.
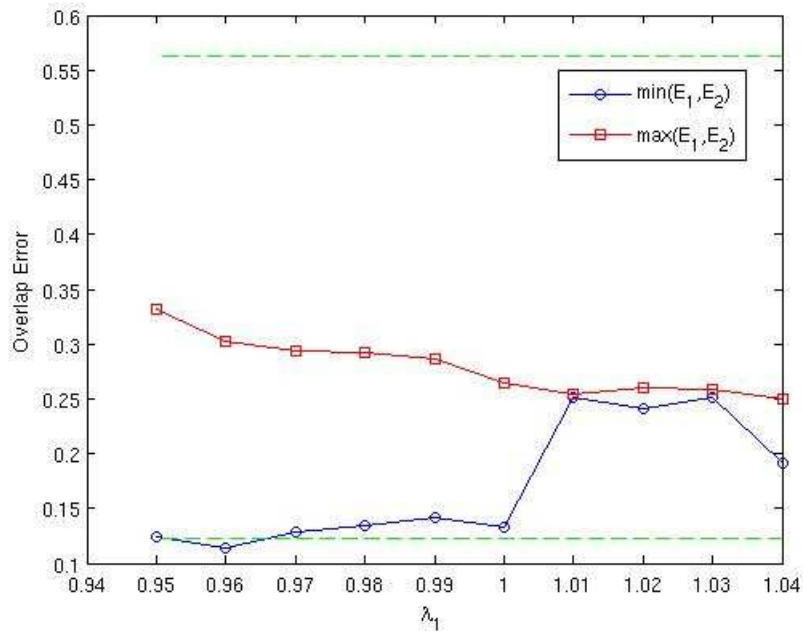

Figure 10. Overlap error vs. λ_1. Changing λ_1 from 1 does not improve the segmentation, since our foreground and background regions both have uniform intensity before noise is added. The results for varying λ_2 are similar.

Here is the segmentation of this image attained by setting $p = 2$, $\mu = 1$, and all the other parameters to their defaults:
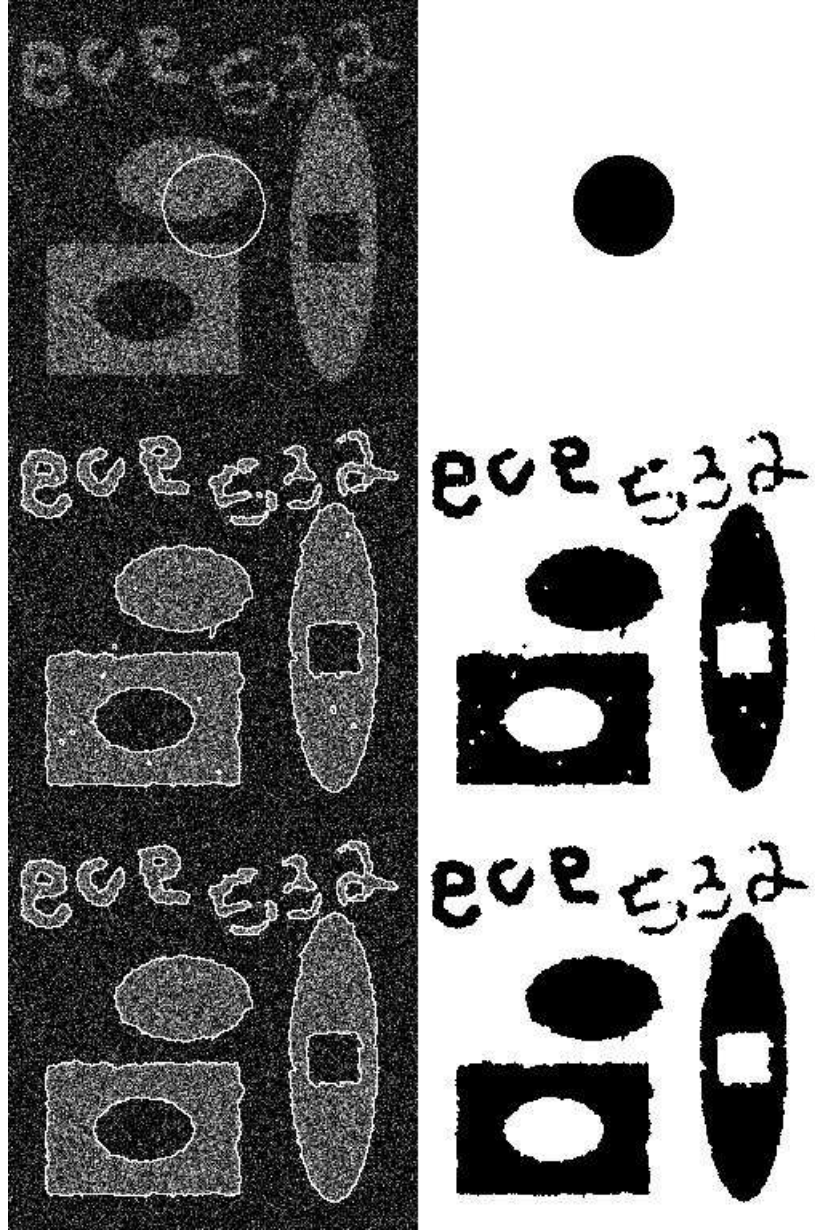
Figure 11. Segmentation of a noisy image. (left) The original image and the detected edge contour after 0, 3, and 5 iterations. (right) The computed segmentation after 0, 3, and 5 iterations. The overlap error is less than five percent.

4.4 Algorithm Running Times

We examine the time it takes to run one iteration of the Chan-Vese algorithm as a function of image size. We use the square bilevel image from section 4.2 at various scales, setting $N = 256, 512, 1024, 2048,$ and 4096. The running time is approximately proportional to the number of pixels, N^2, taking a total of 0.36 seconds for the 256×256 image and 26.2 seconds for the 4096×4096 image. Each of these images is successfully segmented with this one iteration.
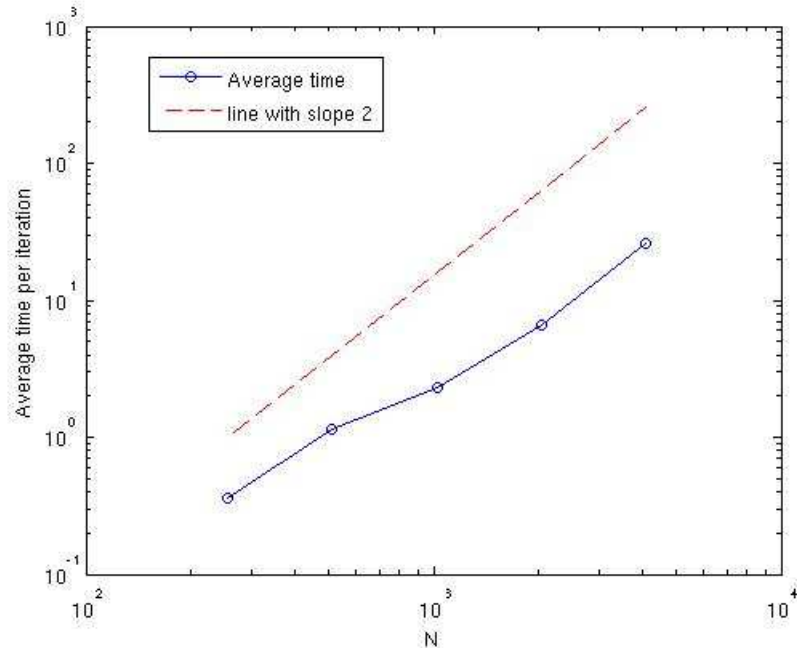
Figure 12. Time (in seconds) to complete one iteration of Chan-Vese algorithm, approximately

For the smallest image, the time is dominated by the step that reinitializes ϕ to the signed distance function to its zero level set; for the 256×256 image this step takes about 85% of the total time. However, for large images the main iteration takes longer, taking up about 75% of the total time for the 4096×4096 image. The time for the main algorithm is almost exactly proportional to N^2, while the time for the reinitialization step is not linearly related to N^2.
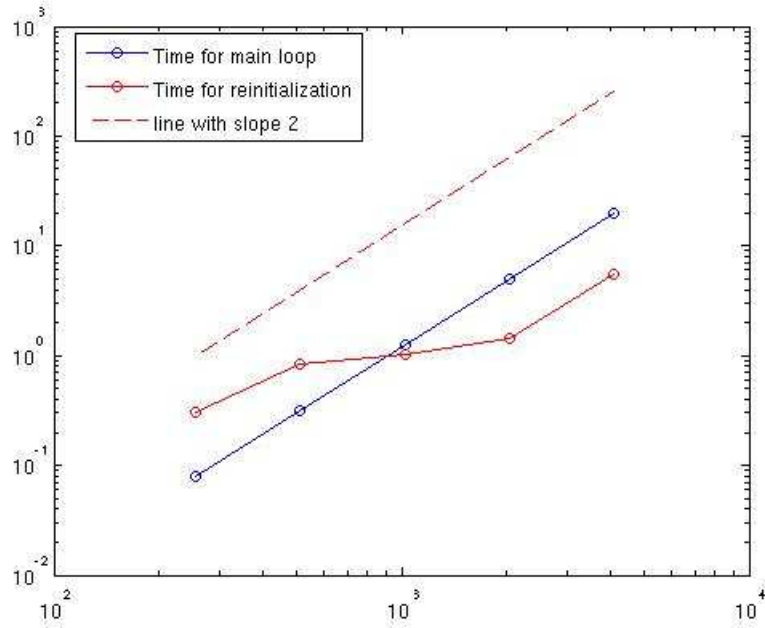

Figure 13. Time (in seconds) to complete the main iteration of the Chan-Vese algorithm (blue) and to complete the reinitialization step (red)

Even ignoring the reinitialization step, the Chan-Vese algorithm takes about 0.1 seconds per iteration on a 256×256 image with 256 gray levels. This makes it unfeasible for use in real time applications such as live video tracking; in such applications the segmentation time must be at most on the order of the video frame rate. The algorithm is more suited to applications where speed and processing power are not of paramount importance, such as in medical image analysis where processing can be done offline after the images are acquired.

4.5 Miscellaneous Scenarios

Finally, we present some segmentation results from various image types to showcase the utility of this algorithm.
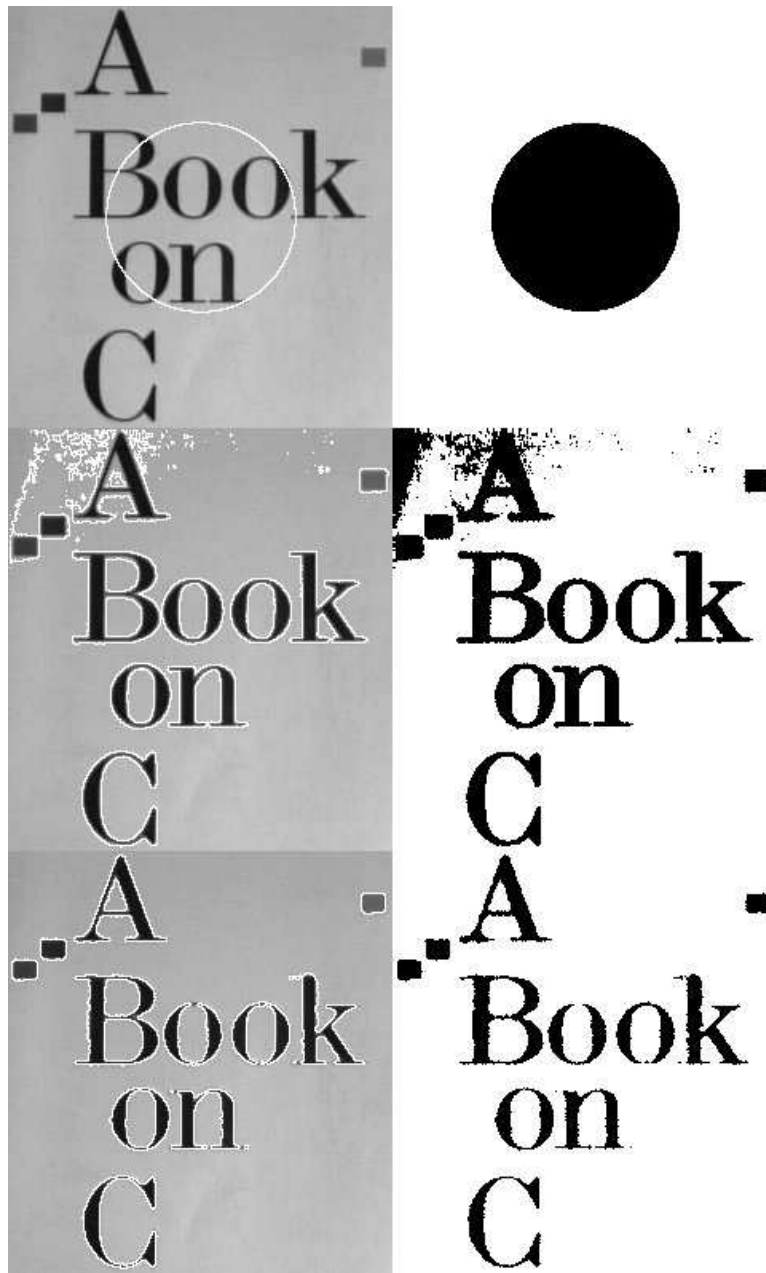

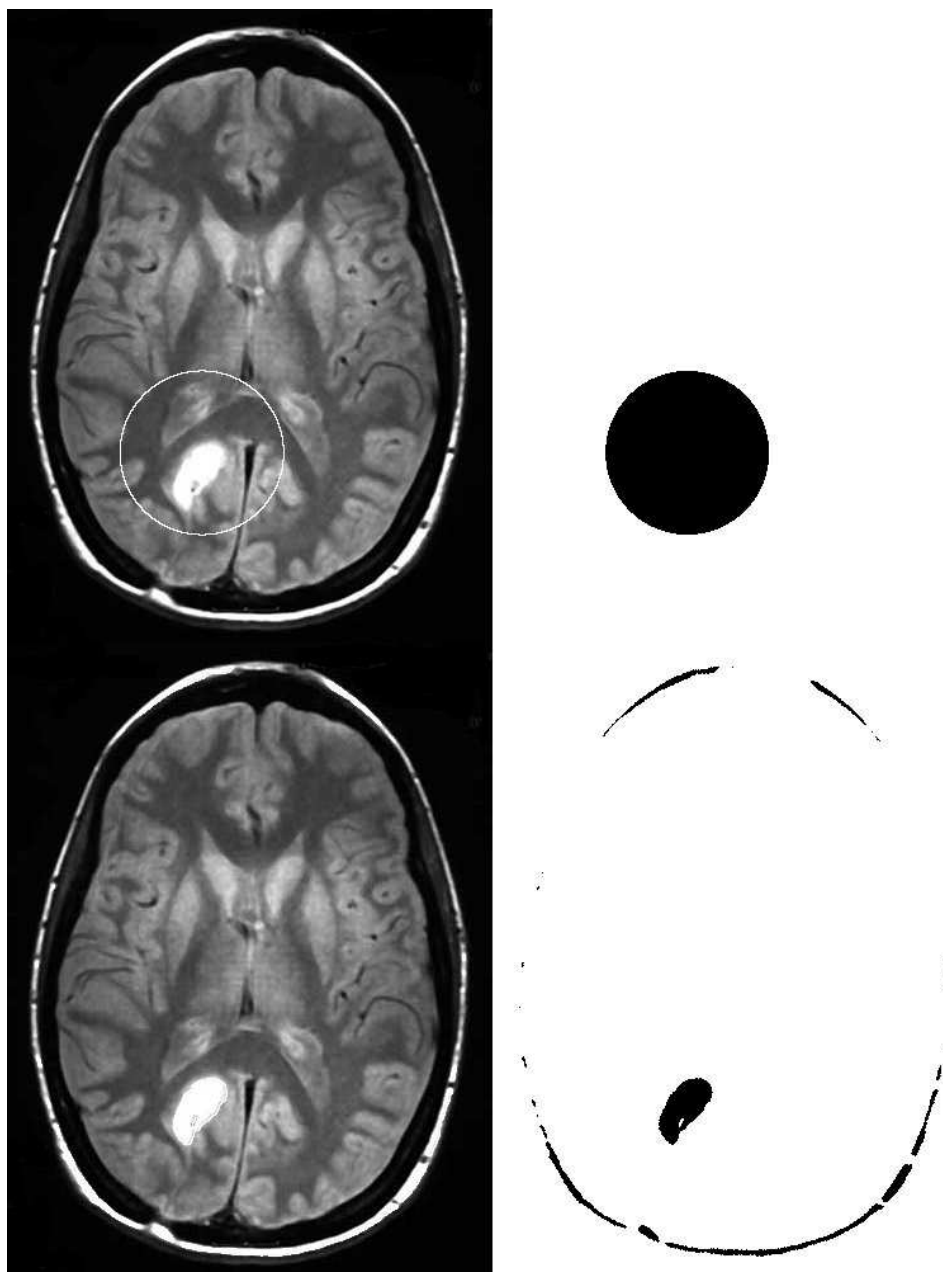**Figure 14.** Segmenting typeset characters. Default parameters, 5 iterations of inner loop.

Figure 15. Segmenting a tumor (bright white spot) in an MRI image. Large area and length penalty terms ($\mu \sim 70$, $\nu \sim 1900$). We set $\lambda_1 = 1.02$ to account for the large graylevel variation in the non-tumor regions. The tumor is segmented well, but we get false positive detection of part of the white skull area that has approximately the same gray level as the tumor (a better selection of parameters might remove these false positives).
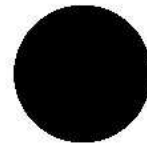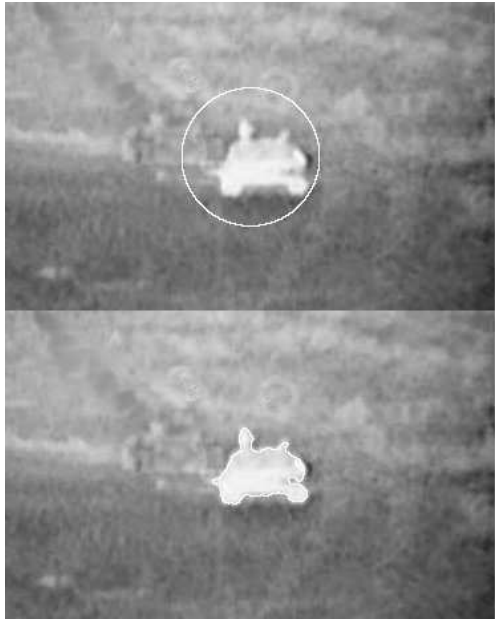
Figure 16. Segmenting an infrared image of a tank. $\mu = 10$, $\nu = 500$
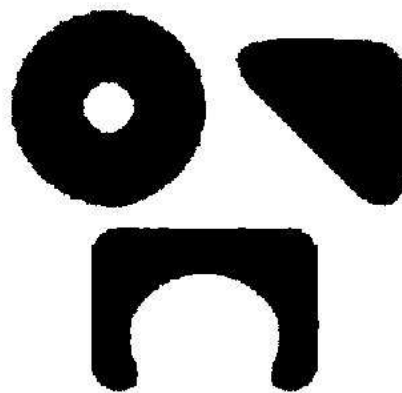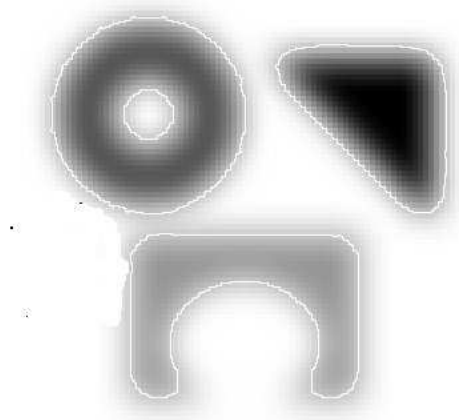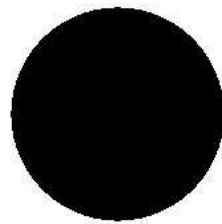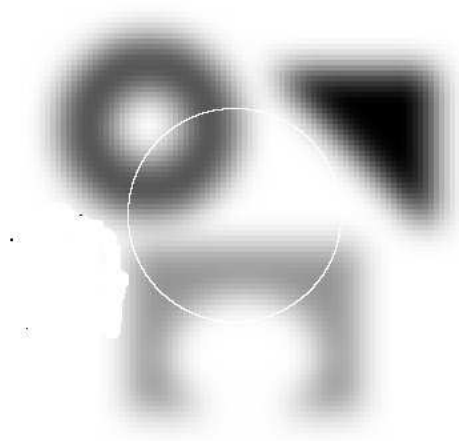

Figure 17. Detecting blurry objects with different average intensities. Default parameters.

5 Extensions and Modifications

5.1 Speeding Up the Algorithm: Reduction to an ODE Method

In [5], Gibou and Fedkiw introduced a method to reduce the segmentation problem to an ordinary differential equation rather than a nonlinear PDE. They ignore the length and area terms, as well as the delta function term, and set $\lambda_1 = \lambda_2 = 1$. This reduces the problem to the ordinary differential equation

$$\phi_t = -(I - c_1)^2 + (I - c_2)^2,$$

with the initial condition $\phi(0) = \phi_0$ for some user selected starting function ϕ_0. We can solve this equation easily with standard ODE methods, for example, using a 4th order Runge-Kutta solver. This method is significantly faster than the Chan-Vese algorithm. Implemented in Matlab, it can segment the 256×256 bilevel image from 4.2 in about 0.07 seconds; a C implementation would be even faster. However, it is effective on a much more restricted class of images; for example, it does not deal well with very noisy images since there are no length or area terms.

5.2 Generalizing to Vector-Valued Images

In this paper we have dealt strictly with gray-level images. There is a rather straightforward generalization to vector-valued images (RGB images, for example), introduced in [7]. It replaces the fitting energy with a new functional [7]

$$F(c^+, c^-, \phi) = \mu \cdot L + \int_{\text{inside}(C)} \frac{1}{N} \sum_{i=1}^{N} \lambda_i^+ |I(x, y) - c_i^+|^2 d\,x\,d\,y$$

$$+ \int_{\text{outside}(C)} \frac{1}{N} \sum_{i=1}^{N} \lambda_i^- |I(x, y) - c_i^-|^2 d\,x\,y.$$

Here we assume each image pixel is a vector in $\mathbb{R}^N$. c_i^+ is the average value of the ith components of all pixels in the foreground, while c_i^- is the average of the ith components in the background. $\lambda_i^\pm$ are parameters analogous to the λ_1, λ_2 terms from the scalar model. An iterative solution method is developed in a similar fashion. This method can then be used to segment RGB color images, or any other image where the pixels are vector-valued.

5.3 Automated Selection of Parameters and Initial Level Set Function

An important consideration in many practical applications is the automated selection of parameter values. In this paper, parameters were selected manually to give qualitatively good segmentations for each test image. In practice, the user would want to be more thorough in selecting a set of parameters that fit a particular expected data set. In some cases, where we expect to work on a fairly homogeneous data, we might be able to get away with a hardcoded set of parameters. For example, if we wish to perform typeset character recognition on scanned book pages with the same font and similar image quality, we would expect the same set of parameters to work equally well for any given page. On the other hand, if we want to segment camera images taken in different lighting conditions and with different backgrounds, it might be helpful to adjust some of the parameters automatically based on, for example, the dynamic range, noise, or other computable properties of the image.

Also, in this paper an initial level set was chosen essentially arbitrarily; we used a sort of "bump" function that gives a circular initial contour. A more intelligent selection of this initial function can help reduce the number of iterations needed, both in the main algorithm and in the reinitialization step. As an example, one such method is to set the value of ϕ_0 at each pixel to $\phi_0(x, y) = I(x, y) - \text{mean}(I)$; this amounts to starting with an initial segmentation given by thresholding the image by its mean gray level.

6 Conclusion

We have introduced the Chan-Vese algorithm for image segmentation and shown that it is effective on a wide variety of images. It is especially useful in cases where an edge-based segmentation algorithm will not suffice, since it relies on global properties (graylevel intensities, contour lengths, region areas) rather than local properties such as gradients. This means that it can deal gracefully with noisy images, blurry images, and images where the foreground region has a complicated topology (multiple holes, disconnected regions, etc).

Although all the test images in this paper took under a minute to segment, the Chan-Vese algorithm is prohibitively slow for some applications. Depending on the type and size of the image and the number of iterations needed, the segmentation can take several seconds, which is too slow to keep up with typical video framerates. A survey of the literature reveals that the predominant uses of this method have been in medical image analysis, particularly for segmentation problems where a diagnostically relevant feature such as a lesion or tumor shows up as a dark or light spot as in figure 15.

Finally, we mention that this algorithm represents an exciting trend in the "modernization" of image processing and analysis. Mathematical subjects and methodologies which have traditionally found little use in image processing, such as partial differential equations and variational calculus, have over the past decade or so found a multitude of applications. From segmentation to image inpainting and denoising problems and beyond, such methods will no doubt play an important role in future image analysis research.

7 References

[1] T. Chan and L. Vese, *Active contours without edges*, IEEE transactions on image processing 10(2) (2001), pp. 266-277

[2] D. Mumford and J. Shah, *Optimal approximation by piecewise smooth functions and associated variational problems*, Comm. Pure Appl. Math. 42, 1989, pp. 577-685

[3] G. Aubert and L. Vese, *A variational method in image recovery*, SIAM J. Num. Anal., 34/5(1997), pp. 1948-1979

[4] M. Sussman, P. Smereka and S. Osher, *A Level Set Approach for Computing Solutions to Incompressible Two-Phase Flow*, J. Comput. Phys., V. 119 (1994), pp. 146-159.

[5] F. Gibou, R. Fedkiw, *Fast hybrid k-means level set algorithm for segmentation*, Proceedings of the 4th Annual Hawaii International Conference on Statistics and Mathematics, 2002. Stanford Technical Report, Nov 2002

[6] L. He, S. Osher, *Solving the Chan-Vese Model by a Multiphase Level Set Algorithm Based on the Topological Derivative*, Lecture notes in Computer Science, no. 4485, 2007, pp. 777-788

[7] T. Chan, L. Vese and Y. Sandberg, *Active contours without edges for vector-valued images*, Journal of Visual Communications and Image Representation, 11, no. 2 (2000), pp. 130-141

8 Appendix: Guide to Code

Attached files:

- TestChanVese.cpp

    Driver program to test the Chan-Vese segmentation algorithm

- ChanVeseSegmentation.cpp/.h

    Implementation of the main algorithm. The primary function is

void ChanVeseSegmentation,

which takes as input a grayscale image, an initial level set function, and a structure pCVinputs which sets the tunable parameters to be used in the segmentation. The output is a level set function giving the final segmentation; the edge contour of this segmentation can be found using the function ZeroCrossings.

- SimpleCVsegment.m

  A Matlab implementation of Gibou & Fedkiw's ODE method. Uses RKstep.m, which performs a single step of an arbitrary explicit Runge-Kutta solver.