Chan-Vese Image Segmentation

Shashank Sharma

December 20, 2023

1 Introduction

On a computer, images are represented using arrays of integers. Arrays for grayscale scale images have two dimensions, whereas for RGB images they have a 3rd dimension called *channel* as well, one for each of the colors Red, Green and Blue. The Image Segmentation problem is then to take such an array of integers and output a segmentation boundary that divides the image into the various objects it contains.

This is a very well researched problem and there exist various methods in literature for doing this. One of the most basic ones is thresholding - classifying pixels as *white* if they are above a threshold, and black instead. Canny edge detection is slightly more complicated - based on convolution against edge detection filters. Nowadays, deep neural network based techniques dominate the field.

In this report, we'll study the Chan-Vese model for image segmentation. It is a continuous model of the digital image, and the segmentation boundary is specified as the solution to a minimization problem. Level set methods can then be used to evolve an initial curve to a local minimum for the problem.

2 The Mumford Shah model

A given grayscale image is modeled as a function $f : \Omega \subset \mathbb{R}^2 \to [0, \infty]$. The Mumford and Shah model [9] approximates f by a piecewise-smooth function u as the solution to the minimization problem

$$\arg\min_{u,C} \mu \, \mathrm{Length}(C) + \lambda \int_{\Omega} (f(x) - u(x))^2 dx + \int_{\Omega \setminus C} |\nabla u(x)|^2 dx, \tag{2.1}$$

where C is an edge set curve where u is allowed to be discontinuous. The first term ensures the regularity of C, the second term encourages u to be close to f, and the third term ensures that u is differentiable and has a small gradient on $\Omega \setminus C$.

This is a very natural way to pose segmentation, but algorithms for solving this minimization model tend to very complicated and computationally expensive. Thus as a simplification, Mumford and Shah also considered a piecewise constant formulation,

$$\arg\min_{u,C} \mu \, \text{Length}(C) + \lambda \int_\Omega (f(x) - u(x))^2 dx \tag{2.2}$$

where u is required to be constant on each connected component of $\Omega \setminus C$.

3 The Chan-Vese Model

The Chan-Vese model chan-veseorig [3] is a further simplification of the piecewise constant Mumford-Shah model. It restricts u to only have two values - c_1 inside C and c_2 outside C. It also adds an additional term penalizing the area inside C. Thus, the model becomes

$$\begin{aligned} \arg\min_{c_1,c_2,C} & \mu \, \text{Length}(C) + \nu \, \text{Area}(\mathit{inside}(C)) \\ & + \lambda_1 \int_{\mathit{inside}(C)} (f(x) - c_1)^2 dx + \lambda_2 \int_{\mathit{outside}(C)} (f(x) - c_2)^2 dx. \end{aligned} \tag{3.1}$$

$\mu, \nu, \lambda_1, \lambda_2$ are hyper-parameters to be determined by experimentation.

4 A semi-implicit gradient algorithm for solving the Chan-Vese minimization problem

To actually solve the minimization problem, it's convenient to represent C as the zero-crossing of a level-set function ϕ such that

$$C = \{x \in \Omega : \phi(x) = 0\} \tag{4.1}$$

and ϕ has different signs inside and outside C. Provided that ϕ is smooth enough and is indeed a distance function, the minimization problem can be rewritten as

$$\begin{aligned} \arg\min_{c_1,c_2,\phi} & \mu \int_\Omega \delta(\phi(x)) |\nabla \phi(x)| dx + \nu \int_\Omega H(\phi(x)) dx \\ & + \lambda_1 \int_\Omega (f(x) - c_1)^2 H(\phi(x)) dx + \lambda_2 \int_\Omega (f(x) - c_2)^2 (1 - H(\phi(x))) dx, \end{aligned} \tag{4.2}$$

where H is the heaviside function and δ the dirac delta distribution.

For a fixed ϕ, the optimal c_1 and c_2 are the region averages

$$c_1 = \frac{\int_\Omega f(x) H(\phi(x)) dx}{\int_\Omega H(\phi(x)) dx}, \tag{4.3}$$

$$c_2 = \frac{\int_\Omega f(x)(1 - H(\phi(x))) dx}{\int_\Omega (1 - H(\phi(x))) dx}. \tag{4.4}$$

For solving the problem numerically, H is approximated by the smooth function

$$H_\epsilon(t) = \frac{1}{2}(1 + \frac{2}{\pi}\arctan(\frac{t}{\epsilon})) \tag{4.5}$$

and δ by its derivative

$$\delta_\epsilon(t) = \frac{d}{dt}H_\epsilon(t) = \frac{\epsilon}{\pi(\epsilon^2 + t^2)}. \tag{4.6}$$

This introduces another hyper-parameter ϵ.

For fixed c_1, c_2, we can use the Euler Lagrange equations to come up with a gradient descent for optimizing ϕ

$$\frac{\partial \phi}{\partial t} = \delta_\epsilon(\phi)\left(\mu \operatorname{div}(\frac{\nabla\phi}{|\nabla\phi|}) - \nu - \lambda_1(f - c_1)^2 + \lambda_2(f - c_2)^2\right) \text{ in } \Omega, \tag{4.7}$$

$$\frac{\delta_\epsilon(\phi)}{|\nabla\phi|}\frac{\partial\phi}{\partial\vec{n}} = 0 \text{ on } \partial\Omega, \tag{4.8}$$

where $\vec{n}$ is the outward normal on the image boundary.

Now, we discretize the problem to solve it numerically. f is sampled on a regular grid $\Omega = \{0, \ldots, M\}\prod\{0, \ldots, M\}$. We discretize in space as (Method of Lines)

$$\frac{\partial\phi_{i,j}}{\partial t} = \delta_\epsilon(\phi_{i,j})\left[\mu\left(\nabla_x^- \frac{\nabla_x^+\phi_{i,j}}{\sqrt{\eta^2 + (\nabla_x^+\phi_{i,j})^2 + (\nabla_y^0\phi_{i,j})^2}} + \nabla_y^- \frac{\nabla_y^+\phi_{i,j}}{\sqrt{\eta^2 + (\nabla_x^0\phi_{i,j})^2 + (\nabla_y^+\phi_{i,j})^2}}\right)\right. \tag{4.9}$$

$$\left. -\nu - \lambda_1(f_{i,j} - c_1)^2 + \lambda_2(f_{i,j} - c_2)^2\right], \quad i, j = 1, \ldots, M-1, \tag{4.10}$$

where ∇_x^- denotes backward difference, ∇_x^+ denotes forward difference and ∇_x^0 denotes central difference in the x dimension. Note that we have introduced another hyper-parameter η to prevent division by 0. The reason for mixing ∇_x^- and ∇_x^+ is to center the combined difference at (i, j).

For convenience, we let

$$A_{i,j} = \frac{\mu}{\sqrt{\eta^2 + (\nabla_x^+\phi_{i,j})^2 + (\nabla_y^0\phi_{i,j})^2}}, B_{i,j} = \frac{\mu}{\sqrt{\eta^2 + (\nabla_x^0\phi_{i,j})^2 + (\nabla_y^+\phi_{i,j})^2}} \tag{4.11}$$

and use Forward Euler to discretize time. Then the fully discretized scheme looks like

$$\begin{aligned}\frac{\phi_{i,j}^{n+1} - \phi_{i,j}^n}{dt} = \delta_\epsilon(\phi_{i,j})\,&[(A_{i,j}(\phi_{i+1,j} - \phi_{i,j}) - A_{i-1,j}(\phi_{i,j} - \phi_{i-1,j})) \\ &+ (B_{i,j}(\phi_{i,j+1} - \phi_{i,j}) - B_{i,j-1}(\phi_{i,j} - \phi_{i,j-1})) \\ &-\nu\lambda_1(f_{i,j} - c_1)^2 + \lambda_2(f_{i,j} - c_2)^2].\end{aligned} \tag{4.12}$$

fully discret

Algorithm 1 A gradient descent based algorithm for Chan-Vese

Initialize ϕ
for n = 1,2, ... **do**
  Compute c_1, c_2 as region averages
  Evolve ϕ using one Forward-Euler time step
  if $\frac{\left\|\phi^{n+1}-\phi^{n}\right\|_2}{\sqrt{|\Omega|}} < tol$ **then**
    break
  end if
end for

The boundary condition is enforced by creating ghost points so that

$$\phi_{-1,j} = \phi_{0,j},\ \phi_{M,j} = \phi_{M-1,j},\ \phi_{i,-1} = \phi_{i,0},\ \phi_{i,M} = \phi_{i,M-1}. \tag{4.13}$$

We terminate when $\frac{\left\|\phi^{n+1}-\phi^{n}\right\|_2}{\sqrt{|\Omega|}}$ is less than a prescribed tolerance tol.

Algorithm 1 describes the full procedure.

To extend this algorithm to segment RGB images, we use the Chan-Sandberg-Vese model [4] instead. It just makes f, c_1, c_2 as d-dimensional, and the new minimization problem becomes

$$\begin{aligned}\underset{c_1,c_2,C}{\arg\min}\, &\mu\, \mathrm{Length}(C) + \nu\, \mathrm{Area}(inside(C)) \\ &+ \lambda_1 \int_{inside(C)} \|f(x)-c_1\|_2^2\, dx + \lambda_2 \int_{outside(C)} \|f(x)-c_2\|_2^2\, dx.\end{aligned} \tag{4.14}$$

The overall algorithm remains the same.

5 Practical Implementation: The Code

The code written works for both grayscale and RGB images of any size.

First, we set the constants and create a function to display the images as shown in Figure 1.

Next, we create functions to compute the dirac delta and heaviside functions, integrate and compute the average of a function using a specified density as shown in Figure 2. The function **extend** will be used to extend ϕ by one row / column on each side of the image, using 0-Neumann boundary conditions. The **restrict** function will be used to restrict any function back to the image domain.

Next, we write functions for computing the forward, backward and central differences of a function in the row and column direction, as shown in Figure 3. The functions are designed so that the output is zero whenever calculating a difference involves indices outside the image bounds. So for example - the row backward difference along the first row of a function would be zero.

```
pkg load image;

file = "apple.jpeg";

% Set constants
neta = 10^(-8);
mu = 0.2;
nu = 0;
lambda1 = 1; lamnda2 = 1;
dt = 0.5;
tol = 0.6*10^(-2);
global epsilon = 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function display_image(f)
  figure(1); clf;
  if (length(size(f)) == 2)
    colormap(gray);
  endif
  imagesc(f);
  %imshow(f);
  caxis([0 1])
endfunction
```

fig:Code 1

Figure 1: Setting constants and display-image function

```

function result = dirac_delta(A)
  global epsilon;
  result = epsilon ./ (pi * (epsilon^2 + A.^2));
endfunction

function result = heaviside(A)
  global epsilon;
  result = (1 + (2 * atan(A / epsilon) / pi)) / 2;
endfunction

function result = integrate(A)
  result = sum(sum(A));
endfunction

function result = average(f, density)
  result = integrate(f .* density) / integrate(density);
endfunction

%Extend f to outside the image boundary using 0 Neumann Boundary Condition
function result = extend(f)
  A = [f(:,1, :), f, f(:,end, :)];
  result = [A(1,:, :); A; A(end, :,:)];
endfunction

function result = restrict(f)
  result = f(2:end-1, 2:end-1,:);
endfunction

```

fig:Code 2

Figure 2: Some basic functions defined

fig:Code 4

Next in Figure 4, we actually read the image from file and then rescale it so that pixel

```
%%% Functions for various differences. Coded in such a way that the output
%% is 0 whenever a difference doesn't make sense

function result = row_forward_diff(f)
  result = [f(2:end,:,:); f(end,:,:)] - f;
endfunction

function result = row_backward_diff(f)
  result = f - [f(1,:,:); f(1:end-1,:,:)];
endfunction

function result = row_central_diff(f)
  result = ([f(2:end, :,:); f(end-1,:,:)] - [f(2,:,:); f(1:end-1,:,:)]) / 2;
endfunction

function result = col_forward_diff(f)
  result = [f(:, 2:end,:), f(:, end,:)] - f;
endfunction

function result = col_backward_diff(f)
  result = f - [f(:, 1,:), f(:, 1:end-1,:)];
endfunction

function result = col_central_diff(f)
  result = ([f(:, 2:end,:), f(:, end-1,:)] - [f(:, 2,:), f(:, 1:end-1,:)]) / 2;
endfunction
```

fig:Code 3

Figure 3: Functions for calculating various differences

```

%% Read an image from a file
f = imread(file);
info = imfinfo(file);

% convert image to double and scale to [0,1]
f = double(f);
f = f / 2^(info.BitDepth);

[n1,n2, n3] = size(f)
LargestPixel = norm(reshape(f, n1*n2*n3, 1), inf)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Initialize phi
x = 1:n2; y = (1:n1)';
[xx, yy] = meshgrid(x, y);
phi = sin(pi * 10* x / n2) .* sin(pi * 10* y / n1);

display_image(f);
```

fig:Code 4

Figure 4: Reading the image from file and initializing ϕ

values lie between 0 and 1. ϕ is initialized using a scaled version of $sin(\pi x)sin(\pi y)$ as suggested in [3] because it leads to fast convergence.

Finally in Figure 5, we come to the Forward Euler time stepping. ϕ is first extended by one unit on each side of the image, and all the computations are carried out with this extended version to ϕ, so that boundary conditions are respected. The computations

```
step = 1;
while (1)

  phi_extended = extend(phi); % Extend using 0 neumann bdd condns

  phi_i_plus = row_forward_diff(phi_extended);
  phi_i_minus = row_backward_diff(phi_extended);
  phi_i_zero = row_central_diff(phi_extended);

  phi_j_plus = col_forward_diff(phi_extended);
  phi_j_minus = col_backward_diff(phi_extended);
  phi_j_zero = col_central_diff(phi_extended);

  A = mu ./ ((neta^2 + phi_i_plus.^2 + phi_j_zero.^2).^(0.5));
  B = mu ./ ((neta^2 + phi_i_zero.^2 + phi_j_plus.^2).^(0.5));

  d = heaviside(phi);
  c1 = average(f, d);
  c2 = average(f, 1 - d);

  % Do all difference computations using phi_extended to enforce 0 Neumann bdd condns
  Dphi = restrict(row_backward_diff(A .* phi_i_plus) + col_backward_diff(B .* phi_j_plus));
  Dphi = Dphi - nu - lambda1 * sum((f - c1).^2,3) + lamnda2 * sum((f - c2).^2, 3);
  Dphi = Dphi .* dirac_delta(phi);

  phi_new = phi + Dphi * dt;

  if (rem(step, 100) == 0)
    step
    display_image(f);
    hold on;
    contour(x, y, phi_new, [0,0], 'r');
    drawnow
    pause(0.001);
    if (norm(reshape(phi_new - phi, n1*n2, 1), 2) / sqrt(n1*n2) < tol)
      printf("Converged\n");
      break;
    endif
  endif

  phi = phi_new;
  step += 1;

endwhile
```

fig:Code 5

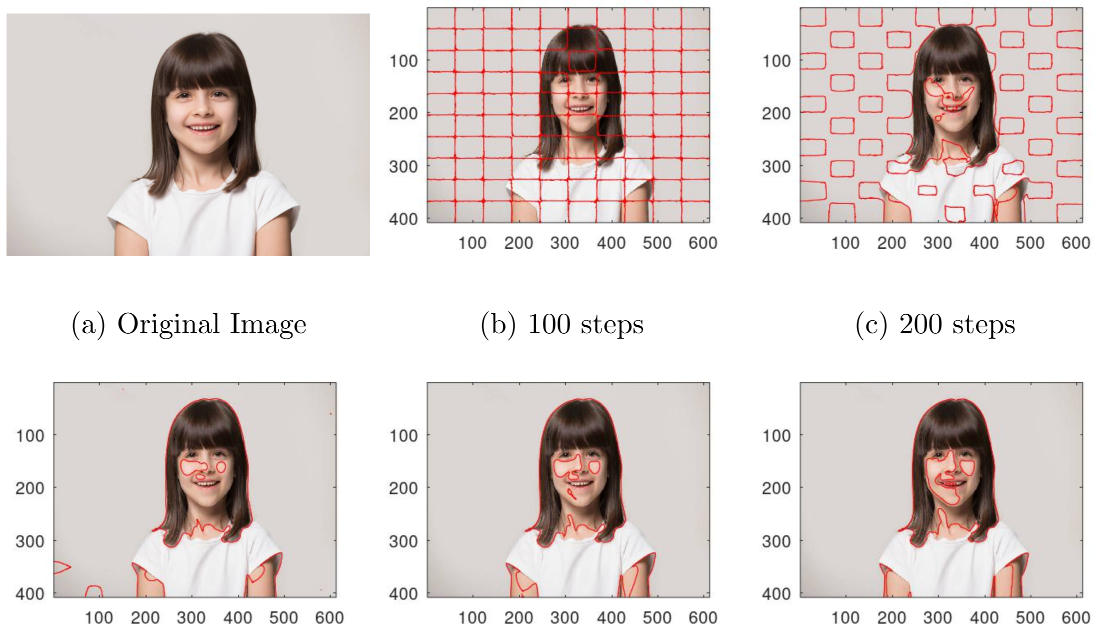
Figure 5: Forward Euler time stepping

carried out are just a fully vectorized version of those described in (4.12) fully discrete scheme. At every 100 steps, we display the contour on the image and also check to see if we satisfy the convergence criteria.

6 Examples

In all the examples, we use the parameters given in Figure 1 fig:Code 1 unless stated otherwise. It's also important to note that the algorithm's runtime is very sensitive to the chosen values

Figure 6: Chan-Vese Segmentation on an RGB image



(a) Original Image (b) 100 steps (c) 200 steps

(d) 300 steps (e) 400 steps (f) 900 steps

of dt, initial condition of ϕ and the image size itself.

6.1 Effect of Noise

Figure 6 demonstrates the algorithm run on an RGB image. It takes 900 steps to converge.

Now, we add a random amount of noise to the image with strengths of $0.2, 0.5$ and 1 respectively, and see how the algorithm performs. Figure 7 shows that the number of steps needed for convergence and the predicted contour remain largely unaffected, thus showing that the algorithm is fairly robust to noise.

6.2 Effect of μ

Figure 8 shows the effect of varying the length parameter μ. We see that as μ increases, the contour output becomes smoother as expected. However, the number of steps required by the algorithm to converge also increase considerably.

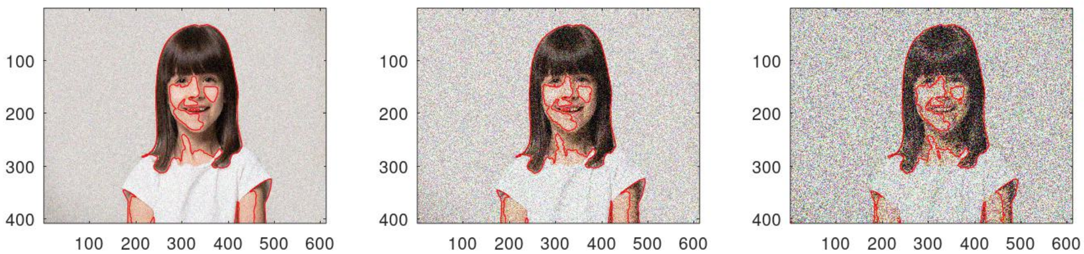
6.3 Effect of ν

Figure 9 shows the effect of varying the area parameter ν. For a negative ν, a bigger area inside the contour is rewarded, whereas for a positive ν it's penalized. We see that for

Figure 7: The algorithm's performance on noisy images

girl noisy

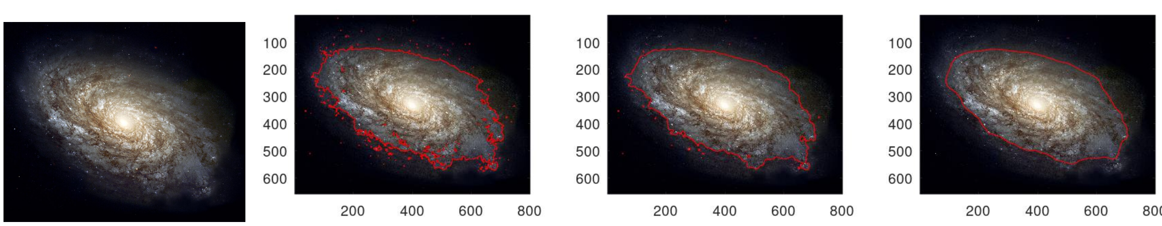
(a) Noise = 0.2 (b) Noise = 0.5 (c) Noise = 1



(d) 800 steps, Noise = 0.2 (e) 800 steps, Noise = 0.5 (f) 900 steps, Noise = 1

Figure 8: The effect of μ

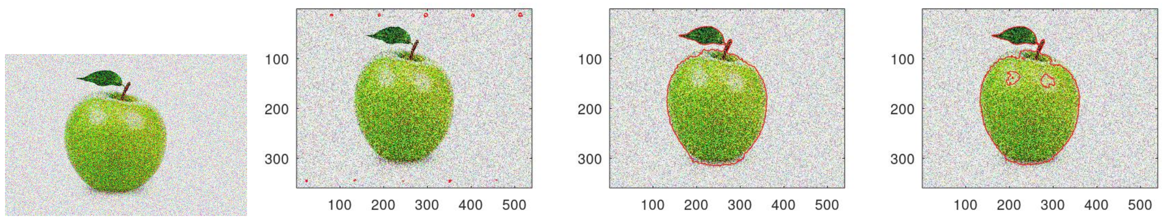
galaxy



(a) Original Image (b) μ=0.05, 400 steps (c) μ=0.2, 700 steps (d) μ=2, 4200 steps

Figure 9: The effect of ν

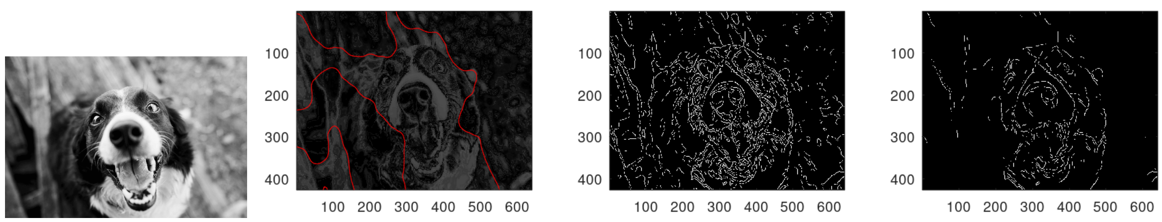
oisy apple



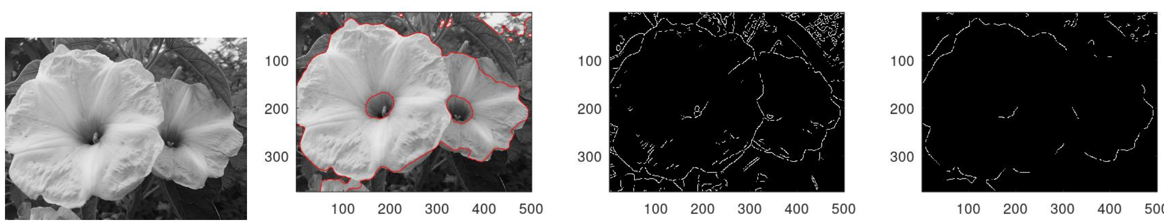
(a) Original (b) ν=-0.04, 300 steps (c) ν=0, 300 steps (d) ν=0.04, 300 steps

Figure 10: Chan-Vese vs Canny

canny



(a) Original (b) 4800 steps (c) Canny thresh=0.3 (d) Canny thresh=0.5



(e) Original (f) 1000 steps (g) Canny thresh=0.2 (h) Canny thresh=0.5

$\nu = -0.04$, the contour almost covers the entire image to maximize the area reward. For $\nu = 0.04$, we see an additional contour boundary inside the apple that decreases the area.

6.4 Comparison against Canny

We compare the method against Canny-Edge detection on grayscale images, using Octave's *edge* function from the *image* package. We note that Canny Edge detection is always much faster than Chan-Vese, with Chan-Vese requiring several minutes of computation compared to Canny outputting in less than a second each time. Figure 10 shows the results.

7 Improvement Directions

One disadvantage of Chan-Vese is that it only segments the image into two phases. An extension of Chan-Vese for nested segmentation is [5] and multiphase segmentation is [10, 8]. Note that by the 3 color theorem, we only need at most 3 phases in general to segment any image.

Another disadvantage is that the gradient descent based algorithm above runs slowly. Other approaches include [7] based on the topological derivative, [1] based on the multigrid method and [6, 2] based on graph cuts.

References

[1] Multigrid method for the chan-vese model in variational segmentation. *Communications in Computational Physics*, 4(2):294–316, 2008.

[2] Egil Bae and Xue-Cheng Tai. *Efficient Global Minimization for the Multiphase Chan-Vese Model of Image Segmentation*, page 28–41. Springer Berlin Heidelberg, 2009.

[3] Tony Chan and Luminita Vese. *An Active Contour Model without Edges*, page 141–151. Springer Berlin Heidelberg, 1999.

[4] Tony F. Chan, B.Yezrielev Sandberg, and Luminita A. Vese. Active contours without edges for vector-valued images. *Journal of Visual Communication and Image Representation*, 11(2):130–141, June 2000.

[5] Ginmo Chung and Luminita A. Vese. Energy minimization based segmentation and denoising using a multilayer level set approach. In Anand Rangarajan, Baba Vemuri, and Alan L. Yuille, editors, *Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 439–455, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[6] Noha El Zehiry, Steve Xu, Prasanna Sahoo, and Adel Elmaghraby. Graph cut optimization for the mumford-shah model. In *The Seventh IASTED International Conference on Visualization, Imaging and Image Processing*, VIIP '07, page 182–187, USA, 2007. ACTA Press.

[7] Lin He and Stanley Osher. *Solving the Chan-Vese Model by a Multiphase Level Set Algorithm Based on the Topological Derivative*, page 777–788. Springer Berlin Heidelberg.

[8] Matthew Keegan, Berta Sandberg, and Tony Chan. A multiphase logic framework for multichannel image segmentation. *Inverse Problems and Imaging*, 1, 02 2012.

[9] David Mumford and Jayant Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *Communications on Pure and Applied Mathematics*, 42(5):577–685, July 1989.

[10] Luminita A. Vese and Tony F. Chan. *International Journal of Computer Vision*, 50(3):271–293, 2002.

Shashank Sharma
Department of Mathematics, University of British Columbia, Canada.
sharm39@math.ubc.ca