M2 - Optimization and Inference Techniques for CV

3. Segmentation

Group 8

Alex Carrillo
Johnny Nuñez
Guillem Martínez

MSc in Computer Vision

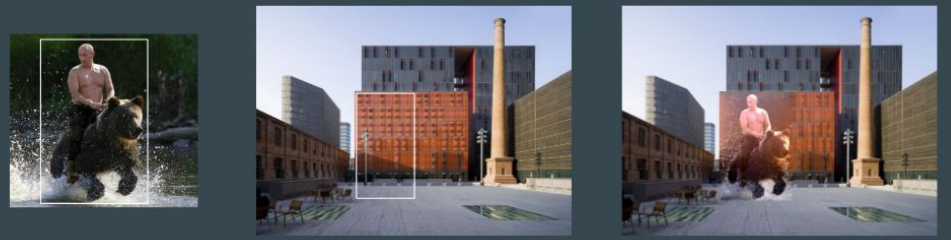
Recap: Week 3

Poisson editing:

- Technique in which damaged, deteriorated or missing regions of images are reconstructed by interpolation of surrounding areas

Conclusions and Results

- Notable results for inpainting small regions
- Bigger region, bigger blur in inpainted region, (may occur because):
  - The content available for inpainting is partial
  - The # of neighbours pixels to look at is not enough (4-connectivity)
- The smooth transition of the Laplacian operator is not enough to “hide” a visible region in an optimal way



Segmentation: Problem Definition

Definition

- Segmentation is the process of finding different regions or segments (set of pixels) that partitionate an image into **meaningful parts**.

Goal

- The goal is to simplify the representation of an image into something that is more meaningful and easier to analyze.

Result

- The result is an **image** that maps the different regions or segments (set of pixels).

original image

Result
segmentation image
(maps the region “horse” and “background”)

original image

Result
segmentation image
(maps multiple regions)

Segmentation: Problem Criteria

1. The segmentation result is an **image**
2. The segmentation image is **similar** to the original image
3. The segmented regions are **homogeneous**
4. The segmented regions have **smooth boundaries**
5. **The segmentation image has **two regions**

Mathematically:

original image

$$f \rightarrow u$$

segmentation image

$$\int_{\Omega} (f(x) - u(x))^2 dx$$

Ω *is the domain to be segmented*

$$\int_{\Omega \setminus C} |\nabla u(x)|^2 dx$$

C *is the edge-set curve where* ***u*** *is allowed to be discontinuous*

$$\arg\min_{u,C} \mu \text{ Length } (C)$$

$$u(x) = \begin{cases} c_1 & \text{where } x \text{ is inside } C \\ c_2 & \text{where } x \text{ is outside } C \end{cases}$$

Segmentation: The Mumford-Shah (simplified) solution

The Mumford-Shah (simplified) solution only considers criterias **1**, **2**, **3** and **4**.

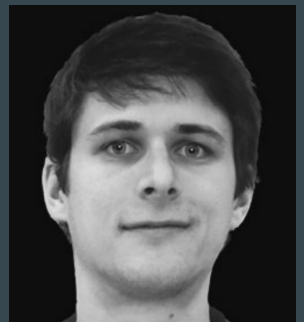
Thus, the result of the segmentation image does not have only two regions (criteria 5).

According to the criteria (mathematical explained in previous slide), the model solution can be described as the following (nonconvex minimization) optimization problem:

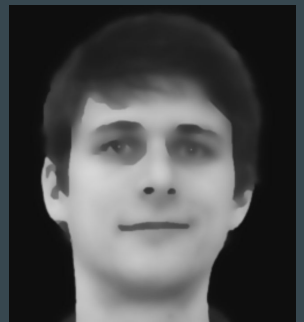
$$\arg\min_{u,C} \mu\,\mathrm{Length}(C) + \lambda \int_{\Omega} (f(x) - u(x))^2 dx + \int_{\Omega \setminus C} |\nabla u(x)|^2 dx$$

energy function

This method suggests selecting this edge-set curve ***C*** as the segmentation boundary.



*original image **f***



Mumford-Shah
piecewise-smooth approximation

Segmentation: The Chan-Vese solution

The Chan-Vese solution also considers criteria **5**, and adds:

1. an additional term **penalizing** the enclosed area
2. a further simplification (binarization) of result image ***u***

Note the hyper-parameters:

- ***μ*** length penalization
- ***v*** enclosed area penalization
- λ_1 fidelity weight term
- λ_2 fidelity weight term

1.

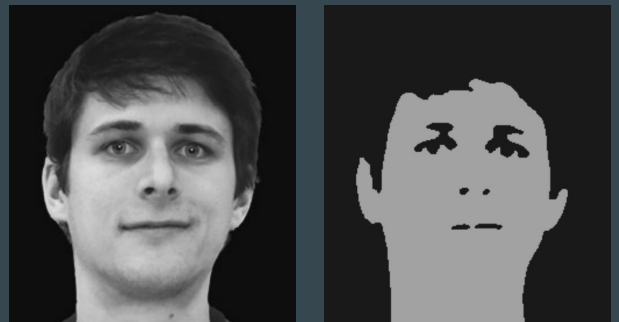
$$\arg\min_{c_1,c_2,C} \mu\,\mathrm{Length}(C) + \nu\,\mathrm{Area}(\mathrm{inside}(C)) + \lambda_1 \int_{\mathrm{inside}\,(C)} |f(x)-c_1|^2 dx + \lambda_2 \int_{\mathrm{outside}\,(C)} |f(x)-c_2|^2 dx.$$

energy function

$$u(x) = \begin{cases} c_1 & \text{where } x \text{ is inside } C \\ c_2 & \text{where } x \text{ is outside } C \end{cases}$$

2.

Chan-Vese binary approximation



*original image **f***

Segmentation: The Chan-Vese solution - Level Set Functions

The Chan-Vese minimization problem requires minimizing over all set boundaries ***C***.

Instead, we can use the zero-crossing of a level set function ***φ***, rather than C explicitly:

$$C = \{x \in \Omega : \varphi(x) = 0\}$$

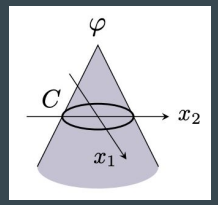
φ is a surface that intersects the image at the position of the contour ***C***.

Note there is more than one possible level set representation (active surface).

Thus, we can choose different initializations of ***φ*** such as:

$$\varphi(x) = r - \sqrt{x_1^2 + x_2^2}$$

a Cone



$$\varphi(x) = \sin\left(\frac{\pi}{5}x\right)\sin\left(\frac{\pi}{5}y\right)$$

a checkerboard shape

Segmentation: The Chan-Vese solution - Heaviside and Dirac

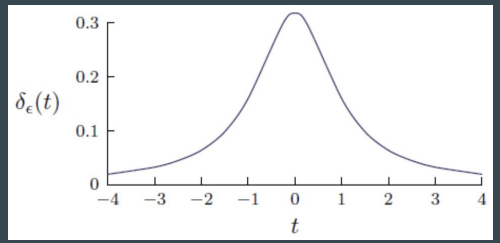
With the new level set function ***φ*** defined, the Chan-Vese minimization can be rewritten on ***φ*** as:

$$\arg\min_{c_1,c_2,\varphi} \mu \int_\Omega \delta(\varphi(x))|\nabla\varphi(x)|dx + \nu \int_\Omega H(\varphi(x))dx$$
$$+ \lambda_1 \int_\Omega |f(x)-c_1|^2 H(\varphi(x))dx + \lambda_2 \int_\Omega |f(x)-c_2|^2 (1-H(\varphi(x)))dx$$

energy function

Where ***H*** denotes the Heaviside function and **δ** the Dirac mass (its distributional derivative):

$$H(t) = \begin{cases} 1 & t \geq 0, \\ 0 & t < 0, \end{cases} \quad \delta(t) = \frac{d}{dt} H(t)$$



Segmentation: The Chan-Vese solution - Resolution implementation

The presented energy function can be optimized using the semi-implicit **Gradient Descent** numerical implementation, according to:

$$c_1 = \frac{\int_\Omega f(x)H(\varphi(x))dx}{\int_\Omega H(\varphi(x))dx}$$
$$c_2 = \frac{\int_\Omega f(x)(1-H(\varphi(x)))dx}{\int_\Omega (1-H(\varphi(x)))dx}$$

averages of each region 1 and 2 (corresponding to 0 or 1)

$$\varphi_{i,j}^{n+1} \leftarrow \Big[\varphi_{i,j}^n + dt\delta_\epsilon(\varphi_{i,j}^n)\Big(A_{i,j}\varphi_{i+1,j}^n + A_{i-1,j}\varphi_{i-1,j}^{n+1} + B_{i,j}\varphi_{i,j+1}^n + B_{i,j-1}\varphi_{i,j-1}^{n+1}$$
$$-\nu - \lambda_1(f_{i,j}-c_1)^2 + \lambda_2(f_{i,j}-c_2)^2\Big)\Big]$$
$$/[1 + dt\delta_\epsilon(\varphi_{i,j})(A_{i,j} + A_{i-1,j} + B_{i,j} + B_{i,j-1})].$$

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

Code Implementation (start.m)

1. Load image (if it is RGB average the channels)

2. Set all the parameters (set-up unless specified):

- Reinitialization to 0, — no reinitialization
- Max iterations to 100.000, — the algorithm will have time to converge
- Set μ to 1 and ν to 0, — length and area regularization parameters
- Set λ_1 and λ_2 to 1, — data fidelity parameters
- dt to 10^{-1} / μ, — time step parameter
- tol to 10^{-2}, — tolerance parameter
- η to 1, — total variation regularization parameter
- φ to a checkerboard initialization
  - It can also be initialize as a cone

3. Call the Chan-Vese algorithm

```
%%Explicit Gradient Descent
seg=sol_ChanVeseIpol_GDExp( I, phi_0, mu, nu, eta, lambda1, lambda2, tol, epHeaviside, dt, iterMax, reIni );
```

Code Implementation: φ Initializations (checkerboard.m and start.m)

- The **checkerboard initialization** is implemented as follows given the image size

```
function result = checkerboard( img_size_i, img_size_j, square_size)
%  Generates a checkerboard level set function.
%    According to Pascal Getreuer, such a level set function has fast
%    convergence.

xv = 1:img_size_j;
yv = (1:img_size_i)';
sf = pi / square_size;
xv = xv .* sf;
yv = yv .* sf;
result = sin(xv) .* sin(yv);
```

- The **cone initialization** is implemented as:

```
phi_0=(-sqrt( ( X-round(ni/2)).^2 + (Y-round(nj/2)).^2)+50);
```

Code Implementation: Heaviside function (sol_diracReg.m)

- The **Heaviside function** is implemented as follows:

```
function y = sol_diracReg( x, epsilon )
%  Dirac function of x
%    sol_diracReg( x, epsilon ) Computes the derivative of the heaviside
%    function of x with respect to x. Regularized based on epsilon.

y = epsilon ./ (pi*(epsilon.^2 + x.^2));
```

Code Implementation: (sol_ChanVeselpol_GDExp.m)

1. Dimensions, φ, difference and nIter set-up

```
[ni, nj]=size(I);
hi=1;
hj=1;

[X, Y]=meshgrid(1:nj, 1:ni);

i = 2:ni-1;
j = 2:nj-1;

phi=phi_0;
dif=inf;
nIter=0;
```

While the algorithm does not converge (difference between old and new φ lower than tolerance) or max iterations are not reached:

2. Set old φ and new iteration

```
phi_old=phi; % phi_old == n, phi == n+1
nIter=nIter+1;
```

3. Estimate the constants

```
%Fixed phi, Minimization w.r.t c1 and c2 (constant estimation)
c1 = sum(I(phi >= 0))/sum(sum(phi >= 0)); %done
c2 = sum(I(phi < 0))/sum(sum((phi < 0))); %done
```

4. Set boundary conditions

```
%Boundary conditions
phi(1,:)   = phi_old(2,:); %done
phi(end,:) = phi_old(end-1,:); %done
phi(:,1)   = phi_old(:,2); %done
phi(:,end) = phi_old(:,end-1); %done
```

5. Compute Δφ

```
%Regularized Dirac's Delta computation
delta_phi = sol_diracReg(phi, epHeaviside);
```

Code Implementation: (sol_ChanVeselpol_GDExp.m)

6. Compute forward, backward and central finite differences

```
%derivatives estimation
%i direction, forward finite differences
phi_iFwd  = DiFwd(phi, hi); %done
phi_iBwd  = DiBwd(phi, hi); %done

%j direction, forward finitie differences
phi_jFwd  = DjFwd(phi, hi); %done
phi_jBwd  = DjBwd(phi, hi); %done

%centered finite diferences
phi_icent = (phi_iFwd + phi_iBwd)/2; %done
phi_jcent = (phi_jFwd + phi_jBwd)/2; %done
```

7. Estimate A and B

```
%A and B estimation (A y B from the Pascal Getreuer's IPOL paper "Chan
%Vese segmentation
A = mu./sqrt(eta^2 + phi_iFwd.^2 +phi_jcent.^2); %TODO 13: Line to complete
B = mu./sqrt(eta^2 + phi_jFwd.^2 +phi_icent.^2); %TODO 14: Line to complete
```

*I(i-1, j-1) because image is NOT padded (but **phi** is padded)*

8. Compute the new φ

```
phi(i, j) = (phi_old(i, j) + dt .* delta_phi(i, j) .* ...
    (A(i, j) .* phi_old(i+1, j) + A(i-1, j).*phi(i-1, j) + ...
    B(i, j) .* phi_old(i, j+1) + B(i, j-1) .* phi(i, j-1) - ...
    nu - lambda1 .* ((I(i-1, j-1)-c1).^2) + lambda2 .* ((I(i-1, j-1)-c2).^2))) ./ ...
    (1 + dt .* delta_phi(i, j) .* (A(i, j) + A(i-1, j) + ...
    B(i, j) + B(i, j-1))); %done
```

9. Depending on nIter, reinitialize if wanted

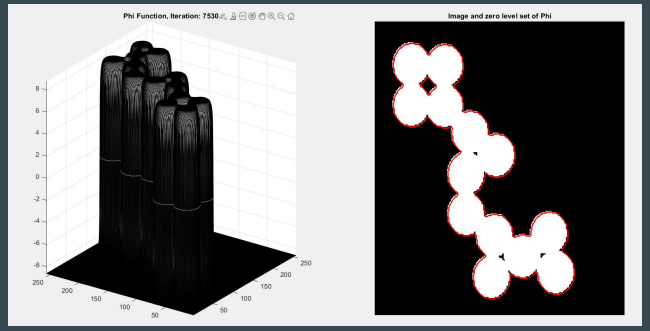
```
%Reinitialization of phi
if reIni>0 && mod(nIter, reIni)==0
    indGT = phi >= 0;
    indLT = phi < 0;

    phi=double(bwdist(indLT) - bwdist(indGT));

    %Normalization [-1 1]
    nor = min(abs(min(phi(:))), max(phi(:)));
    phi=phi/nor;
end
```

10. Calculate the difference between old and new φ

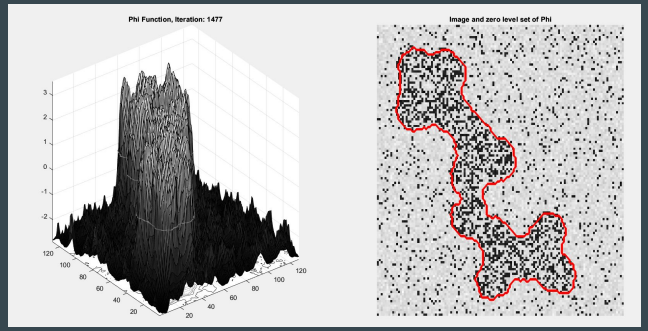
```
dif = mean(sum( (phi(:) - phi_old(:)).^2 ));
```

Results

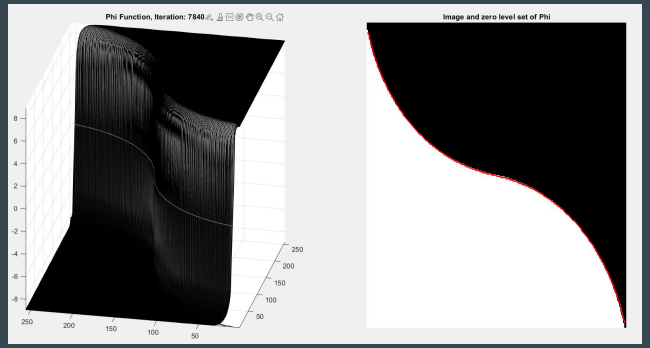
φ Cone - Convergence at 7530 steps



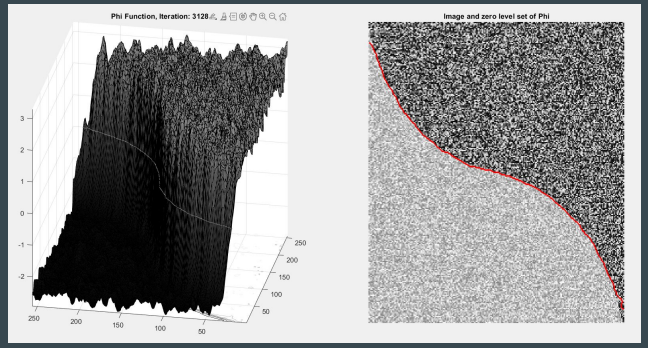
μ = 0.3 | φ Cone - Convergence at 1447 steps



Convergence at 7840 steps

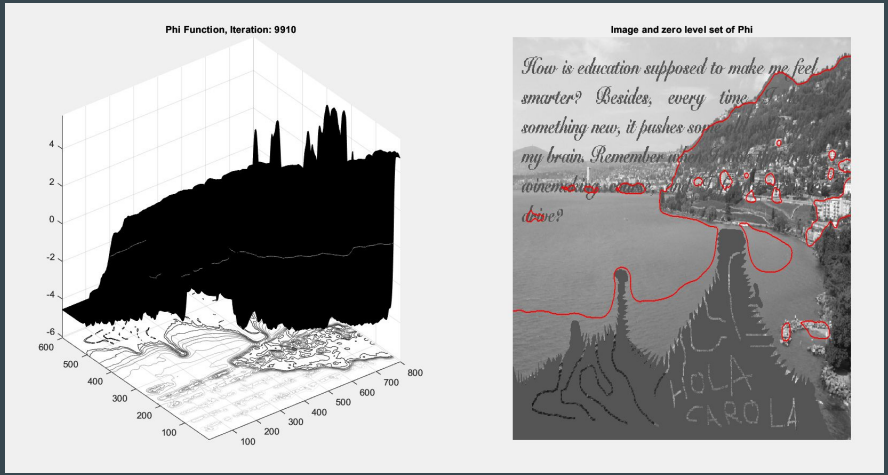


Convergence at 3128 steps

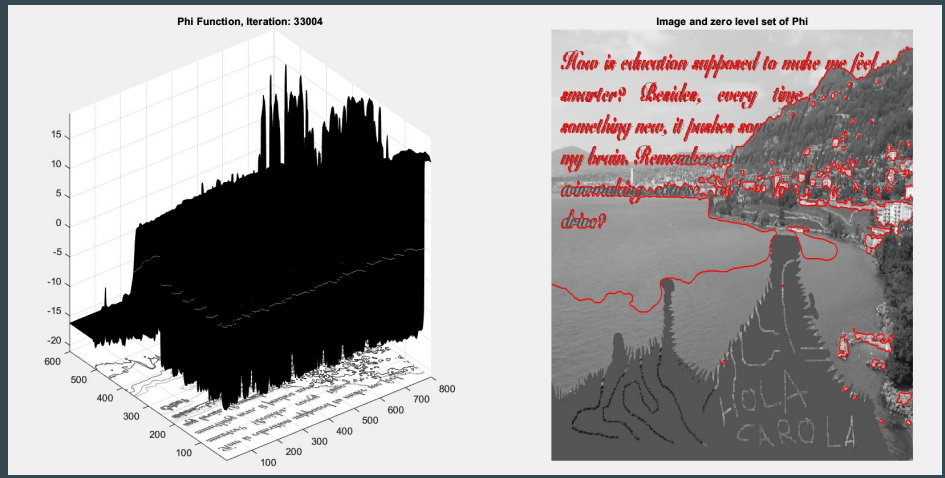


Results: Hola Carola

φ Image - Convergence at 9910 steps

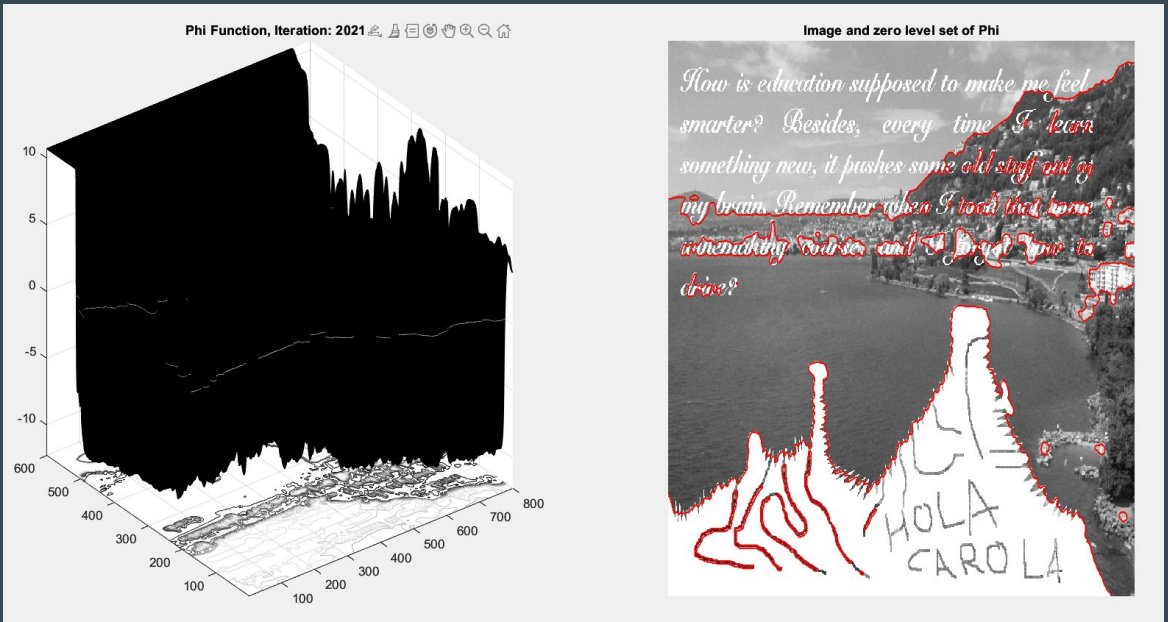


μ = 0.1 | φ Image - Convergence at 33004 steps



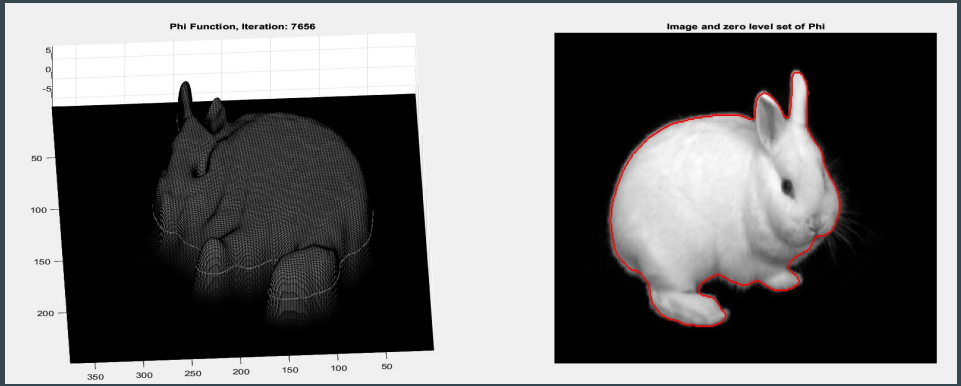
Results: Hola Carola using only the red channel

μ = 0.5 | φ Image - Convergence at 2021 steps

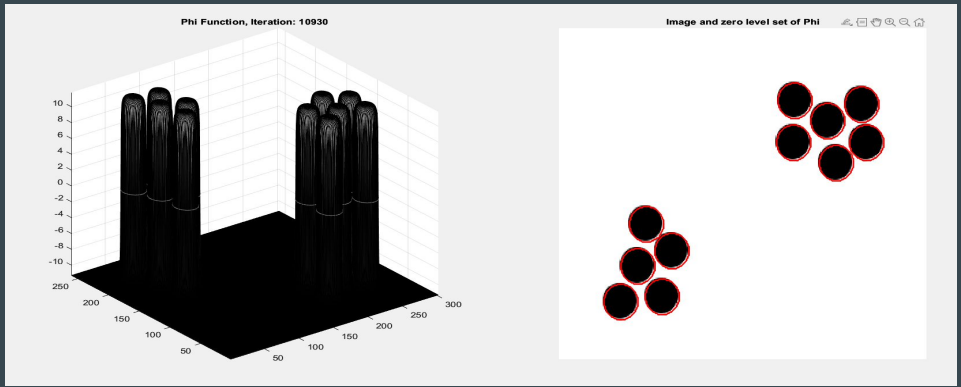


Custom Results

Convergence at 7656 steps

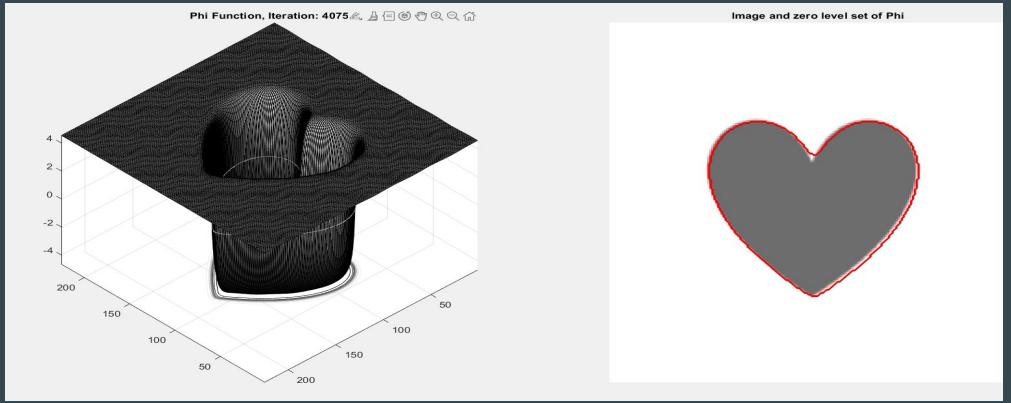


Convergence at 10936 steps



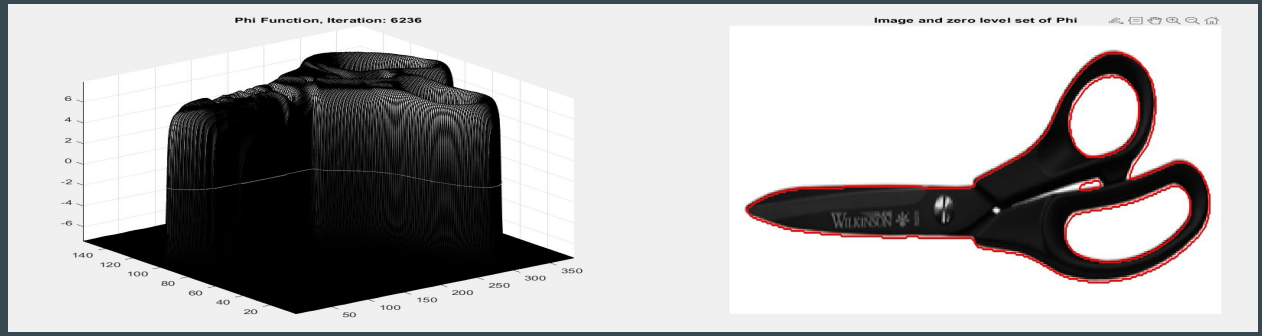
Custom Results

Convergence at 4075 steps

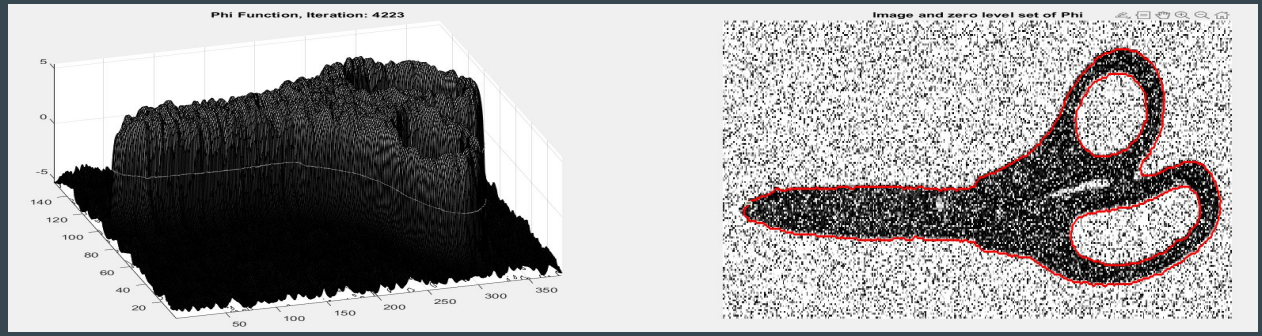


Custom Results

Convergence at 6236 steps

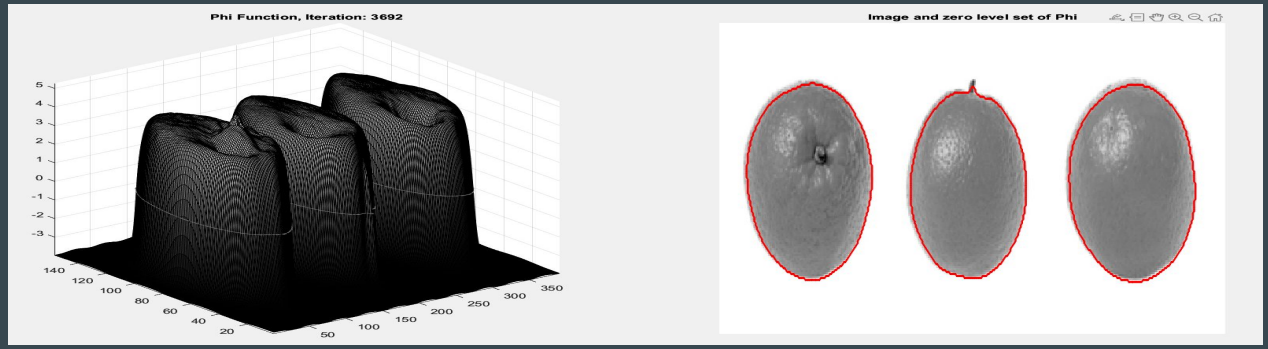


Convergence at 4223 steps

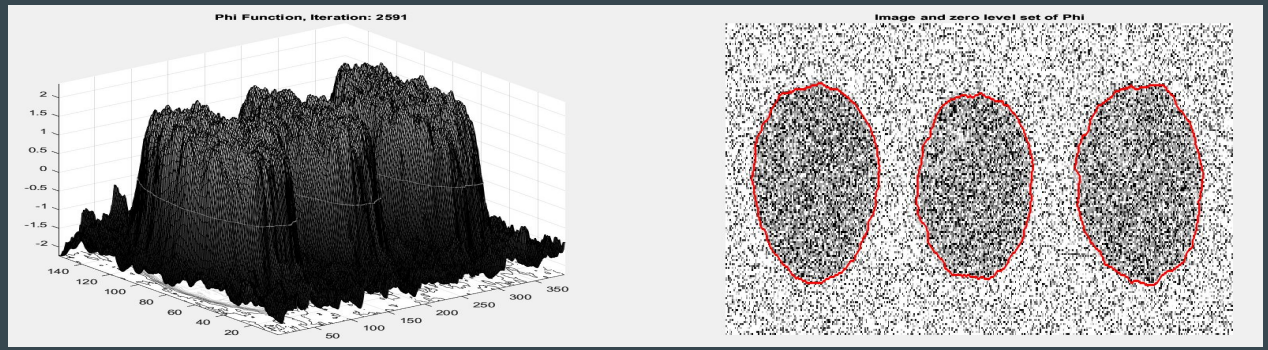


Custom Results

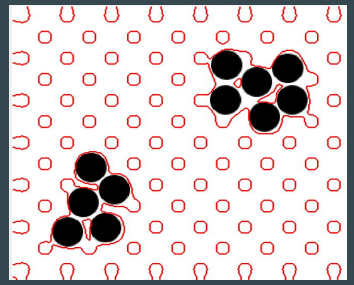
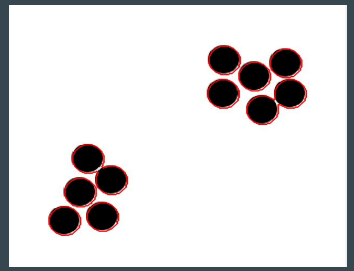
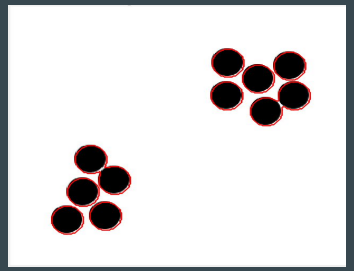
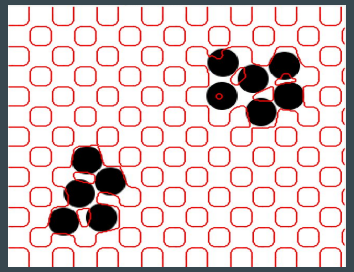
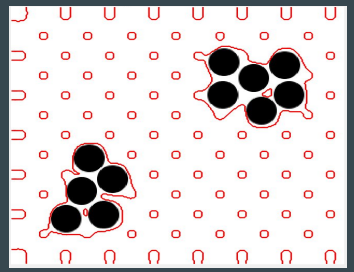
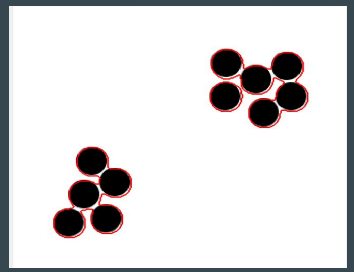
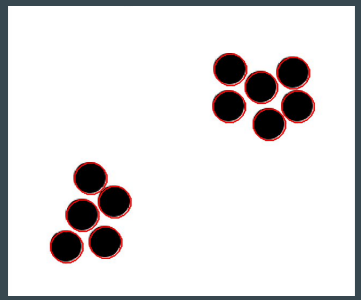
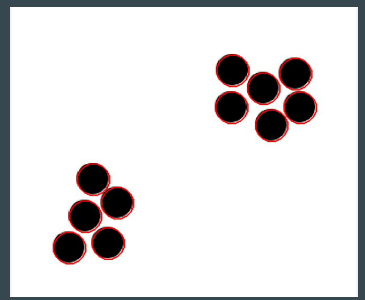
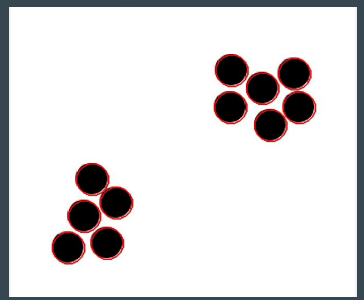
Convergence at 3692 steps



Convergence at 2591 steps

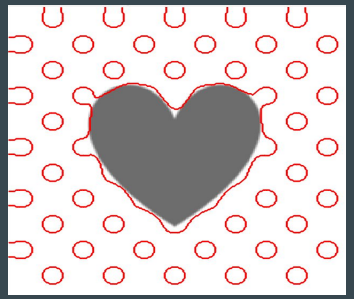
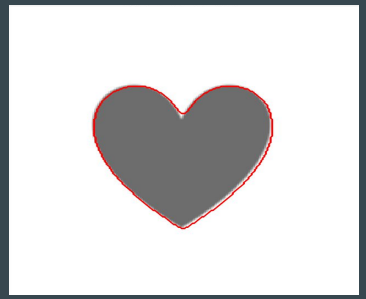
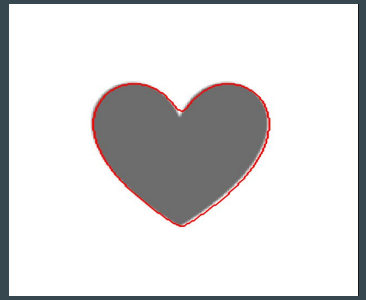
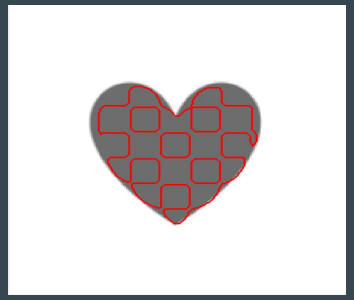
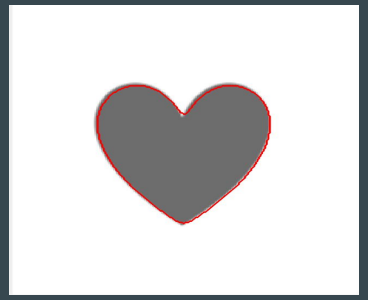
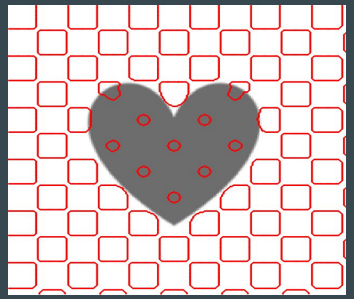
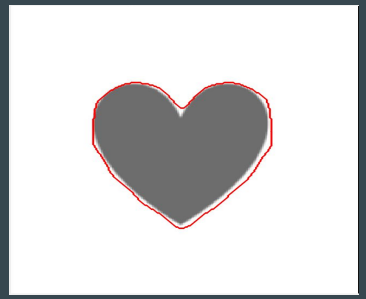
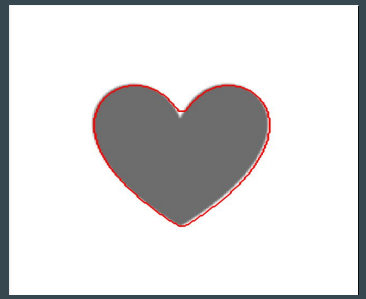


Custom Results - Effect of varying μ

	steps = 400	steps = 800	steps = 1200
μ = 1			
μ = 2.5			
μ = 0.1			

By varying μ we adjust the length penalty. In order to produce smoother or sharper boundaries.

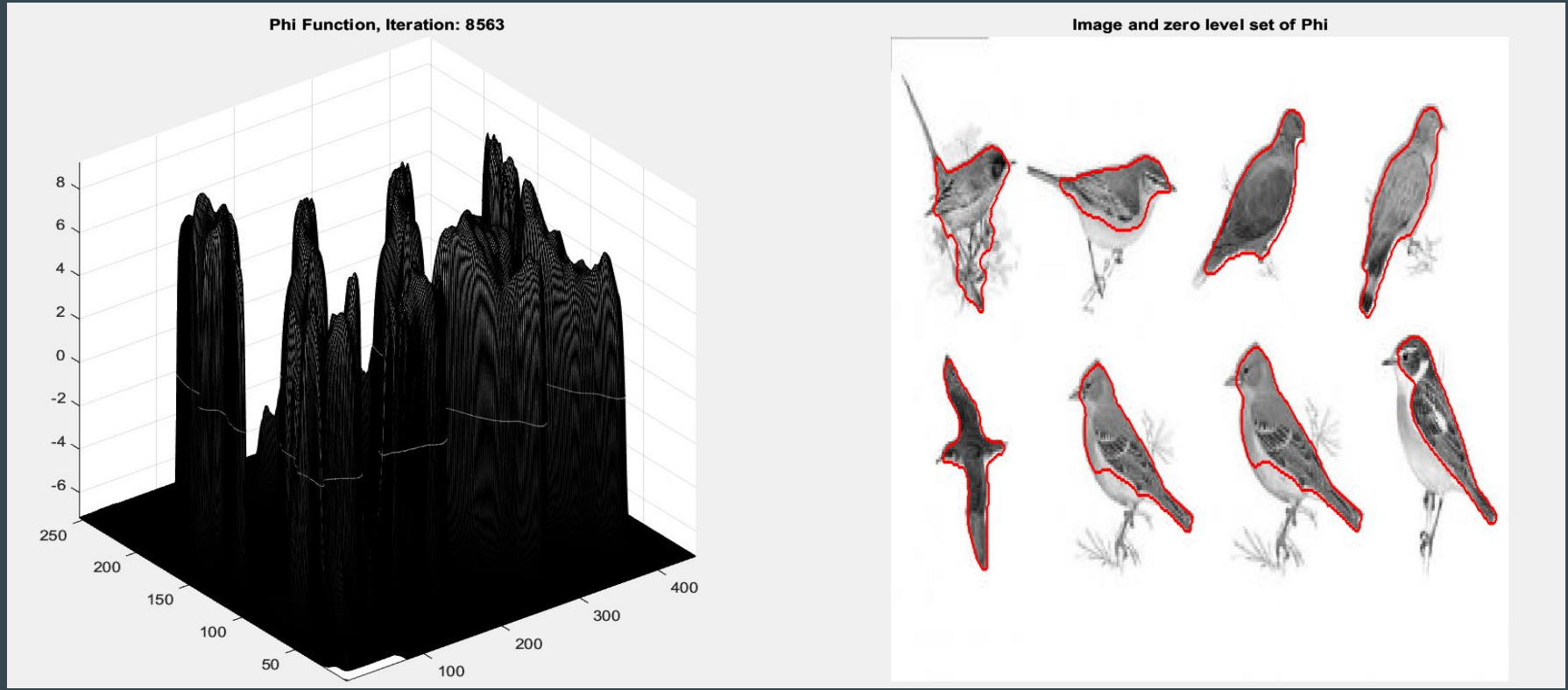
Custom Results - Effect of varying *v*

	steps = 400	steps = 800	Convergence
v = 0			
v = -0.1			
v = 0.01			

By varying ***v*** we adjust the area inside C. Values a bit different than zero can cause the algorithm to adjust the boundary to all the image (< 0) or vanish it (> 0).

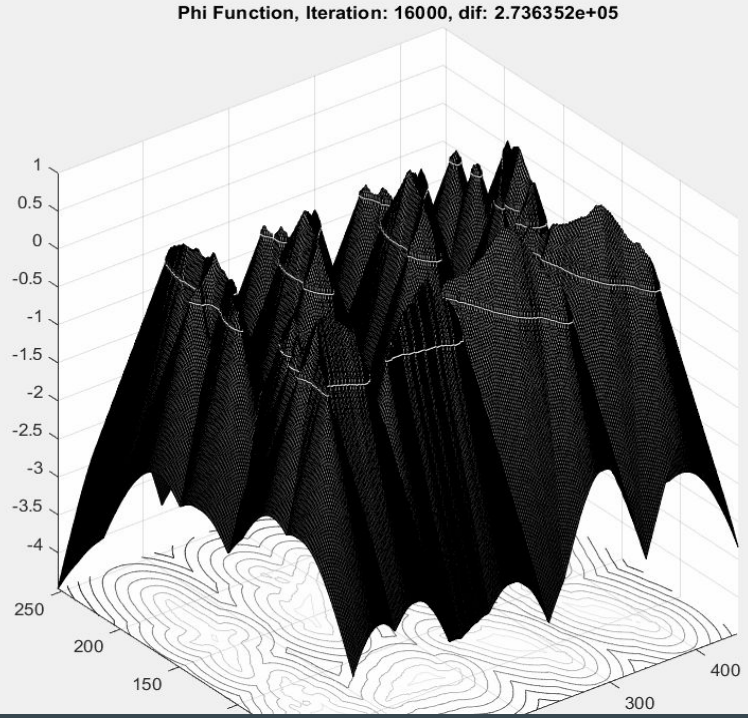
Custom Results - Effect of reinitializing

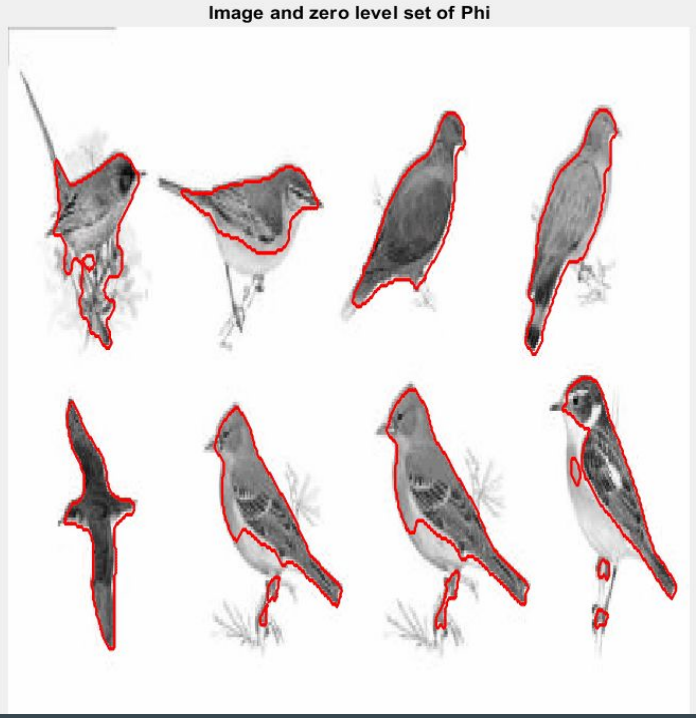
Without reinitialization: Convergence at 8563 steps



Custom Results - Effect of reinitializing

With reinitialization: **No convergence**





Through time, **new** segmentation **areas** appear with the reinitialization. It can be useful to find areas where that the algorithm discarded but at the same time, it can produce an **infinite loop** in some cases.

Conclusions

- The Chan-Vese algorithm is a suitable algorithm for a basic segmentation problem.
- By varying μ we adjust the length penalty. In order to produce smoother or sharper boundaries.
- By varying v we adjust the area inside C. Values a bit different than zero can cause the algorithm to adjust the boundary to all the image (< 0) or vanish it (> 0).
- Reinitialization can have a significant effect on the Chan-Vese algorithm. If the initialization is far from the desired solution, the algorithm may not be able to find the desired solution. However, it can also lead the algorithm to not converge
- The amount of regularization term Φ affects the Chan-Vese algorithm in a few ways. A higher value of Φ will result in a smoother segmentation, while a lower value of Φ will allow for more details in the segmentation.