

An Efficient Lagrangian Algorithm for an Anisotropic Geodesic Active Contour Model

Günay Doğan

Theiss Research, La Jolla, CA 92037, USA

National Institute of Standards and Technology, Gaithersburg, MD 20899, USA

g.dogan@theissresearch.org

Abstract. We propose an efficient algorithm to minimize an anisotropic surface energy generalizing the Geodesic Active Contour model for image segmentation. In this energy, the weight function may depend on the normal of the curve/surface. Our algorithm is Lagrangian, but non-parametric. We only use the node and connectivity information for computations. Our approach provides a flexible scheme, in the sense that it allows to easily incorporate the generalized gradients proposed recently, especially those based on the H^1 scalar product on the surface. However, unlike these approaches, our scheme is applicable in any number of dimensions, such as surfaces in 3d or 4d, and allows weighted H^1 scalar products, with weights may depending on the normal and the curvature. We derive the second shape derivative of the anisotropic surface energy, and use it as the basis for a new weighted H^1 scalar product. In this way, we obtain a Newton-type method that not only gives smoother flows, but also converges in fewer iterations and much shorter time.

1 Introduction

Finding the boundaries of objects or regions in images is a fundamental problem in computer vision. Active contour methods proposed in the eighties have been a very successful approach to solve this problem. One of the pioneering methods of this approach is the Geodesic Active Contour (GAC) model by Caselles et al. In [4], they propose finding the boundaries as curves Γ that correspond to the minima of a weighted length integral

$$J_{GAC}(\Gamma) = \int_{\Gamma} \rho(|\nabla I(x)|) d\sigma, \quad (1)$$

where $\rho(s) = (1 + s^2/\lambda^2)^{-1}$, $\lambda > 0$ and $I : \mathcal{D} \subset \mathbb{R}^2 \rightarrow \mathbb{R}$ is the smoothed image intensity function. Later Caselles et al. extend this to 3d in [5] to extract surfaces in volumetric images.

Following [4], many extensions and intricate energies were proposed to address the challenges of the segmentation problem. However the model (1) has maintained its value and is still widely used. In this paper, we consider an anisotropic variant of (1) in a more general setting

$$J(\Gamma) = \int_{\Gamma} g(x, n) d\sigma, \quad (2)$$

where Γ is an $(d-1)$ -dimensional surface in a d -dimensional space. The energy $J(\Gamma)$ is an anisotropically weighted surface integral. The weight function g depends on the normal n of the surface Γ , as well as the spatial coordinates x of the surface. This variation has found application in edge integration [15] and variational stereo [10, 11, 14].

To devise minimization schemes for (2), we need to compute deformation velocities \mathbf{V} to evolve the surface Γ in a way that will decrease its energy. A crucial step for this is to quantify the effect of a candidate velocity \mathbf{V} on the energy $J(\Gamma)$. For this, we use the concept of shape derivatives [8], which we define in Section 2. The shape derivative of $J(\Gamma)$ with respect to a velocity \mathbf{V} is given by $dJ(\Gamma; \mathbf{V}) = \int_{\Gamma} (g\kappa + \partial_n g + \text{div}_{\Gamma}(g_y)_{\Gamma}) V d\sigma$, where $V = \mathbf{V} \cdot n$ and κ is the mean curvature of Γ , g_y denotes the derivative of g with respect to the normal variable, div_{Γ} is the tangential divergence operator defined in Section 2. This has been known in the area of geometric flows [7], but was recently rederived in computer vision literature using method of moving frame [12] and level set formalism [21]. If we choose

$$V = -(g\kappa + \partial_n g + \text{div}_{\Gamma}(g_y)_{\Gamma}), \quad (3)$$

we ensure $dJ(\Gamma; \mathbf{V}) \leq 0$, hence energy decrease for $J(\Gamma)$. This choice commonly used in computer vision corresponds to using an L^2 metric for the velocity space. Recently H^1 metric has been proposed by Charpiat et al. [6] and Sundaramoorthi et al. [23]. When applied to the surface energy (2), this corresponds to the following velocity equation

$$\alpha_0 \Delta_{\Gamma} V + V = -(g\kappa + \partial_n g + \text{div}_{\Gamma}(g_y)_{\Gamma}), \quad (4)$$

where $\alpha_0 \geq 0$ is a constant and Δ_{Γ} denotes the tangential Laplacian operator defined in Section 2. This equation yields spatially coherent velocities, therefore smoother evolutions. In [6, 23], the H^1 metric is considered in 2d for curves only and a numerical solution is proposed by turning it into an ODE and computing the solution with a convolution.

In this paper, we will consider the weak form of a more general version of (4)

$$\langle \alpha \nabla_{\Gamma} V, \nabla_{\Gamma} \phi \rangle + \langle \beta V, \phi \rangle = -\langle g\kappa, \phi \rangle - \langle \partial_n g, \phi \rangle + \langle (g_y)_{\Gamma}, \nabla_{\Gamma} \phi \rangle, \quad \forall \phi \in H^1(\Gamma), \quad (5)$$

where $\alpha = \alpha(x, n, \kappa) \geq 0$, $\beta = \beta(x, n, \kappa) \geq 0$ and ∇_{Γ} denotes the tangential gradient defined in 2. This more general velocity equation provides new opportunities to compute better descent velocities for (2). In particular, we can derive the second shape derivative of (2) and leverage (5) to implement Newton-type minimization schemes (see [1] for a trust-region-Newton method). *The second shape derivative of (2) is the first contribution of this paper* (see [13] for GAC energy (1)), and it will enable us to achieve faster convergence, and robustness to varying image conditions.

We will also propose a Lagrangian computational scheme based on the finite element method (FEM) to compute the velocity V from (5) and to deform the surface Γ with V decreasing its energy $J(\Gamma)$. *This FEM-based velocity scheme is the main contribution of our paper and has the following advantages:*

- The scheme is applicable in any number of dimensions including surface in 3d and 4d, unlike the schemes proposed in [6] and [23], applicable in 2d only.
- The weights α, β in (5) are not constant as in [6, 23]. They can in fact be more general functions and depend on the position x , the normal n and the mean curvature κ of the surface Γ , and we exploit this to implement fast Newton-type minimization schemes.
- Although the scheme is Lagrangian, it is nonparametric, that is, we do not parametrize the surface Γ . We only work with the list of simplices that represent Γ . This greatly simplifies the implementation.
- To compute the velocity V using (5), we only need first derivatives g_x, g_y of g . This is in contrast to the previous approaches [10], [14] that use (3) within a level set framework. They need to account for the term $\text{div}_\Gamma(g_y)_\Gamma$, which requires second derivatives of g . As level set discretization of this term is very tedious, it is often ignored.

2 Shape Derivatives of the Energy

We use the concept of shape derivatives to understand the change in the energy induced by a given velocity field \mathbf{V} . Once we have the means to evaluate how any given velocity \mathbf{V} affects the energy, we will be able to choose a descent velocity from the space of admissible velocities, namely a velocity that decreases the energy for a given surface Γ .

Before we start deriving the shape derivatives of (2), we need some definitions and concepts from differential geometry. We denote the outer unit normal, the scalar (total or mean) curvature and the curvature vector of surface $\Gamma \in C^2$ by n , κ , $\boldsymbol{\kappa} := \kappa n$ respectively. For given functions $f, \mathbf{w} \in C^2(\mathcal{D})$ on image domain \mathcal{D} , we define the tangential gradient $\nabla_\Gamma f = (\nabla f - \partial_n f n)|_\Gamma$, tangential divergence $\text{div}_\Gamma \mathbf{w} = (\text{div} \mathbf{w} - n \cdot D\mathbf{w} \cdot n)|_\Gamma$, tangential Laplacian $\Delta_\Gamma f = (\Delta f - n \cdot D^2 f \cdot n - \kappa \partial_n f)|_\Gamma$. We define the *shape derivative* of energy $J(\Gamma)$ at Γ with respect to velocity field \mathbf{V} as the limit $dJ(\Gamma; \mathbf{V}) = \lim_{t \rightarrow 0} \frac{1}{t} (J(\Gamma_t) - J(\Gamma))$, where $\Gamma_t = \{x(t, X) : X \in \Gamma\}$ is the deformation of Γ by \mathbf{V} via equation $\frac{dx}{dt} = \mathbf{V}(x(t))$, $x(0) = X$ [8, 20]. For a surface-dependent function $\psi(\Gamma)$, the *material derivative* $\dot{\psi}(\Gamma; \mathbf{V})$ and the *shape derivative* $\psi'(\Gamma; \mathbf{V})$ at Γ in direction \mathbf{V} are defined as follows [20, Def. 2.85, 2.88]:

$\dot{\psi}(\Omega; \mathbf{V}) = \lim_{t \rightarrow 0} \frac{1}{t} (\psi(x(t, \cdot), \Gamma_t) - \psi(\cdot, \Gamma_0))$, $\psi'(\Gamma; \mathbf{V}) = \dot{\psi}(\Gamma; \mathbf{V}) - \nabla_\Gamma \psi \cdot \mathbf{V}$. The 2nd shape derivatives defined as: $\psi''(\Gamma; \mathbf{V}, \mathbf{W}) = (\psi'(\Gamma; \mathbf{V}))'(\Gamma; \mathbf{W})$, and $d^2 J(\Gamma; \mathbf{V}, \mathbf{W}) = d(dJ(\Gamma; \mathbf{V}))(\Gamma; \mathbf{W})$.

With these definitions, we can now calculate the shape derivative of the weighted surface energy (2). In the following, V denotes $\mathbf{V} \cdot n$ the normal component of the vector velocity.

Lemma 1 ([13, Sect. 3]). *The shape derivative $n'(\Gamma; \mathbf{V})$ of the normal n of the surface Γ in direction \mathbf{V} is given by $n' = n'(\Gamma; \mathbf{V}) = -\nabla_\Gamma V$.*

Theorem 1 ([20, Sect. 2.33]). *Let $\psi = \psi(x, \Gamma)$ be given so that $\dot{\psi}(\Gamma; \mathbf{V})$, $\psi'(\Gamma; \mathbf{V})$ exist. Then $J(\Gamma) = \int_\Gamma \psi(x, \Gamma) d\sigma$ is shape differentiable and we have $dJ(\Gamma; \mathbf{V}) = \int_\Gamma \psi'(\Gamma; \mathbf{V}) d\sigma + \int_\Gamma (\partial_\nu \psi + \kappa \psi) V d\sigma$.*

Notice that the shape derivative depends only on the normal component V of \mathbf{V} . Therefore, from this point on, we work with scalar velocities V so that $\mathbf{V} = Vn$.

Theorem 2. Let $\psi = \psi(x, \Gamma)$ be given so that $\psi'(\Gamma; V)$, $\psi''(\Gamma; V, W)$ exist. Then, the 2nd shape derivative of $J(\Gamma) = \int_{\Gamma} \psi(x, \Gamma) d\sigma$ with respect to V, W is

$$\begin{aligned} d^2 J(\Gamma; V, W) &= \int_{\Gamma} \psi''(\Gamma; V, W) d\sigma + \int_{\Gamma} \psi \nabla_{\Gamma} V \cdot \nabla_{\Gamma} W + (\kappa^2 - \sum \kappa_i^2) \psi V W d\sigma \\ &\quad + \int_{\Gamma} (\partial_n \psi'(\Gamma; W) + \kappa \psi'(\Gamma; W)) V + (\partial_n \psi'(\Gamma; V) + \kappa \psi'(\Gamma; V)) W d\sigma. \end{aligned}$$

In the following, g_x, g_y denote the derivatives of $g(x, n)$ with respect to the first variable x and the second variable n respectively.

Proposition 1. The shape derivative of (2) at Γ with respect to V is

$$dJ(\Gamma; V) = \int_{\Gamma} (\kappa g + \partial_n g) V - g_y \cdot \nabla_{\Gamma} V d\sigma = \int_{\Gamma} (\kappa g + \partial_n g + \operatorname{div}_{\Gamma}(g_y)_{\Gamma}) V d\sigma.$$

Proof. We use Theorem 1 with $\psi = g(x, n)$. Then $\psi'(\Gamma; V) = g_y \cdot n' = -g_y \cdot \nabla_{\Gamma} V$ using Lemma 1. We also need to compute the normal derivative of $g(x, n)$. We have $\partial_n \psi = \partial_n(g(x, n)) = \partial_n g + g_y^T \partial_n n = \partial_n g$ (note $\partial_n n = 0$).

We substitute $\psi', \partial_n \psi$ in Thm 1 $\Rightarrow dJ(\Gamma; V) = \int_{\Gamma} (\kappa g + \partial_n g) V - g_y \cdot \nabla_{\Gamma} V d\sigma$. We apply the tangential Green's formula to the last term of the integral, and use the identity $\operatorname{div}_{\Gamma}(\omega)_{\Gamma} = \operatorname{div}_{\Gamma}(\omega) - \kappa \omega \cdot n$ [8, Chap. 8] to obtain the result.

Proposition 2. The 2nd shape deriv. $d^2 J(\Gamma; V, W)$ of (2) w.r.t. velocities V, W

$$\begin{aligned} d^2 J &= \int_{\Gamma} \nabla_{\Gamma} V \cdot ((g - g_y \cdot n) Id + g_{yy}) \cdot \nabla_{\Gamma} W d\sigma + \int_{\Gamma} \left(\frac{\partial^2 g}{\partial n^2} + 2\kappa \frac{\partial g}{\partial n} + (\kappa^2 - \sum \kappa_i^2) g \right) V W d\sigma \\ &\quad - \int_{\Gamma} (\kappa g_y - g_y^T \nabla_{\Gamma} n + n^T g_{xy}^T) \cdot (\nabla_{\Gamma} W V + \nabla_{\Gamma} V W) d\sigma. \end{aligned}$$

Proof. $\psi = g(x, n)$ in Thm.2 $\Rightarrow \partial_n \psi = \partial_n g, \partial_{nn} \psi = \partial_{nn} g, \psi'(\Gamma; V) = -g_y \cdot \nabla_{\Gamma} V$,

$$\begin{aligned} \psi''(\Gamma; V, W) &= -(g_y)' \cdot \nabla_{\Gamma} V - g_y \cdot (\nabla_{\Gamma} V)' = \nabla_{\Gamma} V \cdot g_{yy} \cdot \nabla_{\Gamma} W - g_y \cdot n \nabla_{\Gamma} V \cdot \nabla_{\Gamma} W \\ &= \nabla_{\Gamma} V \cdot (g_{yy} - g_y \cdot n Id) \cdot \nabla_{\Gamma} W \\ \partial_n \psi'(\Gamma; V) &= -\partial_n g_y \cdot \nabla_{\Gamma} V - g_y \cdot \partial_n \nabla_{\Gamma} V = -(g_{yx} n) \cdot \nabla_{\Gamma} V - g_y \cdot (-\nabla_{\Gamma} n \nabla_{\Gamma} V) \\ &= (g_y^T \nabla_{\Gamma} n - n^T g_{yx}^T) \nabla_{\Gamma} V, \end{aligned}$$

since $\partial_n \nabla_{\Gamma} V = -\nabla_{\Gamma} n \nabla_{\Gamma} V$. Substitute deriv.s of ψ in Thm.(2), and reorganize.

3 The Minimization Algorithm

Given shape derivatives of surface energy (2) in Section 2, we can develop an iterative minimization algorithm to compute minimal surfaces as follows:

choose an initial surface Γ^0 .

repeat

 compute the descent velocity \mathbf{V}^k .

 choose step size τ^k .

 update the surface points $\mathbf{X}^{k+1} = \mathbf{X}^k + \tau^k \mathbf{V}^k$, $\forall \mathbf{X}^k \in \Gamma^k$.

until stopping criterion is satisfied.

This is a well-known approach. However, realizing an effective algorithm that will always converge, with a small number of iterations and a small computational cost, is not straight-forward. One needs to design the three main components of the algorithm carefully to ensure effectiveness for a diverse set of image inputs. These are: *stopping criterion*, *step size τ^k selection*, and *the gradient descent velocity \mathbf{V}^k* . Wrong stopping criteria may lead to premature termination of iterations or no convergence. The selection of the step sizes has an impact on convergence as well. Cautious small steps at each iteration can ensure convergence, but may result in too many iterations and long computation times, whereas large steps can enable fast convergence, but might miss the local minima that we need to capture. Moreover, in some iteration schemes, large step sizes can create instabilities in the surface evolution, manifested as noisy or oscillatory geometric patterns on the surface. We describe our solutions for these below.

Step size selection: Minimization problems of this type have traditionally been formulated as surface evolution problems with fixed step sizes. However, with fixed steps, one may miss the minima, or the iterations may not converge. For this reason, we use the Armijo criterion [13, 18] to select a step $10^{-4} \leq \tau^k \leq 1$ ensuring energy decrease at iteration k . Our initial candidate for a step at iteration k is $\min(2\tau^{k-1}, \tau_{\max}^k)$ where is τ_{\max}^k is a safe-guard maximum step that can be taken at iteration k and we set $\tau^0 = 0.01$. We accept and use the step τ^k if it satisfies the following energy decrease condition,

$$J(\Gamma^{k+1}) < J(\Gamma^k) + \eta \tau^k dJ(\Gamma^k; \mathbf{V}^k), \quad (6)$$

where is η a small positive number, $\eta = 10^{-3}$ in our experiments. If condition (6) is not satisfied, we reduce this step (divide by two) until it is satisfied.

Since our goal is to reach a *local* minimum of (2), we do not want to move the surface Γ too much at each iteration (not to miss a minimum). Therefore, we impose an iteration-dependent max step τ_{\max}^k , based on the maximum displacement Γ can take without missing an important feature of energy landscape. If δ_I is the image-dependent bound on the displacement we allow, then the maximum step is given by $\tau_{\max}^k = \delta_I / \max(V)$, in which the bound δ_I can be set to the average width of a valley of the energy weight $g(x, n)$ or the average edge width of the image function $I(x)$. In our examples, we used $\delta_I = 2\sigma$, where σ is the standard deviation of Gaussian $G_\sigma(x)$ applied to the image $I(x)$ to define the isotropic edge indicator function $g(x) = 1/(1 + |\nabla G_\sigma * I(x)|^2 / \lambda^2)$, $\lambda = \frac{1}{4} \max |\nabla G_\sigma * I| > 0$.

Stopping criterion: In finite-dimensional optimization, the iterations are stopped when the norm of the energy gradient is below a given threshold. A typical choice for this norm is the Euclidean norm. This is not applicable to the shape gradient, because it is a mapping defined on the surface, and the

surface itself changes through the iterations. To address these issues, we use the L^2 norm $\|G\|_{L^2(\Gamma)} = (\int_{\Gamma} |G|^2 d\sigma)^{1/2}$ of the shape gradient $G = g\kappa + \partial_n g + \text{div}_{\Gamma}(g_y)_{\Gamma}$ and accompanying shape gradient thresholds $\varepsilon_{abs}, \varepsilon_{rel}$ to realize a stopping criterion. If we require the pointwise value $|G(x)|$ of the shape gradient at the optimal surface Γ^* to be a small fraction of a pointwise maximum value $G_{max} \approx \max_{x \in D, n \in S^1} |G(x, n)|$, then the following threshold for the L^2 norm can be used

$$\|G\|_{L^2(\Gamma)} < (\varepsilon_{abs} + \varepsilon_{rel} G_{max}) |\Gamma|^{1/2}. \quad (7)$$

In our experiments, we set $\varepsilon_{abs} = 0.1$, $\varepsilon_{rel} = 0.01$, and use the quantity $\max |g|/\delta_I$ as an indicator of the scale of G_{max} . In addition to the stopping criterion (7), we monitor the energy change. If step size selection cannot provide an acceptable step τ^k satisfying (6) before convergence (7), then we check energy change in recent iterations. If the change has been very small, i.e. $|J(\Gamma^k) - \bar{J}| < \varepsilon_J |\bar{J}|$ then we terminate iterations. We set $\varepsilon_J = 10^{-4}$, and $\bar{J} = \frac{1}{n_J} \sum_{l=1}^{n_J} J(\Gamma^l)$ is the average energy for the last $n_J = 5$ iterations.

A simple descent velocity: An obvious choice that is commonly used in literature is to set the velocity equal to the negative shape gradient

$$V = -G = -(g\kappa + \partial_n g + \text{div}_{\Gamma}(g_y)_{\Gamma}). \quad (8)$$

This clearly is a descent velocity as $dJ(\Gamma; V) = \int_{\Gamma} GV d\sigma = -\int_{\Gamma} G^2 d\sigma \leq 0$. The velocity (8) was used in [15, 10, 14] for energy minimization. There are two downsides to (8): 1) The curvature term $g\kappa$ makes the geometric evolution unstable, requires very small steps. 2) The term $\text{div}_{\Gamma}(g_y)_{\Gamma}$ requires two derivatives on data function g and is very tedious to discretize when expanded explicitly.

The first downside can be alleviated by pursuing a *semi-implicit stepping scheme*. We recall the identities, $\kappa = -\Delta_{\Gamma} \mathbf{X}$, $\kappa = \kappa \cdot n$, $\mathbf{V} = Vn$, relating the position vector \mathbf{X} , the scalar and vector curvatures $\kappa, \boldsymbol{\kappa}$, the scalar and vector velocities V, \mathbf{V} . Then instead of using the following explicit update sequence at each iteration, $\boldsymbol{\kappa}^k = -\Delta_{\Gamma} \mathbf{X}^k \rightarrow \kappa^k = \boldsymbol{\kappa}^k \cdot n^k \rightarrow V^k = -(g\kappa^k + f(\Gamma^k)) \rightarrow \mathbf{V}^k = V^k n^k \rightarrow \mathbf{X}^{k+1} = \mathbf{X}^k + \tau^k \mathbf{V}^k$ ($f(\Gamma) := \partial_n g + \text{div}_{\Gamma}(g_y)_{\Gamma}$), we keep $\kappa^{k+1}, \boldsymbol{\kappa}^{k+1}, V^{k+1}, \mathbf{V}^{k+1}$ as unknowns to be evaluated at the next iteration. But this requires us solve the following system of equations at each iteration to compute the velocity from the known points \mathbf{X}^k and normals n^k of the current surface Γ^k : $\mathbf{V}^{k+1} - V^{k+1} n^k = 0$,

$$\boldsymbol{\kappa}^{k+1} - \tau \Delta_{\Gamma} \mathbf{V}^{k+1} = -\Delta_{\Gamma} \mathbf{X}^k, \kappa^{k+1} - \boldsymbol{\kappa}^{k+1} \cdot n^k = 0, V^{k+1} + g\kappa^{k+1} = -f^k. \quad (9)$$

This semi-implicit scheme is unconditionally stable, i.e. no stability bound imposed on step size τ^k , but it requires more computation per iteration as we handle a larger number of unknowns at each iteration, and solve a large system of equations, in contrast to the simple update sequence ($\mathbf{X}^k \rightarrow \boldsymbol{\kappa}^k \rightarrow \kappa^k \rightarrow V^k \rightarrow \mathbf{V}^k \rightarrow \mathbf{X}^{k+1}$) in the explicit scheme.

The 2nd downside can be alleviated by writing velocity eqn $V = -G$ in weak form: multiply with a smooth test function ϕ defined on Γ , integrate over Γ

$$\langle V, \phi \rangle = -\langle G, \phi \rangle = -\langle g\kappa + \partial_n g, \phi \rangle + \langle g_y, \nabla_{\Gamma} \phi \rangle, \quad (10)$$

where $\langle u, v \rangle = \int_{\Gamma} uv \, d\sigma$ is the L^2 scalar product on Γ . To obtain (10), we use Green's identity $\int_{\Gamma} v \operatorname{div}_{\Gamma} \mathbf{w} d\sigma = - \int_{\Gamma} \mathbf{w} \cdot \nabla_{\Gamma} v d\sigma$ for integration by parts. Unlike strong form (8), weak form (10) does not require additional derivatives of g_y .

Better descent velocities: The velocity computed with equation (10) is the L^2 gradient descent velocity (as we use the L^2 scalar product on Γ on the left hand side of (10)). We can choose to use other scalar products to define other gradient descent velocities [2, 6, 23, 22]. These velocities can have desirable properties, such as smoother surface evolution, faster convergence. To realize such a framework, we choose a generic scalar product $b(\cdot, \cdot)$ inducing a Hilbert space $B(\Gamma)$ on Γ . Then velocity V computed by the *generalized velocity equation*

$$b(V, \phi) = -\langle G, \phi \rangle = -\langle g\kappa + \partial_n g, \phi \rangle + \langle g_y, \nabla_{\Gamma} \phi \rangle, \quad \forall \phi \in B(\Gamma), \quad (11)$$

is a descent velocity, because it satisfies $dJ(\Gamma; V) = -\langle G, V \rangle = -b(V, V) \leq 0$.

Two possible options for the scalar product are L^2 and weighted H^1 :

$\langle V, W \rangle_{L^2} = \langle V, W \rangle = \int_{\Gamma} VW \, d\sigma$, $\langle V, W \rangle_{H^1} = \langle \alpha \cdot \nabla_{\Gamma} V, \nabla_{\Gamma} W \rangle + \langle \beta V, W \rangle$, where $\alpha(x, \Gamma), \beta(x, \Gamma)$ are spatially-varying weight functions, and may depend on the geometry, e.g. normal, curvature, of Γ as well. The function $\alpha : \mathbb{R}^d \times \Gamma \rightarrow \mathbb{R}^{d \times d}$ is a positive definite matrix-valued, and $\beta : \mathbb{R}^d \times \Gamma \rightarrow \mathbb{R}$ is a positive scalar-valued. The H^1 scalar product with constant weight functions was used in [6, 23]. Our framework, in contrast, enables us to use general non-constant weight functions, which can also depend on the geometry of Γ . This offers more flexibility in the choice of gradient descent velocities.

A particularly useful choice is based on the shape Hessian of the surface energy, derived in Section 2. If we compute the velocity V using the Newton's method, solving equation, $d^2 J(\Gamma; V, \phi) = -dJ(\Gamma; \phi)$ ($= -\langle G, \phi \rangle$), $\forall \phi \in B(\Gamma)$, we find that this velocity leads to faster convergence, quadratic close to the minimum. One easily sees that the 2^{nd} shape derivative

$$d^2 J(\Gamma; V, W) = \langle \alpha \cdot \nabla_{\Gamma} V, \nabla_{\Gamma} W \rangle + \langle \beta V, W \rangle - \langle \gamma \cdot \nabla_{\Gamma} V, W \rangle - \langle \gamma \cdot \nabla_{\Gamma} W, V \rangle, \quad (12)$$

is similar to a weighted H^1 scalar product, when $\alpha(x, \Gamma) = (g - g_y \cdot n)Id + g_{yy}$, $\beta(x, \Gamma) = \partial_{nn}g + 2\kappa\partial_n g + (\kappa^2 - \sum \kappa_i^2)g$, $\gamma(x, \Gamma) = \kappa g_y - g_y^T \nabla_{\Gamma} n + n^T g_{xy}^T$.

The shape Hessian (12) however is not a proper scalar product, and it cannot be used with the generalized velocity equation (11) to compute descent velocities, because it is not positive definite. Still, we can try to use it to create a custom scalar product based on (12) to achieve improved convergence. For this, we propose the thresholded coefficients

$$\alpha_+ = (g - g_y \cdot n)_+ Id + g_{yy}, \quad \beta_+ = (\partial_{nn}g + 2\kappa\partial_n g + (\kappa^2 - \sum \kappa_i^2)g)_+,$$

where $f_+ = \max(f, \varepsilon)$ with $\varepsilon = 1$ for β_+ , $\varepsilon = \min(g)$ for α_+ in our implementation, and g is designed so that the Hessian matrix function g_{yy} is positive semidefinite. Then by neglecting the last two terms in (12), we obtain the following weighted H^1 scalar product

$$\langle V, W \rangle_{H^1} = \langle \alpha_+ \cdot \nabla_{\Gamma} V, \nabla_{\Gamma} W \rangle + \langle \beta_+ V, W \rangle, \quad (13)$$

which acts like a preconditioner on the velocity and results in smoother surface evolutions and convergence in fewer iterations compared to the L^2 velocity (8).

4 Lagrangian Discretization

The minimization algorithm developed so far is in the continuous mathematical realm, and cannot yet be used to compute numerical minima for energy (2). For the actual computation, we need a discrete representation of the surface; moreover, a discretization of the energy (2), and all the relationships used in a minimization iteration: the step size condition (6), the stopping criterion (7), the velocity equation (11). The original algorithm for GAC model [4, 5] relied on a Eulerian level set representation of the surface, and the minimization was carried out as a level set evolution with fixed step size. The discretization was by finite differences on the image grid. This can have high computational cost if all the grid points are evaluated as unknowns in the iterations. In this work, we opt for a Lagrangian discretization of the problem for its efficiency and the flexibility it offers to realize an effective shape optimization algorithm.

Discretization of the geometry: We discretize the surface as a set of simplices $\{\Gamma_i^h\}_{i=1}^m$, namely, line segments making up polygonal curves in 2d, and triangulated surfaces in 3d. The simplices provide a basis for discretization of the minimization equations as well. This simplicial discretization is compact and efficient; a collection of curves used to segment a megapixel image can be represented with only a few hundred nodes, providing orders of magnitude reduction in the number of variables compared to a Eulerian representation.

We can tune the efficiency of the surface representation even further by implementing spatial adaptivity. More nodes are used to increase surface resolution in complicated parts of the geometry or the image, and fewer nodes in flat areas. To manage this dynamically through the evolution of Γ , we use geometrically-consistent surface refinement and coarsening operations [3]. For a line segment, refinement introduces a new node in the middle, and projects it along the normal to match curvature. For a triangle, we use longest-edge bisection approach [19]. Coarsening is just an undo operation for refinement. These adaptations are executed at the end of each iteration to ensure accuracy before the next iteration.

Finite element method for velocity: We use the simplicial discretization $\{\Gamma_i^h\}_{i=1}^{n_\Gamma}$ of the approximate surface Γ^h to introduce a finite element (FE) discretization of the velocity equations. We choose a set of piecewise linear nodal basis functions $\{\phi_i\}_{i=1}^m$ defined on surface elements. The function ϕ_i satisfies $\phi_i(x_i) = 1$ on i^{th} node x_i of Γ^h , but $\phi_i(x_j) = 0$ on the other nodes $x_j, j \neq i$. We use $\{\phi_i\}$ as test functions in the generalized velocity eqn (11):

$$b(V, \phi_i) = -\langle G, \phi_i \rangle = -\langle g\kappa + \partial_n g, \phi_i \rangle - \langle g_y, \nabla_\Gamma \phi_i \rangle, \quad i = 1, \dots, m.$$

Similarly, we take the geometric relationships $\kappa = -\Delta_\Gamma \mathbf{X}$, $\kappa = \kappa \cdot \mathbf{n}$, $\mathbf{V} = V\mathbf{n}$, multiply by ϕ_i , integrate on Γ^h , and write them in weak form: $\langle \kappa, \phi_i \rangle = \langle \nabla_\Gamma \mathbf{X}, \nabla_\Gamma \phi_i \rangle$, $\langle \kappa, \phi_i \rangle = \langle \kappa \cdot \mathbf{n}, \phi_i \rangle$, $\langle \mathbf{V}, \phi_i \rangle = \langle V, \phi_i \cdot \mathbf{n} \rangle$, where ϕ_i is the vector-valued test functions, e.g. $\phi_i = (\phi_i, \phi_i)$ in 2d. We also expand all the critical quantities in terms $\{\phi_i\}_{i=1}^m$, so that they are represented by coefficient vectors $\kappa = \sum_{j=1}^m \mathbf{k}_j \phi_j = \mathbf{K}_j \phi_j$, $\kappa = \mathbf{k}_j \phi_j$, $V = \mathbf{v}_j \phi_j$, $\mathbf{V} = \mathbf{v}_j \phi_j$. In this way, we obtain discretized equations in matrix form: $\mathbf{A}\mathbf{k} = \mathbf{M}\mathbf{x}$, $\mathbf{M}\mathbf{k} = \mathbf{N}\mathbf{k}$, $\mathbf{B}\mathbf{v} = \mathbf{g}$, $\mathbf{M}\mathbf{v} = \mathbf{N}\mathbf{v}$, where $\mathbf{A}_{ij} = \langle \nabla_\Gamma \phi_i, \nabla_\Gamma \phi_j \rangle$, $\mathbf{M}_{ij} = \langle \phi_i, \phi_j \rangle$, $\mathbf{B}_{ij} = \langle \alpha_+ \cdot \nabla_\Gamma \phi_i, \nabla_\Gamma \phi_j \rangle + \langle \beta_+ \phi_i, \phi_j \rangle$, $\mathbf{N}_{ij}^k = \langle \phi_i, \phi_j n_k \rangle$, $\mathbf{g}_i = -\langle g\kappa + \partial_n g, \phi_i \rangle - \langle g_y, \nabla_\Gamma \phi_i \rangle$, and vector versions \mathbf{A}, \mathbf{M} .

The matrices $\mathbf{A}, \mathbf{M}, \mathbf{B}$ consist of tridiagonal blocks in 2d, and they can be inverted in $O(m)$ time. In 3d, they are sparse matrices, and can still be inverted efficiently using a sparse direct solver or the conjugate gradient algorithm. Given the current surface nodes $\mathbf{x} = \{x_i \in \Gamma^h\}_{i=1}^m$, we solve for the discrete velocity \mathbf{v} with the explicit sequence $\mathbf{k} = \mathbf{A}^{-1}\mathbf{M}\mathbf{x} \rightarrow \mathbf{k} = \mathbf{M}^{-1}\mathbf{N}\mathbf{k} \rightarrow \mathbf{v} = \mathbf{B}^{-1}\mathbf{g} \rightarrow \mathbf{v} = \mathbf{M}^{-1}\mathbf{N}\mathbf{v}$. The semi-implicit equations (9) are discretized similarly, then solved as a coupled system of equations with a much larger coefficient matrix.

Adaptivity of discretization: The central concern for the quality of the discretization is accuracy, specifically, how faithfully it captures the geometry, and how well it resolves important features in the image, meanwhile maintaining efficiency as well. Thus we aim to keep accuracy within a reasonable range, not too low or too high. We use two separate criteria for judging the accuracy:

- 1) *Geometry:* Measure geometric discretization error with $\max_{\Gamma_i^h} |\kappa| |\Gamma_i^h|^2$ in 2d, $\max_{\Gamma_i^h} \frac{|n_i - n_j|}{h} |\Gamma_i^h|^2$ in 3d (n_j is the normal of the neighbor element).
- 2) *Data:* To see how well the element Γ_i^h resolves the data func. $g(x, n)$, compare low and high order quadrature approximations of the local integral $\int_{\Gamma_i^h} g(x, n) d\sigma$. Mesh elements are refined if one of two errors is large, coarsened if both are low.

An additional and critical adaptivity criterion is how well we are approximating the shape gradient G , key to velocity computations (11), step size selection (6), and stopping criterion (7). Ideally, the optimization iterations would converge to the optimal surface Γ^* , and we would have the shape gradient $|G(x)| \approx 0, \forall x \in \Gamma^*$. In practice, the energy, the shape gradient, and the velocity will be from different discretizations, and thus may be *inconsistent*. This typically manifests itself with step size selection failures close to the minimum. When this occurs, we identify 10% elements with largest G value, refine them, restart shape optimization to ensure convergence with satisfactory accuracy.

We also enable topological adaptivity in 2d, e.g. merging, splitting of curves, based on intersection detection and local surgery to reconnect curves [9, 16, 17].

5 Numerical Experiments

We test our algorithm with synthetic and real image examples in 2d and 3d. We set $g(x, n) = 1/(1 + |\nabla G_\sigma * I(x)|^2/\lambda^2)$, $\lambda = \frac{1}{4} \max |\nabla G_\sigma * I| > 0$ and $\sigma = 2$ pixels in the Gaussian $G_\sigma(x)$. We minimize (2) using L^2 velocity (8), and the weighted H^1 velocity using (13) (subject to a maximum iteration number of 1000).

We first evaluate the two velocities and our numerical algorithm on two clean synthetic images. The merit of these two images is that it enables us to evaluate the sensitivity of our algorithm with respect to image resolution, thereby the sharpness of the image gradient at the object boundary. For this, we generate the same image at resolutions of $200^2, 400^2, 800^2, 1600^2$ pixels. We examine the number of iterations, the number of energy evaluations, and the actual running times from the velocities (see Fig. 1). We find that H^1 velocity gives superior results in all these cases. It is robust with respect to image resolution, whereas the performance of L^2 velocity deteriorates as the resolution increases. At highest resolution, L^2 does not converge and stops at the maximum iteration number.

resolution	200	400	800	1600
(1) L^2	27, 29, 0.4s	83, 87, 1.2s	299, 319, 4.6s	1000, 1286, 15.4s
H^1	25, 26, 0.2s	41, 42, 0.3s	81, 82, 0.6s	153, 154, 1.1s
(2) L^2	52, 57, 0.7s	169, 176, 2.4s	665, 706, 9.1s	1000, 1337, 14.3s
H^1	27, 28, 0.2s	57, 65, 0.4s	90, 91, 0.7s	184, 190, 1.3s
(3) L^2	145, 184, 2.3s	462, 592, 7.8s	1000, 1138, 16.1s	1000, 1377, 14.2s
H^1	119, 180, 1.3s	101, 108, 1.0s	286, 383, 2.9s	645, 925, 8.3s

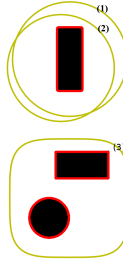


Fig. 1. L^2 and H^1 velocities used to segment two synthetic images of increasing resolution (top right), starting with the different initial curves (1),(2),(3) (yellow), converging to the (red) boundary curves. Iteration numbers, energy evaluations and actual timings (seconds) are compared.

Next we evaluate our algorithm with real image examples (see Fig. 2). We examine the quality of the segmentation, and report the number of iterations, the number of energy evaluations, and the actual running times. We find that our observations from synthetic examples are repeated on the real examples as well. The L^2 velocity takes much longer than the weighted H^1 velocity, both in terms of iteration numbers and running times.

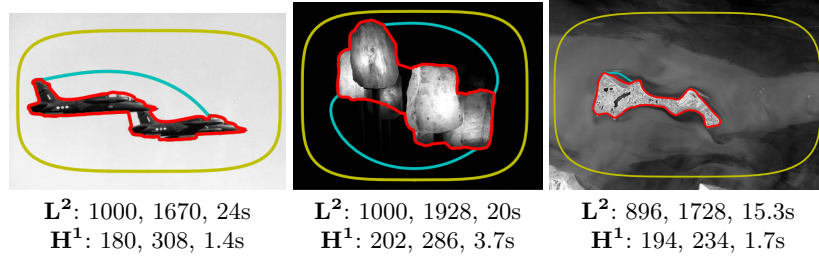


Fig. 2. L^2 and H^1 velocities used to segment three example images. The optimization starts with the yellow curves, and converges to the final cyan final curve with L^2 velocity, and red curve with H^1 velocity. Iteration numbers, energy evaluations and actual timings (seconds) are reported and compared for each example. L^2 velocity terminates by reaching the maximum (1000) number of iterations in examples 1, 2.

Finally, we test our algorithm on synthetic 3d examples (see Fig. 3). We verify that our discrete representation can be successfully used to segment 3d objects with weighted H^1 descent velocities of the surface energy (2). These examples also demonstrate the spatial adaptivity of the triangulated surface mesh. We start the optimization with coarse meshes. These meshes are then adapted and refined as the geometry gets more complicated and as the algorithm senses more

details of the data through the iterations. The final objects are captured in good detail without resorting to an excessively refined mesh.

6 Conclusion

We propose a new minimization algorithm for a more general anisotropic version (2) of the Geodesic Active Contour model [4, 5, 10, 12, 14] in any number of dimensions, supporting alignment penalties on the normal of the surface, in addition to isotropic image-based penalties. The first variation of this energy, used to develop minimization algorithms, is already known. We derive the second variation or second shape derivative, and use it to develop a Newton-type shape optimization algorithm that results in smoother descent velocities, and converges in fewer iterations and in less time. The Newton-type minimization algorithm is more robust; it exhibits consistent convergence behavior for increasing resolution of the same image, with sharper gradients, whereas the performance of the commonly-used L^2 velocity deteriorates.

We use a Lagrangian (not Eulerian e.g. level sets) discretization of the geometry, namely, polygonal curves in 2d and triangulated surfaces in 3d, for computational efficiency and ease of implementation. Our representation is adaptive and captures the objects in the image accurately, but economically, adding more nodes and resolution only when necessary (see Fig.3). The velocity equations are discretized using the Finite Element Method, which can be solved efficiently, in linear time with respect to number of nodes in the case of 2d polygonal curves. Unlike previous approaches [6, 23], our discretization can handle and compute a variety of generalized descent velocities, including L^2 , H^1 and Newton, for any number of dimensions, 2d, 3d, 4d.

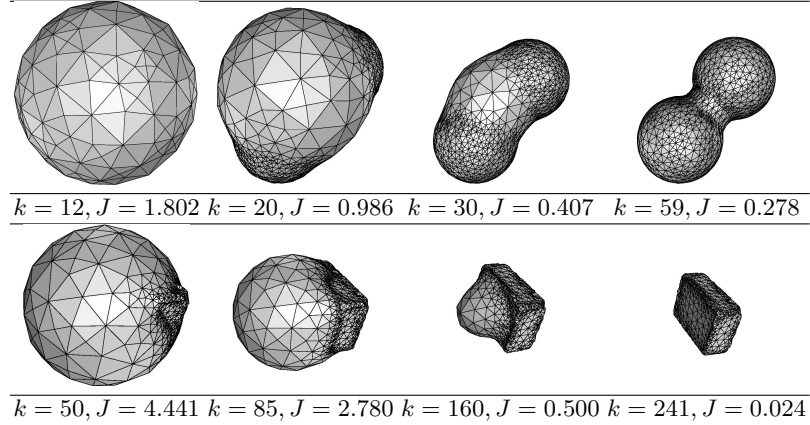


Fig. 3. Iterations from segmentations of two synthetic 3d objects using weighted H^1 velocity. Note the spatial adaptivity of the triangulated surface adjusting to the objects.

Acknowledgement This work was performed under the financial assistance award 70NANB16H306 from the U.S. Department of Commerce, National Institute of Standards and Technology.

References

1. Bar, L., Sapiro, G.: Generalized newton-type methods for energy formulations in image processing. *SIAM Journal on Imaging Sciences* 2(2), 508–531 (2009)
2. Baust, M., Yezzi, A., Unal, G., Navab, N.: Translation, scale, and deformation weighted polar active contours. *J. of Math. Im. and Vision* 44(3), 354–365 (2012)
3. Bonito, A., Nochetto, R.H., Pauletti, M.S.: Geometrically consistent mesh modification. *SIAM Journal on Numerical Analysis* 48(5), 1877–1899 (2010)
4. Caselles, V., Kimmel, R., Sapiro, G.: Geodesic active contours. *IJCV* 22(1), 61–79 (1997)
5. Caselles, V., Kimmel, R., Sapiro, G., Sbert, C.: Minimal surfaces: a geometric three-dimensional segmentation approach. *Numer. Math.* 77(4), 423–451 (1997)
6. Charpiat, G., Maurel, P., Pons, J.P., Keriven, R., Faugeras, O.: Generalized gradients: Priors on minimization flows. *IJCV* 73(3), 325–344 (2007)
7. Deckelnick, K., Dziuk, G., Elliott, C.M.: Computation of geometric partial differential equations and mean curvature flow. *Acta Numer.* 14, 139–232 (2005)
8. Delfour, M.C., Zolésio, J.P.: *Shapes and Geometries. Advances in Design and Control*, SIAM, Philadelphia, PA (2001)
9. Delingette, H., Montagnat, J.: Shape and topology constraints on parametric active contours. *Computer Vision and Image Understanding* 83(2), 140–171 (2001)
10. Faugeras, O., Keriven, R.: Variational principles, surface evolution, pde’s, level set methods and the stereo problem. In: *IEEE Trans. Image Processing.* vol. 7, pp. 336–344 (1998)
11. Goldlücke, B., Magnor, M.: Space-time isosurface evolution for temporally coherent 3d reconstruction. *CVPR I*, 350–355 (June 2004)
12. Goldlücke, B., Magnor, M.: Weighted minimal hypersurfaces and their applications in computer vision. *ECCV* 2, 366–378 (May 2004)
13. Hintermüller, M., Ring, W.: A second order shape optimization approach for image segmentation. *SIAM J. Appl. Math.* 64(2), 442–467 (2003/04)
14. Jin, H., Soatto, S., Yezzi, A.J.: Multi-view stereo beyond lambert. In: *CVPR* (1). pp. 171–178. IEEE Computer Society (2003)
15. Kimmel, R., Bruckstein, A.: Regularized Laplacian zero crossings as optimal edge integrators. *IJCV* 53(3), 225–243 (2003)
16. Lachaud, J.O., Montanvert, A.: Deformable meshes with automated topology changes for coarse-to-fine 3D surface extraction. *Medical Image Analysis* 3(2), 187–207 (1999)
17. McInerney, T., Terzopoulos, D.: T-snakes: Topology adaptive snakes. *Medical Image Analysis* 4(2), 73–91 (2000)
18. Nocedal, J., Wright, S.J.: *Numerical Optimization*. Springer Series in Operations Research, Springer-Verlag, New York (1999)
19. Schmidt, A., Siebert, K.: *Design of Adaptive Finite Element Software*. Springer-Verlag, Berlin (2005)
20. Sokolowski, J., Zolésio, J.P.: *Introduction to Shape Optimization*, Springer Series in Computational Mathematics, vol. 16. Springer-Verlag, Berlin (1992)

21. Solem, J.E., Overgaard, N.C.: A geometric formulation of gradient descent for variational problems with moving surfaces. In: International Conference on Scale-Space Theories in Computer Vision. pp. 419–430. Springer (2005)
22. Sundaramoorthi, G., Yezzi, A., Mennucci, A.: Coarse-to-fine segmentation and tracking using sobolev active contours. PAMI 30(5), 851–864 (2008)
23. Sundaramoorthi, G., Yezzi, A., Mennucci, A.C.: Sobolev active contours. IJCV 73(3), 345–366 (2007)