United Snakes

Jianming Liang[1,2], Tim McInerney[2,3], and Demetri Terzopoulos[2]

[1] Turku Centre for Computer Science, DataCity, Lemminkäisenkatu 14 A, 20520 Turku, Finland
[2] Department of Computer Science, University of Toronto, 6 King's College Road, Toronto, ON M5S 3H5, Canada
[3] Department of Math, Physics and Computer Science, Ryerson Polytechnic University, 80 Gould St., Toronto, ON M5B 2K3, Canada
Email: liang@cs.utu.fi, tim@cs.toronto.edu, dt@cs.toronto.edu

Abstract

Since their debut in 1987, snakes (active contour models) have become a standard image analysis technique with several variants now in common use. We present a portable, reusable, software package called "United Snakes". The package unites the most popular snake variants, including finite difference, B-spline, and Hermite polynomial snakes within the mathematical framework of a general finite element formulation with a choice of shape functions. The package furthermore incorporates a recently proposed snake-like technique known as "livewire". We integrate snakes and livewire by introducing an effective method for imposing hard constraints on snakes. Our experiments demonstrate that snakes and livewire have complementary strengths and that their union offers a more powerful tool for interactive image analysis, especially for medical imaging applications. United Snakes is implemented in Java as a JavaBean so that it can easily be integrated in end-user application systems.

1. Introduction

Snakes (active contour models) quickly gained popularity following their debut in 1987 [7]. They have proven especially useful in medical image analysis [10, 14] and for tracking moving objects in video [17, 3], among other applications. Variants such as finite element snakes [4], B-snakes [11, 3], and Fourier snakes [15], have been proposed in an effort to improve aspects of the original finite difference implementation (e.g., to decrease initialization sensitivity, increase robustness against noise, improve selectivity for certain classes of objects, etc.). No formulation has yet emerged as a "gold standard". Rather, the variants seem well-suited to different applications with particular image modalities and processing scenarios.

Given the confusing array of choices for the user, there is a need for a portable and reusable snakes implementation which unites the best features of the variants while maintaining the simplicity and elegance of the original formulation. To this end, our first contribution in this paper is to unify the most important snakes variants, including finite difference, B-spline, and Hermite polynomial snakes, in a comprehensive finite element framework where a particular type of snake can be derived by simply changing the finite element shape functions.

A technique, known as "livewire" or "intelligent scissors" [12, 2, 13, 5], has recently emerged as an effective interactive boundary tracing tool. Based on dynamic programming [5] or Dijkstra's graph search algorithm [13], it was originally developed as an interactive 2D extension to earlier optimal boundary tracking methods. Livewire features several similarities with snakes, but it is generally considered in the literature as a competing technique. Our second contribution in this paper is the idea that livewire and snakes are in fact complementary techniques that can advantageously be combined via a simple yet effective method to impose hard constraints on snakes.

We call our software implementation *United Snakes*, because it unites several snake variants with livewire to offer a general purpose tool for interactive image segmentation that provides more flexible control while reducing user interactions. We have implemented United Snakes in Java as a JavaBean (reusable Java software component), so that it may easily be integrated into any end-user application system.

We describe our finite element framework in Section 2 and show how several snake variants can be integrated within it. Section 3 describes the livewire technique. We justify the idea of combining snakes with livewire in Section 4 and develop a hard constraint mechanism in Section 5 that makes this combination possible. Section 6 presents results utilizing the United Snakes system in a medical image segmentation scenario. We conclude in Section 7 and propose future extensions of United Snakes.

2. Finite Element Unification of Snakes

A snake is a time-varying parametric contour $\mathbf{v}(s,t) = (x(s,t), y(s,t))^\top$ in the image plane $(x,y) \in \Re^2$, where x and y are coordinate functions of the parameter $s \in [0, L]$ and t is time. The shape of the contour subject to an image $I(x,y)$ is dictated by an energy functional $\mathcal{E}(\mathbf{v}) = \mathcal{S}(\mathbf{v}) + \mathcal{P}(\mathbf{v})$. The first term is the internal deformation energy defined as

$$\mathcal{S}(\mathbf{v}) = \frac{1}{2}\int_0^L \alpha(s)\left|\frac{\partial \mathbf{v}}{\partial s}\right|^2 + \beta(s)\left|\frac{\partial^2 \mathbf{v}}{\partial s^2}\right|^2 ds, \qquad (1)$$

where $\alpha(s)$ controls the "tension" of the contour and $\beta(s)$ regulates its "rigidity". The second term is an external image energy

$$\mathcal{P}(\mathbf{v}) = \int_0^L P_I(\mathbf{v})\, ds, \tag{2}$$

which couples the snake to the image via a scalar potential function $P_I(x, y)$ typically computed from $I(x, y)$ through image processing. The Lagrange equations of motion for a dynamic snake are

$$\mu \frac{\partial^2 \mathbf{v}}{\partial t^2} + \gamma \frac{\partial \mathbf{v}}{\partial t} - \frac{\partial}{\partial s}\left(\alpha \frac{\partial \mathbf{v}}{\partial s}\right) + \frac{\partial^2}{\partial s^2}\left(\beta \frac{\partial^2 \mathbf{v}}{\partial s^2}\right) = \mathbf{q}(\mathbf{v}). \tag{3}$$

The first two terms represent inertial forces due to the mass density $\mu(s)$ and damping forces due to the dissipation density $\gamma(s)$. The next two terms represent the internal stretching and bending deformation forces. On the right hand side are the external forces $\mathbf{q}(\mathbf{v}) = -\nabla P_I(\mathbf{v}) + \mathbf{f}$, where the image forces are the negative gradient of the image potential function. The user may guide the dynamic snake via time-varying interaction forces $\mathbf{f}(s, t)$ (usually applied through a mouse), driving the snake out of one energy minimizing equilibrium and into another. Viewed as a dynamical system, the snake may also be used to track moving objects in a time-varying (video) image $I(x, y, t)$.

2.1. Finite Element Formulation

In a finite element formulation, the parametric domain $0 \le s \le L$ is partitioned into finite sub-domains, so that the snake contour is divided into "snake elements". Each element e is represented geometrically with shape functions $\mathbf{N}(s)$ involving shape parameters $\mathbf{u}^e(t)$. The shape parameters of all the elements are collected together into the snake parameter vector $\mathbf{u}(t)$. This leads to a discrete form of the equations of motion (3) as a system of second order ordinary differential equations in $\mathbf{u}(t)$:

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{C}\dot{\mathbf{u}} + \mathbf{K}\mathbf{u} = \mathbf{g}, \tag{4}$$

where $\mathbf{M}$ is the mass matrix, $\mathbf{C}$ is the damping matrix, $\mathbf{K}$ is the stiffness matrix, and $\mathbf{g}$ is the external force vector. Appendix A details the finite element formulation.

The stiffness matrix $\mathbf{K}$ is assembled from element stiffness sub-matrices $\mathbf{K}^e$ which depend on the shape functions $\mathbf{N}$ (the matrices $\mathbf{M}$, $\mathbf{C}$, and the vector of nodal external forces $\mathbf{g}$ are assembled in a similar way and also depend on $\mathbf{N}$). The shape functions generate different stiffness matrices and, in turn, yield different snake behaviors suitable for different tasks. For example, snakes that use B-spline shape functions are typically characterized by a low number of degrees of freedom, typically use polynomial basis functions of degree 2 or higher, and are inherently very smooth. Therefore, these "B-snakes" [11, 3] can be effective in segmentation or tracking tasks involving noisy images where the target object boundaries may exhibit significant gaps in the images. On the other hand, object boundaries with many fine details or rapid curvature variations may best be segmented by a snake that uses simpler shape functions and more degrees of freedom, such as a finite difference snake [7]. The unification of these different shape functions in a single framework enhances the range of object modeling capabilities.

The following sections address Hermitian shape functions, B-spline shape functions, and "shape functions" for finite difference snakes. Since the two coordinate functions $x(s)$ and $y(s)$ of the snake $\mathbf{v}(s)$ are independent, we shall discuss the shape functions in terms of only one component $x(s)$; the shape functions for $y(s)$ assume an identical form.

2.2. Hermitian Shape Functions

In the case of Hermitian snakes, $x(s)$ ($0 \le s \le l$, where l is the element parametric length) is approximated with a cubic polynomial function, parameterized by position x and slope θ at the endpoints $s = 0$ and $s = l$ of an element. We can show that $x(s) = \mathbf{N}_h \mathbf{u}^{e_i}$, where $\mathbf{u}^{e_i} = [x_i\ \theta_i\ x_{i+1}\ \theta_{i+1}]^\top$ are the shape parameters of element e_i and $\mathbf{N}_h = \mathbf{s}\mathbf{H}$ are the Hermitian shape functions, with $\mathbf{s} = [1\ s\ s^2\ s^3]$ and the *Hermitian shape matrix*

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -3/l^2 & -2/l & 3/l^2 & -1/l \\ 2/l^3 & 1/l^2 & -2/l^3 & 1/l^2 \end{bmatrix}. \tag{5}$$

It is reasonable to assume that the tension function $\alpha(s)$ and rigidity function $\beta(s)$ are constant within the element. Hence, the stiffness matrices associated with the tension and rigidity components for element e_i are respectively

$$\mathbf{K}_\alpha^{e_i} = \frac{\alpha_i}{30l}\begin{bmatrix} 36 & 3l & -36 & 3l \\ 3l & 4l^2 & -3l & -l^2 \\ -36 & -3l & 36 & -3l \\ 3l & -l^2 & -3l & 4l^2 \end{bmatrix}, \tag{6}$$

$$\mathbf{K}_\beta^{e_i} = \frac{\beta_i}{l^3}\begin{bmatrix} 12 & 6l & -12 & 6l \\ 6l & 4l^2 & -6l & 2l^2 \\ -12 & -6l & 12 & -6l \\ 6l & 2l^2 & -6l & 4l^2 \end{bmatrix}. \tag{7}$$

An analytic form of the external forces $\mathbf{q}(\mathbf{v})$ in (3) is generally not available. Therefore, Gauss-Legendre quadrature may be employed to approximate the value of the integral for the element external force vector $\mathbf{F}^e$. For element e_i we have

$$\begin{aligned} \mathbf{F}_x^{e_i} &= \int_0^l {\mathbf{N}_h}^\top \mathbf{q}_x(\mathbf{v}(s))\, ds \\ &= l \sum_j \rho_j \mathbf{N}_h(\xi_j)^\top \mathbf{q}_x(\mathbf{v}(\xi_j)), \end{aligned} \tag{8}$$

where the subscript x indicates the association with coordinate function $x(s)$, and where ξ_j and ρ_j are the jth Gaussian integration point and its corresponding weighting coefficient, respectively. $\mathbf{F}_y^{e_i}$ is derived in a similar fashion.

To make the global matrix assembly process identical for all shape functions, we introduce *assembling matrices*. Suppose that we have a snake with n elements and N nodes ($N = n$ if the snake is closed and $N = n + 1$ if it is open). For the ith element e_i of the snake ($0 \le i \le n - 1$), the assembling matrices are $\mathbf{G}_\alpha^{e_i} = \mathbf{G}_\beta^{e_i} = \mathbf{G}_\mathbf{F}^{e_i} = \mathbf{G}^{e_i}$, where

$$\left(\mathbf{G}^{e_i}\right)_{jk} = \begin{cases} 1 & \text{if } (j + di) \bmod (dN) = k \\ 0 & \text{otherwise,} \end{cases} \tag{9}$$

are $(2 \cdot d) \times (d \cdot N)$ matrices, with d the number of degrees of freedom of each node in an element (here $d = 2$). Hence, $\mathbf{K}_\alpha$, $\mathbf{K}_\beta$ and $\mathbf{F}$ may be assembled as follows:

$$\mathbf{K}_\alpha = \sum_{i=0}^{n-1} (\mathbf{G}_\alpha^{e_i})^\top \mathbf{K}_\alpha^{e_i} (\mathbf{G}_\alpha^{e_i}), \quad (10)$$

$$\mathbf{K}_\beta = \sum_{i=0}^{n-1} (\mathbf{G}_\beta^{e_i})^\top \mathbf{K}_\beta^{e_i} (\mathbf{G}_\beta^{e_i}), \quad (11)$$

$$\mathbf{F} = \sum_{i=0}^{n-1} (\mathbf{G}_\mathbf{F}^{e_i})^\top \mathbf{F}^{e_i}. \quad (12)$$

Only the shape matrix and the assembling matrices are determined by specific polynomial shape functions. Therefore, in the following section we shall focus only on the derivation of the shape matrix and the assembling matrices for B-spline shape functions, and briefly mention other kinds of shape functions which are suitable for snakes.

2.3. B-Spline Shape Functions

For B-spline shape functions, the $x(s)$ coordinate function of $\mathbf{v}(s)$ is constructed as a weighted sum of N_B basis functions $B_n(s)$, $n = 0, ..., N_B - 1$ as follows: $x(s) = \mathbf{B}(s)^\top \mathbf{Q}^x$, where $\mathbf{B}(s) = [\mathbf{B}_0(s), ..., \mathbf{B}_{N_B-1}(s)]^\top$, $\mathbf{Q}^x = [x_0, ..., x_{N_B-1}]^\top$ and x_i are the weights applied to the respective basis functions $B_n(s)$.

A B-spline span serves as an element in our finite element formulation (hence "span" and "element" are interchangeable terms). Consequently, we shall determine the nodal variables (*i.e.* snake shape parameters), the shape matrix, and the assembling matrix associated with a span. When all spans are unit length, the knot multiplicities at the breakpoints are $m_0, ..., m_L$ (L is the number of spans and the total number of knots $N_B = \sum_{i=0}^{L} m_i$), the knot values k_i are determined by $k_i = l$, such that $0 \leq (i - \sum_{j=0}^{l} m_i) < m_{l+1}$. Furthermore, the nth polynomial $B_{n,d}^\sigma$ in span σ can be computed as follows:

$$B_{n,1}^\sigma(s) = \begin{cases} 1 & \text{if } k_n \leq \sigma < k_{n+1} \\ 0 & \text{otherwise} \end{cases} \quad (13)$$

$$B_{n,d}^\sigma(s) = \frac{(s + \sigma - k_n) B_{n,d-1}^\sigma(s)}{k_{n+d-1} - k_n} + \frac{(k_{n+d} - s - \sigma) B_{n+1,d-1}^\sigma(s)}{k_{n+d} - k_{n+1}} \quad (14)$$

For span σ, the index b_σ for the first basis function whose support includes the span can be determined as $b_\sigma = [(\sum_{i=0}^{\sigma} m_i) - d] \bmod N_B$. Therefore, $I = [b_\sigma, (b_\sigma + 1) \bmod N_B, ..., (b_\sigma + d - 1) \bmod N_B]$ are the indices of the nodal variables and also those of the d polynomials $B_{n,d}^\sigma$.[1] Now, the shape matrix for span σ can be constructed by collecting the coefficients of each of the d polynomials $B_{n,d}^\sigma$ as its columns. For example, the shape matrix of a regular cubic B-spline is

$$\mathbf{H} = \begin{bmatrix} 1/6 & 2/3 & 1/6 & 0 \\ -1/2 & 0 & 1/2 & 0 \\ 1/2 & -1 & 1/2 & 0 \\ -1/6 & 1/2 & -1/2 & 1/6 \end{bmatrix} \quad (15)$$

and the element stiffness matrices for element e_i are

$$\mathbf{K}_\alpha^{e_i} = \alpha_i \begin{bmatrix} 0.0500 & 0.0583 & -0.1000 & -0.0083 \\ 0.0583 & 0.2833 & -0.2417 & -0.1000 \\ -0.1000 & -0.2417 & 0.2833 & 0.0583 \\ -0.0083 & -0.1000 & 0.0583 & 0.0500 \end{bmatrix}, \quad (16)$$

$$\mathbf{K}_\beta^{e_i} = \beta_i \begin{bmatrix} 0.3333 & -0.5000 & 0 & 0.1667 \\ -0.5000 & 1.0000 & -0.5000 & 0 \\ 0 & -0.5000 & 1.0000 & -0.5000 \\ 0.1667 & 0 & -0.5000 & 0.3333 \end{bmatrix}. \quad (17)$$

The assembling matrix $\mathbf{G}^{e_i}$ can be defined as

$$(\mathbf{G}^{e_i})_{jk} = \begin{cases} 1 & \text{if } (j + b_\sigma) \bmod N_B = k \\ 0 & \text{otherwise.} \end{cases} \quad (18)$$

In a similar fashion as above, we may construct other kinds of shape functions; for instance, NURBS shape functions [16], Catmull-Rom shape functions, Bézier shape functions and Fourier shape functions. The latter are global shape functions over the whole snake [15], thus the associated assembling matrix becomes an identity matrix.

2.4. Finite Difference Snakes in Element Form

Despite the differences between finite element snakes and finite difference snakes, the finite difference snakes can also be constructed in the finite element fashion, using the Dirac delta function $\delta(s)$ as the shape function. The construction primitives are as follows: For a snake with n nodes, $\mathbf{K}_\alpha^{e_i}$ is a 2×2 matrix and its corresponding assembling matrix $\mathbf{G}_\alpha^{e_i}$ is a $2 \times n$ matrix as follows:

$$\mathbf{K}_\alpha^{e_i} = \alpha_i \begin{bmatrix} -1 & 1 \end{bmatrix}^\top \begin{bmatrix} -1 & 1 \end{bmatrix} = \alpha_i \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \quad (19)$$

$$(\mathbf{G}_\alpha^{e_i})_{jk} = \begin{cases} 1 & \text{if } (j + i) \bmod n = k \\ 0 & \text{otherwise,} \end{cases} \quad (20)$$

where $0 \leq i \leq n - 2$ for an open snake and $0 \leq i \leq n - 1$ for a closed snake. $\mathbf{K}_\beta^{e_i}$ is a 3×3 matrix and with it is associated a $3 \times n$ assembling matrix $\mathbf{G}_\beta^{e_i}$ as follows:

$$\mathbf{K}_\beta^{e_i} = \beta_i \begin{bmatrix} 1 & -2 & 1 \end{bmatrix}^\top \begin{bmatrix} 1 & -2 & 1 \end{bmatrix} = \beta_i \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}, \quad (21)$$

$$\left(\mathbf{G}_\beta^{e_i}\right)_{jk} = \begin{cases} 1 & \text{if } (j + i) \bmod n = k \\ 0 & \text{otherwise,} \end{cases} \quad (22)$$

[1] In an open B-spline snake, d knots are introduced at the two ends. As a result, the index for the first basis function in the first span is zero (*i.e.* $b_0 = 0$) and the index of the last basis function in the last span is $N_B - 1$. For a closed B-spline snake, the index needs to be wrapped properly.

where $0 \leq i \leq n-3$ for an open snake and $0 \leq i \leq n-1$ for a closed snake. The $1 \times n$ assembling matrix $\mathbf{G}_{\mathbf{F}}^{e_i}$ is defined as

$$(\mathbf{G}_{\mathbf{F}}^{e_i})_{0,k} = \begin{cases} 1 & \text{if } i = k \\ 0 & \text{otherwise,} \end{cases} \quad (23)$$

where $0 \leq i \leq n-1$ for both open and closed snakes.

With the above formulation of finite difference snakes, we have a uniform finite element construction for a variety of snake representations, which leads to a relatively straightforward United Snakes implementation in an object-oriented programming language, such as Java.

3. Livewire

Livewire is a recently proposed interactive boundary tracing technique [12, 2, 13, 5]. It has two essential components, a local cost function that assigns lower cost to image features of interest, such as edges, and an expansion process that forms optimal boundaries for objects of interest based on the cost function and seed points provided interactively by the user.

3.1. Trace Formation

Boundary finding in livewire can be formulated as a directed graph search for an optimal (minimum cost) path using Dijkstra's algorithm. First, nodes in the graph are initialized with the local costs as described in the next section. Once the user selects a seed point, it will be used as the starting point for a recursive expansion process. In the expansion process, the local cost at the seed point is summed into its neighboring nodes. The neighboring node with the minimum cumulative cost is then further expanded and the process produces a dynamic "wavefront". The wavefront expands in order of minimum cumulative cost. Consequently, it propagates preferentially in directions of highest interest (*i.e.* along edges). This process requires only n iterations over the wavefront for paths of length n. Furthermore, since the wavefront is maintained in a sorted list, the expansion happens at interactive rates.

Therefore, for any dynamically selected goal node (*i.e.*, the free point) within the wavefront, the optimal path back to the seed point which forms a livewire (trace) can be displayed in real time. When the cursor (the free point) moves, the old livewire trace is erased and a new one computed and displayed in real time. The expansion process aims to compute an optimal path from a selected seed point to *every* other point in the image and lets the user choose among paths interactively, based on the current cursor position.

Livewire may be implemented very efficiently in multithreaded programming languages, such as Java, because the expansion process and the user interface can execute in separate, parallel threads. Since the free point is generally near the target object boundary, the expansion process will most likely have already advanced beyond that point and the livewire trace can be displayed immediately. That is, the livewire trace can typically be displayed before the expansion process has finished sweeping over the entire image.

3.2. Local Cost Function

There are different ways to define the local cost function. We follow the definition given by Mortensen and Barrett [12]. The local cost $l(\mathbf{p}, \mathbf{q})$ on the directed link from $\mathbf{p}$ to a neighboring pixel $\mathbf{q}$ is defined as a weighted sum of three local component costs created from various edge features:

$$l(\mathbf{p}, \mathbf{q}) = \omega_Z \mathsf{f}_Z(\mathbf{q}) + \omega_D \mathsf{f}_D(\mathbf{p}, \mathbf{q}) + \omega_G \mathsf{f}_G(\mathbf{q}), \quad (24)$$

where $\mathsf{f}_Z(\mathbf{q})$ is the Laplacian zero-crossing function at $\mathbf{q}$, $\mathsf{f}_D(\mathbf{p}, \mathbf{q})$ is the gradient direction from $\mathbf{p}$ to $\mathbf{q}$, $\mathsf{f}_G(\mathbf{q})$ is the gradient magnitude at $\mathbf{q}$, and ω_Z, ω_D and ω_G are their corresponding weights. The Laplacian zero-crossing function $\mathsf{f}_Z(\mathbf{q})$ is a binary function defined as

$$\mathsf{f}_Z(\mathbf{q}) = \begin{cases} 0 & \text{if } I_L(\mathbf{q}) = 0 \\ 1 & \text{otherwise,} \end{cases} \quad (25)$$

where $I_L(\mathbf{q})$ is the Laplacian of the image I at pixel $\mathbf{q}$. The gradient magnitude serves to establish a direct connection between edge strength and cost. The function f_G is defined as an inverse linear ramp function of the gradient magnitude G

$$\mathsf{f}_G = \frac{\max(G') - G'}{\max(G')} = 1 - \frac{G'}{\max(G')} \quad (26)$$

where $G' = G - \min(G)$. When calculating $l(\mathbf{p}, \mathbf{q})$, the function $\mathsf{f}_G(\mathbf{q})$ is further scaled by 1 if $\mathbf{q}$ is a diagonal neighbor to $\mathbf{p}$ and by $1/\sqrt{2}$ if $\mathbf{q}$ is a horizontal or vertical neighbor. The gradient direction $\mathsf{f}_D(\mathbf{p}, \mathbf{q})$ adds a smoothness constraint to the boundary by associating a higher cost for sharp changes in boundary direction. With $\mathbf{D}(\mathbf{p})$ defined as the unit vector normal to the gradient direction at pixel $\mathbf{p}$ (*i.e.*, $\mathbf{D}(\mathbf{p}) = [I_y(\mathbf{p}), -I_x(\mathbf{p})]$), the formulation of the gradient direction cost is

$$\mathsf{f}_D(\mathbf{p}, \mathbf{q}) = \frac{2}{3\pi} \left\{ \arccos[d_{\mathbf{p}}(\mathbf{p}, \mathbf{q})] + \arccos[d_{\mathbf{q}}(\mathbf{p}, \mathbf{q})] \right\}, \quad (27)$$

where $d_{\mathbf{p}}(\mathbf{p}, \mathbf{q}) = \mathbf{D}(\mathbf{p}) \cdot \mathbf{L}(\mathbf{p}, \mathbf{q})$ and $d_{\mathbf{q}}(\mathbf{p}, \mathbf{q}) = \mathbf{L}(\mathbf{p}, \mathbf{q}) \cdot \mathbf{D}(\mathbf{q})$ are vector dot products and

$$\mathbf{L}(\mathbf{p}, \mathbf{q}) = \frac{1}{||\mathbf{p} - \mathbf{q}||} \begin{cases} \mathbf{q} - \mathbf{p} & \text{if } \mathbf{D}(\mathbf{p}) \cdot (\mathbf{q} - \mathbf{p}) \geq 0 \\ \mathbf{p} - \mathbf{q} & \text{if } \mathbf{D}(\mathbf{p}) \cdot (\mathbf{q} - \mathbf{p}) < 0 \end{cases} \quad (28)$$

is the normalized bidirectional link or unit edge vector between pixels $\mathbf{p}$ and $\mathbf{q}$.

4. Combining Snakes and Livewire

With livewire, the user has no control of traces between seed points. When the shape of the object boundary is complex, or when it is near other strong but uninteresting object boundaries, many seed points are needed in order to generate an acceptable result. Furthermore, when a section of the desired object boundary has a weak edge relative to a nearby strong edge, the livewire snaps to the strong edge rather than the desired weaker boundary. In order to mitigate this problem, Mortensen and Barrett proposed *on-the-fly training* [13]. However, this method relies on the assumption that the edge property is relatively consistent along the object boundary. For example, in the lung image of Figure 1(a), the

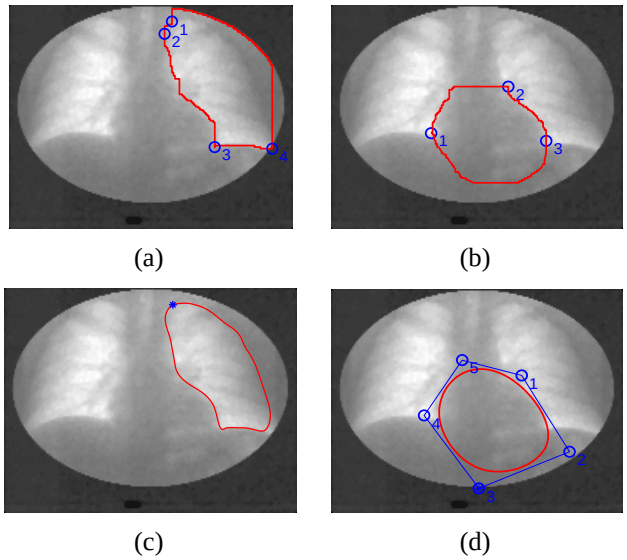

Figure 1. Delineation of the lung (a) and heart (b) in X-ray fluoroscopy images using livewire (seed points are shown). (c) Delination with Hermite snake constructed from livewire trace using first seed point as a hard constraint. (d) Result using B-snake (and control polygon) constructed from livewire trace.

livewire snaps to the strong edges of the elliptical viewport rather than the desired lung boundary. In this case, on-the-fly training is ineffective since the edge property of the lung boundary varies considerably over it extent.

Livewire is fundamentally image-based. Thus, it cannot effectively bridge gaps where the desired object boundaries are missing, and the smoothness of the traces can hardly be guaranteed. For instance, in Figure 1(b), part of the livewire trace from seed point 1 to seed point 2 is a straight line where the cardiac boundary is missing. The livewire technique does not generate an acceptable cardiac boundary from seed point 3 to seed point 1, and we have manually drawn a rough curve between the points.

Therefore, it is desirable to allow the user to exercise control over the livewire traces between seed points, impose smoothness on livewire traces, and bridge gaps along object boundaries. This is what snakes are very good at doing. Snakes adhere to edges with sub-pixel accuracy and they may also be adjusted interactively as parametric curves with intuitively familiar physical behaviors. Furthermore, snakes have the power to track moving objects, while livewire does not.

In most cases, however, livewire can quickly give much better results than casual manual tracing. Hence, the resulting livewire boundary can serve to initialize a snake. The livewire seed points reflect the user's prior knowledge of the object boundary. They can therefore serve as either hard or soft point constraints for the snake, depending on the user's confidence in the accuracies of these seed points.

Because a livewire-traced initial object boundary is more accurate than a hand-drawn boundary, and with the further incorporation of the seed points as snake constraints, the snake will very quickly lock onto the desired object boundary. If necessary, the user may correct mistakes inherited from the livewire-generated boundary by applying mouse-controlled spring forces to the snake. Because the user still has the opportunity to correct the mistakes on the traces as the snake is deforming, the number of seed points needed by livewire to generate the object boundary can be further reduced. Consequently, a coarse object boundary can be generated very quickly using livewire.

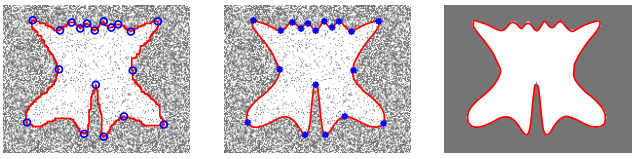

Figure 2. Performance of United Snakes demonstrated using a noisy synthetic image. (a) A livewire is sensitive to noise (the required seed points are shown). (b) The United Snake is robust against noise. (c) The segmented boundary accurately conforms to the ideal boundary.

Other hard or soft constraints may be added during the snake deformation process as well. Because constrained values may be changed dynamically, the user may adjust the seed points to further refine object boundaries as the snake deforms. In the lung segmentation example, a Hermite snake constructed from the livewire traces with the first seed as a hard constraint can firmly adhere to the lung apex, and it can easily be pulled out of the strong edge and lock onto the lung boundary as shown in Figure 1(c) without on-the-fly training. For the heart, a least squares approximation to the initial livewire curve with a 5-knot cubic B-spline is used to initialize a B-snake. A hard constraint may be further imposed on the control polygon node 3 to effectively bridge the gap along the heart boundary. The result is shown in Figure 1(d) after only a few iterations. The user may move the control polygon node 3 to refine that segment of the heart boundary if necessary.

As further evidence that United Snakes improves upon the robustness and accuracy of its component techniques, Fig. 2 shows a synthetic image of a known curve degraded by strong Gaussian white noise (variance 0.25). Given its image-based nature, the livewire is sensitive to noise as shown in Fig. 2(a). A snake initialized with the livewire gives a better result (Fig. 2(b)). Fig. 2(c) shows that the United Snakes result is very close to the boundary in the ideal image, despite the strong noise. This performance is a consequence of the imposed hard constraints, without which the snake would slip away from high curvature points.

5. Hard Constraints

The combination of snakes and livewire relies on an efficient constraint mechanism. A constraint on a snake may be either soft or hard. Hard constraints generally compel the snake to pass through certain positions or take certain shapes (generic hard constraints are discussed in [6]), while soft constraints merely encour-

age a snake to do so. Two kinds of soft constraints, springs and volcanos, were described in the original snakes paper [7] and they are incorporated into our finite element formulation. Hard constraints have been used to prevent snake nodes from clustering in dynamic programming snakes [1]. In this section, we propose a convenient mechanism, called *pins*, as a simple yet effective way to impose hard constraints on snakes for the integration of snakes and livewire.

Suppose that we wish to guarantee that the snake node i sticks at position (x_i^c, y_i^c) in the Hermitian parameterization. Recall that in the Hermitian parameterization, the polynomial shape of each element is parameterized by the position and slope of $x(s)$ and $y(s)$ at the two nodes (position and slope variables occupy alternating positions in the nodal variable vector **u**). Therefore, the snake stiffness matrix **K** may be updated with

$$\mathbf{K}_{2\cdot i,j} = \begin{cases} 1 & \text{if } 2i = j \\ 0 & \text{otherwise} \end{cases} \tag{29}$$

where $0 \leq j \leq 2(N-1)$ and N is the number of snake nodes, and the system force vector **F** is updated as

$$\mathbf{F}^x_{2\cdot i} = x_i^c, \qquad \mathbf{F}^y_{2\cdot i} = y_i^c, \tag{30}$$

where x and y indicate coordinate function $x(s)$ and $y(s)$, respectively. It is then guaranteed that the snake node i is always at position (x_i^c, y_i^c).

A drawback of this simple technique, however, is that the updated system stiffness matrix is no longer symmetric. Consequently, we are unable to economically save the stiffness matrix using skyline storage, nor factorize it in $\text{LDL}^\top$ form (see Appendix A). Nevertheless, since the position of node i is given, a constant force may be derived from the stiffness matrix for each degree of freedom and subtracted from its corresponding position in the system force vector so that we can restore the symmetry of the stiffness matrix while keeping the system in balance. That is, the system force vector **F** and the stiffness matrix are further updated with

$$\mathbf{F}^x_j = \mathbf{F}^x_j - \mathbf{K}_{j,2i} \times x_i^c \qquad \text{if } j \neq 2i \tag{31}$$

$$\mathbf{F}^y_j = \mathbf{F}^y_j - \mathbf{K}_{j,2i} \times y_i^c \qquad \text{if } j \neq 2i \tag{32}$$

$$\mathbf{K}_{j,2i} = \begin{cases} 1 & \text{if } 2i = j \\ 0 & \text{otherwise} \end{cases} \tag{33}$$

We can constrain the slope in the same way. If we constrain two node variables of an element in both position and slope, this element will be frozen. Its two neighboring elements will also be influenced by the constraint. The constraints on a B-snake are imposed on the nodes of its control polygon. Imposing hard constraints in this manner also lessens computational cost, in terms of both memory and time, since the number of entries in the skyline storage of the stiffness matrix is reduced. Consequently, the $\text{LDL}^\top$ factorization and forward/backward substitutions can be performed more efficiently (see Appendix A). It is also possible to apply more general constraints to any point on the snake as is described in [16].

In the formulation above, the updated stiffness matrix only indicates which degrees of freedom of the snake are constrained, it does not contain any constraint values. These are recorded in the system force vector. As a result, the constraint values may be updated dynamically during snake deformation. In other words, the user has the ability to move the constraint points around the image plane to refine the object boundary as the snake is deforming. This property is very useful when integrating snakes with livewire. While a snake is deforming, additional hard constraints may be imposed on the snake to restrict its deformation. Because these constraints are unknown before the snake is constructed, they may be incorporated on-the-fly using reaction forces on the system force vector without changing the stiffness matrix. However, small time steps are required to ensure the stability of the snake. In our implementation, we create a new snake from the current snake plus the hard constraints, since the $\text{LDL}^\top$ factorization is fast.

6. Applying United Snakes

In the United Snakes system, the user begins an image segmentation task using a livewire. An initial seed point is placed near the boundary of the object of interest. As the cursor, or free point, is moved around, the livewire or trace, is interactively displayed from the seed point to the free point. If the displayed trace is acceptable, the free point is collected as an additional seed point. For example, we can roughly capture a cell boundary in Figure 3(a) with just three seeds.

The livewire tends to stick to the object boundary using the seed points as a guide. The trace between the two adjacent seeds is frozen. The user has no further control over these traces other than backtracking. In order to generate a more accurate result in the area indicated by a rectangle, more seed points may be placed as in Figure 3(b). Although the livewire boundary is somewhat jagged and exhibits small errors, it is in general as accurate as manual tracing, but more efficient and reproducible.

Next we construct a snake using the livewire-generated boundary to initialize the snake and the seed points to constrain it. The user may select a shape function for the snake which is suitable for the object boundary. In our cell segmentation example, if the livewire result with five seed points is used to construct a finite difference snake, it is able to tolerate the livewire errors and very quickly and accurately lock onto the cell boundary (Figure 3(c)) without any need for further user interaction (the asterisks indicate the pins—imposed hard constraints). Using the livewire result with three seed points, the snake becomes "stuck" in the problematic area (Figure 3(d)) due to the livewire-generated boundary errors. However, this situation can be easily remedied using the mouse spring (Fig. 3(e)). Furthermore, as the snake is deforming, the hard constraints may be adjusted to refine the snake boundary. In Fig. 3(f) for example, constraint point 2 is moved to illustrate this snake boundary adjustment capability. By contrast, it is not nearly as easy to adjust a seed point in the livewire algorithm.

This form of livewire-snake integration is referred to as *static* integration—once the livewire result is used to initialize a snake, the segmentation process continues using only the constrained, user-controlled snake. The user may also set the United Snakes system to a more *dynamic* integration "mode": Once the livewire trace between the last seed point and the free point is formed, a corresponding open snake with constraints at the seed point and the

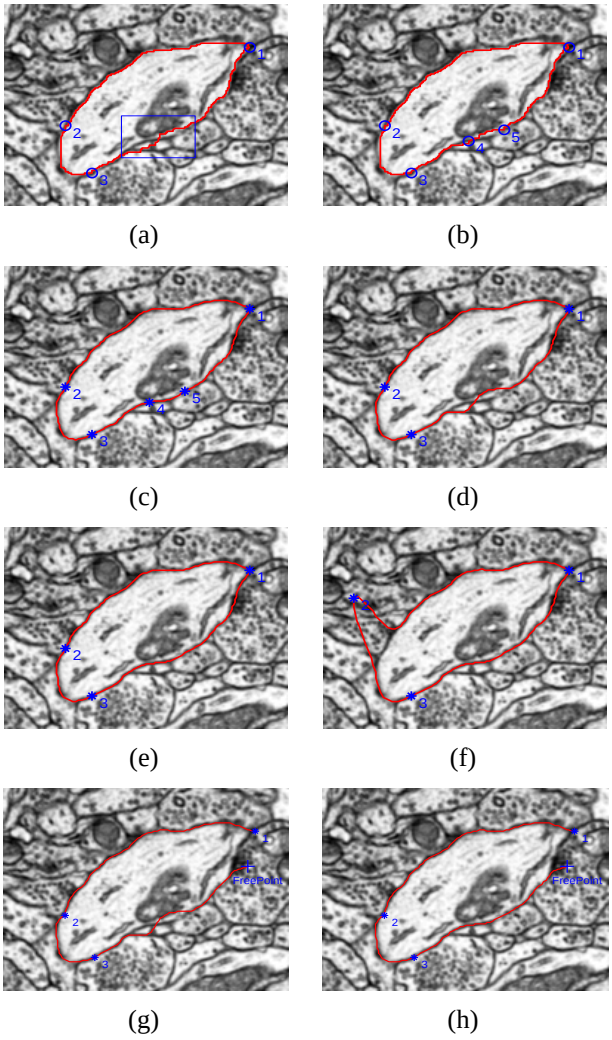

Figure 3. Using United Snakes to segment neuronal EM images (see text).

free point is constructed and automatically activated. When the free point is collected as a seed point, this open snake is merged with the snake constructed from the previous livewire traces (if they exist). All seed points are automatically applied as constraints. Fig. 3(g–h) illustrates this process, where "+" indicates the current free point. Since the snake is automatically set in motion, the user may use the mouse spring to correct it in any problematic areas along the snake (Fig. 3(h)).

In summary, the information from livewire including the user guidance and expert prior knowledge is fully utilized by the snake; the snake very quickly locks onto the image features of interest with reasonable tolerance to mistakes in the livewire traces. Thus, the integration of snakes and livewire creates an efficient, reproducible, accurate, semiautomatic segmentation tool which combines the power and flexibility of both techniques. We have applied United Snakes to several different medical image analysis projects in [8], demonstrating the generality, accuracy, robustness, and ease of use of the tool.

7. Conclusion

We have unified several snake variants in a finite element framework, argued that snakes and livewire are complementary to one another, and through an effective hard constraint mechanism, demonstrated that a snakes/livewire union enhances the power of snakes for interactive image segmentation. Furthermore, to meet the demand for a portable, reusable, comprehensive snake software package, we have implemented our work as a JavaBean which may easily be integrated into application systems.

We the creators of the United Snakes, in order to form a more perfect union of snake technologies, plan to incorporate within our framework, affine cell image decomposition methods for snake topological adaptability [9], advanced snake motion tracking mechanisms [17, 3], and other snake techniques. We anticipate that such efforts will further enhance the effectiveness of this image segmentation tool.

Acknowledgments

JL gratefully acknowledges the valuable comments and suggestions of Prof. Timo Järvi as well as the support of the Turku Centre for Computer Science and the Instrumentarium Foundation. The chest image was collected by Dr. Raimo Virkki and provided by Dr. Aaro Kiuru. The cell image was obtained from Dr. Kristen Harris of the Harvard Medical School.

A. Finite Element Snakes Formulation

To develop the finite element formulation and the corresponding matrix equations, we apply Galerkin's method to the Euler-Lagrange equation

$$-\frac{\partial}{\partial s}\left(\alpha\frac{\partial \mathbf{v}}{\partial s}\right)+\frac{\partial^2}{\partial s^2}\left(\beta\frac{\partial^2 \mathbf{v}}{\partial s^2}\right)-\mathbf{q}(\mathbf{v})=\mathbf{0}, \qquad (34)$$

which expresses the necessary condition for the snake at equilibrium. The average weighted residual is

$$I=\int_0^L\left(-\frac{\partial}{\partial s}\left(\alpha\frac{\partial \mathbf{v}}{\partial s}\right)+\frac{\partial^2}{\partial s^2}\left(\beta\frac{\partial^2 \mathbf{v}}{\partial s^2}\right)-\mathbf{q}(\mathbf{v})\right)w(s)\,ds=\mathbf{0}, \qquad (35)$$

where $w(s)$ is an arbitrary test function. By performing integrations by parts once for the first term and twice for the second term of equation (35), we arrive at the weak formulation of the snake model:

$$\int_0^L\left(\alpha\frac{\partial \mathbf{v}}{\partial s}\frac{\partial w}{\partial s}\right)ds+\int_0^L\left(\beta\frac{\partial^2 \mathbf{v}}{\partial s^2}\frac{\partial^2 w}{\partial s^2}\right)ds-\int_0^L \mathbf{q}w\,ds+\mathbf{B}=\mathbf{0}, \qquad (36)$$

where

$$\mathbf{B}=\left[-w\alpha\frac{\partial \mathbf{v}}{\partial s}+w\frac{\partial}{\partial s}\left(\beta\frac{\partial^2 \mathbf{v}}{\partial s^2}\right)-\frac{\partial w}{\partial s}\beta\frac{\partial^2 \mathbf{v}}{\partial s^2}\right]_0^L, \qquad (37)$$

are the boundary conditions at the two boundary points, $s = 0$ and $s = L$. We approximate $\mathbf{v}$ as

$$\mathbf{v} = \mathbf{N}\mathbf{u}, \tag{38}$$

where $\mathbf{N} = [N_1(s), N_2(s), ..., N_n(s)]$ are the shape functions, $\mathbf{u} = [u_1, u_2, ..., u_n]^\top$ the n nodal variables (degrees of freedom) of the snake model. In Galerkin's method, the arbitrary test function w takes the form

$$w = \mathbf{N}\mathbf{c}, \tag{39}$$

where $\mathbf{N}$ are the same shape functions as in equation (38), and $\mathbf{c}$ is an arbitrary vector. As w is a scalar, we have

$$w = w^\top = \mathbf{c}^\top \mathbf{N}^\top. \tag{40}$$

Substituting (38) through (40) into (36) yields

$$\mathbf{K}\mathbf{u} - \mathbf{F} + \mathbf{P} = \mathbf{0}, \tag{41}$$

where $\mathbf{K}$ is the stiffness matrix, $\mathbf{F}$ the force vector, and $\mathbf{P}$ the boundary forces, defined as follows:

$$\mathbf{K} = \mathbf{K}_\alpha + \mathbf{K}_\beta \tag{42}$$

$$\mathbf{K}_\alpha = \int_0^L \left(\frac{\partial \mathbf{N}}{\partial s}\right)^\top \alpha \left(\frac{\partial \mathbf{N}}{\partial s}\right) ds \tag{43}$$

$$\mathbf{K}_\beta = \int_0^L \left(\frac{\partial^2 \mathbf{N}}{\partial s^2}\right)^\top \beta \left(\frac{\partial^2 \mathbf{N}}{\partial s^2}\right) ds \tag{44}$$

$$\mathbf{F} = \int_0^L \mathbf{N}^\top \mathbf{q} ds \tag{45}$$

$$\mathbf{P} = \left[-\mathbf{N}^\top \alpha \frac{\partial \mathbf{N}}{\partial s} + \mathbf{N}^\top \frac{\partial}{\partial s}\left(\beta \frac{\partial^2 \mathbf{N}}{\partial s^2}\right) - \left(\frac{\partial \mathbf{N}}{\partial s}\right)^\top \beta \frac{\partial^2 \mathbf{N}}{\partial s^2} \right]_0^L \mathbf{u}. \tag{46}$$

Equation (41) gives the finite element formulation for the whole snake. To achieve acceptable accuracy in the finite element approximation, the integration domain should be discretized into a number of small subdomains resulting in the finite element mesh. That is, the snake contour is divided into small segments (elements), each of which can still be considered a snake. Applying equation (41) to an element, we have $\mathbf{K}^e\mathbf{u}^e - \mathbf{F}^e + \mathbf{P}^e = \mathbf{0}$, where $\mathbf{K}^e$ is the element stiffness matrix, $\mathbf{F}^e$ the element force vector, and $\mathbf{P}^e$ the element boundary forces applied to the boundary points of the element. Assembling the element matrices results in the system matrix equation

$$\mathbf{K}\mathbf{u} = \mathbf{F}, \tag{47}$$

where $\mathbf{F}$ is the generalized system force vector.

To solve the discrete form of the equations of motion (4), we replace the time derivatives of $\mathbf{u}$ with the backward finite differences $\ddot{\mathbf{u}} = (\mathbf{u}^{(t+\Delta t)} - 2\mathbf{u}^{(t)} + \mathbf{u}^{(t-\Delta t)})/(\Delta t)^2$, $\dot{\mathbf{u}} = (\mathbf{u}^{(t+\Delta t)} - \mathbf{u}^{(t)})/\Delta t$, where the superscripts denote the quantity evaluated at the time given in the parentheses and the time step is Δt. This yields the update formula

$$\mathbf{A}\mathbf{u}^{(t+\Delta t)} = \mathbf{b}\mathbf{u}^{(t)} + \mathbf{c}\mathbf{u}^{(t-\Delta t)}, \tag{48}$$

where $\mathbf{A} = \mathbf{M}/(\Delta t)^2 + \mathbf{C}/\Delta t + \mathbf{K}$ and $\mathbf{b} = 2\mathbf{M}/(\Delta t)^2 + \mathbf{C}/\Delta t$ and $\mathbf{c} = -\mathbf{M}/(\Delta t)^2$. Because $\mathbf{A}$ is symmetric and banded, it can be economically saved in skyline storage, and efficiently factorized uniquely into the form $\mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^\top$, where $\mathbf{L}$ is a lower triangular matrix and $\mathbf{D}$ is a diagonal matrix. The solution $\mathbf{u}^{(t+\Delta t)}$ to equation (48) is obtained by first solving $\mathbf{L}\mathbf{s} = \mathbf{b}\mathbf{u}^{(t)} + \mathbf{c}\mathbf{u}^{(t-\Delta t)}$ with forward substitution, then $\mathbf{L}^\top\mathbf{u} = \mathbf{D}^{-1}\mathbf{s}$ with backward substitution. Since $\mathbf{A}$ is constant, only a single factorization is necessary. Therefore, at each time step only the forward/backward substitutions are performed to integrate the snake forward through time.

References

[1] A. Amini, T. Weymouth, and R. Jain. Using dynamic programming for solving variational problems in vision. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 12(9):855–867, 1990.

[2] W. Barrett and E. Mortensen. Interactive live-wire boundary extraction. *Medical Image Analysis*, 1(4):331–341, 1997.

[3] A. Blake and M. Isard. *Active Contours*. Springer-Verlag, 1998.

[4] L. Cohen and I. Cohen. Finite element methods for active contour models and balloons for 2D and 3D images. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 15(11):1131–1147, November 1993.

[5] A. X. Falão, J. K. Udupa, S. Samarasekera, and S. Sharma. User-steered image segmentation paradigms: Live wire and live lane. *Graphical Models and Image Processing*, 60:233–260, 1998.

[6] P. Fua and C. Brechbühler. Imposing hard constraints on deformable models through optimization in orthogonal subspaces. *Computer Vision and Image Understanding*, 65:148–162, 1997.

[7] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

[8] J. Liang, T. McInerney, and D. Terzopoulos. Interactive medical image segmentation with united snakes. In *Proc. Second International Conf. on Medical Image Computing and Computer-Assisted Intervention (MICCAI 99)*, Cambridge, England, September 1999. Springer.

[9] T. McInerney and D. Terzopoulos. Topologically adaptable snakes. In *Proc. Fifth International Conf. on Computer Vision (ICCV'95), Cambridge, MA, June, 1995*, pages 840–845, Los Alamitos, CA, 1995. IEEE Computer Society Press.

[10] T. McInerney and D. Terzopoulos. Deformable models in medical image analysis: A survey. *Medical Image Analysis*, 1(2):91–108, 1996.

[11] S. Menet, P. Saint-Marc, and G. Medioni. B-snakes: Implementation and application to stereo. In *Proceedings DARPA*, pages 720–726, 1990.

[12] E. N. Mortensen and W. A. Barrett. Intelligent scissors for image composition. In *Proceedings of Computer Graphics (SIGGRAPH'95)*, pages 191–198, Los Angeles, CA, August 1995.

[13] E. N. Mortensen and W. A. Barrett. Interactive segmentation with intelligent scissors. *Graphical Models and Image Processing*, 60:349–384, 1998.

[14] A. Singh, D. Goldgof, and D. Terzopoulos, editors. *Deformable Models in Medical Image Analysis*. IEEE Computer Society Press, 1998.

[15] L. Staib and J. Duncan. Boundary finding with parametrically deformable models. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 14(11):1061–1075, November 1992.

[16] D. Terzopoulos and H. Qin. Dynamic NURBS with geometric constraints for interactive sculpting. *ACM Transactions on Graphics*, 13(2):103–136, 1994.

[17] D. Terzopoulos and R. Szeliski. Tracking with Kalman snakes. In A. Blake and A. Yuille, editors, *Active Vision*, pages 3–20. MIT Press, Cambridge, MA, 1992.