Level Set Active Contours on Unstructured Point Cloud

Hon Pong Ho†, Yunmei Chen‡, Huafeng Liu†,* and Pengcheng Shi†
†Department of Electrical and Electronic Engineering
Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong
‡Department of Mathematics, University of Florida, Gainesville, Florida, USA
*State Key Lab of Modern Optical Instrumentation, Zhejiang University, Hangzhou, China
{garyho@ust.hk, yun@math.ufl.edu, eeliuhf@ust.hk, eeship@ust.hk}

Abstract

We present a novel level set representation and front propagation scheme for active contours where the analysis/evolution domain is sampled by unstructured point cloud. These sampling points are adaptively distributed according to both local data and level set geometry, hence allow extremely convenient enhancement/reduction of local front precision by simply putting more/fewer points on the computation domain without grid refinement (as the cases in finite difference schemes) or remeshing (typical in finite element methods). The front evolution process is then conducted on the point-sampled domain, without the use of computational grid or mesh, through the precise but relatively expensive moving least squares (MLS) approximation of the continuous domain, or the faster yet coarser generalized finite difference (GFD) representation and calculations. Because of the adaptive nature of the sampling point density, our strategy performs fast marching and level set local refinement concurrently. We have evaluated the performance of the method in image segmentation and shape recovery applications using real and synthetic data.

1 Introduction

Active contour models have been among the most successful strategies in object representation and segmentation [4, 7, 14]. While in general the parametric active contours have explicit representations as parameterized curves in a Lagrangian formulation [7, 25], the geometric active contours (GACs) are implicitly represented as level sets of higher-dimensional, scalar level set functions and evolve in an Eulerian fashion [4, 14, 18]. The geometric nature of the GACs gives them several often desired properties, i.e. independence from parametrization, easy computation of curvatures, and natural compatibility with topological changes.

In this paper, we are going to present numerical schemes for geometric active contours, via level set representation, on adaptively distributed cloud of points. The key issues include (1) unstructured, point-based domain sampling adaptive to local internal (model geometry) and external (input data) features, and (2) obtaining level set surface gradient needed for front evolution, using local surface patches within an influence domain of the fiducial point through moving least square approximation or generalized finite difference schemes. We want to point out that we are focusing, in this paper, on the numerical implementations of any forms of GACs on unstructured point cloud in order to improve their accuracy, efficiency, and flexibility. On the other hand, appropriate evolution criteria still play essential roles in achieving satisfactory results and will be separately addressed in our related efforts.

Figure 1. Domain and curve representation schemes. Finite difference (top): regular grid (left), refined grid (middle), and moving deformation grid (right); Finite element (bottom): regular triangular mesh (left), and refined triangular mesh (middle); Adaptive, unstructured point cloud (bottom, right).
1.1 Previous Works

One key issue in GACs is the tradeoff between computational efficiency and contour precision/segmentation accuracy, and several algorithmic improvements such as the fast marching scheme [23] and the narrowband search [14] have provided some reliefs. However, since most GACs rely on the finite difference computational schemes, the resolutions of the final results are fundamentally dependent on the computational grids used to solve the discreteized GAC partial differential equations (PDEs). While a highly refined grid allows high accuracy, the computational cost may be prohibitively expensive for practical applications. On the other hand, coarse grid allows fast computation, at the expense of low fidelity to the original continuous contours.

A popular solution to the gridding problem has been the use of adaptive grid techniques to locally modify the grid structure based on certain criteria [2]. The local refinement methods place additional grid nodes when (i.e. during narrowband refinement) and where (i.e. high gradient areas) they are needed to avoid excessive global refinement procedure [5, 22]. More recently, moving grid methods have been proposed as simpler alternatives to the local refinement methods [6, 9]. Instead of adding grid nodes as needed, these methods move grid lines towards salient feature locations to increase the sampling rates at those regions while the overall sampling rate remains constant. The overhead cost is an additional mapping between the deformed level set grid and the original grid, and the physical distance among grid points on the deformed grid is needed to calculate the derivatives using finite difference since spatial grid points separation are no longer unique. An efficiency gain can be justified, for example, by having the overall sampling rate much lower than the original image grid, while having local rate similar to image grid for high gradient areas. But the local sampling rate for moving grid strategy is still upper-bounded by the number of initial grid lines used, which occurs at the extreme situation when all grid lines moved towards one point on the image. The top row of Fig. 1 illustrates the regular, locally refined, and the moving grid finite difference schemes for contour representation.

Since the purpose of domain representation is to offer an appropriate embedding for various calculations needed by the front representation and evolution, one is obviously not limited to any specific finite difference sampling. Finite element mesh has emerged as an alternative scheme for GAC algorithms by offering a piecewise continuous approximation of the domain where the level set computations are performed [1, 24]. Similar to the local refinement procedures in finite difference, the triangular mesh can be enhanced through adaptive element subdivision to provide higher sampling rate at interesting areas [24]. The remeshing process, however, could be computationally expensive and algorithmically difficult, especially for 3D problems. The left and middle figures (bottom row) in Fig. 1 show the regular and refined FEM representations.

1.2 Contributions

In the Lagrangian formulation of parametric active contour models and motion tracking, meshfree particle methods have been introduced as more efficient and possibly more effective object representation and computation alternatives because of their trivial h-p adaptivity [11]. In addition, it has been shown that, for arbitrary shape modeling, point-sampled representation allows user-defined large deformations on the object shape [20]. Following the same spirit, it is easy to see that if general level set PDE computation is conducted in a meshfree environment for both shape and evolution (deformation) domains, we can freely adjust the sampling rate of the domains without laborious extra efforts. This can be achieved through point-wise continuous approximations of the domain and of the level set functions through polynomials fitting using local point cloud.

Hence, we introduce a unified numerical scheme for level set representation and evolution on feature-adaptive cloud of points. The contour evolution process is performed on the point-sampled analysis domain through either moving least squares (MLS) approximation or generalized finite difference (GFD) schemes. Because of the h-p adaptive nature of the sampling point, our strategy naturally possesses multi-scale property and performs fast marching and local refinement concurrently.

2 Methodology

2.1 Implicit Surface Modeling

Suppose we want to model a dynamic deformable interface $C(t) \in \mathbb{R}^d$ under a known velocity field $F(x, C(t))$, where $x \in \mathbb{R}^d$ with given initial value $C(0)$. For example, if $d = 2$ then $x = (x, y)$ is a 2D point and $C(t)$ is a 2D contour; and if $d = 3$ then $C(t)$ is a 3D surface where $x = (x, y, z)$ represents a particular 3D point. The dynamic equation for the moving interface can be written as:

$$\frac{dC(t)}{dt} = F(x, C(t)) \quad (1)$$

If $F$ is relatively simple, the above equation can be solved numerically by first discretizing $C(t)$ using a line/surface mesh, and then iteratively updating the positions of the vertices on the mesh under the inference of $F$. This can be referred to the classical finite element method (FEM).

However, if $F$ is more complex such that $C(t)$ may undergo structural changes or have singularities, i.e. breaking into two or more pieces or non-differentiable when a
corner is formed, then direct discretization on $C(t)$ surface may result in tedious issues while handling the $C(t)$ mesh. Therefore, changing the representation of $C(t)$ into implicit form had been developed, known as the level set methods [17, 21], where the basic idea is to implicitly represent the moving interface $C(t)$ by a higher dimensional hypersurface $\phi$ (the level set function). Let $\phi$ to be the signed distance transformation of $C(t)$ in the $\mathbb{R}^d$ domain:

$$\phi(x, t) = H(x, C(t)) \min\{|x - C(t)|\}$$  \hspace{0.5cm} (2)

$$H(x, C) = \begin{cases} 
1 & \text{if } x \text{ outside } C \\
-1 & \text{if } x \text{ inside } C \\
0 & \text{if } x \in C 
\end{cases}$$  \hspace{0.5cm} (3)

where $\cdot$ denotes the norm of vector. In other words, $\phi(x, t)$ is the scalar distance between $x$ and its nearest $C(t)$ location. So the zero level set of $\phi$ is actually the set of $C(t)$ such that:

$$C(t) = \{x \mid \phi(x, t) = 0 \ \forall x \in \mathbb{R}^d\}$$  \hspace{0.5cm} (4)

Differentiate the above equation with respect to $t$, and assume all points on $\phi$ move along their normal directions $n$ controlled by the scalar velocity field $F = n \cdot x'(t)$ with $n = \nabla \phi$, we have

$$\frac{\partial \phi}{\partial t} = F \cdot \nabla \phi$$  \hspace{0.5cm} (5)

where $\nabla \phi$ represents the norm of the gradient of $\phi$.

The curve evolution is now driven by Eqn. (5) and we are going to solve for convergence of this dynamic equation through time-domain finite difference:

$$\phi^{t+\Delta t} = \phi^t + \Delta t F \cdot \nabla \phi$$  \hspace{0.5cm} (6)

$$\nabla \phi = \left[ \frac{\partial \phi}{\partial x}, \frac{\partial \phi}{\partial y}, \frac{\partial \phi}{\partial z} \right]$$  \hspace{0.5cm} (7)

$$|\nabla \phi| = \sqrt{\left(\frac{\partial \phi}{\partial x}\right)^2 + \left(\frac{\partial \phi}{\partial y}\right)^2 + \left(\frac{\partial \phi}{\partial z}\right)^2}$$  \hspace{0.5cm} (8)

Eqn. (6) is the basic level set formulation for modeling front propagation. In fact, there are many motion governing formulations for level set active contour modeling. Since proper construction of $F$ is not the main concern of this paper, we will discuss it for specific applications later. Our emphasis is to offer an alternative domain representation scheme using point cloud such that spatial calculations can be conducted in a more efficient and more accurate fashion.

The discrete samples $z$ of the continuous zero level set curve $C(t)$ can be found through [17, 21]:

$$z = x - \phi(x, t) \frac{\nabla \phi}{|\nabla \phi|}$$  \hspace{0.5cm} (9)

where $z_i \subset C(t)$. Another way to find $z$ is by detecting the sign changes (crossings) of $\phi$ values between any neighboring pair $\{\phi(x), \phi(x \pm \Delta x)\}$, or $\{\phi(x), \phi(x \pm \Delta y)\}$, or $\{\phi(x), \phi(x \pm \Delta z)\}$, which can be done with the Marching Cubes algorithm [13]. Regardless of the detection methods, the requirement for $z$ is that $\phi(z)$ must be numerically very close to zero at any time $t$.

The accuracy of the solution for $C(t)$ depends on how small $\Delta x$ can be in Eqn. (11), which is ultimately constrained by the density of $\phi$ (i.e. the sampling rate). Theoretically, increasing the density of $\phi$, either globally or locally (i.e. through grid refinement/moving grid), can give better results for $C(t)$, with added computational costs and bookkeeping of data structure.

### 2.2 Spatial Discretization: Finite Difference Grid vs. Meshfree Point Cloud

In order to numerically implement Eqn. (6) for a given $F$, we need to address three key issues: 1). proper discrete representation of the hypersurface $\phi$ so that computational operations can be performed; 2). obtaining the derivatives of $\phi$ (i.e. $\nabla \phi$); and 3). detecting the zero level set of $\phi$ (i.e. $C(t)$). The work flow for one temporal step of level set evolution can be summarized as:

| signed gradient levelset zero crossing distance approx. evolution detection |
|-----------------------------|-----------------------------|-----------------------------|
| $C(t)$ → $\phi(x, t)$ → $\nabla \phi$ → $\phi(x, t + \Delta t)$ → $C(t + \Delta t)$ |

**Finite difference grid.** Here, domain sampling can be interpreted as keeping a finite subset of $\phi$ values at a finite subset of grid locations $\mathbb{N}^d$:

$$\phi_i \subset \phi \mid \phi_i(x_i, t) = \phi(x_i, t) \forall x_i \in \mathbb{N}^d \subset \mathbb{R}^d$$  \hspace{0.5cm} (10)

where $i$ is an integer index for the sample of $\phi$ at location $x_i$. If we have $N$ samples in total, then $i = 1...N$. For 2D problem (i.e. $d = 2$), we typically do not use $i$ but instead use a finite set of $(x, y)$ pairs where $x, y$ are confined to be integer values.

$\nabla \phi$ can be obtained using Eqn. (7) through spatial discrete approximation of the partial derivative:

$$\frac{\partial \phi}{\partial x} \approx \frac{\Delta \phi}{\Delta x} = \frac{\phi(x_i + \Delta x, t) - \phi(x_i, t)}{\Delta x}$$  \hspace{0.5cm} (11)

where $\Delta x$ is a small spatial displacement along $x$ direction. Derivatives along $y, z$ directions can be similarly computed.

One favored strategy to reduce the size of $\mathbb{N}^d$ without losing much accuracy is that we can use a subset $\mathbb{N}_{NB}^d \subset \mathbb{N}^d$ which only includes samples near the current $C(t)$ [14]:

$$\mathbb{N}_{NB}^d(C(t)) = \{x_i \mid |x_i - C(t)| < w \forall x_i \in \mathbb{N}^d\}$$  \hspace{0.5cm} (12)

where $w$ is a small constant. The elements in $\mathbb{N}_{NB}^d$ will occupy a band shape space around $C(t)$, therefore this reduction of samples is usually called narrow band speed-up,
and \( w \) the bandwidth. Normally \( \mathbb{N}^d \) is a static set that only related to input data, like pixel or voxel locations, while \( \mathbb{N}^d_{NB} \) is a dynamic set that also relates to the status of current \( C(t) \). Note that the sampling density within \( \mathbb{N}^d_{NB} \) has not been improved.

**Meshfree point cloud.** To aim for better accuracy, we define a new dynamic set of sampling locations to replace \( \mathbb{N}^d_{NB} \) in Eqn. (12):

\[
x_i \in P^d_{NB}(f, C(t)) \subset \mathbb{R}^d
\]

where \( f : \mathbb{R}^d \rightarrow \mathbb{R} \) is a function of the input data. By using the narrow band concept, members of set \( P^d_{NB} \) form a point cloud surrounding \( C(t) \) with its local sampling density determined by the input data and the geometry of \( C(t) \). The term point cloud means that these sampling points are not structured by pre-established grid or mesh. Fig. 4 shows different types of point during evolution process. While this unstructured point cloud sampling gives us flexibility in distributing more/fewer points where the data and front geometry demand, the detection of zero level set can no longer be performed using the Marching Cubes algorithm, which requires the grid structure, and we have used Eqn. (9) instead.

**2.3 Feature-Driven Domain Sampling: Unstructured Point Cloud**

Here, we discuss the detailed steps of our input data/front geometry driven domain representation with meshfree point cloud \( x_i \). We would like to name \( x_i \) the level set supporting nodes (or just nodes) to differentiate them from other types of points on the image. It is worthwhile to note that there is no spatial ordering between \( x_i \) and \( x_{i\pm n} \) for any integer \( n \).

It is essential to get a higher precision representation, with more sampling nodes, at those regions where the input data or the current front \( C(t) \) contains rich information, i.e. high image gradient or complicated geometry. But exactly what kind of features needed to be considered is application dependent, and we may even include prior knowledge (i.e. the ear positions when performing shape recovery of the bunny in Fig. 6) in the construction of \( P^d(f, C(t)) \). In the following paragraphs we only discuss adaptive node distribution driven by the data-derived \( f \), but similar procedures have been done for the front geometry and prior information in our implementation and experiment.

We quantify our requirement by setting the node density around a point \( f(x) \) proportional to its data feature \(^1\):

\[
\frac{\text{Node Density at } x}{\text{Area}} \propto f(x) \quad (14)
\]

\[
\frac{\text{Nodes}}{\text{Area}} = \rho f(x), \text{ where } \rho \text{ is constant} \quad (15)
\]

\(^1\)Bilinear interpolation is needed if \( f \) is only defined on grid positions

In Eqn. (15), determining the Area for node density measurement at any point \( x \) would be an issue of selecting proper scale, so we would like to let the interpolant of \( \phi \) to control it by fixing the number of Nodes within an Area to be the minimum amount of neighboring nodes \( x \) that the interpolant needs to interpolate the \( \phi(x) \) value. Assume Area to be circular, we get

\[
\text{Area} = \pi R^2 = \frac{k}{\rho f(x)} \quad \text{or} \quad R = \sqrt{\frac{k}{(\pi \rho) f(x)}} \quad (16)
\]

where \( k \) is the minimum number of neighboring nodes to interpolate \( \phi(x) \), and \( R \) is the radius that we are going to spread \( k \) nodes around \( x \). Before going to the algorithm, there is a little problem with Eqn. (16) when \( f(x) \rightarrow 0 \), which implies that there is little data-driven information at \( x \) (such as image background or smooth regions), the spreading radius goes unbounded. To guarantee a reasonable coverage of samples at these areas, we first initialize a base set of nodes which have a uniform separation \( R_{min} = \sqrt{\frac{k}{(\pi \rho) \text{mean}(f(x))/2}} \) apart. To further speed up the distributing process, we also like to include the highest 1-2% \( f(x) \) points in the base set. Note that while the base set might look like grid points, we do not assume any ordering relationship among these nodes in the set.

If the problem space \( \mathbb{R}^d \) is large, exhaustive put-and-check for node density at every possible \( x \) would be very slow. We have thus modified an existing mesh refining algorithm [12] to a node distribution algorithm. The original algorithm assumes the actual boundary of the to-be-meshed object is known and the node density at a point is controlled by how far that point is away from the known boundary. Regions closer to actual boundary will have finer triangulation. The two main differences of our algorithm are that, first, the point-to-boundary measurement is replaced by the \( f(x) \) function, since the actual boundary is unknown in our problems. Second, the triangulation is not performed as the mesh is not needed in our method. Nevertheless, our method is still recursive and requires an initial coarse node distribution. The idea is that we check every initial nodes on the problem space \( \mathbb{R}^d \) by counting the amount of neighboring nodes already existed within an area of radius \( R \) defined by Eqn. (16). If any node failed to have \( k \) neighbors, then that node is removed and additional new nodes are randomly added to fill up the shortage. During the checking, all newly created nodes are stored in a list for next iteration. After checking all initial nodes, the same checking procedures are performed on the list of new nodes and produces another list of new nodes. The process terminates until an empty list is found at the end of an iteration which implies no new node has been added in the last iteration. See Fig. 2 for an example of the process. Then all nodes have at least \( k \) neighbors in its influence domain, see next section.
Algorithm 1 (Feature-Driven Node Distribution)

CurrrList ← Base Node Set
NodeSet ← Base Node Set
While (CurrrList is not empty )
    Randomize the order of node in CurrrList
    NextList ← Empty
    For every node x in CurrrList
        Rx ← min(sqr(x/(k*p)sfx(x))),Rmin)
        N ← # of Nodes in NodeSet within Rx circle
        If( k-N > 0 )
            Remove x from NodeSet
            NewNodes ← Random (k-N) Nodes in Rx circle
            Add NewNodes into NextList
            Add NewNodes into NodeSet
        End If
    End For
End While

2.4 Level Set Gradient Approximation: Moving Least Square Method

Influence domain: In order to implement Eqns. (6) and (8), we need not only those \( \phi(x_i) \) samples but also its derivatives: \( \phi_x, \phi_y, \phi_z \). Traditionally, these quantities are approximated by taking spatial finite difference on grid-based samples of \( \phi(x,y,z) \). Since now we have changed our data structure to randomly distributed set of points instead of structured grid points, we no longer have the ordered neighboring relationship in x-, y-, and z-directions to conduct such calculations.

Assuming that the \( \phi \) function at any location \( x \) is a well-behaved polynomial surface within a small area around \( x \), we can use the neighboring \( \phi(x_i) \) nodes in that area to recover the coefficients of \( \phi \) surface by performing a least square fitting. Calling such local area the influence domain or support region of \( x \), all nodes inside the influence domain will have certain amount of effects on the recovered coefficients, which depend on how the least square fitting is performed. In general data fitting problem, the size of the influence region and the order of polynomial basis function control the complexity of the reconstructed surface. In our framework, since node distribution has already been feature-driven, fixing the polynomial order and choosing a fixed number of nearest neighboring nodes for \( x \) have implicitly the same meaning as determining a feature-adaptive size for the influence domain. Hence, we define the Area, where the \( k \) neighboring nodes \( x_i \) are found, to interpolate the \( \phi \) value at \( x \) to be the influence domain of \( x \) (see the left figure of Fig. 3 for an illustration).

Moving Least Square Approximation: Originally developed for data fitting and surface construction [8], the moving least square (MLS) approximation has found new applications in emerging meshfree particle methods [10, 11, 15]. MLS approximation is chosen because it is compact which means the continuity of the approximation within overlapping influence domains is preserved. In this paper, we use \( \phi \) to denote the MLS approximated \( \phi \) for better description. Using MLS, \( \phi \) surface is defined to be in the form:

\[
\phi(x) = \sum_{j} p_j(x^T) a_j(x) \equiv p^T(x) a(x) \tag{17}
\]

where \( p \) is a vector of polynomial basis, \( m \) is the order of polynomial, and \( a \) is a vector of coefficients for the \( \phi \) function at location \( x \) using the polynomial basis. For example, if \( p(x) = [1, x, y, z] \), then \( a(x) = [a_1(x), a_2(x), a_3(x), a_4(x)] \) and they are arbitrary function of \( x \). To get \( \phi(p) \) at arbitrary location \( x \), we determine the coefficients \( a \) through the \( k \)-nearest neighboring nodes \( x_i \) and their known \( \phi(x_i) \) values. The weighted approximation error is defined as :

\[
J = \sum_{i} W(\frac{|x-x_i|}{R_x}) |p^T(x_i)a(x) - \phi(x_i)|^2 \tag{18}
\]

where \( W \) is a cubic spline decaying weight function (see the right figure of Fig. 3), \( |x-x_i| \) is the spatial distance...
between $x$ and $x_i$, and $T$ denotes vector transpose. Solve for $a(x)$ through minimizing $J$ we can get:

$$\dot{\phi}(x) = N(x)^T \mathbf{U}_s$$

(20)

$$\mathbf{U}_s = [\phi(x_1), \phi(x_2), \ldots, \phi(x_k)]^T$$

(21)

$$N(x) = \mathbf{p}^T(x) \mathbf{A}(x)^{-1} \mathbf{B}(x)$$

(22)

$$\mathbf{A}(x) = \sum_k W(|x - x_i|) \mathbf{p}(x_i) \mathbf{p}^T(x_i)$$

(23)

$$\mathbf{B}(x) = W(|x - x_i|) \mathbf{p}(x_1), \mathbf{p}(x_2), \ldots, \mathbf{p}(x_k)]$$

(24)

The solution of $a(x)$ has already embedded in $N(x)$ of Eqn. (22) for simplification. The $N(x)$ is called the shape function that interpolates the known vector $\mathbf{U}_s$, which is a collection of known $\phi(x_i)$ values, to get the approximated $\dot{\phi}(x)$ value.

To get the first order derivatives of $\dot{\phi}$, let

$$\mathbf{A}(x) \mathbf{Y}(x) = \mathbf{p}(x)$$

(25)

$$\mathbf{Y}_x = \mathbf{A}^{-1}(\mathbf{p}_x - \mathbf{A}_x \mathbf{Y})$$

(26)

$$\dot{\phi}_x = \mathbf{Y}_x \mathbf{B} + \mathbf{Y} \mathbf{B}_x$$

(27)

while $\dot{\phi}_x$ is the shorthanded notation for $\frac{\partial \dot{\phi}(x)}{\partial x}$, and $\dot{\phi}_y, \dot{\phi}_z$ can be obtained in similar fashion. Finally, $|\nabla \dot{\phi}|$ can be obtained from Eqn. (8).

In many practical 2D problem like image segmentation, curvature measurement $\kappa$ is also needed, see later Eqn. (30). For the second order terms $\phi_{xx}, \phi_{yy}$ and $\phi_{xy}$ in $\kappa$:

$$\mathbf{Y}_{xx} = \mathbf{A}^{-1} (\mathbf{p}_{xx} - 2 \mathbf{A}_x \mathbf{Y}_x - \mathbf{A}_{xx} \mathbf{Y})$$

(28)

$$\mathbf{Y}_{xy} = \mathbf{A}^{-1} (\mathbf{p}_{xy} - (\mathbf{A}_x \mathbf{Y}_y + \mathbf{A}_y \mathbf{Y}_x + \mathbf{A}_{xy} \mathbf{Y}))$$

(29)

$$\dot{\phi}_{xx} = \mathbf{Y}_{xx} \mathbf{B} + 2 \mathbf{Y}_x \mathbf{B}_x + \mathbf{Y} \mathbf{B}_{xx}$$

Moreover, $\dot{\phi}_{yy}$ can be obtained in similar way as $\dot{\phi}_{xx}$. With $\phi_{xx}, \phi_{yy}$ and $\phi_{xy}, \kappa$ can be computed using Eqn. (32).

Generalized Finite Difference Method. While the MLS approximation produces very nicely estimated surface $\phi$ at $x$ (see bottom of Fig. (4) for an example), it demands quite a lot computations in order to get the interpolation function in Eqn. (22). A faster and less expensive alternative to MLS is to remove the weighting function $W$ in Eqn. (18) and to solve $a$ by least square solution of a set of simultaneous equations. This can be in fact considered as a generalized finite difference scheme, and some related details can be found in [3].

3 Applications

As discussed earlier, while the main focus of this paper is not on the $F$ function, we do want to point out that appropriate application-dependent $F$ should be used in Eqn. (6) in order to achieve best possible results [16, 26].

In the implementation and experiment results presented in this paper, we have selected the $F$ formulation for contour evolution from [19]:

$$F = \beta g \kappa - (1 - \beta) g (\hat{v} \cdot \nabla \phi)/|\nabla \phi|$$

(30)

$$g = 1/(1 + |\nabla G * I|)$$

(31)

Eqn. (30) consists of mainly two terms. The first term is the product of image gradient force $g$ and contour curvature $\kappa$, serving respectively as external and internal energy constraints during evolution. The second term is the additional external force using the inner product between image gradient vector flow (GVF) and contour normal direction. The incorporation of GVF enhances the flexibility of choosing initial contour position and handling concave object boundary. The parameter $\beta \in [0, 1]$ is just a constant weighting between the two main terms. $G$ denotes the gaussian smoothing kernel and finally contour curvature $\kappa$ can be obtained from the divergence of the gradient of the unit normal vector to the contour [14]:

$$\kappa = \nabla \cdot \nabla \phi/|\nabla \phi| = -\frac{\phi_{xx}\phi_y^2 - 2\phi_{xy}\phi_y\phi_{xy} + \phi_{yy}\phi_x^2}{(\phi_x^2 + \phi_y^2)^{3/2}}$$

(32)

where $\phi_x$ is the shorthanded notation for first order derivative $\frac{\partial \phi}{\partial x}$, $\phi_{xx}$ denotes $\frac{\partial^2 \phi}{\partial x^2}$ and $\phi_{xy}$ stands for $\frac{\partial^2 \phi}{\partial x \partial y}$.

We have implemented and tested the level set active contours on unstructured point cloud using both MLS and GFD. Quantitative comparisons are made between traditional finite difference implementation and our efforts on segmentation of synthetic data with known ground truth (see Table 1). In addition, application results to brain tissue segmentation (Fig. 5) and to shape recovery of the Stanford Bunny (Fig. 6) are presented.

4 Discussions

In this paper, we have not introduced new curve evolution formula for level set active contours. Instead, we have presented strategies on the numerical implementations of any forms of GACs on unstructured point cloud in order to improve their accuracy, efficiency, and flexibility.

The advantages of this point-based level set implementation over finite difference schemes (regular, local refinement, and moving grid) are evident. To achieve higher precision at interesting locations, we can adaptively distribute more sampling nodes in the associated solution domain. If holding the order of surface polynomial constant, areas with more nodes can have smaller influence domains since the amount of nodes needed for surface reconstruction can be kept constant. The areas of the solution domain which have
Figure 4. Top: Different types of sampling nodes during contour evolution process (left to right). Bottom: A visualization of the actual computational domain, MLS interpolated initial, intermediate and final $\phi$.

Table 1. Comparison between point-based and regular-grid level set implementations for noisy synthetic image segmentation.

<table>
<thead>
<tr>
<th>SNR</th>
<th>Images with different levels of noise</th>
</tr>
</thead>
<tbody>
<tr>
<td>10dB</td>
<td></td>
</tr>
<tr>
<td>6.9dB</td>
<td></td>
</tr>
<tr>
<td>5.2dB</td>
<td></td>
</tr>
<tr>
<td>3.9dB</td>
<td></td>
</tr>
</tbody>
</table>

Point-based Level Set Results with MLS Interpolation

Point-based Level Set Results with GFD Interpolation

Regular Grid Level Set Results with Finite Difference

<table>
<thead>
<tr>
<th>3 Objects</th>
<th>Noise</th>
<th>MSE</th>
<th>S.D.</th>
<th>Min Err</th>
<th>Max Err</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLS</td>
<td>10dB</td>
<td>0.222</td>
<td>0.324</td>
<td>0.006</td>
<td>2.093</td>
<td>372</td>
</tr>
<tr>
<td>MLS</td>
<td>6.9dB</td>
<td>0.402</td>
<td>0.4</td>
<td>0.002</td>
<td>2.084</td>
<td>295</td>
</tr>
<tr>
<td>MLS</td>
<td>5.2dB</td>
<td>0.405</td>
<td>0.410</td>
<td>0.002</td>
<td>1.820</td>
<td>224</td>
</tr>
<tr>
<td>MLS</td>
<td>3.9dB</td>
<td>0.504</td>
<td>0.464</td>
<td>0.003</td>
<td>2.133</td>
<td>317</td>
</tr>
<tr>
<td>GFD</td>
<td>10dB</td>
<td>0.480</td>
<td>0.443</td>
<td>0.000</td>
<td>2.166</td>
<td>164</td>
</tr>
<tr>
<td>GFD</td>
<td>6.9dB</td>
<td>1.324</td>
<td>0.606</td>
<td>0.001</td>
<td>2.954</td>
<td>106</td>
</tr>
<tr>
<td>GFD</td>
<td>5.2dB</td>
<td>1.218</td>
<td>0.654</td>
<td>0.006</td>
<td>3.322</td>
<td>148</td>
</tr>
<tr>
<td>GFD</td>
<td>3.9dB</td>
<td>2.119</td>
<td>1.039</td>
<td>0.014</td>
<td>6.057</td>
<td>126</td>
</tr>
<tr>
<td>FD-LS</td>
<td>10dB</td>
<td>1.074</td>
<td>0.617</td>
<td>0.008</td>
<td>3.318</td>
<td>341</td>
</tr>
<tr>
<td>FD-LS</td>
<td>6.9dB</td>
<td>2.11</td>
<td>0.999</td>
<td>0.000</td>
<td>5.298</td>
<td>340</td>
</tr>
<tr>
<td>FD-LS</td>
<td>5.2dB</td>
<td>2.128</td>
<td>0.994</td>
<td>0.002</td>
<td>5.952</td>
<td>344</td>
</tr>
<tr>
<td>FD-LS</td>
<td>3.9dB</td>
<td>5.343</td>
<td>1.300</td>
<td>0.080</td>
<td>7.183</td>
<td>294</td>
</tr>
</tbody>
</table>

smaller influence domains will have higher precisions due to the increased node density, the equivalent effect as grid refinement. A key difference is that our partitions (through the influence domains) overlap with each other such that they are loosely inter-related and individual error at each location would be smoothed out by the neighbors within its influence domain when applying the MLS/GFD approximation. The smoothing scale is controlled by the influence domain size which is inversely proportional to the node density. Therefore, the smoothing would be automatically adjusted whenever the node density changes. By making the sampling rate adaptive to input data/ front geometry/prior knowledge, we can have greater smoothing for the contour point at low information area and little smoothing at information-rich locations. For example, for figures in Table 1, we need to apply large filters to de-noise the image but boundary leakage occurs when clear edges are missing. It does not happen to our point-based strategy because node distribution also governs the de-noising capability of MLS/GFD approximation. Although MLS is computational expensive, working on fewer nodes can make the speed of point based strategy comparable to traditional regular grid finite difference level set (FD-LS). Moreover, the running speed is also adaptive simply because fewer node values computation around low information areas, meaning a fast marching and level set refinement performed concurrently.

Experiment on brain image is shown in Fig. 5. The image gradient is used as the feature for node distribution. We start from an initial guess inside the brain structure. It evolves outward and stop at high gradient areas. The point-sampled implicit surface is visualized by increasing the node density around the final $C(t)$ to increase the density of $z$. In Fig. 6, since we used distance to the nearest data point as the $F$ function and the original range scanned data set is huge, in order to speedup the point searching in $F$, we quantized all data points to integral positions in a 100x100x100 grid.

Acknowledgement. This work is supported by NBRPC 2003CB716104, NSFC 60403040, and HKUST6252/04E.

References

Figure 5. Image segmentation results from medical data. The sampling rate has been increased when the evolving surface is closer to the target features. At the final stage, we use excessive amount of samples to create a better visualization of the implicit surface without meshing.


Figure 6. Shape recovering of the Stanford Bunny. The F function is the distance toward nearest data point.