2.5D Active Contour for Surface Reconstruction

Ye Duan and Hong Qin

Department of Computer Science
State University of New York at Stony Brook
Stony Brook, New York, USA.
Email: yduan|qin@cs.sunysb.edu

Abstract

In this paper, we present a new deformable model — 2.5D Active Contour— that is capable of directly extracting shape geometry from 3D unorganized point cloud datasets. The reconstructed surfaces are either open or closed. Furthermore, the new model can reconstruct and discover non-manifold geometry hidden in the point-cloud dataset. Our algorithm starts from a simple seed (e.g. a triangle) that can be automatically initialized, and always enlarges the model boundary outwards along its tangent direction suggested by the underlying dataset. This mechanism ensures that our novel models "flow" directly over the data boundary through the expansion of its bounding contour. To maintain the regularity of the model and the stability of the numerical integration process, commonly used mesh optimization techniques are employed throughout the deformation process. In addition, the new model can recover fine details of the underlying shape through local/adaptive refinements. We demonstrate both the accuracy and the robustness of our algorithm through a number of experiments on both real and synthetic datasets.

1 Introduction

Advances in 3D laser-range scanning technologies have given rise to a massive amount of datasets, and the obtained range dataset often contains sampling artifacts such as noises and gaps. How to efficiently and accurately reconstruct surfaces from these datasets remains challenging to researchers in computer vision, geometric modeling and visualization. In general, there are two kinds of approaches: static, geometry-based approaches, and dynamic, deformable model-based approaches. Among the static approaches, there are either explicit methods or implicit methods. Explicit methods reconstruct a triangulated surface using Delaunay triangulations and Voronoi diagrams. These include the Alpha-Shape algorithm [8], the Crust algorithm [1], and the Ball-Pivoting algorithm [3]. Since the noise in the dataset becomes embedded in the reconstruction, these explicit methods are usually difficult to handle non-uniformity and noises. Implicit methods use the input points to either define a scalar function by the use of a combination of smooth basis function such as Radial Basis Functions (RBF) [5, 7], or define a signed distance function on rectangular grids [11]. To create the output mesh, the Marching Cubes algorithm [17] is then used to polygonalize the iso-contour. Implicit methods are less sensitive to noises because they produce approximating rather than interpolating meshes. However, implicit methods in general require more computation. Fundamentally different from static techniques are deformable model based techniques, i.e., methods that perform reconstruction by deforming an initial seed model to fit the dataset. Deformable models are more robust to sampling artifacts such as noise and gaps because of their inherent continuity and smoothness and are more suitable for reconstructing time-evolving datasets. However, conventional parameterized deformable models [13, 25, 23, 24] are not amenable to arbitrarily complicated topology and geometry. Recently, researchers have proposed implicit level-set based models for shape reconstruction such as [18, 6, 22, 27]. Level-set models are topologically very flexible, however, they can only handle shapes that are closed manifolds. In contrast, our new model can represent both closed and open surfaces that are either manifold or non-manifold.

In this paper, we propose a new deformable model—2.5D Active Contour—that is capable of recovering shapes from 3D unorganized point cloud

datasets. The recovered shape is either open or closed, therefore, it is topologically flexible. More importantly, our model can be applied to extract non-manifold shape directly from point clouds. These unique advantages are achieved by using the following techniques: Starting from a simple seed (e.g. a triangle) that is automatically initialized by the system, our new model can enlarge itself and flow directly over the object boundary through the expansion of its boundary contour outwards along the surface tangent (i.e., the surface grows tangentially and always expand itself directly over the dataset). The deformation behavior of the boundary contour is governed by partial differential equation (PDE). To ensure the regularity of the model and the stability of the numerical integration process, commonly used mesh optimization techniques are employed throughout the deformation process. In addition, the new model can recover fine details of the underlying shape through local/adaptive refinements.

2 Algorithm

The entire pipeline of the algorithm consists of the following major steps:

- Model initialization.
- Model growing.
- Model relaxation.
- Collision detection.
- Mesh stitching.

2.1 Model Initialization

Before the deformation begins, a seed model needs to be initialized on the boundary of the object. This can be done either interactively by the user or automatically by the system. Any polygon (e.g. a triangle) can be used as a seed model. A triangle can be automatically initialized by the system by finding three data points that are closest to each other: the first data point is randomly chosen, and its closest data point serves as the second data point. The third data point is chosen as the closest point to the middle position of the two previous data points and is not collinear with the two previous data points. It is possible that the underlying object may consist of multiple disjoint components. Hence, even after the model deformation stops, the model initialization may still be needed. Therefore, the system must check whether there are still some un-visited data points yet to be processed. If that is the case, then a new seed model will be initiated to handle those data points via the following model deformation procedure. In principle, the model initialization is capable of repeating several times until all the data points have been visited. A data point will be marked as visited if the distance from the data point to the reconstructed model is smaller than a threshold. Otherwise, it is marked as unvisited.

2.2 Model Growing

After the model is initialized, the model will grow and deform. At each deformation step, only the boundary contour of the model is active and is allowed to move. The deformation behavior of the model boundary is governed by an evolutionary system of differential equations with the following form:

$$\frac{\partial C(p,t)}{\partial t} = F\vec{n}(p,t), C(p,0) = C_0(p), \quad (1)$$

where t is the time parameter, $C_0(p)$ is the initial surface, $\vec{n}(p,t)$ is the tangential normal vector of the current position p on its tangent plane. F is an application-dependent speed function. In this paper, for the purpose of surface reconstruction, we used the following curvature-based speed function:

$$F = (v + k)g(p), \quad (2)$$

here k is the tangential curvature at the current position p on its tangent plane and is acting as a smoothing constraint. v is the constant velocity, which will enable the convex initial shape to capture non-convex, arbitrarily complicated shapes. This definition of speed function is similar to the one used by Malladi *et al.* [18] for the purpose of medical image segmentation. Figure 1 illustrates the definition of the tangential curvature k and speed function $F = k \cdot \vec{n}$ in Eq. 2. $g(p)$ is the scalar value stopping function. By default, $g(p)$ is set as 1. The value of $g(p)$ will become 0 when the model is moving over the surface border of an open object. In this case the deformation will stop because the speed F is zero. We will explain how to detect the surface border in more detail in Section 3.1.

The surface evolution process is approximated using an explicit iterative equation:

$$C(p, t + \Delta t) = C(p,t) + F(p,t)\vec{n}(p,t)\Delta t. \quad (3)$$

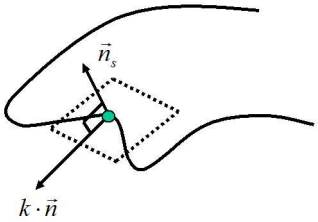

Figure 1: Front propagation on the object boundary (shown in solid curve). $\vec{n}_s$ is the surface normal vector at the current vertex position p, k is the tangential curvature, $\vec{n}$ is the tangential normal vector, and $F = k \cdot \vec{n}$ is the speed function.

We will explain how to approximate the tangential normal and tangential curvature in Section 5. Because the model is only moving along the direction of the surface tangential normal, in order to ensure the model always stays on the object boundary, at each deformation we need to project all the active boundary vertices back onto its closest point over the object boundary. Figure 2 shows a 2D illustration. The vertex position at the current time step t is $C(p,t)$; the calculated position at the next time step $t + \Delta t$ from Eq. 3 is $C(p, t + \Delta t)$ (Fig. 2(a)). We then project the vertex from position $C(p, t + \Delta t)$ back onto the object boundary at $C_\perp(p, t + \Delta t)$ (Fig. 2(b)). So the final vertex position at the next time step $t + \Delta t$ is $C_\perp(p, t + \Delta t)$ (Fig. 2(c)). The propagation will stop when all the data points are "flowed over". (Fig. 2(d)). Since the boundary of the underlying object is implicitly represented by point clouds, the project step is done by projecting the current vertex onto its local tangent plane. We will explain how to approximate the local tangent plane in Section 5.

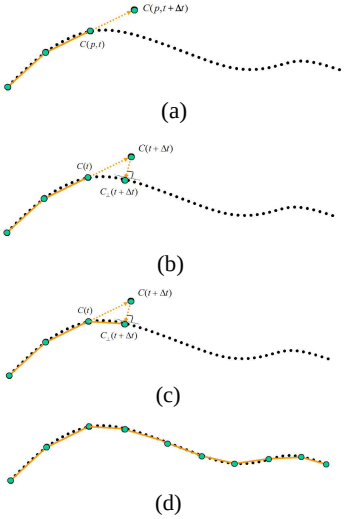

Figure 2: 2D illustration of front propagation. Data points are shown in black dots, green dots are the nodes of the propagating fronts, while the edges connecting the nodes are shown in golden color.

2.3 Model Relaxation

To ensure the numerical simulation of the deformation process to proceed smoothly, a relaxation step is necessary after each deformation step to maintain the regularity of the mesh such that the mesh has a good node distribution, a proper node density, and a good aspect ratio of the triangles. This is achieved through the use of local subdivision scheme and three mesh optimization operations [12]: edge split, edge collapse, and edge swap. A face is subdivided if its area is larger than a certain user-defined threshold and is still active (i.e. at least one of its three vertices is an active boundary vertex). Local subdivision is achieved through triangle quadrisection, i.e. each triangle is divided into four smaller triangles at the middle position of its three edges. Edge split and edge collapse are used to keep an appropriate node density. An edge split is triggered if the edge length is bigger than the maximum edge length threshold. Similarly, an edge will be collapsed if its length is smaller than the minimum edge length threshold. Edge swapping is used to ensure a good aspect ratio of the triangles. As suggested by Kobbelt *et al.* [15], this can be achieved by forcing the average valence to be as close to 6 as possible. An edge is swapped if and only if the quantity $\sum_{p \in \Delta} (valence(p) - 6)^2$ is minimized after the swapping. Note that mesh optimization operations are only applied to regions that are still active.

2.4 Collision Detection

There are three kinds of collision that may occur during the model deformation process: vertex-

vertex collision, vertex-edge collision, and vertex-face collision. In this paper, a simple distance-based collision detection scheme is used. A vertex-vertex collision is detected if the distance of any two non-neighboring active boundary vertices is smaller than a threshold. These two vertices will be marked as non-active and will not be allowed to move further. Vertex-edge collision happens between an active boundary vertex and a non-adjacent active boundary edge. If the distance between an active boundary vertex and the middle position of a non-neighboring boundary edge is smaller than a threshold, a vertex-edge collision will be detected and the boundary vertex will be marked as non-active and will not be allowed to move further. These non-active vertices will be merged with their corresponding vertices in the next mesh-stitching step which we will address in the next section. Vertex-face collision happens between an active boundary vertex and a non-adjacent interior face. This can only occur when the underlying object contains some non-manifold geometric structures. Vertex-face collision is detected if the distance between an active boundary vertex and a non-adjacent interior face is smaller than a threshold. The distance from the vertex to the face is approximated by the smallest distance between the vertex and the sampling points of the face including the face center, the three corner points of the face, and the three middle points of the three boundary edges of the face. If a vertex-face collision is detected, then the vertex will be marked as a non-manifold vertex and will not be allowed to move further. After the deformation stops, a post processing step will be invoked to attach all the non-manifold vertices onto the model (Section 3.2).

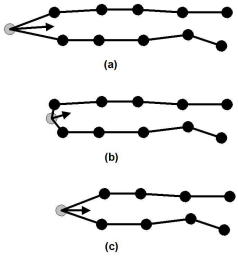

Figure 3: On-the-fly stitching. Dark-colored circles are non-active boundary vertices. Grey-colored circle is the active boundary vertex. (a)-(b) The active boundary vertex is moving towards its two adjacent chains of non-active boundary vertices. (b)-(c) the active vertex is merged with its two adjacent non-active boundary vertices by the edge collapse operation, and is moving towards its two new adjacent non-active boundary vertices.

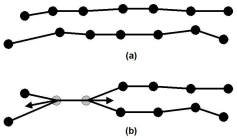

Figure 4: Post stitching. (a) Two disconnected chains of non-active boundary vertices (shown in dark circles). (b) The closest pair of boundary edges are merged together. The two endpoints of the newly merged edge are activated and are moving towards their two adjacent non-active boundary vertices.

2.5 Mesh Stitching

Mesh stitching is used to merge all the vertices that have been marked as non-active in the previous collision detection step. There are two possible cases: "on-the-fly stitching" and "post stitching". On-the-fly stitching occurs when there is an active boundary vertex that is adjacent to two chains of non-active boundary vertices. The movement of the active boundary vertex and the aforementioned mesh optimization operations will iteratively merge the two chains of non-active boundary vertices together. See Figure 3 for an illustration.

If the two chains of non-active boundary vertices that are merging with each other are disconnected, then the "post-stitching" method will be used. First, a pair of closest boundary edges from these two chains is found and is merged together. Then, the two endpoints of the new edge are activated. Finally, the two active endpoints will iteratively merge the two chains of non-active boundary vertices together by the aforementioned "on-the-fly stitching" method. Note that "post-stitching" is used only after the model deformation stops. See Figure 4 for an illustration.

3 Open Manifolds and Non-manifolds

We have described the main steps of the algorithm in the previous section. Besides closed manifolds, our model can also recover shapes of open manifolds, non-manifolds, and multiple disjoint components. We have explained how to recover multiple disjoint components in section 2.1. In this section, we will explain how to handle open manifolds and non-manifolds, respectively.

3.1 Open Manifolds

Open manifolds are manifolds that have openings (i.e. open boundaries). To detect the possible openings within the underlying object, after each deformation, for each active boundary vertex, we will calculate the distance between the vertex and the center position of its k-nearest data points. These are the k-nearest points used for local tangent plane approximation, which we will discuss in section 5. Since the model is flowing over the data points, the distance is usually very small if the underlying object is a closed shape. Hence, a much bigger distance would indicate that the vertex is moving over the border of an open object. In this case, the vertex will be marked as non-active and will be identified as a vertex that is on the border of an open object. After the deformation stops, all the vertices that have been identified as being on the object border will be moved back to the border by updating the position of the vertex using the position of its closest data points.

3.2 Non-Manifolds

As we mentioned in section 2.4, a non-manifold vertex is identified if there is an occurrence of vertex-face collision between the current active boundary vertex and a non-adjacent interior face. After the deformation stops, the system will check whether there are any non-manifold vertices being identified. If this is the case, then the non-manifold vertices will be put into a linked list. And each non-manifold vertex will be projected onto the part of model it is colliding with. This is done by the ray-triangle intersection method frequently used in the computational geometry community. In the interest of the space, we will omit the details here, please refer to the book written by O'Rourke [20] for more details.

After all the non-manifold vertices have been projected onto the model, they needed to be merged with the region of the model they are colliding with. The basic idea is: for each non-manifold vertex, find a corresponding vertex on the model and merge them together. Figure 5 shows an illustration. Here, the non-manifold vertices are shown in dark circles. The non-manifold edges are shown in dark lines (both solid and dotted). The interior edges of the model are shown as gray solid lines. Starting from the first vertex (shown in gray-colored circle in Fig. 5(a)) in the linked list of non-manifold vertices, find its closest vertex on the model shown in small white circle), snap it to the position of the first non-manifold vertex and merge these two vertices. Now (Fig. 5(b)) the second vertex (shown in gray-colored circle) in the linked list becomes the current vertex. Check whether it is located in the 2-neighborhood of the previous non-manifold vertex. If yes, then snap the closest one-neighborhood vertex (shown in small white circle) of the previous non-manifold vertex to the current vertex and merge them. Otherwise (Fig. 5(c)), insert a new non-manifold vertex in the middle between the current non-manifold vertex and the previous non-manifold vertex (Fig. 5(d)). Repeat the above steps (Fig. 5(e)) until all the non-manifold vertices on the linked list are merged with the interior region of the model (Fig. 5(f)).

4 Levels Of Detail Control

Once an initial shape of the object is recovered, the model can be further refined several times to improve the fitting accuracy. In this paper, we employed two refinement approaches: global refinement and local/adaptive refinement. The decision of which method to use can be made either interactively by the user (whether he/she prefer a more uniformed mesh or an adaptively sampled mesh), or automatically by the system. The system can make a technically sound decision by calculating the variance of the fitting accuracy of the current model. If the variance of the fitting accuracy is very low, then the underlying object must be relatively smooth and global refinement will be a good choice. Otherwise, adaptive refinement will be used to recover the fine details embedded in the underlying object.

Global refinement is conducted by subdivision scheme. Since the recovered shape may contain

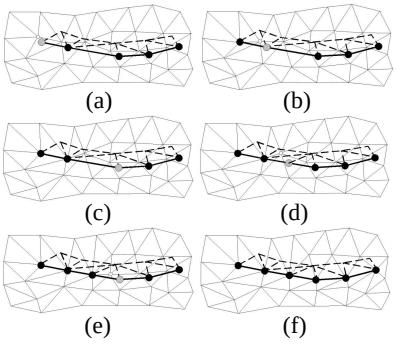

Figure 5: Merge the chain of non-manifold vertices with the corresponding vertices on the interior region of the model by vertex snapping. The non-manifold vertices are shown in shaded circles, the non-manifold edges are shown in dark colors (both solid and dotted lines), and the interior edges of the model are shown in gray-colored lines. In each subfigure, the gray-colored circle is the current non-manifold vertex that is merging with its corresponding vertex (small white circle) on the model. (a) A chain of non-manifold vertices have been projected onto the region of the model they are colliding with. (b)-(e) Merge the chain of non-manifold vertices with the model iteratively by vertex snapping. (f) The chain of non-manifold vertices has been merged with the interior region of the model.

open boundaries and even non-manifold geometry, in this paper, we use the piecewise smooth Loop's scheme proposed by Hoppe *et al.* [10]. In order to refine non-manifold geometry, we treat all the non-manifold edges as sharp edges and all the non-manifold vertices as corner vertices.

Adaptive refinement is guided by the fitting accuracy. Various kinds of metrics can be used to evaluate the fitting accuracy over each triangle, for example, the maximum distance to the object boundary, the average distance to the boundary, the curvature of the triangle, etc. We use the variance of the distance from the triangle to the boundary of the object as the metric of the fitting accuracy as suggested by Wood *et al.* [26]. The distance to the boundary of the object is estimated using the distance between the current sampling position and its local tangent plane approximated by its k-nearest neighbors, which we will discuss in more details in section 5. The variance of a discrete set of distances is computed in the standard way: $V_T[d] = E[d^2] - E[d]^2$, where E denotes the mean of its argument. To calculate the variance of the distance samples for a given triangle, we temporarily quadrisect the triangle T into four smaller triangles and for each smaller triangle, calculate the distance at its barycentric center. If the variance of the distance samples for a given triangle is bigger than a user defined threshold, then this triangle will be refined since the object boundary underneath this particular triangle must have high curvature. Several levels of refinements (both global refinement and local/adaptive refinement) can be applied until a user specified fitting accuracy has been met. All the new vertices generated at each level of refinement will be directly projected onto its local tangent plane approximated by its k-nearest neighbors.

5 Numerical Implementation

5.1 Local Tangent Plane Approximation

The local tangent plane of any given position p is estimated by using the method of principle component analysis (PCA) of its k-nearest data points as suggested by Hoppe *et al.* [11]: For any position p, its local tangent plane is represented by a center point c and a unit normal vector n. The center point c is the centroid of the k-nearest data points of position p, which is denoted as $Nbhd(p)$. The normal vector $\vec{n}$ is computed by doing eigen analysis of the covariance matrix C of $Nbhd(p)$, which is a symmetric 3 x 3 positive semi-definite matrix:

$$C = \sum_{p_i \in Nbhd(p)} (p_i - c) \otimes (p_i - c). \quad (4)$$

Here, $\otimes$ denotes the outer product vector operator, and the normal vector n is the eigenvector associated with the smallest eigenvalue of the covariance matrix C. In our experiments, k is set to be five. In order to efficiently find the closest data points of a given position p, we preprocess the point clouds by putting them into a uniform regular grid and connecting all the points inside one grid cell by a linked list. If the point clouds are relatively uniformly sampled, then the time complexity of the nearest point search is almost linear.

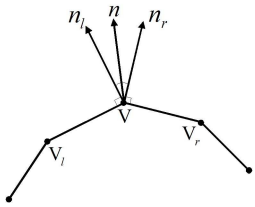

Figure 6: The tangential normal vector of the boundary vertex.

5.2 Boundary Vertex Tangential Normal

We define the tangential normal vector $\vec{n}$ of a boundary vertex V as the normalized sum of the two unit vectors $\vec{n}_l$ and $\vec{n}_r$ that are perpendicular to its two adjacent boundary edges VV_l and VV_r:

$$\vec{n} = \frac{\vec{n_l} + \vec{n_r}}{\|\vec{n_l} + \vec{n_r}\|} \tag{5}$$

Note that all the three vectors $\vec{n}$, $\vec{n}_l$ and $\vec{n}_r$ are in the same plane defined by the three vertices V, V_l and V_r and are pointing outwards from the boundary contour. In fact, we can easily verify that the tangential normal vector $\vec{n}$ is the bisector vector of the outer angle V_lVV_r. This is consistent with the commonly used definition of vertex normal in planar curve evolutions [2]. See Figure 6 for an illustration.

5.3 Boundary Vertex Tangential Curvature

Since the speed of the boundary vertex depends on the tangential curvature, we need to be able to accurately estimate the tangential curvature at the boundary vertex. The boundary vertex curvature $\vec{K}$ is defined as:

$$\vec{K} = k\vec{n}. \tag{6}$$

Here $\vec{n}$ is the tangential normal vector of the vertex V defined in the previous section and k is the scalar value of the curvature. We use the discrete curvature estimator proposed by [22] to calculate the absolute value of k:

$$\|k\| = \frac{4 * S_{\triangle VV_lV_r}}{\|VV_l\| * \|V_lV_r\| * \|V_rV\|}, \tag{7}$$

where $S_{\triangle VV_lV_r}$ is the face area of the triangle $\triangle VV_lV_r$, $\|VV_l\|$, $\|V_lV_r\|$, and $\|V_rV\|$ are the edge lengths of the three corresponding edges VV_l, V_lV_r, and V_rV, respectively. The sign of the curvature k, i.e. whether it is pointing outwards or inwards, is decided by checking the local convexity at the current boundary vertex V. If it is locally convex at the vertex V, then the curvature k is negative (i.e. pointing opposite to the direction of the tangential normal vector $\vec{n}$). Otherwise, the curvature k is positive (i.e. pointing to the same direction as $\vec{n}$). See Figure 7 for an illustration.

6 Experimental Results On Surface Reconstruction

In this section, we will show some experimental results obtained by our new model. All the experiments are conducted on a Pentium 4M 1.6GHZ Notebook PC with 512MB memory. Note that in all the figures, all active boundary contours of the models are shown in red, non-active interior regions of the models are shown in blue. The input of Figure 8 is the point cloud dataset of Stanford bunny (shown in golden color in Fig. 8(a)). It has two holes and several small gaps in the bottom. Fig. 8(b) to Fig. 8(d) are the three snapshots of the growing stage. Fig. 8(e) shows the mesh stitching process where the chain of non-active boundary vertices are stitched together by the "post stitching" process. Fig. 8(f) is the initial recovered shape. Fig. 8(g) is the wireframe view of the same shape in Fig. 8(f). Fig. 8(h) shows the bottom of the bunny. The two holes are correctly recovered. The small gaps in the bottom are automatically filled in. Fig. 8(i) is the refined shape of the bunny. Adaptive refinement is used to recover the fine details. Fig. 8(j) is the rendered view of the same shape in Fig. 8(i). The input of Figure 9 is the mannequin dataset, which is an open manifold. Fig. 9(a) shows the initial seed triangle (shown in red) staying on the dataset. Fig. 9(b) is a snapshot of the model growing stage. Fig. 9(c) is the initial recovered shape shown in wireframe. Fig. 9(d) shows the adaptively refined shape in wireframe. Again, adaptive refinement is used to recover the fine details. Fig. 9(e) is the rendered view of the same shape in Fig. 9(d). The input of Figure 10 is a synthetic point cloud dataset that consists of eight disjoint tori. Fig. 10(a) is the point cloud dataset shown in golden color. Fig. 10(b) and Fig. 10(c) are two snapshots of the model growing stages. Fig. 10(d) is the initial recovered shape shown in wireframe. Fig. 10(e) is the refined shape

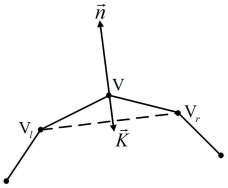

Figure 7: The boundary vertex tangential curvature.

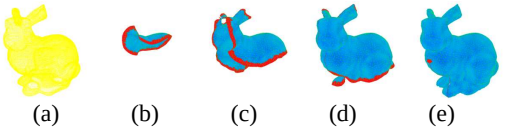

Figure 8: Surface reconstruction from the point cloud dataset of the Stanford bunny.

after one level of global subdivision, also shown in wireframe. Figure 11 shows a simple example of a non-manifold shape. The input is a synthesized Saturn-like point clouds. Fig. 11(a) shows the model flowing through the dataset. After the first seed model stops, a new seed model is initialized around the region of the dataset that have not been visited and start growing again (Fig. 11(b)). Fig. 11(c) and Fig. 11(d) are the top view of the two snapshots of the growing stages before and after the two parts of the model are merged together, respectively. Fig. 11(e) is the same shape as Fig. 11(d) shown in the original view direction.

7 Conclusion

We have described a new deformable model—2.5D Active Contour—that is capable of discover-

Figure 9: Surface reconstruction from the point cloud dataset of the Mannequin.

Figure 10: Surface reconstruction from a synthetic point cloud dataset of eight-tori.

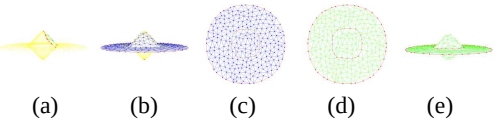

Figure 11: Surface reconstruction of a non-manifold geometry.

ing shapes of both closed and open surfaces that are either manifold or non-manifold. Our model directly works on unorganized data points (as an input) and correctly reconstructs shape geometry without any data conversion. Starting from a simple seed (e.g. a triangle) that is automatically initialized by the system, our new model can enlarge itself and flow directly over the object boundary through the expansion of its boundary contour outwards along the surface tangent. In addition, the new model can recover very fine details of the underlying shape through adaptive refinement.

References

[1] N. Amenta, M. Bern, and M. Kamvysselis. A new Voronoi-based surface reconstruction algorithm. In *Proceedings of ACM SIGGRAPH 98*, pages 415–421, 1998.

[2] A. G. Belyaev, E. V. Anoshkina, S.Yoshizawa, and M.Yano. Polygonal curve evolutions for planar shape modeling and analysis. *International Journal of Shape Modeling*, 5(2):195–217, 1999.

[3] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, 1999.

[4] J. D. Boissonnat. Geometric structures for three-dimensional shape rep-resentation. *ACM Transactions on Graphics*, 3(4):266–286, 1984.

[5] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *Computer Graphics (Proceedings of SIGGRAPH 2001)*, pages 67–76, August 2001.

[6] V. Caselles, R. Kimmel, G. Sapiro, and C. Sbert. Minimal surfaces based object segmentation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19:394–398, 1997.

[7] H. Q. Dinh, G. Slabaugh, and G. Turk. Reconstructing surfaces using anisotropic basis functions. In *International Conference on Computer Vision (ICCV) 2001*, pages 606–613, Vancouver, Canada, July 2001.

[8] H. Edelsbrunner and E. P. Mücke. Three-Dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, Jan. 1994. ISSN 0730-0301.

[9] A. Hilton, A. Stoddart, J. Illingworth, and T. windeatt. Marching triangles: Range image fusion for complex object modeling. In *proceedings of IEEE international conference on image processing*, pages 381–384, 1996.

[10] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise smooth surface reconstruction. In *Computer Graphics (SIGGRAPH 94 Proceedings)*, pages 295–302, July 1994.

[11] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In E. E. Catmull, editor, *Computer Graphics (SIGGRAPH 92 Proceedings)*, volume 26, pages 71–78, July 1992.

[12] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In J. T. Kajiya, editor, *Computer Graphics (SIGGRAPH 93 Proceedings)*, volume 27, pages 19–26, Aug. 1993.

[13] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, pages 321–331, 1988.

[14] R. Kimmel and J. A. Sethian. Computing geodesic paths on manifolds. In *Proceedings of National Academy of Sciences*, pages 8431–8435, July 1998.

[15] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. Interactive multi-resolution modeling on arbitrary meshes. In *Computer Graphics (SIGGRAPH'98 Proceedings)*, pages 105–114, 1998.

[16] H. Lee, L. Kim, M. Meyer, and M. Desbrun. Meshes on fire. In *Eurographics workshop on computer animation and simulation*, 2001.

[17] W. E. Lorensen and H. E. Cline. Marching cubes: a high resolution 3D surface construction algorithm. In *Computer Graphics (SIGGRAPH 87 Proceedings)*, volume 21, pages 163–169, July 1987.

[18] R. Malladi, J. Sethian, and B. Vemuri. Shape modeling with front propogation: A level set approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(2):158 – 175, 1995.

[19] R. Mencl. A graph-based approach to surface reconstruction. In *EUROGRAPHICS*, pages 445–456, 1995.

[20] J. O'Rourke. *Computation geometry in C*. Cambridge University Press, 1998.

[21] K. Polthier and M. Schmies. Straightest geodesics on polyhedral surfaces. In H. Hege and K. Polthier, editors, *Mathematical visualization*. Springer Verlag, 1998.

[22] J. A. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge University Press, second edition, 1999.

[23] D. Terzopoulos and K. Fleischer. Deformable models. *The Visual Computer*, 4(6):306–331, Dec. 1988.

[24] D. Terzopoulos and D. Metaxas. Dynamic 3D models with local and global deformations: deformable superquadrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(7):703–714, 1991.

[25] D. Terzopoulos, A. Witkin, and M. Kass. Symmetry-seeking models and 3d object reconstruction. *International Journal of Computer Vision*, 1(3):211–221, 1987.

[26] Z. Wood, M. Desbrun, P. Schroder, and D. Breen. Semi-regular mesh extraction from volumes. In *Proceedings of IEEE Visualization*, pages 275–282, 2000.

[27] H. Zhao, S. Osher, and R. Fedkiw. Fast surface reconstruction and deformation using the level set method. In *Proceedings of the IEEE Workshop on Variational and Level Set Methods in Computer Vision (VLSM 2001)*, 2001.