Reconstructing the Surface Mesh Representation for Single Neuron

Ivar Ekeland[1(✉)] and Roger Temam[2]

[1] Princeton University, Princeton, NJ 08544, USA
I.Ekeland@princeton.edu
[2] Université de Paris-Sud, Laboratoire d'Analyse Numérique, Bâtiment 425, Orsay, France

Abstract. In this paper, we present a pipeline to reconstruct the membrane surface of single neuron. Based on the abstract skeleton described by points with diameter information, a surface mesh representation is generated to approximate the neuronal membrane. The neuron has multi-branches (called neurites) connected together. Using a pushing-forward way, the algorithm computes a series of non-parallel contour lines along the extension direction of each neurite. These contours are self-adaptive to the neurite's cross-sectional shape size and then be connected sequentially to form the surface. The soma is a unique part for the nerve cell but is usually detached to the neurites when reconstructed previously. The algorithm creates a suitable point set and obtains its surface mesh by triangulation, which can be combined with the surface of different neurite branches exactly to get the whole mesh model. Compared with the measurements, experiments show that our method is conducive to reconstruct high quality and density surface for single neuron.

Keywords: Triangle mesh · Surface reconstruction · Neuron

1 Introduction

Individual neuron is the starting point during the exploration of the brain in modern neuroscience [1]. It is recognized to be the basic functional unit of nervous system. The immediate study for single neuron is its morphological structure, which mainly consists of a cell body (also called soma) and the neurites. However, the neuron is hard to be observed directly by human eyes because it is microscale, having tiny geometry, and is semitransparent. Visualizing the neuron is therefore not trivial.

Modern electron microscope can clearly observe biological specimens and save them as images, while state of the art laser microscope even can directly image living brain tissue with super-resolution. Forming the images achieves persistent preservation of neuronal structure. An advanced computer technology, known as neuron tracing [2,3], allows researchers to extract the neuron from microscopy images. The tracing actually converts the image data sets into a much more parsimonious representation of neuronal topology and geometry, described as a

N. Magnenat-Thalmann et al. (Eds.): CGI 2022, LNCS 13443, pp. 170–182, 2022.
https://doi.org/10.1007/978-3-031-23473-6_14

set of sample points. These points with their intrinsic connectivity express the morphology of single neuron as 3D skeleton, i.e. the medial axis lines generated by inward contraction of the neurites.

The skeleton representation keeps a well-behaved abstraction of neuronal structure, but the lack of neurite's thickness brings some limitations. The most drawback is that the skeleton does not provide a continuous surface representation. Nevertheless, the neuron is the cell with smooth and continuous membrane surface. The membrane separates both inside and outside of the neuron, which gives a particular 3D appearance of the nerve cell. Reconstructing the corresponding surface allows perceiving the neurite thickness (and therefore volume) immediately. Here we describe a simple and general method to provide a surface reconstruction pipeline of neuronal membrane. This approach results in a surface mesh representation, which is made up of triangles with high quality and density. For one thing, the reconstruction in this paper can further improve the visual presentation power of the neuron and be a supplement to the ball-and-stick model in some visualization software [4]. For another, the 3D representation may benefit neuroscience researches, such as helping the computational neuroscientists to build brain function model, simulating electrophysiological experiments of voltage dynamics [5] and so on.

The rest of the paper is structured as follows. Section 2 reviews some related works. In Sect. 3, we describe our method in detail and Sect. 4 shows the experimental results. The paper ends with a conclusion.

2 Related Work

Creating an accurate closed surface based on a skeleton is a significant research topic in computer graphics. It has been applied in various modeling domains, such as trees [6–8] and blood vessels [9–11]. The relevant techniques usually are divided into implicit and explicit, corresponding to implicit surface and explicit surface [12].

An implicit surface is defined as an isosurface that all of the points have the same given scalar field, satisfying a specific implicit field function [13]. The implicit surface-based modeling is able to reconstruct surfaces for any objects theoretically. However, its modeling potential is limited by the exact definition of an implicit function, which is closely related the shape of the object. Yet neurons are diverse and it is impossible to find one or several functions to describe all neurons. Additional, an implicit surface needs to be polygonized through isosurface extraction algorithms (like marching cube [14]) for visualizing and rendering. In contrast, the explicit surface is rendered directly in computer's graphic system. The basic explicit element is isolated point. For example, point clouds obtained by 3D scanning can be observed immediately as long as they are input. They are regarded the original data in many cases as well for surface reconstruction [15]. The surface for explicit form is represented by polygonal mesh, which is widely applied to approximate geometric objects in computer graphics.

In recent years, there are a few research studies creating surface meshes from neuronal skeleton. Lasserre et al. [16] used mesh extrusion starting from

a fixed soma to obtain a coarse mesh with quadrilateral faces and refined it by subdivision for a detailed mesh. Carcia-Cantero et al. [17] also extruded the meshes of neurites but applied an improved Finite Element Method [18] to the fixed soma, making it more realistic. Abdellah et al. [19] developed a tool named Skin Modifiers for high fidelity neuronal meshes but they even need to complete reconstruction in Blender software.

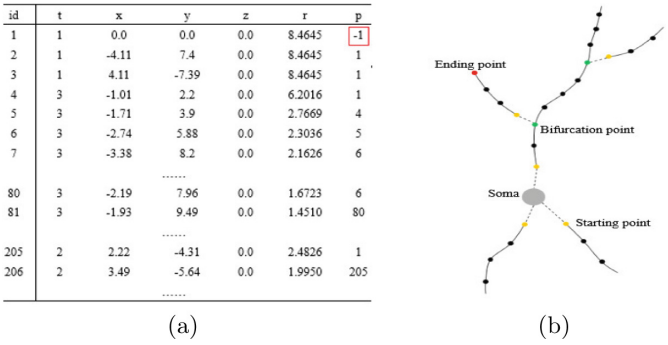
However, in this current work, the method inputs the skeleton and directly outputs a refined surface mesh with high quality. It is simple and intuitive, without subdivision operation or other software.

3 Method

The main goal of the method presented here is reconstructing a 3D polygonal model that represents the neuronal membrane surface approximately. The first step takes as input a single neuron and divides it into individual neurite branches. The second step computes adaptive contours and connects them sequentially to form the surface mesh for each branch, together with saving the connectivity information that makes for the surface of different branches to be spliced subsequently. As for the soma, the algorithm constructs a suitable point set used to triangulate and the triangulation result can be combined exactly to the branches' surface. The following sections describe above steps in detail.

3.1 Branch Identification

The source of neuronal morphologies is from a public and online database NeuroMorpho.Org [20]. Each digital neuron in this repository is stored in text file with SWC format (Fig. 1(a)), which contains a hierarchical morphology skeleton described as a set of connected sample points (Fig. 1(b)). Each point provides several components, including its sample number (id), type (t), coordinates (x, y, z), local thickness (r) and connectivity information (p) which links this point to a parent one.



id	t	x	y	z	r	p
1	1	0.0	0.0	0.0	8.4645	-1
2	1	-4.11	7.4	0.0	8.4645	1
3	1	4.11	-7.39	0.0	8.4645	1
4	3	-1.01	2.2	0.0	6.2016	1
5	3	-1.71	3.9	0.0	2.7669	4
6	3	-2.74	5.88	0.0	2.3036	5
7	3	-3.38	8.2	0.0	2.1626	6
			.....			
80	3	-2.19	7.96	0.0	1.6723	6
81	3	-1.93	9.49	0.0	1.4510	80
			.....			
205	2	2.22	-4.31	0.0	2.4826	1
206	2	3.49	-5.64	0.0	1.9950	205
			.....			

(a) (b)

Fig. 1. a Example of a SWC file. **b** The skeleton abstraction of single neuron

In Fig. 1(b), the abstract skeleton shows that the neuronal morphology is multi-branches structure. Here a branch is defined as successive samples from a starting point to an ending point. However, the bifurcation point will bring ambiguous during dividing different branches, because there are two child points connected to it in most cases. Thus, the process of branch identification need to determine which child point is the best successor of the bifurcation point.

The selection of the best successor is in light of the potential that brings convenience to the contour connection between the bifurcation point and its child. There are two constraints to be considered. First, the algorithm priorities the largest angle condition. For instance, point **B** in Fig. 2(a) is selected due to $\alpha_1 < \alpha_2$. If the difference value between those two angles is less than a given threshold t, the algorithm considers the selected child whose r-component is closer to the bifurcation point (see point **D** in Fig. 2(b)). Unselected child as a starting point radiates a new branch.

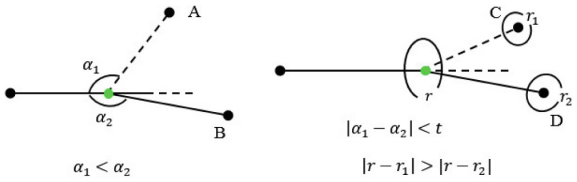


Fig. 2. Selection of the best successor. α and r represent the angle and the local thickness, respectively

Currently the neuronal morphology can be regarded as a collection of individual branches. For convenience of description later, the primary branch is used to signify a branch that includes the bifurcation point, while the secondary branch means a new branch starting from a child point. The concept of those two categories is relatively changing, especially during the process of mesh splicing. That is, a primary branch in one case may be a secondary branch in another case. In addition, the soma branch is used to signify a branch that radiates from the soma.

3.2 Surface Reconstruction of Neurites

This section describes that how to obtain the surface mesh for individual neurite branch and splice different branches, including four subsections in the following.

Resampling. Original sample points of each branch are lower density and cannot meet the requirements of surface reconstruction in this article. This leads to resampling operation for original data. The resampling utilizes the thoughts of interpolation curve fitting. It not only increases the data's density while preserving the old points but also maintains the consistency of neuronal shape before and later.

The Catmull-Rom [21] fitting method is adopted to construct a spline curve for every branch. This method is piecewise fitting, so it is necessary to calculate

the interpolation precision (i.e., the number of resampling points) for each two original adjacent points. Assuming that an individual branch is denoted as $\mathbf{B} = \{(P_i, r_i)|i = 1, 2, \ldots, n\}$, where $P_i = (x_i, y_i, z_i)$. The interpolation precision δ between P_i and P_{i+1} is calculated as follows:

$$\delta_{(P_i,P_{i+1})} = \frac{D(P_i, P_{i+1})}{r_{avg}} + k_1, i \in [1, n) \tag{1}$$

In formula 1, $D(P_i, P_{i+1})$ represents the Euclidean distance and r_{avg} represents the average value of r-component of n points on branch $\mathbf{B}$. A control parameter k_1 is used due to the Catmull-Rom method needs to consider endpoint condition during piecewise calculation. If P_i is the starting point of $\mathbf{B}$, the value of k_1 equals to number 2; otherwise, it equals to number 1.

Besides, the tangent vector is calculated by the first-order derivative of the fitting curve, determining the contour's orientation of each sampling point.

Contour Generation. The contour characterizes the cross-sectional shape of a neurite branch at some local position. At point P_i, it is defined as an inscribed polygon of the circle whose center is P_i and radius is r_i. The contour's vertices are the sampling points on the circle so that the contour approximates gradually to the circle as the number of vertices growing. This is consistent with the cognition that neurites are tubular branches with circular cross-section. The plane in which the contour lies is stated by $P_i \cdot \boldsymbol{o}_i$, where $\boldsymbol{o}_i$ represents the orientation vector of P_i's contour.

A pushing-forward way is used to calculate the contour for each point on the same branch and every contour is self-adaptive to its own radius and orientation. A contour at point P_i with m_i vertices can be written as $C_i = \{C_i^j | j = 1, 2, \ldots, m_i\}$, where C_i^1 is the first vertex on it. Then, the corresponding vertex C_{i+1}^1 on the contour at P_{i+1} can be obtained according to the following steps (see Fig. 3(a)).

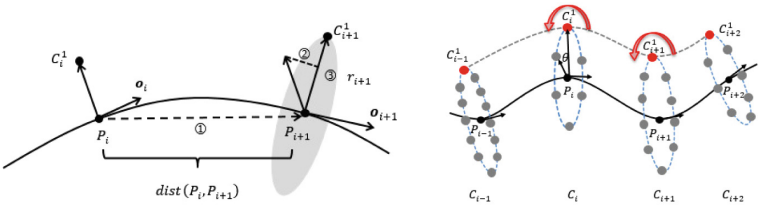


Fig. 3. a The computation of the first sample vertex on each contour. **b** The schematic view of contour generation using the pushing-forward way

Step 1 Vector $\boldsymbol{P_iC_i^1}$ from point P_i to its contour's vertex C_i^1 is translated to point P_{i+1} along the direction of $\boldsymbol{P_iP_{i+1}}$;
Step 2 The vector after translation is projected onto the plane $P_{i+1} \cdot \boldsymbol{o}_{i+1}$;

Step 3 The vector after projection is normalized, and then is multiplied by the radius of point P_{i+1} to obtain the vector $\boldsymbol{P}_{i+1}\boldsymbol{C}_{i+1}^{1}$.

Similarly, the first vertex of the other contours can be obtained in the same manner. The rest vertices on the same contour can be calculated using the Rodrigues rotation formula [22].

$$\boldsymbol{v}_{rot} = \boldsymbol{v}\cos\theta + (\boldsymbol{k}\times\boldsymbol{v})\sin\theta + \boldsymbol{k}(\boldsymbol{k}\cdot\boldsymbol{v})(1-\cos\theta) \tag{2}$$

Given a vector $\boldsymbol{v}$ in R^3, formula 2 rotates it with a specific angle θ around a fixed rotation axis $\boldsymbol{k}$ to get a new vector $\boldsymbol{v}_{rot}$. For P_i's contour C_i, the vector $\boldsymbol{P}_i\boldsymbol{C}_i^1$ is regarded as $\boldsymbol{v}$ and C_i's orientation vector $\boldsymbol{o}_i$ is regarded as $\boldsymbol{k}$. The angle θ starts from zero and increases $\frac{2\pi}{m_i}$ for each rotation. Figure 3(b) depicts the pushing-forward way and rotation process.

The contour adheres that the bigger the point's r-component is, the more the number of vertices is. The vertices' number of P_i's contour is calculated as follows:

$$m_i = ceil(\frac{2\pi}{\arccos\frac{1-r_{avg}^2}{2r_i^2}}) + k_2, k_2 = 0, 1 \tag{3}$$

where $ceil()$ is a rounding function to obtain an integer and k_2 is another control parameter to ensure that the number m_i keeps an even all the time.

Adjacent contour may intersect with each other causing self-intersection in the resulting mesh. The algorithm marks and ignores these intersected contours when contour connection. Additionally, each contour keeps track of its center point so that associated information (like orientation) can be accessed in the following stage conveniently.

Contour Connection. The surface of an individual branch is reconstructed by contour connection. This technique sequentially connects the vertices on adjacent contours of a branch. The contours obtained above are non-parallel in 3D space and have different number of vertices. The algorithm converts adjacent contours from non-parallel to parallel state temporarily before connecting them. For adjacent contours C_i and C_{i+1}, the former is projected on to the plane $P_i \cdot \boldsymbol{o}_{i+1}$. Now the connection process between C_i and C_{i+1} is converted to C_{i_pro} and C_{i+1}.

The connection here belongs to one-to-one case [23], which needs to select a vertex on adjacent contours respectively as initial condition for starting the generation process of mesh triangle. Traditional method selects arbitrary vertex on one contour and uses "shortest distance" principle to select the other on another contour [24]. They may make mistakes for 3D contours (Fig. 4). In contrast, our pushing-forward way to calculate contours makes the 1–1 correspondence among contour's first vertex, avoiding the potential errors.

The first vertex $C_{i\_pro}^1$ on contour C_{i_pro} and C_{i+1}^1 on contour C_{i+1} can be regarded as the initial condition directly for constructing the first triangular

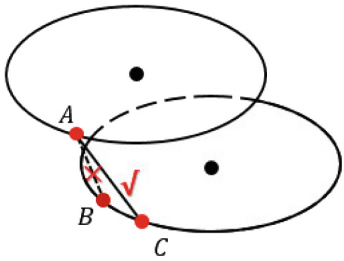


Fig. 4. Point a should correspond to point C, but may to point B

patch. There are two choices to select the third vertex of this triangle. The "minimum included-angle" criterion is used to determine whether the third vertex is $C^2_{i\_pro}$ or C^2_{i+1}. If the included angle formed by $\boldsymbol{C}^1_{i\_pro}\boldsymbol{C}^2_{i+1}$ and $\boldsymbol{P}_i\boldsymbol{P}_{i+1}$ is smaller than the angle formed by $\boldsymbol{C}^2_{i\_pro}\boldsymbol{C}^1_{i+1}$ and $\boldsymbol{P}_i\boldsymbol{P}_{i+1}$, the third vertex is C^2_{i+1}; otherwise, it is $C^2_{i\_pro}$. Then, the vertices $C^1_{i\_pro}$ and C^2_{i+1} or $C^2_{i\_pro}$ and C^1_{i+1} are regarded as new initial condition to construct the next triangular patch. The local surface mesh between two contours is reconstructed by traversing the vertices in the same winding order and connecting them. Finally, the vertex coordinates of the projected contour C_{i_pro} are replaced back to the coordinates of the contour C_i correspondingly.

Following the same procedure, the algorithm processes all of the adaptive contours in sequence to complete the surface reconstruction for individual branch.

Mesh Splicing. The transition surface between different branches is formed by mesh splicing. But the surface of a secondary branch may intersect with the surface of the corresponding primary branch near the bifurcation area (Fig. 5(a)). This problem is handled before the real surface generation of the secondary branch. If any vertex of a contour of the secondary branch is situated in the

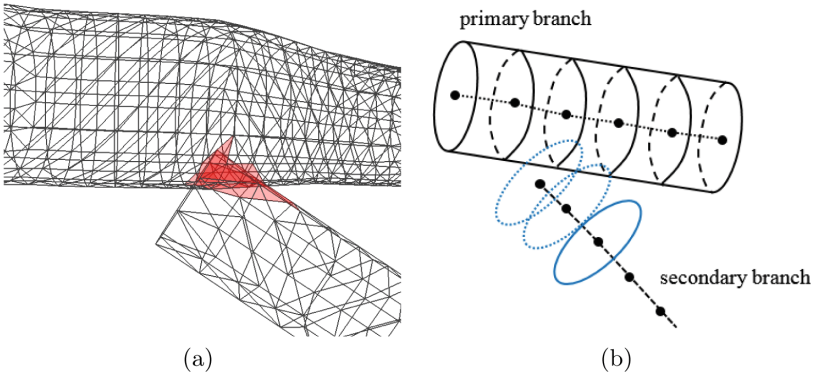
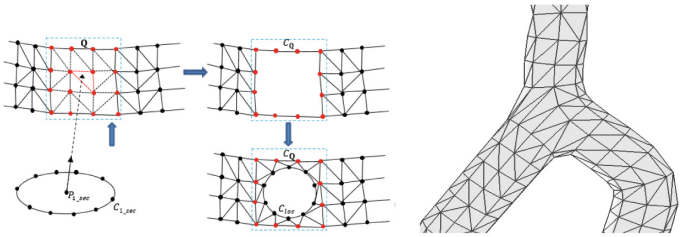


Fig. 5. Preprocessing intersection. **a** Mesh intersection. **b** Excluding dashed contours



(a) (b)

Fig. 6. a Splicing area determination. **b** Splicing example

volume of the primary branch, the algorithm does not deal with this contour when performing contour connection for the secondary branch (Fig. 5(b)).

The algorithm selects from the surface of the primary branch a suitable area $\mathbf{Q}$, which orients toward to the starting point's contour of the secondary branch. As shown in Fig. 6(a), a ray is emitted from the starting point P_{1_sec} of the secondary branch to the bifurcation point. Along the ray path, there exists some intersection points on the surface of the primary branch. The triangle where the closest intersection to point P_{1_sec} is marked as target. The algorithm finds other triangles around the target to form the area $\mathbf{Q}$. The splicing in this part is hence to form the transition mesh between the border of $\mathbf{Q}$ (denoted as $C_{\mathbf{Q}}$) and the contour of P_{1_sec} (denoted as C_{1_sec}). The distance from $C_{\mathbf{Q}}$ to C_{1_sec} may be so large that directly connecting them may cause lower quality mesh. Therefore, the algorithm inserts some middle contours between $C_{\mathbf{Q}}$ to C_{1_sec}.

To begin with, the algorithm determines a middle contour for locating, denoted as C_{loc} whose center P_{loc} and orientation $\mathbf{o}_{loc}$ are same with the barycenter and normal of the marked triangle. The center of each middle contour is positioned on the line from P_{loc} to P_{1_sec}, and the vertices are obtained by performing vector operations (such as projection, multiplication) for C_{1_sec}'s vertices. Secondly, the algorithm computes the distance of each pair of vertices on contours C_{loc} and C_{1_sec} separately and records the minimum value. This value is used to make modulus operation with E_{ave} to get the number of middle contours (denoted as N_{mids}). The E_{ave} represents an average edge-length of the polygon $C_{\mathbf{Q}}$. The plane in which each middle contour lies is represented by $P_t \cdot \boldsymbol{o}_t$, where P_t is this contour's center ($P_t = P_{t-1} + \frac{1}{N_{mids}} \cdot |\boldsymbol{P}_{loc}\boldsymbol{P}_{1\_sec}|$) and $\boldsymbol{o}_t$ is the orientation vector ($\boldsymbol{o}_t = \frac{\boldsymbol{o}_{t-1}+\boldsymbol{P}_{loc}\boldsymbol{P}_{1\_sec}}{2}$). The parameter t is from 0 to $N_{mids} - 1$ and when $t = 0$, $P_0 = P_{loc}$, $\boldsymbol{o}_0 = \boldsymbol{o}_{loc}$. Finally, $C_{\mathbf{Q}}$ and those middle contours as well as C_{1_sec} are connected to each other based on the contour connection algorithm described in previous subsection. Figure 6(b) gives an example of mesh splicing between two branches.

3.3 Surface Reconstruction of the Soma

The soma is only a point in original SWC data, without other more detailed information to describe its geometry. For generating its surface, the solution presented here constructs a suitable point set to be the input of Delaunay triangulation.

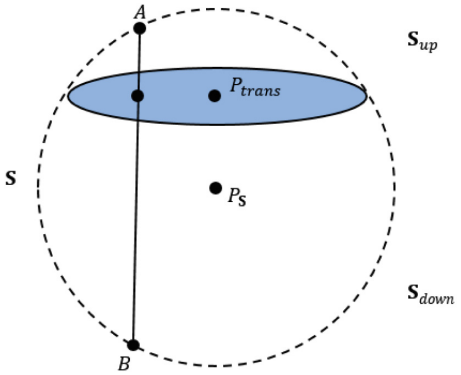
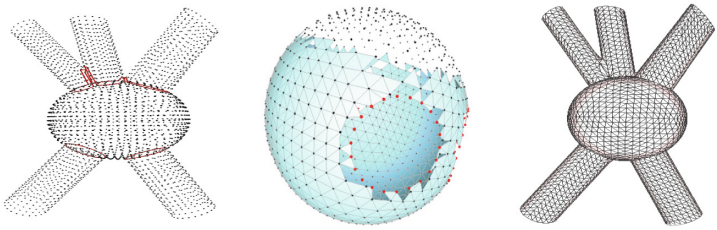


Fig. 7. Both A and B have the same projection on the plane in which C_{trans} lies. Point A is removed because of $A \in \mathbf{S}_{up}$

In the beginning, a collection of discrete points sampled uniformly on a sphere is as the initial point set, denoted as $\mathbf{S}$ with its center $P_{\mathbf{S}}$. For a soma branch, the first contour C_{1_sec} and its orientation vector $\boldsymbol{o}_{1\_sec}$ as well as center point P_{1_sec} are known. A copy (denoted as C_{trans} with its center P_{trans}) of contour C_{1_sec} is translated along the direction of vector $\boldsymbol{P}_{1\_sec}\boldsymbol{P}_{\mathbf{S}}$ until the copy is equivalent to a small circle on the sphere. The plane in which the C_{trans} lies divides $\mathbf{S}$ into two subsets ($\mathbf{S}_{up}$ and $\mathbf{S}_{down}$). Then, the algorithm removes from $\mathbf{S}$ the points that belong to $\mathbf{S}_{up}$ (Fig. 7) and inserts to $\mathbf{S}$ the points of the copy.

The planes in which the translated copies of different soma branches lies may lead to intersection. The algorithm removes the vertices in $\mathbf{S}_{up}$ of them respectively. After that, the algorithm removes the vertices after projection that belong to the common area formed by the intersected copies and inserts the remaining vertices.



(a) (b) (c)

Fig. 8. The basic idea for updating the $\mathbf{S}$ set and the triangulation

For achieving a better appearance after triangulation, point interpolation is used to add extra points between the vertices of the first contour of each soma branch and the vertices of its translated copy. Finally, the vertices of every soma branch's first contour and those extra points are inserted to update the $\mathbf{S}$ set (Fig. 8(a)). Furthermore, the vertices of the first contours and of their

copies have a new attribute to signify that they are boundary points. During Triangulation, no triangles are formed inside the area enclosed by the boundary points (Fig. 8(b)). Figure 8(c) shows the triangulation result, which can be merge exactly into the surface mesh of the branches.

4 Experiments and Results

The experiments are implemented by C++ language in this paper and the results are exported as OBJ format to visualize and render through a famous visualization software MeshLab [25]. Figure 9 shows part of a neurite branch, from the discrete points to its surface mesh representation, including the original points in (a), the spline curve in (b), the resampling in (c) and their adaptive contours in (d) as well as the surface mesh in (e).

Figure 10 shows the whole reconstructions of three neurons, which belong to the brain stem of the mouse. We evaluate the quality and validity of the reconstructed mesh through comparing with the provided measurement information. The measurements from NeuroMorpho.Org are the Soma Surface, the Total Surface and the Total Volume. The comparison results are listed in Table 1. It is obvious that there are some errors between our computations and the measurements. As approximated representations, the mesh model allows these errors. But performing some extra-processing operations like smooth on the mesh may help to improve and reduce these errors probably.

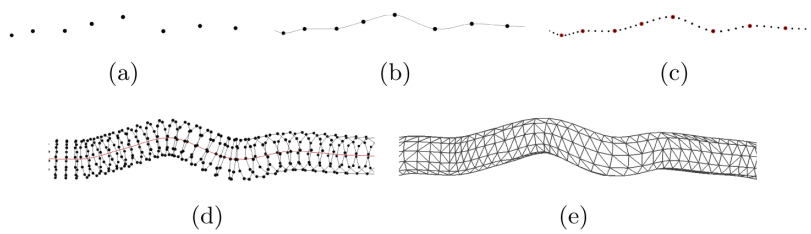


Fig. 9. Surface reconstruction of a neurite branch

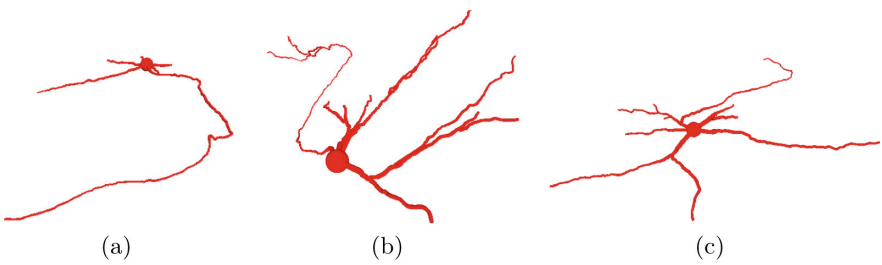


Fig. 10. Examples. **a** NMO_45929. **b** NMO_45928. **c** NMO_45927

Table 1. Comparing with the measurements. M (measurement); O (ours); E (error)

ID	Soma surface			Total surface			Total volume		
	M	O	E	M	O	E	M	O	E
NMO_45927	899.92	849.44	5.61%	8310.37	8766.53	5.49%	6398.93	7211.06	12.69%
NMO_45928	1287.75	1246.71	3.18%	13449.30	13299.55	1.11%	10517.60	11142.51	5.94%
NMO_45929	1256.04	1207.07	3.90%	8659.32	9379.83	8.32%	8080.25	8923.37	10.43%
Average	–	–	4.23%	–	–	4.97%	–	–	9.69%

5 Conclusion

In this paper, we have proposed a surface reconstruction pipeline to generate a mesh representation, which approximates the membrane of single neuron. The method converts the discrete points with radius and position information into a manifold mesh model with high density and good quality. For one thing, considering the differences of each points, our method uses a pushing-forward way to calculate adaptive contours for those points so that simplifying the branch's mesh generation based on contour connection. However, the splicing does not involve the uncommon case where there are more than two children point at the bifurcation point. For another, the surface reconstruction for complex topology near the soma is solved to a certain extent. This is accomplished by constructing a suitable point set and then performing the 3D triangulation directly. Nevertheless, the construction relies on the assumption that the soma is a sphere. In fact, the real shape of the soma is diverse, like star, cone or pear, etc. Additionally, original SWC data lacks detailed information about the soma. As a result, how to represent the soma precisely is a challenge task all the time.

When reviewing relevant literatures, we found that the convolution surfaces have great potential for modelling objects with complex geometries and are well suited to surface reconstruction of neuronal soma consequently. But we need to solve the problem that the convolution surface are not easily control as a kind of implicit surface. Even if it solved, we still have to consider how the convolution surface merges with the surface mesh of neurites. All these questions are the main directions in our future work.

References

1. Bear, M.F., Connors, B.W., Paradiso, M.A.: Neuroscience: Exploring the Brain, 4th edn. Wolters Kluwer (2015). https://doi.org/10.1007/BF02234670
2. Merjering, E.: Neuron tracing in perspective. J. Cytimetry Part A **77**(7), 693–704 (2010). https://doi.org/10.1002/cyto.a.20895
3. Donohue, D.E., Ascoli, G.A.: Automated reconstruction of neuronal morphology: an overview. J. Brain Res. Rev. **67**, 94–102 (2010). https://doi.org/10.1016/j.brainresrev.2010.11.003
4. Peng, H.C., Bria, A., Zhou, Z., Iannello, G., Long, F.H.: Extensible visualization and analysis for multidimensional images using Vaa3D. J. Nat. Protoc. **9**(1), 193–208 (2014). https://doi.org/10.1038/nprot.2014.011

5. Glesson, P., Steuber, V., Sliver, R.A.: neuroConstruct: a tool for modeling networks of neurons in 3D space. J. Neuron **54**(2), 219–235 (2007). https://doi.org/10.1016/j.neuron.2007.03.025
6. Livny, Y., Yan, F.L., Olson, M., Zhang, H., Chen, B.Q., EI-Sana J.: Automatic reconstruction of tree skeletal structures from point clouds. J. ACM Trans. Graph. **29**(6), 1–8 (2010). https://doi.org/10.1145/1882261.1866177
7. Zhu, X.Q., Guo, X.K., Jin, X.G.: Efficient polygonization of tree trunks modeled by convolution surfaces. J. Sci. China Inf. Sci. **56**, 1–12 (2013). https://doi.org/10.1007/s11432-013-4790-0
8. Xie, K., Yan, F.L., Sharf, A., Oliver, D., Huang, H., Chen, B.Q.: Tree modeling with real tree-parts examples. J. IEEE Trans. Vis. Comput. Graph. **22**(12), 2608–2618 (2016). https://doi.org/10.1109/TVCG.2015.2513409
9. Luboz, V., et al.: A segmentation and reconstruction technique for 3D vascular structures. In: Duncan, J.S., Gerig, G. (eds.) MICCAI 2005. LNCS, vol. 3749, pp. 43–50. Springer, Heidelberg (2005). https://doi.org/10.1007/11566465_6
10. Wu, X.L., Luoz, V., Krissian, K., Cotin, S., Dawson, S.: Segmentation and reconstruction of vascular structures for 3D real-time simulation. J. Med. Image. Anal. **15**(1), 22-34 (2015). https://doi.org/10.1016/j.media.2010.06.006
11. Yu, S., Wu, S.B., Zhang, Z.C., Chen, Y.L., Xie, Y.Q.: Explicit vascular reconstruction based on adjacent vector projection. J. Bioeng. Bugs **7**, 365–371 (2016). https://doi.org/10.1080/21655979.2016.1226667
12. De-Araújo, B.R., Lopes, D.S., Jepp, P., Jorge, J.A., Wyvill, B.: An survey on implicit surface polygonization. J. ACM Comput. Surv. **47**(4), 1–39 (2015). https://doi.org/10.1145/2732197
13. Wyvill, B., Wyvill, G.: Field functions for implicit surfaces. J. Vis. Comput. **5**(1), 75–82 (1989). https://doi.org/10.1007/BF01901483
14. Newman, T.S., Yi, H.: A survey of the marching cubes algorithm. J. Comput. Graph. **30**(5), 854–879 (2006). https://doi.org/10.1016/j.cag.2006.07.021
15. Yin, K.X., Huang, H., Zhang, H., Gong, M.L., Cohen-Or, D., Chen, B.Q.: Morfit: interactive surface reconstruction from incomplete point clouds with curve-driven topology and geometry control. J. ACM Trans. Graph. **33**(6) (2014). https://doi.org/10.1145/2661229.2661241
16. Lasserre, S., Hernando, J., Hill, S., Schümann, F., de Miguel Anasagati, P., Jaoudé, G.A., Markram, H.: A neuron membrane mesh representation for visualization of electrophysiological simulations. J. IEEE Trans. Vis. Comput. Graph. **18**(2), 214–227(2011). https://doi.org/10.1109/TVCG.2011.55
17. Carcia-Cantero, J.J., Brito, J.P., Mata, S., Pastor, L.: NeuroTessMesh: a tool for the generation and visualization of neuron meshes and adaptive on-the-fly refinement. J. Front. Neuroinform. **11** (2017). https://doi.org/10.3389/fninf.2017.00038
18. Brito., J.P., Mata, S., Bayona, S., Pastor, L., DeFelipe, J., Benavides-Piccione, R.: Neuronize: a tool for building realistic neuronal cell morphologies. J. Front. Neuroanat. **7**(15) (2013). https://doi.org/10.3389/fnana.2013.00015
19. Abdellah, M., Favreau, C., Hernando, J., Lapere, S., Schürmann, F.: Generating high fidelity surface meshes of neocortical neuronss using Skin Modifiers. In: Tam, G.K.L., Roberts, J.C. (eds.) Computer Graphics & Visual Computing(CGVC) 2019 (2019). https://doi.org/10.2312/cgvc.20191257
20. Ascoli, G.A., Donohue, D.E., Halavi, M.: NeuroMorpho.Org: a central resource for neuronal morphologies. J. Neurosci. **27**(35), 9247–9251 (2007). https://doi.org/10.1523/jneurosci.2055-07.2007

21. Eyiyurekli, M., Breen, D.E.: Localized editing of Catmull-rom splines. J. Comput. Aided Des. Appl. **6(**3), 307–316 (2009). https://doi.org/10.3722/cadaps.2009.307-316
22. Liang, K.K.: Efficient conversion from rotating matrix to rotation axis and angle by extending Rodrigues' formula. J. Comput. Sci. (2018). https://doi.org/10.48550/arXiv.1810.02999
23. He, J.G.: The correspondence and branching problem in medical contour reconstruction. In: 2008 IEEE International Conference on Systems, Man and Cybernetics, pp. 1591–1595. IEEE (2008). https://doi.org/10.1109/ICSMC.2008.4811514
24. Ekoule, A.B., Peyrin, F.C., Odet, C.L.: A triangulation algorithm from arbitrary shaped multiple planar contours. J. ACM Trans. Graph. **10**(2), 182–199 (1991). https://doi.org/10.1145/108360.108363
25. Cignoni, P, Callieri, M, Corsini, M, Dellepiane, M, Ganovelli, F, Ranzuglia, G.: MeshLab: an open-source mesh processing tool. In: Eurographics Italian Chapter Conference, vol. 2008, pp. 129–136 (2008). https://doi.org/10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136