Cylindrical approximation of a neuron from reconstructed polyhedron *

Wenhao Lin[†] Binhai Zhu[‡] Gwen Jacobs[§] Gary Orser[¶]

Abstract

In this paper, we investigate the problem of approximating a neuron (which is a disconnected polyhedron P reconstructed from points sampled from the surface of a neuron) with minimal cylindrical segments. The problem is strongly NP-hard when we take sample points as input. We present a general algorithm which combines a method to identify critical vertices of P and useful user feedback to decompose P into desired components. For each decomposed component Q, we present two algorithms, the first one tries to minimize the symmetric difference of volume between Q and the approximate cylindrical segment and the second one tries to minimize the radius of the approximate enclosing cylindrical segment. Previously, this process can only be done manually by researchers in computational biology. Empirical results show that the algorithm is very efficient in practice.

Keywords: Biological modeling, Approximation algorithms, Smallest enclosing cylinder

*This research is supported by NSF CARGO grant DMS-0138065.

[†]Department of Computer Science, Montana State University, Bozeman, MT 59717-3880, USA. Email: lin@cs.montana.edu.

[‡]Department of Computer Science, Montana State University, Bozeman, MT 59717-3880, USA. Email: bhz@cs.montana.edu.

[§]Center for Computational Biology, Montana State University, Bozeman, MT 59717, USA. Email: gwen@cns.montana.edu.

[¶]Center for Computational Biology, Montana State University, Bozeman, MT 59717, USA. Email: orser@cns.montana.edu.

1 Introduction

Approximating a neuron with cylindrical segments is an important process in constructing neural maps to study their functionality (so as to further study human/animal behavior) [JT96, JT00, PDJ99]. In practice, researchers would use commercial software to compute a polyhedron from points sampled from the surface of a neuron and then manually estimate the problem with cylindrical segments. Unfortunately, due to the complex structure of a neuron, the polyhedron reconstructed is not usually simple. In general it could be a disconnected polyhedron with extra errors (like an edge is incident to three triangles). See Figure 1 for an example of reconstructed polyhedron.

Theoretically, if we use sample points as input, i.e., given n points in 3D, compute the minimum number of cylindrical segments so as to minimize the sum of radii of these segments (or the sum of volume, or the volume of the union of the segments), then the problem is strongly NP-hard [Zh02]. This negative result probably explains why computational biologists prefer using reconstructed polyhedron instead of sample points as input since even this erroneous polyhedron carries important geometric and topological information.

In this paper, we try to solve this problem in a semiautomatic fashion. We take a reconstructed polyhedron P, whose surface is triangulated, as input. The user input/feedback is minimized and is used to speed up the computation. We identify critical vertices of P around which the polyhedron encounters great geometric/topological changes (like making a turn or branching into more small branches). We then decompose P into a number of smaller polyhedra Q's. To approximate each Q, we design an efficient heuristic algorithm for computing the smallest enclosing cylinder of a set of points in 3D which does not use any complex subroutines. We also propose a method which tries to minimize the symmetric difference of volume between Q and an approximate cylindrical segment.

The paper is organized as follows. In Section 2, we make necessary definitions. In Section 3 we discuss the smallest enclosing cylinder and related problems. In Section 4, we discuss how to identify critical vertices and we explain why at some stages user feedback is helpful in improving system efficiency. In Section 5, we conclude the paper.

2 Preliminaries

In this section we make some necessary definitions regarding geometry and approximation algorithms which are related to the problem to be studied. Throughout this paper, the distance metric is Euclidean unless otherwise specified.

An approximation algorithm for a minimization optimization problem Π provides a **performance guarantee** of λ if for every instance I of Π, the solution value returned by the approximation algorithm is at most λ of the optimal value for I. (*Notice that following the above definitions, λ is at least 1.*) For the simplicity of description, we simply say that this is a factor λ approximation algorithm for Π.

Given a set S of n points in 3D, the *diameter* of S is the maximum distance $d(p_i, p_j)$, $p_i, p_j \in S$, over all points in S. We denote it as $D(S)$. A cylinder $\mathcal{C}$ is an infinite set of points which have at most a distance R to a given line l in 3D. The line l is called the *center* of $\mathcal{C}$. The section area of $\mathcal{C}$ which is vertical to l corresponds to a disk with radius R.

Given a segment s_1s_2 in 3D, let the distance from a point q to the line through s_1s_2 be $d(q, r)$. The distance from q to s_1s_2 is $d(q, r)$ if r is on the segment s_1s_2, otherwise the distance from q to s_1s_2 is infinite. A cylindrical segment $\mathcal{S}$ is an infinite set of points which have at most distance R to a given line segment s_1s_2 in 3D. Similarly, the segment s_1s_2 is called the *center* of $\mathcal{S}$ and R is called the *radius* of segment. The two section areas through s_1, s_2 are called the *bases* of $\mathcal{S}$. The length of s_1s_2, $d(s_1, s_2)$, is called the *length* of $\mathcal{S}$ and $2R$ is called the *width* of $\mathcal{S}$. We denote them as $length(\mathcal{S})$ and $width(\mathcal{S})$ respectively and we assume that $length(\mathcal{S}) \geq width(\mathcal{S})$. The ratio $length(\mathcal{S})/width(\mathcal{S})$ is called the *aspect ratio* of $\mathcal{S}$, denoted by $\alpha(\mathcal{S})$. It is easy to see that $\alpha(\mathcal{S}) \geq 1$.

3 The Smallest Enclosing Cylinder and Related Problems

In this section, we revisit the problem of computing the smallest enclosing cylinder of a set of points in 3D. This problem has found applications in CAD and CAM. Given a set of n points in 3D, Schömer et al. presented an $O(n^4 \text{polylog } n)$ time algorithm [SSTY00] and Agarwal et al. obtained an $O(n^{3+\epsilon})$ time algorithm [AAS97] to solve the problem. When n is sufficiently large, both of the algorithms are not practical. Agarwal et al. also obtained a $(1+\delta)-$approximation which runs in $O(n/\delta^2)$ time [AAS97]. However, their algorithm used several subroutines (like computing the transversal of a set of 3D cubes and computing the smallest enclosing circle of a set of 2D points) which also make it hardly practical in terms of implementations. Recently, Chan present another $(1+\delta)-$approximation which runs in $O(n/\delta)$ time using convex programming [Ch00]. Chan's algorithm is also not known to be practical.

In [Zh02], an $O(n \log n + n/\delta^4)$ time algorithm is proposed to compute a $(1+\delta)$-approximation for the smallest enclosing cylinder problem. The algorithm is completely elementary and the only slightly complex subroutine is computing the diameter of the input points. The idea behind the algorithm is to first compute the diameter of the input points, which gives a rough orientation of the smallest enclosing cylinder, and then use two grids to locate the final approximate center of the smallest enclosing cylinder. (The purpose for this algorithm was to have a PTAS for the general problem when the number of cylindrical segments is given in advance.) However, even though it is easy to implement, for relatively large data sets with small δ it seems to be inefficient.

In this section, we first propose another $(1+\delta)$-approximation for the smallest enclosing cylinder problem. The running time is $O(n/\delta^2)$. The algorithm is easy to implement compared with previous ones. The only slightly complex subroutine used in the algorithm is to compute the smallest enclosing circle of a set of 2D points, which can be implemented efficiently [Ga].

Note that in practice the sampled points we obtain from the surface of a neuron usually contain errors. Therefore, using smallest enclosing cylinder (cylindrical segment) might not be all the biologists want. In this section, we also investigate another problem, namely, given a polyhedron Q compute a cylindrical segment C' such that the symmetric difference of volume of Q and C' is minimized. We prove a necessary condition on the optimality of C' and hence can obtain a close approximation of C'.

3.1 The smallest enclosing cylinder problem revisited

In this subsection we first propose a simple and practical approximation algorithm for the smallest enclosing cylinder problem. Given a set S of n points, let C^* be the smallest enclosing cylinder of

S (i.e., S is completely contained in C^* and the radius of C^* is minimized). Also, let the center of C^* be u^*v^*. We first have the following simple approximation algorithm:

Algorithm 1.

(1) Discretize a unit ball centered at the origin into an δ by δ grid and use all unit vectors through a grid point and the origin as candidate vectors.

(2) Project all the points in S on a plane vertical to a candidate vector and compute the corresponding smallest enclosing circle of the projected points. Among all solutions, pick up the one with the smallest enclosing circle.

The idea behind this algorithm is a brute-force method: simply search over all possible discrete directions for one which has the smallest angle with u^*v^*. As computing the smallest enclosing disk of a set of n points in 2D can be done in $O(n)$ time time [MSW92, We91], it is easy to see that this algorithm runs in $O(n/\delta^2)$ time and returns a $(1+\delta)$-approximation for the smallest enclosing cylinder of S.

Notice that the radius of the enclosing cylinder of S is a function $\mathcal{F}$ of all (unit) vectors. The radius of C^* is basically the global minimum of this function. In some applications when n is large and δ has to be very small $O(n/\delta^2)$ time might not be efficient enough. Unfortunately, we do not know of any practical $O(n/\delta)$ time approximation algorithm yet. We present below a heuristic algorithm which runs in $O(n/\delta^2)$ time and always converges to a local minimum of $\mathcal{F}$. However, it seems that with practical data sampled from a neuron it runs in $O(n/\delta)$ time and almost always converges to the global minimum. We first present following simple approximation algorithm.

Algorithm 2.

(1) Pick any point p' of S. Let the furthest point from p' be p".

(2) Project all points orthogonally on a plane vertical to $p'p$".
Compute the smallest enclosing disk D of these projected points. Let the line passing through the center of D which is also parallel to $p'p$" be ρ.
Return twice of radius of D as the width of A and ρ as the center of A.
The length of A can be returned in an extra of $O(n)$ time by finding the two points furthest along $p'p$".

We have the following lemma.

Lemma 3.1 Algorithm 2 presents a factor-3 linear time approximation for the smallest enclosing cylinder problem.

Proof: Let the length of C^* be $length(C^*)$. We consider two cases: (a) $length(C^*) \leq 2\times width(C^*)$, and (b) $length(C^*) > 2\times width(C^*)$. In case (a), it is easy to show that A has a width at most $\sqrt{5}\times width(C^*)$. In case (b), a calculation shows that the width of A is bounded by $(2+\sqrt{1-4/\alpha^2})\cdot width(C^*)$, which is always bounded by $3\times width(C^*)$. The running time of Algorithm 1 is $O(n)$ as computing the smallest enclosing disk of a set of n points in 2D can be done in $O(n)$ time

[MSW92, We91]. □

In practice, especially in biology related applications, a factor-3 approximation is hardly useful. In fact, what we really want in our neuron simulation project is an approximation algorithm which runs in $O(n/\delta)$ time and returns an $(1+\delta)$-approximation. However, the only known theoretical algorithm achieving this objective [Ch00] does not seem to be implementable. In the following we present a heuristic algorithm which always converges to a local minimum of $\mathcal{F}$. Moreover, even though its theoretical running time is $O(n/\delta^2)$, for practical data we handle in our neuron simulation project the practical running time seems to be $O(n/\delta)$. The empirical results will be presented later in this subsection.

Let C be a minimal enclosing cylindrical segment of S with center γ, i.e., when γ is fixed the radius of C is minimum, and let C_γ be the orthogonal projection of C along γ. Without loss of generality, assume that three points in S whose orthogonal projection along γ, $a_\gamma, b_\gamma, c_\gamma$, uniquely determine the smallest enclosing disk C_γ among all projected points. Let the three corresponding points in S be a, b, c.

We define the local rotation of γ toward a point a on C (but not on its bases) by ϕ as follows. We pick any two points u', v' on γ which are not contained in C such that $C \cap \gamma$ is between u', v' and $d(u', v')$ is at least $D(S)$. (Algorithmically, this can be done in $O(n)$ time by computing the minimum axis parallel bounding box $\mathcal{B}$ of S and picking up u', v' on γ such that $\mathcal{B} \cap \gamma$ is between u' and v' and $d(u', v')$ is equal to the length of the diagonal of $\mathcal{B}$. Clearly $d(u', v') = D(\mathcal{B}) \geq D(P)$.) We rotate $u'v'$ on the plane $\triangle(u'v'a)$ around u' by an angle of ϕ such that after the rotation a is closer to the new $u'v'$, i.e., the rotation is toward a (Figure 2). Symmetrically we can perform this local rotation around v'. We have the following lemma.

Lemma 3.2 If we can always obtain a smaller enclosing cylindrical segment by a local rotation of γ toward either a, b or c, then the radius of C is not a local optimum of $\mathcal{F}$.

Proof: If C is a local optimum of $\mathcal{F}$, then by definition we cannot find a point $a \in S$ such that the local rotation of the center of C toward a results in a better C. □

We now present the following heuristic for the smallest enclosing cylindrical segment problem.

Heuristic 1.

(1) Compute the approximate cylindrical segment A of S using Algorithm 2.
Let the orthogonal projection of A along its center ρ be D.

(2) Compute the minimum axis-parallel bounding box $\mathcal{B}$ of S.
Let its diagonal be $D(\mathcal{B})$.

(3) Perform six possible local rotations of the center of A toward a, b and c by an angle of $\delta \cdot width(A)/3D(\mathcal{B})$. This results in 6 new directions $\rho_1, ..., \rho_6$. In each case, project S orthogonally along ρ_i and compute the smallest enclosing disk D_i. If none is smaller than D, then return D, ρ as the approximation solution. Otherwise, Update $D \leftarrow D_j, \rho \leftarrow \rho_j$ such

that D_j is of the smallest size among $D, D_1, ..., D_6$ and repeat step (3).

We have the following theorem regarding Heuristic 1. The proof is trivial and hence omitted.

Theorem 3.3 Heuristic 1 runs in $O(n/\delta^2)$ time and always converges to a local minimum of $\mathcal{F}$.

The reason why we use Heuristic 1 is that in our neuron simulation project the cylindrical segments used are usually of large aspect ratio. Moreover, in this case, its subroutine, Algorithm 2, presents a good approximation for the center of C^*. In the following tables we compare the running times (T) and the radii (R)of the approximate smallest enclosing cylinder returned by the three $(1+\delta)$-approximations: the one proposed in [Zh02], Algorithm 1 and Heuristic 1. Table 1 shows the empirical results when the data are from our neuron simulation project. For Zh02, since the running time is virtually $O(n/\delta^4)$ so we cannot set δ to be too small as that will take several hours to run when $n \approx 2000$. So basically we compare Algorithm 1 and Heuristic 1, for which we set $\delta = 3°$. In all cases, the radius of a cylindrical segment is the measured distance (as in the table) multiplying a scaled constant (76.79/1024) millimeters. For convenience we simply use mm' as the length measure. (For example, in the fifth row, second column we have 2.543. The actual radius of the cylindrical segment is $2.543 \times 76.79/1024$ millimeters.) The data sets we use are drawn from branches in the neuron shown in Figure 1 and the platform of the testing is Java3D on which our system is built. Typically, in our application each cylindrical segment encloses less than 2000 points.

n	127	233	509	1131	1479	2344	5453
$T_{Zh02}(s)$	7.735	13.906	30.609	84.484	110.563	175.781	355.578
$T_{Algorithm\ 1}(s)$	6.656	8.672	35.281	98.047	125.256	186.344	1211.609
$T_{Heuristic\ 1}(s)$	0.281	2.156	6.375	5.860	4.469	3.875	90.969
$R_{Zh02}(mm')$	2.543	6.265	6.958	9.242	9.370	12.427	27.897
$R_{Algorithm\ 1}(mm')$	2.480	6.212	6.665	8.932	9.046	12.100	27.247
$R_{Heuristic\ 1}(mm')$	2.481	6.214	6.654	9.082	8.995	12.096	27.198

Table 1. Empirical results for Zh02, Algorithm 1 and Heuristic 1. $\delta = 3°$ for Algorithm 1 and Heuristic 1.

We can observe that Heuristic 1 outperforms Algorithm 1 in all the 7 cases. (Notice that when $n \approx 1100$, Algorithm 1 already needs about 90 seconds to run, which is not acceptable in an interactive system.) Moreover, in 6 out of 7 cases it converges to the global minimum of the corresponding problems. (In the case when n=1131, it is not known whether Heuristic 1 converges to the global minimum or not as we did not try a smaller δ. But even if it does not, the returned cylinder is good enough—less than 2% off the optimum). In fact, we have tested Heuristic 1 for several months over different neurons and we have yet found a practical example for which it fails (i.e., misses the global minimum by more than 5%) or takes a long time (i.e., over 10 seconds) to run. (It is easy to construct some examples for which the heuristic either converges to a local minimum or takes $O(n/\delta^2)$ time to run.) In Table 2 we list the empirical results for Algorithm 1 and Heuristic 1 only, over exactly the same data sets used in Table 1, with $\delta = 1°$ for Algorithm 1 $\delta = 3°$ for Heuristic 1.

n	127	233	509	1131	1479	2344	5453
$T_{Algorithm\ l}(s)$	56.969	74.625	304.438	877.453	1111.031	1654.734	10793.984
$T_{Heuristic\ 1}(s)$	0.281	2.156	6.375	5.860	4.469	3.875	90.969
$R_{Algorithm\ 1}(mm')$	2.455	6.211	6.658	8.924	9.040	12.097	27.202
$R_{Heuristic\ 1}(mm')$	2.481	6.214	6.654	9.082	8.995	12.096	27.198

Table 2. Empirical results for Algorithm 1 with $\delta = 1°$ and Heuristic 1 with $\delta = 3°$.

It is easily seen that the accuracy of Algorithm 1 does not change much compared with Table 1 (where $\delta = 3°$). However, the running time for Algorithm 1 becomes horrible. Notice that when $n \approx 2300$, Algorithm 1 needs about 25 minutes to run, which makes it useless in any practical system! This makes Heuristic 1 a clear winner over Algorithm 1, even though this is not supported by the theoretical results.

3.2 The minimal cylindrical segment problem

As we mentioned at the beginning of this section, because the sample points S obtained from the surface of a neuron (hence the reconstructed polyhedron P) always contain some errors, when we decompose P into sub-polyhedra it might not be good enough to approximate a sub-polyhedron Q using the smallest enclosing cylinder of Q. A different approximation might be needed.

The minimal cylindrical segment problem is defined as follows. Given a simple polyhedron Q, compute a cylindrical segment C' such that the symmetric difference of volume between them, $Vol(Q - C') \cup Vol(C' - Q)$ is minimized. Let the part of the surface of C' which is inside Q be C^- and the part of C' which is outside Q be C^+. Also, let the area of C^- and C^+ be $A(C^-)$ and $A(C^+)$ respectively. We have the following lemma which is similar to a lemma proved in [Zh97].

Lemma 3.4 $Vol(Q - C') \cup Vol(C' - Q)$ is minimized only if $A(C^-) = A(C^+)$.

Proof: The lemma claims that if $Vol(Q-C') \cup Vol(C'-Q)$ is minimized then $A(C^-) = A(C^+)$. Assume this is not the case, i.e., if $A(C^-) > A(C^+)$, then we can increase the radius and length of C by ϵ to obtain C" such that $Vol(Q-C") \cup Vol(C"-Q)$ is smaller than $Vol(Q-C') \cup Vol(C'-Q)$, which is a contradiction. (If $A(C^-) < A(C^+)$ then we can obtain a similar claim.) $\square$

This lemma gives us a heuristic algorithm for finding an approximate version of C', H, to approximate Q. (Notice that $A(C^-)$ and $A(C^+)$ are difficult to compute.) Assume that the direction of the center of H is given (if not, we can discretize the space to have a finite number of directions for H), then we can first compute an approximate version of H, H', such that H' is a m-prism (i.e., the section area of H' is a regular m-polygon instead of a circle) and the center of H' is the same as that of H. Certainly, this involves the computation of the intersection of H' and Q. Once H' is obtained then we simply return the smallest enclosing cylindrical segment of H' as H.

4 Decomposing a Reconstructed Polyhedron and Empirical Results

In this section, we discuss how to decompose a reconstructed polyhedron P into sub-polyhedra. We present a semi-automatic algorithm which combines the identification of critical vertices (edges) of

P together with some user input.

From the user point of view, certainly an automatic solution would be the most desirable. However, due to the following reasons, for the moment this is very hard to do. First, the sample points obtained from the surface of a neuron contain errors. Second, due to the complex structure of a neuron, the reconstructed polyhedron (from erroneus sample points) induce further errors. For example, in P some edges might be incident to three faces and moreover; P might not be connected at all, i.e., it could contain several connected components.

We now review the method in [LGTW98] to identify critical edges in a polyhedron. This method was originally proposed to simplify a polyhedron using edge contraction. Let T be a simple polyhedral surface (or polyhedron) with n vertices in 3D. Let (v_1, v_2) be an edge of T. We say that (v_1, v_2) is a *feasible* edge if contracting (v_1, v_2) from v_2 induces no self-intersection of the surface, i.e., after the edge contraction the resulting new surface is still simple (Figure 3). In addition, for each pair of triangles, $\triangle(uwv_2)$ and $\triangle(uwv_1)$, before and after the edge contraction, the angle between the outer norms of the two triangles is bounded by a constant (which is $\pi/2$ in the implementation). The latter constraint makes sure that the contraction of an edge should not change the local geometry too much. We follow [LGTW98] with the following definition.

The *weight* of a feasible edge e is defined as the product of the length of e, $|e|$, and the importance factor of e, which is related to the local geometry of e, over a reference length L_e. Intuitively, if the importance value of e is big then we should delay the contraction of e to a late stage. In [LGTW98], the importance value (or weight) of a vertex v is defined as the sharpness of v. To simplify the calculation, it is calculated as $I(v) = (x_{max} - x_{min}) + (y_{max} - y_{min}) + (z_{max} - z_{min})$ where $(x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max})$ are the minimum and maximum values of all triangle normal vectors around v. The importance factor of $e = (v_1, v_2)$ is defined as the minimum of $I(v_1)$ and $I(v_2)$. The reference length L_e is the maximum length of the axis-parallel bounding box of the model.

In [LGTW98], it was proposed that we always contract the lightest feasible edge, i.e., the one with the smallest weight as defined above. The empirical results are very promising although there is no theoretical proof that explains why an edge with a large weight is important in describing the topology of T (say, it is at the junction of two branches of T).

In our algorithm, we only use $I(v)$ defined above to classify important and common vertices in P. We set a threshold on $I(v)$'s such that all the important vertices of interest are identified. However, to achieve this, it is found that typically about 20% of the vertices are identified as well (though most of them are introduced by the errors mentioned above, i.e., most of the vertices in P with large weight should have a smaller weight if P contained less errors). Because of this reason, we ask the user to first input the following information to the system: (1) Connect different disconnected components to make P completely connected. (2) Around each branch where a cut should take place, input two vertices (this does require some user cooperation).

We now mention the algorithm to cut a branch our of P using the two input vertices u and v. (In our current implementation u, v are important vertices in P. But it is easy to let user choose two flat vertices and walk from those two vertices to two points with large weights.) All we want to compute is a plane which passes through u and v and cuts the branch at right angle. What we do is as follows. From u (or v) we classify all other important vertices around u into several classes.

We are interested in those points $w_i, i = 1, 2, ..., W$, such that vector $\widehat{uw_i}$ and vector $\widehat{uv}$ has large angle. (Intuitively, these points are at the 'ridge' of a branch.) Let Z be the median vector of all those vectors $\widehat{uw_i}$. Then the target plane is the one passing through u, v and has the maximum angle with vector Z (Figure 4). In the system, we also give user a chance to undo this process if he/she is not satisfied with the target plane, which is usually caused by a bad choice of u and v.

Notice that in the above procedure we do not leave all the work to users by asking them to input three vertices (which uniquely defines a cutting plane). The reason is that this operation involves the rotation of P and it is always easier for user to identify two point from one side of P. Therefore, even though this procedure is simple, it is meaningful.

Our system is built on Java 3D which supports various graphics operations. The system takes a (possibly disconnected) polyhedron P and asks users to first make it connected. Then it computes all vertices with large weight. The user can input two important vertices at which he/she wants to cut a sub-polyhedron Q out of P. The system applies the above algorithm to cut Q from P. Finally the system takes Q and computes the approximate cylinder for Q using Heuristic 1 presented in Section 3. This process continues until P are completely decomposed into desirable sub-polyhedra. (Alternatively, the last loop can be implemented in a batch fashion; i.e., the user can input all pairs of important vertices first and leave all the computation to the system.)

Finally we show some empirical results. In Figure 5, we show the output of our system when we cover part of P with eight cylindrical segments. To make the image more readable, we only show a part of the approximation. (So far, for each sub-polyhedron we compute and display the approximate smallest enclosing cylinder.) In this case, P has 8954 vertices and about twice as many faces. To manually fit this model completely, it takes a technician one to two days—it is hard to have an exact measure of time to the seconds though. With our system, it takes less than one hour.

5 Concluding Remarks

In this paper we describe a semi-automatic method to approximate a neuron with minimal cylindrical segments from a reconstructed polyhedron. Our process is much more efficient than the previous manual estimation, which takes a few days to manually fit a reconstructed polyhedron with cylindrical segments. An immediate question is whether we could have a completely automatic method. We are currently working on smoothing a reconstructed polyhedron and on studying new methods to define critical vertices (edges). Another interesting question is to further investigate why Heuristic 1 works better than what the corresponding theoretical result implies in practice. An analysis similar to the Simplex method for linear programming might be necessary.

References

[AAS97] P. Agarwal, B. Aronov and M. Sharir. Line transversals of balls and smallest enclosing cylinders in three dimensions. In *Proc. 8th ACM-SIAM Symp on Discrete Algorithms (SODA'97)*, New Orleans, LA, pages 483-492, Jan, 1997.

[Ch00] T. Chan. Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. In *Proc. 16th ACM Symp on Computational Geometry (SCG'00)*, Hong Kong, pages 300-309, June, 2000.

[MSW92] J. Matoušek, M. Sharir and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16:498-516, 1992.

[Ga] B. Gärtner. `http://www.inf.ethz.ch/personal/gaertner/miniball.html`

[JT96] G. Jacobs and F. Theunissen. Functional organization of a neural map in the cricket cercal sensory system. *J. of Neuroscience*, 16(2):769-784, 1996.

[JT00] G. Jacobs and F. Theunissen. Extraction of sensory parameters from a neural map by primary sensory interneurons. *J. of Neuroscience*, 20(8):2934-2943, 2000.

[LGTW98] R. Lau, M. Green, D. To and J. Wong. Real-time Continuous Multi-Resolution Method for Models of Arbitrary Topology. *Presence: Teleoperators and Virtual Environments*, 7:22–35, 1998.

[PDJ99] S. Paydar, C. Doan and G. Jacobs. Neural mapping of direction and frequency in the cricket cercal sensory system. *J. of Neuroscience*, 19(5):1771-1781, 1999.

[PS85] F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 1985.

[SSTY00] E. Schömer, J. Sellen, M. Teichmann and C.K. Yap. Smallest enclosing cylinders. *Algorithmica*, 27:170-186, 2000.

[We91] E. Welzl. Smallest enclosing disks (balls and ellipsoids). In *New results and new trends in computer science*, LNCS 555, pages 359-370, 1991.

[Zh97] B. Zhu. Approximating convex polyhedra with axis-parallel boxes. *Intl. J. of Computational Geometry and Applications*, 7(3):253-267, 1997.

[Zh02] B. Zhu. Approximating 3D points with cylindrical segments. In *Proc. 8th Intl. Computing and Combinatorics Conf.(COCOON'02)*, LNCS 2387, pages 400-409, Aug, 2002.

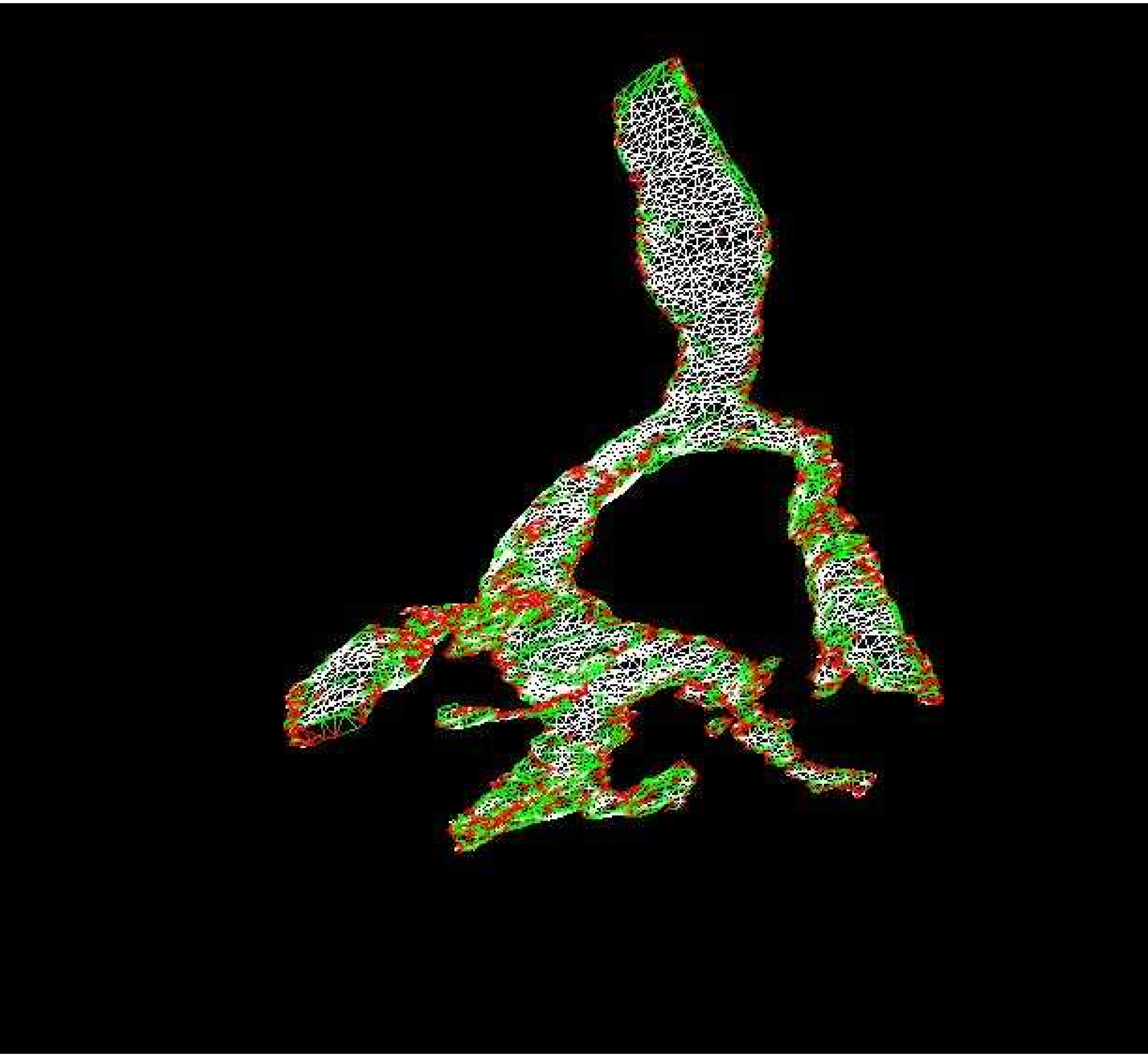

Figure 1. A neuron polyhedron with 8954 vertices.

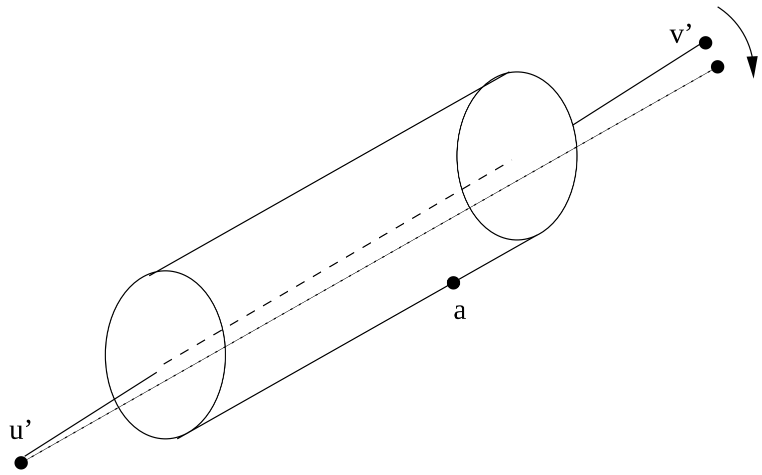

Figure 2. Local rotation.

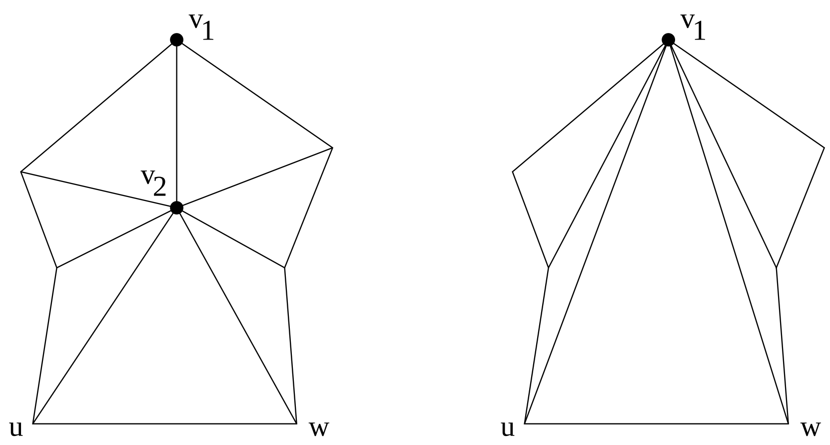

Figure 3. An example of edge contraction.

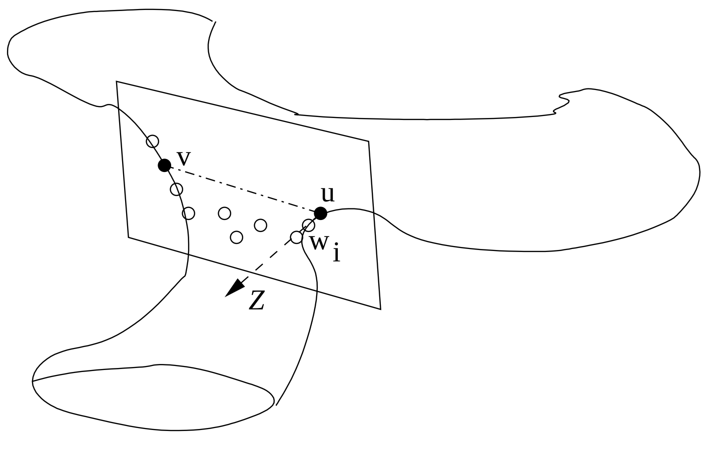

Figure 4. Cutting a branch with user input.

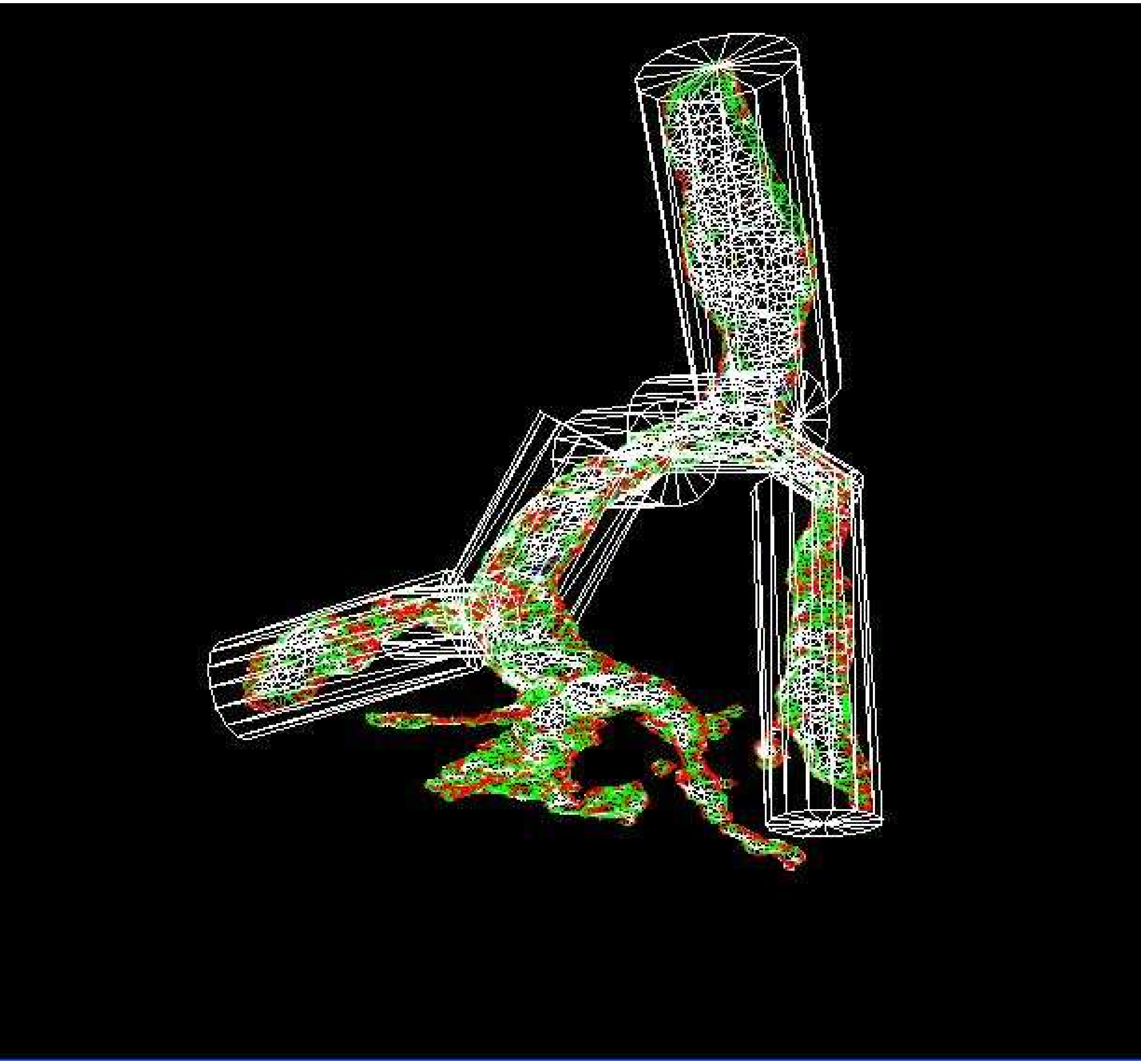

Figure 5. Fitting part of a neuron polyhedron with 8 cylindrical segments.