Distributed Algorithms for Bottleneck Identification and Segmentation in 3D Wireless Sensor Networks

Hongyu Zhou, Ning Ding, Miao Jin, Su Xia and Hongyi Wu

***Abstract*—Segmentation decomposes a network with complex and irregular shape into a set of subnetworks, each under a simple boundary condition without bottlenecks. It has a wide spectrum of applications in routing, coverage, localization, backbone construction and maintenance, and in-network data centric storage and retrieval. To our best knowledge, this is the first work that tackles the segmentation problem in 3D wireless sensor networks. We propose a fully distributed 3D segmentation scheme with mere network connectivity information. Each node on boundary computes its injectivity radius, which reflects the narrowness of the corresponding boundary area and thus is employed to locate the undesired bottlenecks. A cluster of connected boundary nodes with similar smallest injectivity radii form a bottleneck segment. A recursive process is applied to identify a set of such bottlenecks, which together divide the network boundary into segments. An internal non-boundary node simply joins the nearest segment, thus completing the segmentation of the entire 3D sensor network. Our simulations show that the proposed algorithm works efficiently under various sensor network models with different boundary conditions and noise levels, always yielding appropriate segmentation results. We further demonstrate that segmentation can effectively promote the performance of a range of applications in 3D wireless sensor networks.**

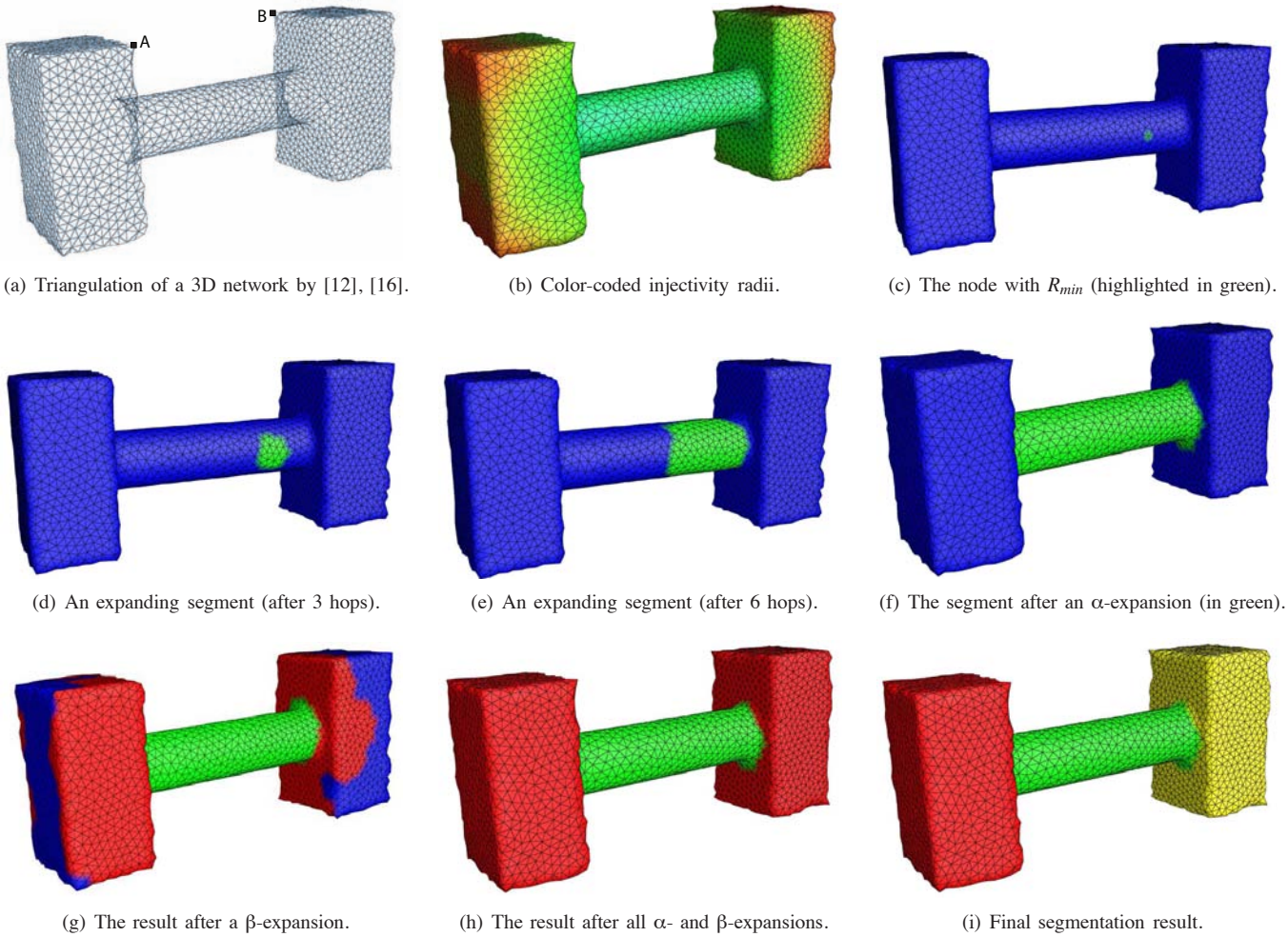
I. Introduction

Due to randomness in sensor deployment and possible dynamics during its operation, a sensor network usually exhibits irregular shape (or boundary conditions), leading to undesired intractability in many applications, especially under the emerging 3D sensor network settings [1]–[12]. For example, greedy routing is one of the most promising routing schemes for wireless sensor networks, where a node makes its routing decision by standard distance calculation based on a small set of local coordinates only, thus achieving scalable data delivery. However, greedy routing fails at irregular concave boundaries. For example, there does not exist a greedy routing path from Node A to Node B in the network shown in Fig. 1(a). This is mainly due to the bottleneck area in the middle of the network that creates dead-ends for greedy routing. While face routing can be applied in 2D networks to recover such greedy routing failures [13], [14], there is no effective, deterministic solution for 3D networks as proven in [15]. Similarly, irregular network shape may lead to unbalanced load in distributed in-network data storage and query. In such systems, data are mapped to some coordinates via a geographic hash function to enable efficient storage and retrieval. To perform geographic hashing, however, a bounding box of the network must be identified to define the range of the coordinates. If the network has an irregular shape, a significant part of the bounding box is empty, i.e., unoccupied by any sensors. As a result, the boundary nodes, especially at the bottleneck areas (see the middle part of the network illustrated in Fig. 1(a)) must store the data hashed to nearby empty space, thus experiencing high storage load and communication overhead (for both data storage and retrieval). The bottlenecks in an irregular network also introduce challenges in coverage, localization, and backbone construction and maintenance.

The hassles due to the bottleneck areas in irregular sensor networks naturally motivate us to decompose the network into segments, each under a desired simple boundary condition without bottlenecks. The problem of segmentation has been studied in 2D sensor networks [17]. It employs a shape segmentation scheme based on flow complex introduced in [18]. More specifically, each boundary node initiates a flooding packet inward the network to create flows, and a node on a flow is assigned a flow direction. A node with no flow direction is called a sink. It, together with other sensors that flow into it, forms a segment. The effectiveness of the algorithm depends on the accuracy in the computation of flows and sinks in a discrete 2D sensor network. A great deal of special care and delicate strategies are required as shown in [17]. Under the discrete 3D setting with mere connectivity of a volumetric points cloud, however, it is extremely difficult to identify and control noises for computing flows and sinks, in order to achieve efficient segmentation.

In this research, we propose a distributed algorithm to segment 3D wireless sensor networks. It is based on a parameter called injectivity radius, which is calculated by each individual sensor on the network boundary. The injectivity radius measures the narrowness of the corresponding boundary area, and thus is employed to locate the undesired bottlenecks. In particular, a cluster of connected boundary nodes with similar smallest injectivity radii form a bottleneck segment. A recursive process is applied to identify a set of such bottlenecks, which together divide the network boundary into segments. An internal non-boundary node simply joins the nearest segment, thus completing the segmentation of the entire 3D sensor network. While segmentation does not completely solve all problems in 3D sensor networks that we have discussed earlier, our simulations show that it can effectively improve the performance of several applications such as greedy routing and distributed in-network data storage and query, benefitted from the smoothed boundary conditions

The authors are with the Center for Advanced Computer Studies, University of Louisiana at Lafayette, P.O. Box 44330, Lafayette, LA 70504. This work is supported in part by the National Science Foundation under grant CNS-1018306.



(a) Triangulation of a 3D network by [12], [16]. (b) Color-coded injectivity radii. (c) The node with R_{min} (highlighted in green).

(d) An expanding segment (after 3 hops). (e) An expanding segment (after 6 hops). (f) The segment after an α-expansion (in green).

(g) The result after a β-expansion. (h) The result after all α- and β-expansions. (i) Final segmentation result.

Fig. 1. Illustration of the segmentation algorithm. (a) Triangulation of a 3D sensor network constructed by [12], [16]. (b) Color-coded injectivity radii, where the radii increase as color changes from blue to green to red. (c)-(h) Nodes with $\alpha_i = FALSE$ and $\beta_i = FALSE$ are colored in blue; nodes with $\alpha_i = FALSE$ and $\beta_i = TRUE$ are colored in red; and nodes with $\alpha_i = TRUE$ are colored in green. (i) Three segments are colored in red, yellow and green, respectively.

of individual segments. The main contributions of this work are summarized below:

- It introduces the first effective solution to segment 3D wireless sensor networks. It is fully distributed and based on network connectivity information only.
- Our simulations show that the proposed algorithm is robust. It always yields appropriate segmentation results for a wide variety of 3D wireless sensor networks.
- We further show that segmentation can effectively promote the performance of a range of applications in 3D wireless sensor networks.

The rest of this paper is organized as follows: Sec. II introduces the proposed algorithms for calculating injectivity radius and performing segmentation. Sec. III presents the results of segmentation and its application. Finally, Sec. IV concludes the paper.

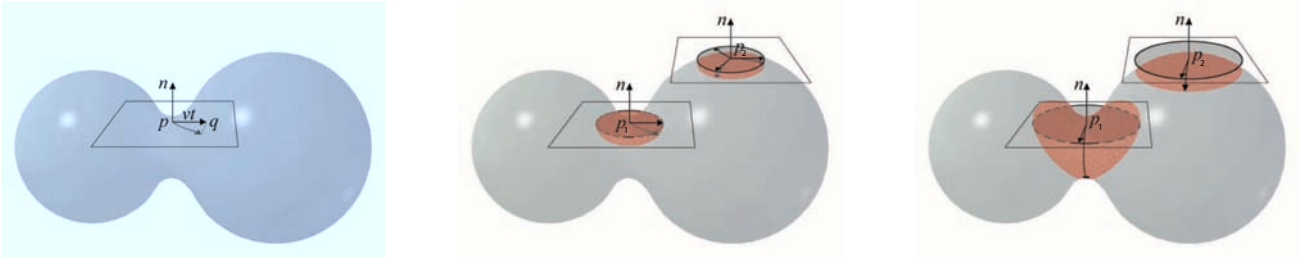
II. Proposed Segmentation Algorithm

The objective of this work is to locate the undesired bottlenecks in a 3D wireless sensor network and accordingly partition the network into segments with simple boundary conditions. For example, the network model in Fig. 1(a) can be largely divided into three segments, as colored in red, yellow and green, respectively, as illustrated in Fig. 1(i). In this research, we introduce a parameter named injectivity radius for each boundary node to effectively measure the narrowness of the corresponding boundary area and propose a distributed algorithm to segment the network based on such nodal injectivity radii.

A. Computation of Injectivity Radius

The injectivity radius is a key concept in Riemannian geometry. Under a 3D network setting, it indicates the narrowness of a given part of the network boundary. In this subsection, we first present the theoretic background of injectivity radius under a continuous setting, and then propose a distributed algorithm for each boundary node to determine its injectivity radius in discrete wireless sensor networks.

1) Injectivity Radius: We first introduce injectivity radius on a continuous smooth surface, denoted by M. The injectivity radius at Point p of M is the largest radius for which the



(a) Geodesic. (b) Exponential map. (c) Injectivity radius.

Fig. 2. Illustration of injectivity radius.

exponential map at p is a diffeomorphism [19]. It is defined according to geodesic, a generalized notion of a "straight line" in "curved spaces". More specifically, the geodesic from Point p to Point q (that are both on the surface) can be mapped to the corresponding tangent line segment as illustrated in Fig. 2(a). The tangent line segment is on the tangent plane of Point p, and starts from Point p along a tangent direction v for a distance equal to t : $p+tv \rightarrow q$. A geodesic circle centering at Point p can be drawn in a similar way on the surface, such that all points on the circle have equal geodesic distance to p. The geodesic circle is mapped to a circle on the tangent plane, and this map is called an exponential map (see Fig. 2(b) for two examples of exponential map at two different points on the surface). Once the geodesic circle grows large and touches itself, the exponential map is no longer 1-to-1 at the touching point, and accordingly becomes non-diffeomorphic (see the exponential map of p_1 in Fig. 2(c)). The injectivity radius at Point p of a smooth surface is defined as the largest radius for which the exponential map at p is a diffeomorphism. Different points on the surface have different injectivity radii, which are determined by the geometric shape of the surface. Clearly, points around the bottleneck areas of the surface have small radii. For example, p_1 has a smaller injectivity radius than p_2 does as shown in Fig. 2(c).

2) Distributed Algorithm to Estimate Injectivity Radius: While the definition of injectivity radius on continuous surface is given above, it remains challenging to calculate it under a discrete sensor network setting, where individual sensors must perform computation in a decentralized manner.

The boundary nodes of a 3D sensor field can be detected by distributed algorithms introduced in [12], and a triangulated boundary surface can be constructed correspondingly according to [16]. An example of the triangulated boundary surface is illustrated in Fig. 1(a). Both algorithms in [12] [16] can work under different communication models, e.g., unit disk graph(UDG), Quasi-UDG, and Log-normal [16]. So is the proposed segmentation algorithm. Note that we do not require Delaunay triangulation. In fact, we do not even need a rigorous triangulation. A CW-complex structure [20], where a face could be any simple n-polygon instead of a triangle, can work effectively for the proposed segmentation algorithm. In addition, a boundary surface is always closed for a 3D sensor field. And given two adjacent boundary nodes on the triangular surface, e.g., Nodes v_i and v_j, there is an edge with double directions connecting them: e_{ij} and e_{ji}, which belong to two neighboring triangular faces f_{ijk} and f_{jil}, respectively.

The proposed algorithm is outlined as follows with six steps and exemplified in Fig. 3. To facilitate our illustration, we "cut open" the triangulation of the 3D surface along Edge e_{nq} and "flatten" it onto 2D. Thus Fig. 3 shows two copies of Node n and Node q, which stand for the same nodes, respectively. One can imagine to bend the 2D illustration and seal it along Edge e_{nq} to reconstruct the original 3D surface.

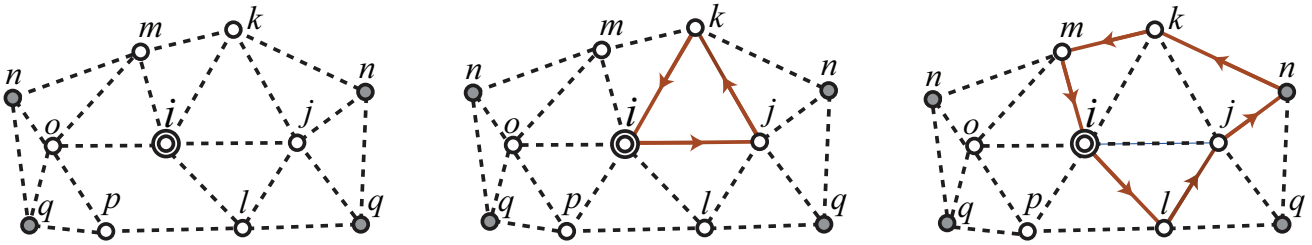
Step (a). Each node, e.g., Node v_i, is associated with a geodesic circle boundary list (GCBL), which is denoted by L_i and initialized as $L_i = \emptyset$ (see Fig. 3(a)). L_i represents an approximated geodesic circle that centers at Node v_i.

Step (b). Node v_i randomly marks a face (e.g., Face f_{ijk}) that contains itself, and updates its GCBL, $L_i = \{e_{ij}, e_{jk}, e_{ki}\}$ as highlighted in brown in Fig. 3(b). Note that when we consider a face, all edges of the face follow the CCW (counter-clockwise) direction.

Step (c). "Glue" the neighbor faces of Face f_{ijk}, if they have not been marked by Node v_i. For example, to glue Face f_{jil}, the edge between Faces f_{ijk} and f_{jil}, i.e., Edge e_{ij}, in L_i is replaced by the other two edges of f_{jil}, i.e., e_{il} and e_{lj}. Similarly, Faces f_{jnk} and f_{kmi} are glued to Face f_{ijk}, resulting in an updated GCBL of $L_i = \{e_{il}, e_{lj}, e_{jn}, e_{nk}, e_{km}, e_{mi}\}$, as shown in Fig. 3(c).

Step (d). Then, L_i is checked to remove any successive edges which connect the same nodes but in opposite directions. For example, after two iterations of gluing by Step (c), we arrive at the results shown in Fig. 3(d), where $L_i = \{e_{ip}, e_{pl}, e_{lq}, e_{qj}, e_{jq}, e_{qn}, e_{nk}, e_{km}, e_{mo}, e_{oi}\}$. Thus Edges e_{qj} and e_{jq} should be removed, yielding Fig. 3(e) where the GCBL is reduced to $L_i = \{e_{ip}, e_{pl}, e_{lq}, e_{qn}, e_{nk}, e_{km}, e_{mo}, e_{oi}\}$. Note that the first edge is considered successive to the last edge in L_i. Moreover, after a pair of edges are removed, L_i is checked again, until no such removable edges exist.

Step (e). Repeat Steps (c) and (d) to grow up the chart centered at Node v_i. The algorithm stops when two edges that connect the same nodes appear in L_i twice but are not successive, indicating that the chart has met itself from two different directions. For example, Edges e_{no} and e_{on} are identified in Fig. 3(f), where $L_i = \{e_{nk}, e_{km}, e_{mn}, e_{no}, e_{op}, e_{pl}, e_{lq}, e_{qo}, e_{on}\}$. By now, L_i represents the approximated geodesic circle that just touches itself.



(a) Triangulation graph. $L_i = \emptyset$. (b) $L_i = \{e_{ij}, e_{jk}, e_{ki}\}$. (c) $L_i = \{e_{il}, e_{lj}, e_{jn}, e_{nk}, e_{km}, e_{mi}\}$.

(d) L_i=$\{e_{ip}, e_{pl}, e_{lq}, e_{qj}, e_{jq}, e_{qn}, e_{nk}, e_{km}, e_{mo}, e_{oi}\}$. (e) $L_i = \{e_{ip}, e_{pl}, e_{lq}, e_{qn}, e_{nk}, e_{km}, e_{mo}, e_{oi}\}$. (f) $L_i = \{e_{nk}, e_{km}, e_{mn}, e_{no}, e_{op}, e_{pl}, e_{lq}, e_{qo}, e_{on}\}$.

Fig. 3. The proposed distributed algorithm for calculation of injectivity radius. To facilitate our illustration, we "cut open" the triangulation of the 3D surface along Edge e_{nq} and "flatten" it onto 2D. Thus each figure shows two copies of Node n and Node q, which stand for the same nodes, respectively.

Step (f). Node v_i can easily find its hop-count distance to each of the nodes involved in L_i. The largest value among such hop-count distances serves as the approximated injectivity radius of Node i, denoted by R_i. For example, in Fig. 3(f), the longest such hop-count distance is 2 (between Nodes v_i and v_q). Thus, $R_i = 2$.

Every boundary node runs the above algorithm by itself to obtain its approximated injectivity radius. Fig. 1(b) gives an example to illustrate the radii of the boundary nodes of a 3D wireless sensor network. It is color-coded, where radii increase as color changes from blue to green to red.

B. Segmentation Based on Injective Radius

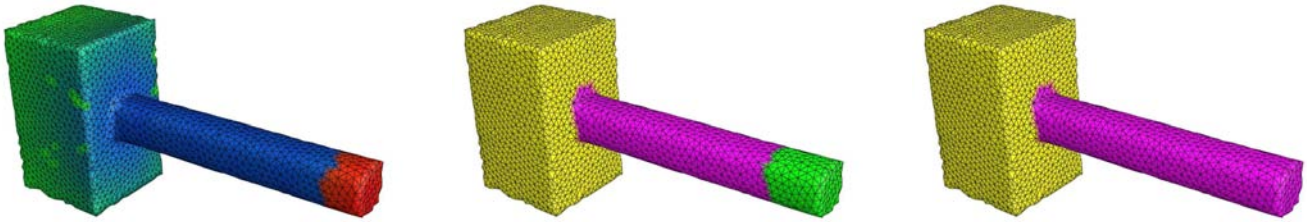
Based on the injectivity radii of boundary nodes, the basic idea for segmentation is to identify a cluster of connected boundary nodes with similarly smallest injectivity radii to form a bottleneck segment. A recursive process is applied to identify a set of such bottlenecks, which together divide the network boundary into segments. An internal non-boundary node simply joins the nearest segment, thus completing the segmentation of the entire 3D sensor network.

Each boundary node, e.g., Node v_i, is associated with three parameters: SID_i, α_i, and β_i. SID_i represents the segment ID of Node v_i. α_i is a boolean variable to indicate whether Node v_i has been marked as a bottleneck or not. β_i is also a boolean variable that signifies if Node v_i should be excluded from further processing by the algorithm as to be elaborated below. The three parameters are initialized as $SID_i = -1$, $\alpha_i = FALSE$, and $\beta_i = FALSE$.

The proposed algorithm then follow the steps outlined below, with the example in Fig. 1 to facilitate our discussion. Step (a) *Identification of minimum radius*. Let $\Phi = \{v_i | \alpha_i = FALSE, \beta_i = FALSE\}$. Note that Φ consists of boundary nodes only. The node in Φ with the minimum radius (denoted by R_{min}) is identified via a controlled flooding process. More specifically, each boundary node in Φ floods its injectivity radius to other boundary nodes on the surface of the network. At the same time, it records the currently known smallest radius (denoted by $\hat{R}_{min}$) according to the packets it receives. If a boundary node receives a flooding packet with a radius greater than $\hat{R}_{min}$, it simply drops it. Clearly, the node with the minimum radius (denoted by v_o) can identify itself when the controlled flooding process terminates. For example, Fig. 1(c) shows v_o highlighted in green.

Step (b) α-*expansion*. Node v_o initiates a segment by setting SID_o as its own ID and $\alpha_o = TRUE$. It then expands the segment by merging any neighboring node whose radius is no greater than $R_{min} + \delta$ into its segment, where δ is a small constant. Such a node, e.g., Node v_i, is *marked* by setting $\alpha_i = TRUE$ and $SID_i = SID_o$. Once Node v_i joins the segment (i.e., is marked), it checks its neighbors and repeats the above process until no nodes can be merged into the segment. Figs. 1(c)-1(f) illustrate the process of α-expansion of a segment, which is highlighted in green.

Step (c) *Identification of segment boundary*. Node v_i with $\alpha_i = TRUE$ is on the boundary of its segment if it has at least one neighbor that is not marked as a node in the segment (e.g., see the green nodes that are next to the blue area in Fig. 1(f)). A trivial signaling protocol can be devised to identify such segment boundaries and the average radius of the nodes on each segment boundary. If a segment has only one boundary, it must be at a tip of the network, and doesn't need to be considered as a separate segment for now. As a result, a node in such a segment, e.g., Node v_i, simply sets $SID_i = -1$, $\alpha_i = FALSE$, and $\beta_i = TRUE$. Having $\beta_i = TRUE$ is to keep it



(a) Color-coded radii. (b) Before refinement. (c) After refinement.

Fig. 4. Refinement of segments. (a) The color-coded radii, where the values of radii increase as color changes from blue to green to red. (b) Before refinement, three segments are colored in yellow, purple and green, respectively. (c) Two segments remain after refinement.

from being reselected again in the next round of the algorithm, aiming to identify another bottleneck.

Step (d) β-*expansion*. If a segment has two boundaries (e.g., the segment shown in green in Fig. 1(f)), the nodes on each segment boundary perform a hop-by-hop β-expansion. More specifically, assume Node v_i is on a segment boundary with an average radius of $\bar{R}$. If Node v_i (with $\alpha_i = TRUE$) has a neighbor Node v_j with $\alpha_j = FALSE$ and $R_j > \bar{R}$, then Node v_j sets $\beta_j = TRUE$. After all nodes on the segment boundary have completed such one-hop expansion, the newly identified nodes with $\beta_j = TRUE$ are treated as new boundary nodes. The average radius (i.e., $\bar{R}$) is updated, and the above process repeats, until no further expansion is possible (e.g., because there is no neighboring node whose radius is greater than $\bar{R}$). The result of this step is illustrated in Fig. 1(g), where the red areas indicated the nodes with $\beta_j = TRUE$ after β-expansion.

Step (e) *Assignment of segment ID*. Steps (a)-(d) repeats based on updated Φ, until $\Phi = \emptyset$. Till now, Node v_i has either $\alpha_i = TRUE$ (e.g., a node highlighted in green in Fig. 1(h)) or $\alpha_i = FALSE$ and $\beta_i = TRUE$ (such as a node highlighted in red in Fig. 1(h)). The former has already been assigned to a segment, with a known segment ID. For the latter, its segment is yet to be determined. To this end, Node v_i with $\alpha_i = FALSE$ temporarily sets its node ID as its segment ID, and sends it to other nodes via controlled flooding similar to what we have discussed in Step (a). Such a flooding packet is dropped by any marked node (i.e., a node with $\alpha_i = TRUE$), and thus limited within a connected set of nodes with $\alpha_i = FALSE$ only. Each node in this set records the lowest node ID as its segment ID. Fig. 1(i) shows three segments yielded by the algorithm, which are colored in red, yellow and green, respectively.

Step (f) *Segmentation of internal nodes*. Till now, the network boundary has been segmented. An internal node simply assigns itself to its nearest segment on the boundary surface. In other words, it finds the nearest boundary node, and joins the segment of the latter.

C. Time Complexity and Communication Cost

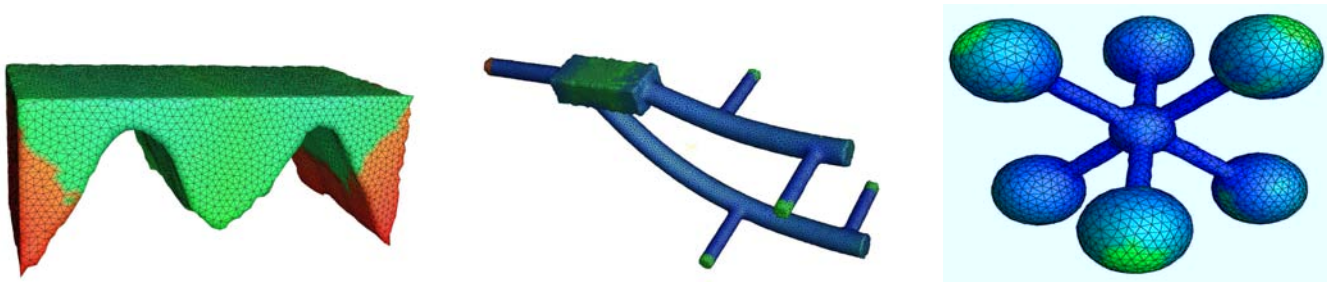
The proposed segmentation algorithm has a linear time complexity and communication cost (measured by messages sent) with respect to the size of the network. More specifically, the boundary nodes are detected by the algorithm introduced in [12] with a complexity of $O(k^3)$ and communication cost of $O(k)$, where k is the average nodal degree (i.e., the average number of neighbors per node). The triangulation mesh of the boundary can be constructed with a complexity of $O(m)$ and communication cost of $O(m^2)$, where m is the number of boundary nodes [16]. Since the injectivity radius is calculated by individual nodes on the boundary in a decentralized manner, it introduces a complexity of $O(m)$ and communication cost of $O(m^2)$. Finally, the time complexity and communication cost of the algorithm for segmentation and refinement are both $O(n)$, where n a total number of nodes in the network. Given $k \ll m \ll n$, the overall time complexity and communication cost of the segmentation algorithm are dominated by $O(n)$.

D. Further Discussions

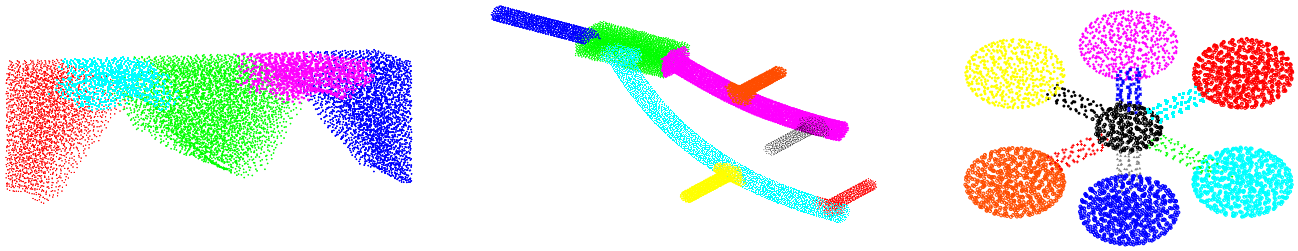
We have successfully applied the proposed algorithm to various network models to demonstrate its effectiveness (see Sec. III for examples). In this subsection, we discuss several observations gained from our implementation, which are insightful and lead to improvement of segmentation results.

1) Noise in Estimated Injectivity Radius: The performance of segmentation highly depends on the accuracy of injectivity radii. The injectivity radius can be accurately calculated for each point on a continuous smooth surface. Under a discrete setting, however, it becomes extremely challenging to obtain a precise injectivity radius because a geodesic between two nodes can only be approximately measured by hop counts along their shortest path. As illustrated in Fig. 3, the final L_i is not a perfect circle that centers at Node v_i, and thus the injectivity radii are inaccurate and discontinuous. In general, higher accuracy of injectivity radii can be achieved under a denser and more uniform triangulation.

2) Noise Filtering: Based on the above observation, we propose two techniques to filter out the noise in injectivity radii. First we introduce a small constant δ in α-expansion (i.e., Step (b) of the segmentation algorithm). In contrast to a continuous surface where two close points always have similar radii, the radii of two neighboring nodes in a sensor network may be very different. δ is employed to cope with such discontinuity. As a rule of thumb, we set $\delta = 0.3R_{min}$ with bounds of $2 \leq \delta \leq 5$.



(a) Model 1: seabed (color-coded radii). (b) Model 2: coalmine tunnel (color-coded radii). (c) Model 3: air port (color-coded radii).



(d) Model 1: seabed (5 segments). (e) Model 2: coalmine tunnel (8 segments). (f) Model 3: air port (12 segments).

Fig. 5. Examples of segmentation (Part I). Models 1-3 depict an underwater sensor network above undersea mountains, an underground sensor network in coalmine tunnels, and a sensor network deployed at a modern airport, respectively. (a)-(c) The color-coded radii, where the values of radii increase as color changes from blue to green to red. (d)-(f) Segmentation results, where a color indicates a segment.

Second, due to possible noise in radius calculation, a node may have very different radius in comparison with its neighbors. Thus a post-processing is employed after α-expansion. More specifically, if $\alpha_i = FALSE$, but Node v_i finds all of its neighbors are marked, it sets $\alpha_i = TRUE$ and joins the segment, to filter out such noise. Similarly, if Node v_o fails to merge any neighboring nodes into its segment, it unmarks itself by setting $\alpha_o = FALSE$.

3) Refinement of Segments: A segment is merged with a neighboring segment if it consists of a very small set of nodes and has only one boundary (i.e., has only one neighboring segment). This scheme effectively avoids creating a small segment at a tip of a network boundary, where a node usually exhibits a long injectivity radius because the geodesic circle may grow unusually large before it touches itself. For example, the nodes in the red part in Fig. 4(a) have large radii. By applying Steps (a)-(f) of the segmentation algorithm to this network, we arrive at three segments illustrated in Fig. 4(b). The segment colored in green in Fig. 4(b) is clearly unwanted. Since it has only one boundary and consists of a small set of nodes (in comparison with its neighboring segment), it is merged with the neighboring segment, yielding the final results with two segments as shown in Fig. 4(c).

III. Segmentation Results and Applications

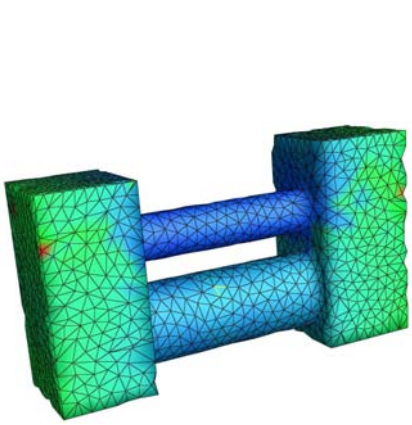
To evaluate our proposed algorithm, we have applied it in various network models as illustrated in Figs. 5 and 6. As can be seen, the algorithm can always locate the bottlenecks and accordingly partition the network into segments under desired simple boundary condition without bottlenecks.

Segmentation effectively regulates network shape, and thus can improve the performance of a range of applications, from routing to backbone construction as discussed in Sec. I. In this section, we introduce just two of such applications to demonstrate the effectiveness of the segments obtained by our proposed algorithm.

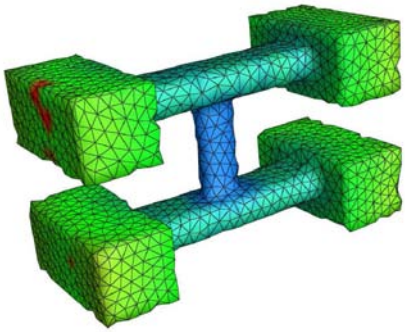
A. Segment-Based Routing

Greedy routing is a promising technology to achieve scalable data communication in wireless sensor networks. However it does not guarantee delivery under complex (especially concave) boundary conditions. In a 2D network, face-routing or its alternatives can be employed to recover greedy routing failures [13], [14]. However, it has been proven that there is no deterministic solution that can ensure successful data delivery based on local information only in 3D networks [15]. For example, greedy routing fails to deliver data packets from a source node located at an upper corner of the left-side segment in Fig. 1(i) (i.e., the segment colored in red) to a destination at an upper corner of the right-side segment (colored in yellow), because of local minimum, i.e., a node that is not the destination but closer to the destination than all of its neighbors.

To address this problem, we exploit our segmentation results, where the network is divided into multiple segments with simple boundary conditions. A graph is established, where each vertex represents a segment and two vertices are connected by an edge if the corresponding segments are adjacent. Subsequently, the Dijkstra algorithm is applied based on the graph to create a routing table, which consists of a list

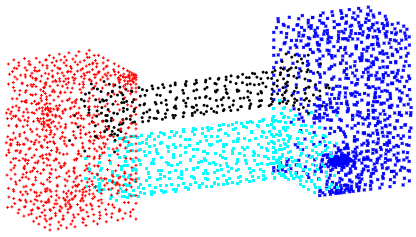


(a) Model 4 (color-coded radii).

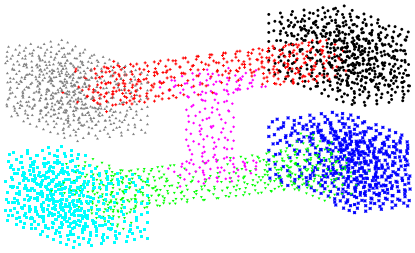


(b) Model 5 (color-coded radii).

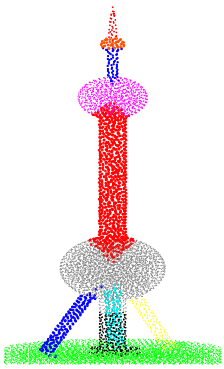
(c) Model 6 (color-coded radii).



(d) Model 4 (4 segments).



(e) Model 5 (7 segments).



(f) Model 6 (11 segments).

Fig. 6. Examples of segmentation (Part II). The color codes are the same as Fig. 5.

of segments and the next hops to reach them. Such a table has a small size bounded by the number of segments, and thus can be made available to all sensor nodes in the network.

Each sensor node is associated with its node ID and physical or virtual coordinates similar to the setting under any typical greedy routing protocol. In addition, it maintains its segment ID and a gateway to each neighboring segment, where the latter is the closest or a randomly chosen node on the shared boundary of the two adjacent segments. Note that the local information maintained by a sensor is bounded by a small constant and obtained with limited signaling overhead during network initialization only.

If both source and destination are within the same segment, greedy routing is applied. Otherwise, the routing across segments relies on the established routing table. More specifically, the source node looks up the routing table to find the next segment toward the destination. Then the data packet is greedily routed toward the gateway to the next segment. Whenever it enters the next segment, the above process repeats, until it researches the segment of the destination, where greedy routing is applied to deliver the packet.

We have simulated the segment-based routing and evaluated its performance based on the 3D sensor networks illustrated in Figs. 5 and 6. As shown in Table I, segmentation significantly improves routing success rate, because it eliminates bottlenecks and consequently reduces the undesired dead-ends in greedy routing, especially under the networks with complex, concave shapes where greedy routing between most segments clearly fails.

It is interesting to observe that the average path length is longer under segment-based routing. This is because greedy routing without segmentation often fails for long routes across segments. This phenomenon is evident from Fig. 7 that shows the success rate of routing paths with different lengths. The segment-based routing achieves a perfect delivery rate. On the other hand, the greedy routing without segmentation only well support short paths. With the increase of path length (i.e., when the source and destination become farther away from each other), the delivery rate of greedy routing without segmentation dramatically decreases to as low as 20%.

Finally, Fig. 8 illustrates the distribution of traffic load. Here we assume one unit of traffic is sent between a pair of nodes. Since the segment-based routing enables more routes through the bottlenecks, it is natural that the nodes at bottleneck areas

TABLE I
COMPARISON OF ROUTING SUCCESS RATE.

	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6	Overall
With segmentation	100%	86%	100%	95.28%	99.92%	99.96%	99.72%
Without segmentation	96.07%	71.43%	57.69%	92.44%	62.78%	88.92%	86.69%

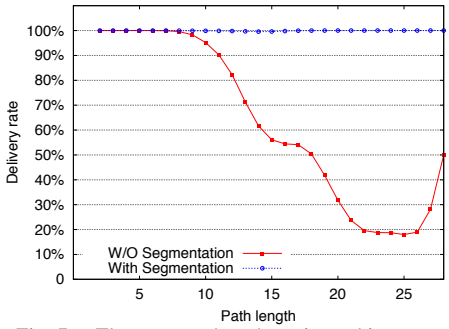


Fig. 7. The segment-based routing achieves a perfect delivery rate, while the greedy routing without segmentation only support short paths (Model 5).

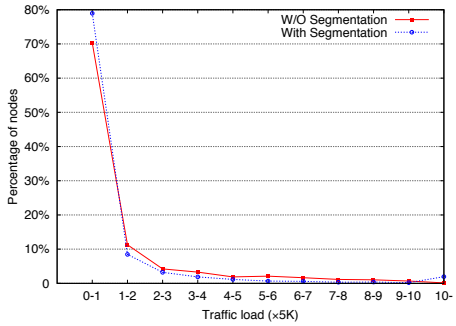


Fig. 8. Routing load distribution (Model 3). The segment-based routing results in similar load distribution as the greedy routing without segmentation.

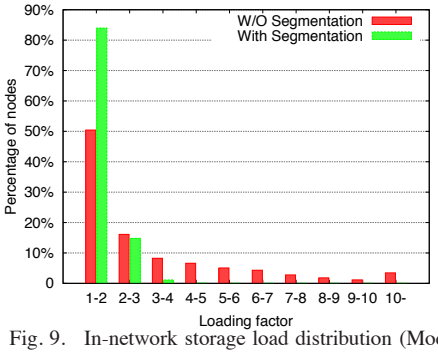


Fig. 9. In-network storage load distribution (Model 3), where segmentation leads to nicely distributed loading factor around one, i.e., the ideal load.

must carry more traffic. However, note that although gateways are employed to support routing across segments, the routing paths do not have to pass through the gateways. Instead, a packet is always routed toward the gateway. Whenever it approaches the boundary between two segments, it is forwarded to the nearest node in the next segment, without actually going through the gateway node. Therefore, the segment-based routing results in similar load distribution as the greedy routing without segmentation.

B. Segment-Based Bounding in In-network Data Centric Storage and Retrieval

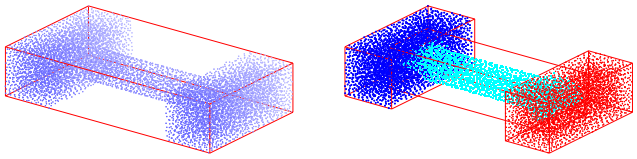
While it is assumed in most sensor networks that data are collected by sensors and transmitted to sinks for storage and further processing, in-network data centric storage has been investigated in the literature [21]–[23], aiming to reduce energy consumed for communication and to establish a self-contained data acquisition, storage and retrieval sensor system. In such systems, data are consistently mapped to some coordinates via a locality-preserving geographic hash function, which allows efficient retrieval of data. An underlying geographic routing protocol (e.g., [13], [14]) is employed to route data and query packets to their corresponding nodes.

To perform geographic hashing, however, a bounding box of the network must be identified to define the range of coordinates to be used by the hash function. For example, in a 2D sensor network, we usually use the smallest rectangle that contains the network as its bounding box. Similarly, the smallest hexahedron can be identified as the bounding box for a 3D sensor network. Of course, other regular shapes of bounding boxes can be employed too, in order to make the closest match with the actual shape of the network.

A datum is hashed to a location in either 2D or 3D space within the bounding box, and stored by the node that is closest to the location. Clearly, if the network has an irregular shape, a significant part of the space in the bounding box is "empty", i.e., unoccupied by any sensors (see Fig. 10(a) for example). As a result, the nearby boundary nodes must store the data hashed to such empty space, thus experiencing high storage load and communication overhead (for both data storage and retrieval).

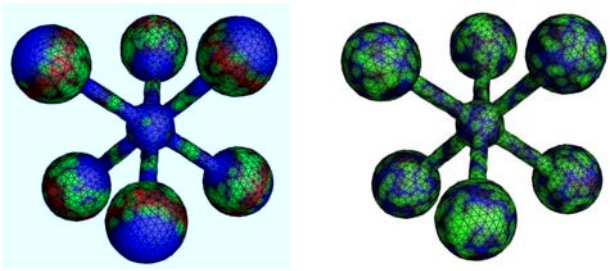
To this end, we exploit the segmentation results, to employ multiple bounding boxes with one for each segment (as shown in Fig. 10(b)). While hexahedra are adopted in our simulation for simplicity, more sophisticated shapes can be used in practice, as long as they are "regular", such that geographic hashing can be applied. But note that, it is extremely challenging, if not impossible, to construct a single bounding box with regular shape to tightly fit a whole network with complex boundary conditions (e.g., the networks shown in Figs. 5-6).

We have carried out simulations to demonstrate the effectiveness of segment-based bounding in in-network data centric storage and retrieval. An example is illustrated in Fig. 11 to visually compare the storage load with or without segmentation, which is color-coded with the load increasing as color changing from blue to red. Here we only show the nodes on the boundary of the network, which carry heavier load than internal nodes do. As can be seen, the results with segmentation (i.e., Figs. 11(b)) are less reddish, exhibiting lower storage load compared with the results without segmentation (i.e., Figs. 11(a)). A quantitative comparison of load distribution is given in Fig. 9, where the x-axis shows the loading factor that is defined as follows. Let the total load of the entire network be normalized as 1. The ideal load distribution is $1/N$ on every node, where N is the number of nodes in the network. The loading factor of a node is the ratio of its actual load to the ideal load. To illustrate the nodes that experience high load, Fig. 9 shows the distribution of the nodes with loading factor greater than one only. As can be seen, the loading factor is widely spread under the case without segmentation, indicating uneven distribution of load. Some nodes even suffer a loading factor of 10 or higher. In a sharp contrast, the segment-based



(a) Simple bounding. (b) Segment-based bounding.

Fig. 10. (a) In the network with an irregular shape, a significant part of the space in the simple bounding box is "empty", i.e., unoccupied by any sensors. (b) Segment-based bounding reduces "empty" space, and consequently lowers the storage and communication load at the boundary nodes.



(a) Model 3 without segmentation. (b) Model 3 with segmentation.

Fig. 11. Load distribution in in-network storage. The storage load is color-coded, where the load increases as color changes from blue to red.

approach leads to nicely distributed loading factor around one, i.e., the ideal load.

IV. Conclusion and Future Work

In this paper, we have proposed the first effective solution to identify bottleneck areas in a 3D wireless sensor network and segment it into a set of subnetworks, each under a desired simple boundary condition without bottlenecks. It is fully distributed and based on network connectivity only. Each node on boundary computes its injectivity radius, which reflects the narrowness of the corresponding boundary area. Then a set of bottlenecks are identified based on injectivity radii, which together divide the network into segments. Our simulation has shown that the proposed algorithm works efficiently under various sensor networks with different boundary conditions and noise levels, always yielding appropriate segmentation results. We have further demonstrated that segmentation can effectively promote the performance of routing and in-network data centric storage in 3D wireless sensor networks.

We have focused on solid 3D networks (i.e., without internal holes) in this paper. The network models with holes will be studied in our future work.

References

[1] X. Bai, C. Zhang, D. Xuan, J. Teng, and W. Jia, "Low-Connectivity and Full-Coverage Three Dimensional Networks," in *Proc. of ACM International Symposium on Mobile Ad hoc Networking and Computing (MobiHOC)*, pp. 145–154, 2009.

[2] X. Bai, C. Zhang, D. Xuan, and W. Jia, "Full-Coverage and K-Connectivity (K=14, 6) Three Dimensional Networks," in *Proc. of IEEE Int'l Conference on Computer Communications (INFOCOM)*, 2009.

[3] C. Liu and J. Wu, "Efficient Geometric Routing in Three Dimensional Ad Hoc Networks," in *Proc. of IEEE International Conference on Computer Communications (INFOCOM)*, 2009.

[4] T. F. G. Kao and J. Opatmy, "Position-Based Routing on 3D Geometric Graphs in Mobile Ad Hoc Networks," in *Proc. of The 17th Canadian Conference on Computational Geometry*, pp. 88–91, 2005.

[5] J. Opatrny, A. Abdallah, and T. Fevens, "Randomized 3D Position-based Routing Algorithms for Ad-hoc Networks," in *Proc. of Third Annual International Conference onMobile and Ubiquitous Systems: Networking & Services*, pp. 1–8, 2006.

[6] R. Flury and R. Wattenhofer, "Randomized 3D Geographic Routing," in *Proc. of IEEE International Conference on Computer Communications (INFOCOM)*, pp. 834–842, 2008.

[7] F. Li, S. Chen, Y. Wang, and J. Chen, "Load Balancing Routing in Three Dimensional Wireless Networks," in *Proc. of IEEE International Conference on Communications (ICC)*, pp. 3073–3077, 2008.

[8] D. Pompili, T. Melodia, and I. F. Akyildiz, "Routing Algorithms for Delay-insensitive and Delay-sensitive Applications in Underwater Sensor Networks," in *Proc. of The ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pp. 298–309, 2006.

[9] W. Cheng, A. Y. Teymorian, L. Ma, X. Cheng, X. Lu, and Z. Lu, "Underwater Localization in Sparse 3D Acoustic Sensor Networks," in *Proc. of IEEE International Conference on Computer Communications (INFOCOM)*, pp. 798–806, 2008.

[10] J. Allred, A. B. Hasan, S. Panichsakul, W. Pisano, P. Gray, J. Huang, R. Han, D. Lawrence, and K. Mohseni, "SensorFlock: An Airborne Wireless Sensor Network of Micro-Air Vehicles," in *Proc. of The International Conference on Embedded Networked Sensor Systems (SenSys)*, pp. 117–129, 2007.

[11] J.-H. Cui, J. Kong, M. Gerla, and S. Zhou, "Challenges: Building Scalable Mobile Underwater Wireless Sensor Networks for Aquatic Applications," *IEEE Network, Special Issue on Wireless Sensor Networking*, vol. 20, no. 3, pp. 12–18, 2006.

[12] H. Zhou, S. Xia, M. Jin, and H. Wu, "Localized Algorithm for Precise Boundary Detection in 3D Wireless Networks," in *Proc. of The 30th International Conference on Distributed Computing Systems (ICDCS)*, pp. 744–753, 2010.

[13] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia, "Routing with Guaranteed Delivery in Ad Hoc Wireless Networks," in *Proc. of Third Workshop Discrete Algorithms and Methods for Mobile Computing and Communications*, pp. 48–55, 1999.

[14] B. Karp and H. Kung, "GPSR: Greedy Perimeter Stateless Routing for Wireless networks," in *Proc. of The ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pp. 1–12, 2001.

[15] S. Durocher, D. Kirkpatrick, and L. Narayanan, "On Routing with Guaranteed Delivery in Three-Dimensional Ad Hoc Wireless Networks," in *Proc. of International Conference on Distributed Computing and Networking*, pp. 546–557, 2008.

[16] H. Zhou, H. Wu, S. Xia, M. Jin, and N. Ding, "A Distributed Triangulation Algorithm for Wireless Sensor Networks on 2D and 3D Surface," in *Proc. of Annual IEEE Conference on Computer Communications (INFOCOM)*, 2011.

[17] X. Zhu, R. Sarkar, and J. Gao, "Shape segmentation and applications in sensor networks," in *Proc. of Annual IEEE Conference on Computer Communications (INFOCOM)*, pp. 1838–1846, 2007.

[18] T. K. Dey, J. Giesen, and S. Goswami, "Shape Segmentation and Matching with Flow Discretization," in *In Proc. Workshop on Algorithms and Data Structures*, pp. 25–36, 2003.

[19] K. Grove and P. Petersen, *Comparison Geometry*. Cambridge University Press, 1997.

[20] M. Henle, *A Combinatorial Introduction to Topology*. Dover Publications, 1994.

[21] J. Li, J. Jannotti, D. S. J. De Couto, D. R. Karger, and R. Morris, "A Scalable Location Service for Geographic Ad Hoc Routing," in *Proc. of The Annual International Conference on Mobile Computing and Networking (MobiCom)*, pp. 120–130, 2000.

[22] X. Li, Y. J. Kim, R. Govindan, and W. Hong, "Multi-dimensional Range Queries in Sensor Networks," in *Proc. of the 1st International Conference on Embedded Networked Sensor Systems (SenSys)*, pp. 63–75, 2003.

[23] C. Yu-Chi, S. I-Fang, and L. Chiang, "Supporting Multi-Dimensional Range Query for Sensor Networks," in *Proc. of International Conference on Distributed Computing Systems (ICDCS)*, pp. 35–35, 2007.