Efficient Geometric Routing in Three Dimensional Ad Hoc Networks

Cong Liu and Jie Wu
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431

***Abstract*—Efficient geometric routing algorithms have been studied extensively for two-dimensional ad hoc networks, or simply, 2D networks. They are not only efficient but have also been proven to be worst-case optimal localized routing algorithms. However, few prior works have focused on efficient geometric routing in 3D networks due to the lack of an efficient method to bound the search once the greedy routing algorithm encounters a local minimum, like face routing in 2D networks. In this paper, we tackle the problem of efficient geometric routing in 3D networks. We propose routing on hulls, a 3D analogue to face routing, and present the first 3D partial unit Delaunay triangulation (PUDT) algorithm to divide the entire network space into a number of closed subspaces. Our proposed greedy-hull-greedy (GHG) routing is efficient because it limits the local-minimum recovery process on the hull of only one of these subspaces. Simulation results verify the efficiency of our proposed algorithms.**

***Index Terms*—Delaunay triangulation, geometric routing, ad hoc networks, three-dimensional (3D) networks.**

I. Introduction

In this paper, we focus on efficient geometric routing algorithms for three-dimensional ad hoc networks, or simply, 3D networks. Exploiting the geometry of the network to perform routing is a commonly-used approach for overcoming the challenges posed by resource-limited ad hoc networks. An important property of geometric routing algorithms is that they are based on local information which can easily be refreshed to reflect unavoidable topology changes in mobile networks.

Most geometric routing protocols start with greedy forwarding, which is simple and close to optimal. In greedy forwarding, each node knows the positions of its neighbors, and the node forwards each message to the neighbor that is the closest to the message's destination. However, greedy forwarding is not always successful: it fails when a message reaches a *local minimum* node whose neighbors are all further away from the destination than the node itself. For 2D networks, face routing is the most prominent solution to recover from local-minima. In the greedy-face-greedy (GFG) approach and its variants [1], [2], [3], [4], and [5], a message stuck in a local minimum is routed along the face intersecting the source-destination line (i.e. the empty network space between the local-minimum and the destination) until it finds a node that is closer to the destination than the local minimum, at which point it continues with greedy forwarding. [4] and [5] show that geometric routing protocols are not only efficient in the average case; they are also worst-case optimal localized

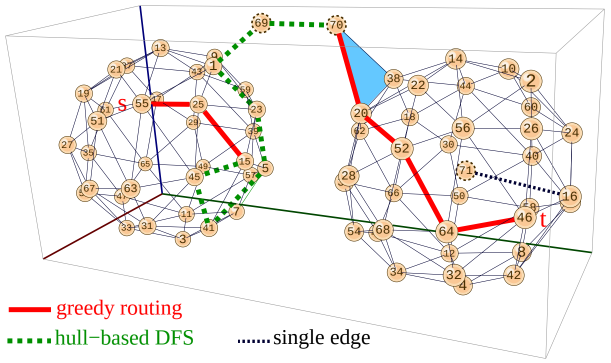


Fig. 1. An example of greedy-hull-greedy routing.

routing protocols with the hop-counts of the resulting paths bounded by $O(d^2)$, where d is the distance between the source and destination.

However, the detection of the face requires a planarized network graph which can only be constructed in 2D networks. Localized planar graph constructions in 2D networks include relative neighbor graph (RNG) [6], Gabriel graph (GG) [7], and Delaunay triangulation (DT) [8], [9]. Few previous research works have attempted to discover a similar structure in 3D networks. In [10], Flury and Wattenhofer used virtual cubes to capture the surface of holes, and they used random walk to recover from local minima. Their work proved the optimal worst-case bound $O(d^3)$ for localized routing algorithms and provided the first feasible solution for geometric routing in 3D. However, the virtual cube approach is expensive (requiring 3-hops of information), and the random walk is inefficient.

In this paper, we propose a low-cost, localized 3D *partial unit Delaunay triangulation* (PUDT) algorithm for capturing the empty 3D network subspaces in order to perform an efficient local-minimum recovery search. Our PUDT algorithm requires just over 1-hop of information.

Local recognition of the network subspaces and the nodes in each subspace is the main challenge. In this paper, (1) we define a triangle as a small plane delimited by the three edges of three connected nodes; (2) we divide the entire network space into a number of subspaces delimited by these triangles; (3) analogous to the planar graph construction process which removes intersecting edges, we use a PUDT algorithm which removes intersecting triangles (more precisely, an intersecting triangle and edge); and (4) we define the *hull* of a subspace as a structure that contains the triangles shared by the adjacent subspace and the *single edges* (none of which belong to any triangle), and we devise localized algorithms to recognize hulls and identify the triangles and single edges belonging to each hull.

Analogous to face routing, once the message reaches a local-minimum, we use hull-based routing to constrain the local-minimum recovery search on the hull of a particular subspace instead of wandering aimlessly throughout the entire network. It switches to greedy mode when a node that is closer to the destination than the local-minimum is reached. Our *greedy-hull-greedy* (GHG) routing contains this hull-based routing and is shown to be efficient through simulation.

An example routing process of our GHG protocol is shown in Figure 1, where greedy routing sends the message from node 55 (source) to 15. On local-minimum 15, hull-based routing sends the message from nodes 15 to 70 where greedy can be continued (since 70 is closer to 46 than 15). Finally, greedy routing sends the message from 70 to 46 (destination).

Our contributions in this paper are summarized as follows:

- We first propose to use partial unit Delaunay triangulation (PUDT) to define network hulls in 3D networks.
- We present the first localized PUDT algorithm and a low-cost local hull construction algorithm.
- We devise a 3D geometric routing protocol, greedy-hull-greedy (GHG), which efficiently recovers from local-minima on a target hull.
- We perform simulations to show that the overhead of our low-cost PUDT is just over 1-hop of information, and that our GHG protocol is efficient.

This paper is organized as follows. Section II introduces some preliminaries and reviews related works. Section III covers the main ideas in this paper. Section IV proposes the low-cost PUDT algorithm. Section V presents the localized hull construction and target hull selection. Section VI illustrates the elements in the GHG routing algorithm. Simulation evaluations are shown in Section VII. Finally, Section VIII concludes the paper with future research directions.

II. Preliminaries & Related Works

A. Geometric routing

This paper considers an ad hoc network with all nodes distributed in a 3D space. Following the traditional line of research, all wireless nodes have distinctive identities, each wireless node knows its location information (i.e. through a GPS receiver), and all wireless nodes have the same transmission range which is normalized to one unit. Consequently, all wireless nodes (V) together define a unit-disk graph $UDG(V)$ in a 2D network or a *unit-ball graph* $UBG(V)$ in a 3D network. Several planar network topologies are proposed, including RNG [6], GG [7], and DT [8], [9], which can be used as the underlying structure for *face routing*.

In [11], a depth-first-search geometric routing protocol is proposed, which is also applicable in 3D networks. In this algorithm, messages only need to keep $O(1)$ routing state information; all other information is stored in the nodes.

Other geometric routing protocols that can be extended to run in 3D networks include those that use virtual coordinates and those that rely on virtual global structures [12], [13], [14], and [15]. The problem with all of these approaches is that they require global operations on the network. Other regionized approaches to improve routing performance and tolerate location errors include [16], [17], and [18].

B. Localized planar graphs in 2D networks

A planar graph construction is localized if every wireless node u can determine the edges of G incident on u using only the information of the nodes within a constant hop. The relative neighborhood graph [6], denoted by $RNG(V)$, consists of all edges uv such that $\|uv\| < 1$ and there is no point w such that $\|uw\| < \|uv\|$ and $\|wv\| < \|uv\|$. Let $disk(u, v)$ be the closed disk with diameter uv. The Gabriel graph [7], denoted by $GG(V)$, consists of all edges uv such that $\|uv\| < 1$ and the interior of $disk(u, v)$ does not contain any other node w.

Delaunay triangulation $DT(V)$ [8], [9] for a set V of points in the plane is a triangulation such that the circumcircle of a triangle in $DT(V)$ formed by three points in V does not contain vertices other than the three that define it. In d-dimensional Euclidean space, a Delaunay triangulation is a triangulation $DT(V)$ such that no point in V is inside the circum-hypersphere of any d-simplex in $DT(V)$. Here, a d-simplex is the d-dimensional analogue of a triangle. We are interested in 3-dimensional spaces where the 3-simplex is a tetrahedron. In Euclidean space, the Delaunay triangulation of V corresponds to the dual graph of the Voronoi tessellation for V.

A unique Delaunay triangulation (DT) exists if V is a set of points in general position. That is, no three points are on the same line and no four are in the same circle for a set of points on a 2D plane, or, no four points are on the same tetrahedron and no five points are in the same sphere for a 3-dimensional set of points. To simplify the proofs, from now on we assume that V is a set of points in general position. Otherwise, a very small random perturbation to their coordinates allows the assumption above without causing any problems in the actual network. *Unit Delaunay triangulation* (UDT) differs from DT in that UDT only contains the edges which are shorter than one. *Partial unit Delaunay triangulation* ($PUDT$) differs from UDT in that $PUDT$ might contain extra edges and fewer triangles to guarantee routing delivery.

III. Our Proposed Approach

Our proposed geometric routing protocol starts with greedy forwarding. Once the message is forwarded to a local-minimum, a recovery process is started by searching nodes in the subspace containing the segment between the local-minimum and destination. Once the routing algorithm is recovered from the local minimum (by forwarding the message to a node that is closer to the destination than the local minimum), it continues with greedy forwarding.

There are several challenges in this paper. The first challenge concerns dividing the network space into subspaces so that we can limit the local-minimum recovery search in one of these subspaces to improve recovery efficiency. In 2D networks, faces are divided by non-crossing edges. Local-minimum recovery consists of searching along consecutive edges bordering a particular face. We first propose to use triangles consisting of three connected nodes to divide the 3D network space into subspaces. The entire network space is divided by triangles into one outer subspace and a number of inner subspaces. In Figure 1, the network consists of two spherical inner subspaces and one outer subspace which fills the rest of the entire space.

The second challenge concerns removing intersecting triangles. In 2D planarization, crossing edges need to be removed such that the network plane can be divided into faces. Similarly, non-overlapping subspaces cannot be divided with intersecting triangles. We propose a low-cost, localized PUDT algorithm in Section IV to remove intersecting triangles which uses just over 1-hop of information.

The third challenge concerns identifying nodes in different subspaces since our local minimum recovery search is limited to the nodes within a particular subspace. This essentially consists of grouping the triangles and edges into different subspaces. We define a hull for a subspace as a structure which contains the triangles bordering the subspace and the triangles and single edges (edges not belonging to any triangle) inside the subspace. We propose a local hull construction in Section V in which each node locally groups its triangles and single edges into different hulls. When distinguishing each triangle by two sides, each triangle and single edge must belong to exactly one hull.

In Figure 1, the hull of the right-side inner subspace contains all of the triangles on the right-side ball and the single $edge(p_{16}, p_{71})$ inside the ball. Some triangles have their two sides belonging to different hulls, and others have both sides belonging to the same hull. For example, both sides of the blue triangle belong to the outer hull (for the outer subspace). Each single edge belongs to only one hull.

The last two important challenges deal with finding (Section V) and searching (Section VI) the target hull once the message is in a local-minimum. For a particular destination, a target hull is one of the local hulls of the local minimum which contains all or part of the segment between the local-minimum and the destination. Recovery from a local-minimum is guaranteed when searching the target hull. For enhanced performance, we propose a hull-based connected dominating set (CDS) in Section VI, to further limit the node in the local minimum recovery search.

IV. Partial Unit Delaunay Triangulation (PUDT)

In a PUDT for 3D networks, intersecting triangles and edges are logically removed such that the entire network space can be divided into a number of subspaces.

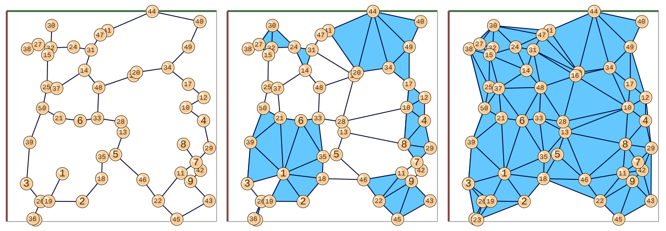

(a) RNG (b) GG (c) PUDT

Fig. 2. Triangles in RNG, GG, and PUDT.

A. The basic approach and its correctness

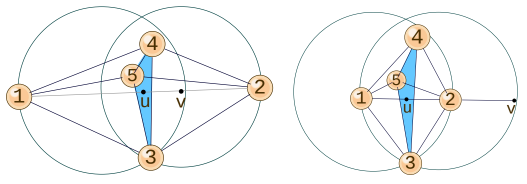
GG and RNG are more popular than PUDT in 2D networks because they are simpler. However, in 3D networks, PUDT is the only feasible choice among the three because GG and RNG remove most of the triangles. As can be observed in the random network in Figure 2, PUDT conserves as many triangles as possible while GG and RNG destroy most triangles because they remove excessive edges. It can be proved that RNG $\subseteq$ GG $\subseteq$ PUDT in terms of triangles as well as edges.

We denote nodes as $p_1, p_2, \ldots$, an edge between nodes p_1 and p_2 as $edge(p_1, p_2)$, a triangle determined by three points not in a line as $\Delta(p_1, p_2, p_3)$, a tetrahedron as $T(p_1, p_2, p_3, p_4)$, and a ball determined by four points not on the same plane as $ball(p_1, p_2, p_3, p_4)$. Note that $ball(p_1, p_2, p_3, p_4)$ is the circumsphere of $T(p_1, p_2, p_3, p_4)$.

When the edges can be arbitrarily long, the result of applying Delaunay triangulation to a set of vertexes is a space uniquely divided into a number of non-intersecting tetrahedra and a single outer subspace. The rule is that only the tetrahedra without a fifth vertex inside its circumsphere is a valid Delaunay tetrahedron. Two tetrahedra are intersecting if there is a common point inside them. For example, in Figure 3(a), $T(p_1, p_3, p_4, p_5)$ and $T(p_1, p_2, p_3, p_4)$ are intersecting, and Delaunay triangulation ensures that only one of them is valid. In this example, $T(p_1, p_2, p_3, p_4)$ is not valid since p_5 is inside $ball(p_1, p_2, p_3, p_4)$.

The PUDT algorithm in 3D networks is analogous to planar graph construction in 2D networks: the latter is used to remove intersecting edges, while the former is used to remove intersecting triangles. It can be proved that if there is no intersecting edge and triangle, then there is no intersecting tetrahedra. This is because when two tetrahedra intersect, one of the four triangles on the first tetrahedron must intersect a triangle on the second tetrahedron; moreover if two triangles intersect, an edge of one of the triangles must intersect the other triangle. When the network is very dense, the non-removed triangles partition the network space into a number of tetrahedra. Otherwise, there are some irregular polyhedra.

Our basic PUDT algorithm logically removes the edges and triangles, which are defined in Definition 1, and it is illustrated in Figures 3(a) and 3(b). Definition 1 guarantees that an edge and a triangle cannot exist at the same time if they intersect. Note that we define edges and triangles as two kinds of objects. Removing any of the three edges of a triangle



(a) Case 1 (b) Case 2

Fig. 3. Illustrations for Theorem 1, where i denotes p_i.

results in removing the triangle. However, a triangle can be removed while all of its three edges are retained in the graph.

Definition 1 (Invalid edge & triangle): If $edge(p_1,\ p_2)$ intersects $\Delta(p_3, p_4, p_5)$ and p_2 is outside $ball(p_1, p_3, p_4, p_5)$, then $edge(p_1, p_2)$ is an invalid edge (invalidated by $\Delta(p_3, p_4, p_5)$). Otherwise, if p_2 is inside $ball(p_1, p_3, p_4, p_5)$, then $\Delta(p_3, p_4, p_5)$ is an invalid triangle (invalidated by $edge(p_1, p_2)$). $\Delta(p_3, p_4, p_5)$ is also invalid if any of its three edges are invalid or if there exists a vertex u such that the radius of $ball(u, p_3, p_4, p_5)$ is greater than 1.

The last constraint in the definition is required for guaranteeing delivery (in Theorem 5) which otherwise is not guaranteed in Delaunay triangulation [3]. Theorem 1 will show that invalid edges and triangles determined distributively are consistent among the nodes. Theorems 2 and 3 will show that the position information of a 2-hop neighbor is sufficient for the correctness of Theorem 1. Theorem 4 shows that network connectivity is conserved when invalid edges are removed.

Theorem 1: If $edge(p_1, p_2)$ intersects $\Delta(p_3, p_4, p_5)$, then invalid edges and triangles determined distributively are consistently among the nodes.

Proof: To prove this theorem, we need to prove that p_1 and p_2 make consistent decisions, which is to prove the follows: (1) if p_2 is outside $ball(p_1, p_3, p_4, p_5)$, then p_1 is outside $ball(p_2, p_3,\ p_4, p_5)$ as shown in Figure 3(a), and (2) if p_2 is inside $ball(p_1, p_3, p_4, p_5)$, then p_1 is inside $ball(p_2, p_3, p_4, p_5)$ as shown in Figure 3(b). The detailed proof can be found in the Appendix. ■

In case 1 (Figure 3(a)), $edge(p_1, p_2)$ is invalid and there are two tetrahedra: $T(p_1, p_3, p_4, p_5)$ and $T(p_2, p_3, p_4, p_5)$. In case 2 (Figure 3(b)), $\Delta(p_3, p_4, p_5)$ is invalid, and there are three tetrahedra: $T(p_1, p_2, p_3, p_4)$, $T(p_1,\ p_2, p_3, p_5)$, and $T(p_1, p_2, p_4, p_5)$. There are no intersecting triangles in either case: $\Delta(p_3, p_4, p_5)$ is valid iff all of $\Delta(p_1, p_2, p_3)$, $\Delta(p_1, p_2, p_4)$, and $\Delta(p_1, p_2, p_5)$ are invalid, and vice versa.

Nodes can determine invalid edges and triangles consistently only if they have sufficient information about the other nodes. To detect an invalid edge consistently, both of its nodes must know about any triangle invalidating this edge. To detect an invalid triangle Δ consistently, all three of its nodes must know of (1) any edge invalidating Δ, and (2) any other triangle invalidating any edge of Δ.

Theorem 2: In $UBG(V)$, if an edge intersects a triangle, then all five of the vertexes are at most 2-hops away.

Proof: We can prove this theorem by proving: if $edge(p_1, p_2)$ intersects $\Delta(p_3, p_4, p_5)$, then either (1) p_1 or p_2 connects to all of p_3, p_4, and p_5, or (2) p_3, p_4, or p_5 connects to both p_1 and p_2. The detailed proof can be found in the Appendix. ■

Theorem 3: In $UBG(V)$, if one of the three edges of a triangle is invalidated by another triangle, then all six of the vertexes are 2-hops away.

Proof: The proof can be found in the Appendix. ■

Theorem 4: The connectivity of the network is conserved after all invalid edges are removed.

Proof: We need to prove that if $edge(p_1, p_2)$ is invalidated by $\Delta(p_3, p_4, p_5)$, both p_1 and p_2 connect to at least one of p_3, p_4, and p_5. This follows directly from Theorem 2. ■

B. A Low-cost PUDT algorithm

We have showed a basic PUDT algorithm in which the nodes propagate 2-hops of position information and then remove all invalid edges and triangles. In this subsection, we will calculate the PUDT with just over 1-hop of position information. In our low-cost PUDT algorithm, each node sends its own position and might also send some *advertised information* to its neighbors, which includes the positions of some of a node's 1-hop neighbors. The basic PUDT algorithm is a special case in which each node's advertised information includes the position information for all of the neighbors, while the low-cost PUDT algorithm includes as little information as possible. Simulation results in Section VII shown that the average amount of advertised information is less than 3% of 2-hop information and hence we say that it is just over 1-hop.

We have five rules on selecting advertised information. The first four rules are regarding edges invalidated by triangles or triangles invalidated by edges. Figures 4(a) and 4(b) show the two situations of connectivities between the five vertexes of an edge and a triangle that intersect each other. Other situations are impossible, as shown in the proof of Theorem 2. Rules 1-4 are illustrated as follows. In Figure 4(a) for rules 1 and 2, if p_1 is not connected with any of p_3, p_4, and p_5, then (1) p_2 should advertise p_2 to p_3, p_4, and p_5 when $edge(p_1, p_2)$ invalidates $\Delta(p_3, p_4, p_5)$, and (2) p_2 should advertise $\{p_3, p_4, p_5\}$ to p_2 when $\Delta(p_3, p_4, p_5)$ invalidates $edge(p_1, p_2)$. In Figure 4(b) for rules 3 and 4, if p_1 is not connected with p_3, and p_2 is not connected with p_4, then (3) p_5 should advertise $\{p_1, p_2\}$ to p_3 and p_4 when $edge(p_1, p_2)$ invalidates $\Delta(p_3, p_4, p_5)$, and (4) p_5 should advertise $\{p_3, p_4\}$ to p_1 and p_2 when $\Delta(p_3, p_4, p_5)$ invalidates $edge(p_1, p_2)$.

The fifth rule is regarding triangles invalidated by other triangles. In Figure 4(c) for rule 5, if $\Delta(p_3, p_4, p_5)$ invalidates $edge(p_1, p_2)$ and p_6 is not connected with some vertexes in $\Delta(p_3, p_4, p_5)$, then p_1 should advertise these vertexes to p_6.

To reduce the size of the advertised information, we optimize the algorithm by (1) sending advertised information only when necessary, and (2) selecting minimized advertised information to send. For example, in Figure 4(a), if $\Delta(p_3, p_4, p_5)$ is

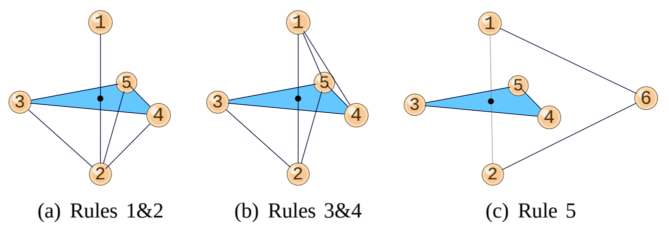


Fig. 4. Illustrations of our low-cost PUDT algorithm.

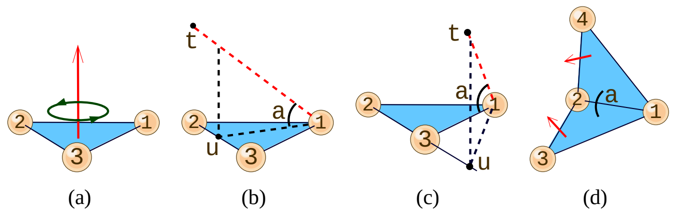


Fig. 5. Illustrations of triangles and angles.

also invalidated by another, say $edge(u, v)$, of which all of the three vertexes in $\Delta(p_3, p_4, p_5)$ are aware, then it is unnecessary for p_1 to advertise $\{u, v\}$ to the vertex nodes in $\Delta(p_3, p_4, p_5)$.

V. Local Hulls and Target Hull

After the PUDT algorithm, each node knows all of its adjacent valid edges and triangles. Note that greedy forwarding uses all nodes in the network whereas hull-based routing only uses nodes on the valid edges and triangles. Therefore, in the section, only valid edges and triangles are used and we refer to them simply as edges and triangles. The entire network space is divided into subspaces by triangles. This section will present the solutions to two problems: (1) how each node locally groups single edges and triangles into hulls of different subspaces (i.e., identify local hulls); and (2) given a destination, how the target hull is selected. Note that we only identify local hulls since it is impossible to determine whether two *objects* are on the same global hull with local information. We define objects as either triangles or single edges. Local hulls can be combined when messages are routing on the hull (in Section VI-B) which guarantees that every node in a global hull can be traveled to by messages.

A. Construction of Local hulls

To identify nodes in different subspaces consists essentially of identifying the triangles and edges in different subspaces. We define a hull for a particular subspace as a structure which contains the triangles bordering the subspace and the triangles and *single edges* (edges not belonging to any triangle) inside the subspace. First, we will present some concepts.

We distinguish triangles by both vertexes and sides, and a triangle with its side touching a subspace belongs to the hull of the subspace. Two sides of a triangle can belong to either two different hulls or the same hull. In the following, we refer to a triangle as a triangle with a particular side. The side of a triangle is defined by the order of the vertexes using the right hand rule. In Figure 5(a), $\Delta(p_1, p_2, p_3)$ is the triangle facing upward, while $\Delta(p_1, p_3, p_2)$ is the one facing downward.

We define the angle between two triangles as the angle to flip one of the triangles along their common edge until they are on the same plane and face-to-face. In Figure 5(d), the angle between $\Delta(p_1, p_2, p_3)$ and $\Delta(p_1, p_4, p_2)$ is α; the angle between $\Delta(p_1, p_3, p_2)$ and $\Delta(p_1, p_2, p_4)$ is $\pi - \alpha$; and the angle between $\Delta(p_1, p_2, p_3)$ and $\Delta(p_1, p_2, p_4)$ is undefined.

We define that two triangles are neighboring triangles if they can be flipped to become face-to-face, and when flipping one of the triangles, it does not pass through any other triangle. That is, if Δ_1 and Δ_2 are neighboring triangles, then(1) Δ_1 and Δ_2 have an angle α, and (2) for any Δ_3 that also share the common edge of Δ_1 and Δ_2, if Δ_3 has an angle β with Δ_1 (or Δ_2), then $\beta > \alpha$.

To group the objects into different local hulls, we first find the components, which consist of either a single edge or a set of neighboring triangles, then we group different components into different hulls. An example of a component consisting of neighboring triangles is shown in Figure 6(a). Starting from any triangle, say $\Delta(p_1, p_3, p_4)$, we can find a sequence of consecutive neighboring triangles adjacent to p_1: $\Delta(p_1, p_4, p_5)$, $\Delta(p_1, p_5, p_6)$, $\Delta(p_1, p_6, p_7)$, and $\Delta(p_1, p_7, p_3)$, which are in a component of p_1. We can find another component for node p_1 consisting of the opposite triangles in the first component. The third component for p_1 is the single edge, $edge(p_1, p_2)$. Similarly, we can find two components for p_1 in Figure 6(b), and there are three components for the p_1 in Figure 6(c).

Before combining components, we define two concepts. (1) The angle α between an edge (or a segment) and a triangle that share a common vertex is defined as the follows. As illustrated in Figure 5(b), let u be the intersection point of $edge(p_2, p_3)$ and the projection of the $edge(p_1, t)$ on $\Delta(p_1, p_2, p_3)$. If u is on $edge(p_2, p_3)$, as in Figure 5(b), the angle α is the angle between $segment(p_1, t)$ and its projection $edge(p_1, u)$. Otherwise, as in Figure 5(c), α is undefined. (2) The closest object to an edge (or segment) e is a triangle or an edge that has the smallest angle with e. The closest object to $segment(p_1, t)$ in Figure 5(b) is $\Delta(p_1, p_2, p_3)$, and the closest object to $segment(p_1, t)$ in Figure 5(c) is $edge(p_1, p_3)$.

The following rule determines whether two components belong to the same hull. Once two components are determined as belonging to the same hull, we combine them by putting their objects together. (1) If two components C_1 and C_2 have two triangles that are opposite, these two components belong to different hulls (though C_1 and C_2 can belong to the same global hull). (2) If C_1 and C_2 belong to the same hull and C_2 and C_3 belong to different hulls, then C_1 and C_3 belong to different hulls. (3) For each edge (not necessarily a single edge) in C_1, we select its closest object in the components that were not determined as belonging to different components. If the closest object is found in component C_2, then C_1 and C_2 belong to the same hull.

In Figure 6(a), let C_1 be the component of p_1 consisting of triangles facing node t, C_2 be the component consisting of

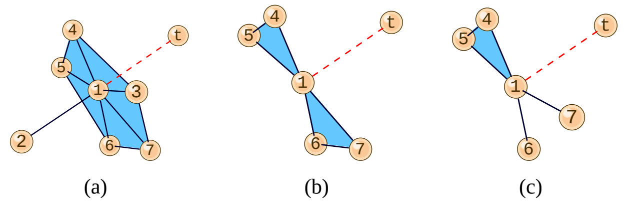


Fig. 6. Components and hulls for p_1.

triangles facing p_2, and C_3 be the component consisting of the single edge $edge(p_1, p_2)$; C_1 and C_2 belong to different hulls because they contain opposite triangles, such as $\Delta(p_1, p_3, p_4)$ and $\Delta(p_1, p_4, p_2)$; C_3 and C_2 belong to the same hull because one of the triangles or edges in C_2 must be the closest object to $edge(p_1, p_2)$ in C_3; finally, C_3 and C_1 belong to different hulls because C_2 and C_1 belong to different hulls. Therefore, p_1 has two local hulls. Similarly, in Figures 6(b) and 6(c), we can see that p_1 has only one hull.

B. Determine the target hull

When a message reaches a local minimum, one of the adjacent hulls of the local minimum (the target) is selected such that the message can recover from the local minimum by searching this hull (searching the nodes in or on the hull's subspace). We define the *target hull* as the hull whose subspace contains all or part of the segment connecting the local-minimum m and destination t, or simply the m-t segment.

Since each object belongs to only one hull, to determine the target hull is to find a representative object (a triangle or a single edge) of the target hull. First we find the closest object to the s-t segment. If the closest object is a triangle or a single edge, then this object is the representative of the target hull. Otherwise, the closest object is an edge, say e, on some triangle. In this case, we flip the virtual triangle, which consists of the two vertexes of e and the destination, along e to find the triangle which has the smallest angle with the virtual triangle. This triangle is the representative object of the target hull. In Figure 6(a), suppose p_1 is the local minimum, t is the destination, and $edge(p_1, p_3)$ is the closest object of $segment(p_1, t)$. We can flip the virtual triangle $\Delta(t, p_1, p_3)$ (or $\Delta(t, p_3, p_1)$) to find the first triangle $\Delta(t, p_4, p_3)$ (or $\Delta(t, p_3, p_7)$) on the target hull. It can be proved that all or part of the m-t segment is in the subspace of the target hull, since there cannot be a triangle dividing the m-t segment and its closest object into different subspaces.

When routing on a target hull, each node only forwards the message to its neighbors adjacent to the triangles and single edges on the target hull. Therefore, in hull routing, when a node forwards a message to another node, the sender needs to tell the receiver which hull is the target hull by piggybacking the information about an object on the target hull that the receiver knows. Theorem 5 shows that hull-based routing can always make progress.

Theorem 5: If the subspace of the target hull contains all or part of the m-t segment, then either the destination is on the target hull, or at least one node on the target hull is closer to the destination than the local minimum.

Proof: If the destination is reachable and is not on the target hull, then there must be a triangle, say $\Delta(u, v, w)$, that intersects the m-t segment. In this case, at least one vertex in $\Delta(u, v, w)$ must be closer to t than m. Otherwise $\Delta(u, v, w)$ is not a valid triangle. This is because if all of the vertexes of $\Delta(u, v, w)$ are outside $ball(m, \|mt\|)$ (which is the ball centered at m with radius equal to the distance between m and t, $\|mt\|$), then $ball(u, v, w, m)$ contains $ball(m, \|mt\|)$. This follows that the radius of $ball(u, v, w, m)$ is greater than 1 (since the radius of the contained ball $\|mt\| > 1$) and $\Delta(u, v, w)$ is invalid, according to Definition 1. ■

VI. GHG and Extension

A. Greedy-hull-greedy (GHG) routing

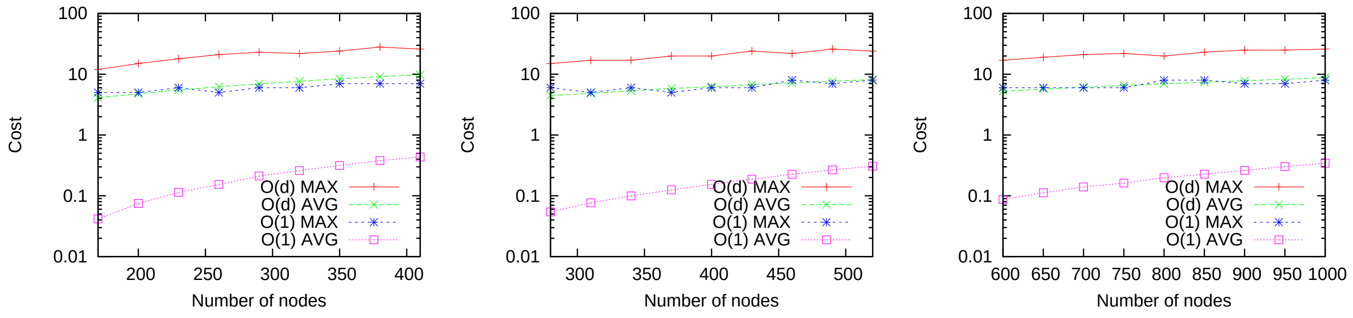
Greedy-hull-greedy (GHG) routing is analogous to greedy-face-greedy (GFG) routing. All geometric routing algorithms contain a greedy routing algorithm and a recovery algorithm, since greedy routing (which forwards the message ever closer to its destination) is the simplest and most efficient. An execution of GHG is a repetitive alteration between greedy forwarding and hull-based local-minimum recovery. GHG can be easily extended with a bounded circle as in [5] to achieve the worst case bound $O(d^3)$, where d is the distance between the source and destination. Delivery is guaranteed since hull routing can always make progress (Theorem 5).

B. Efficient searching on the target hull

In 2D, searching the border of a face for a recovery node is a trivial one-dimensional search. Random walk is proposed in [10] to search the virtual cube structure also proposed in [10]. We use a more efficient hull-based, depth-first search, in which each message is forwarded at most twice the number of the nodes on a target hull (leaf nodes forward at most once and non-leaf nodes forward at most $k+1$ times, where k is the number of children in the search tree). Therefore, we conserve the worst-case bound of $O(d^3)$.

In [11], a depth-first search (DFS) has been proposed for use in geometric routing where depth can be defined as the reciprocal of the distance between the nodes and the destination. In this algorithm, messages only store $O(1)$ routing state information. This algorithm can be improved by allowing each node to overhear the messages of its neighbors.

We assume that each message has a unique ID. Whenever a node u overhears or receives a message for the first time, it creates a record for the message. This record is removed when the message expires. Each record stores (1) a set of nodes that were overheard forwarding the message to some other nodes, (2) an ancestor node of u that first forwards the message to u, and (3) a set of nodes that u forwarded the message to. The DFS rules are: (1) when u receives a message from v, if it is the first time that u receives it (v is the ancestor) or u forwarded the message to v before, u sends the message to the next neighbor w that is the closest to the destination among the neighbors on the target hull and that does not have the



(a) Z=100. (b) Z=200. (c) Z=400.

Fig. 7. Comparison of PUDT costs.

message (known from overhearing); (2) if w does not exist, u forwards the message back to the ancestor; and (3) if v is not the ancestor and u did not send the message to v, u returns the message to v.

We associate each message with a counter which increases whenever it starts to travel to a new hull. When a node finds that a message travels to two of its local hulls with the same counter value, the node combines these two local hulls into one hull. This ensures that each forwarding node forwards the message to all of its neighbors on the target hull when the target hull contains different local hulls of the forwarding node. It can be proved that this algorithm guarantees that any recovery node on the target hull can be traveled to.

C. Extension: CDSs on hulls

In [19], the Gabriel Graph (GG) is constructed on the connected dominating set (CDS) of the network nodes to reduce the number of nodes on each face. We use CDS to reduce the number of nodes on each hull to make searching more efficient. Our CDS nodes are hull-specific. Note that we cannot construct a CDS on all nodes since it results in removing most of the triangles.

An optional step can be applied before the hull-based CDS selection to reduce the CDS size by allowing nodes to construct larger local hulls through exchanging local hull information and combining hulls when hulls share objects.

VII. Simulation

A. Evaluation of our PUDT algorithm

PUDT is performed in random 3D networks of size $1,000 \times 1,000 \times Z$, where Z varies among 100, 200, and 400. For each Z, networks containing a varied amount of nodes are generated. For each Z and each network density, 100 networks are generated to repeat the simulation by randomly selecting an (x, y, z) coordinate for each node within the specific space.

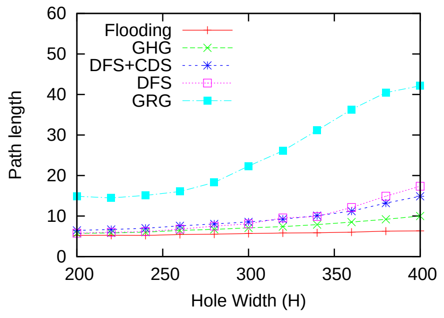
We compare the cost of the basic PUDT (denoted by $O(d)$ in the simulation results) and the low-cost PUDT (denoted by $O(1)$) in terms of the size of position information exchanged among the nodes. The size is measured by the volume of position information, each of which contains three integers describing the x, y, and z coordinates of a node. The simulation results are plotted in log scale. The measurement does not include information about the node's own position. Simulation results in Figures 7(a)-7(c) show that cost of the PUDT algorithms under different height Z and number of nodes. The results show that the average cost of the low-cost PUDT is only around 3% of the 2-hop information which is required in the basic PUDT. Also, the maximum cost of the low-cost PUDT is almost equal to the average cost of the basic PUDT.

B. Routing performance

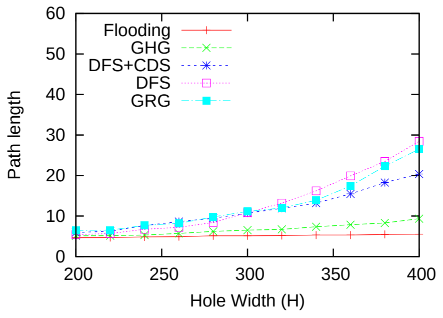
We compare the routing performances of Flooding (which finds the optimal paths), DFS [11] and DFS+CDS (DFS runs on the connected dominating set of the network), greedy-random-greedy (GRG) [10] which performs its random walk local-minimum recovery search on the hull we constructed, and GHG. Our simulation metric is in terms of hop-count.

We generate networks with randomly placed nodes and artificial holes to emulate obstacles in practical situations. The size of all networks is $500 \times 500 \times 500$ and the transmission range of the nodes is 100. A number of N nodes are randomly placed in each network whose degree D ranges between 8, 12, or 16 neighbors per node, and N is calculated from D as $N = 500^3/(\pi \times 100^2/(D+1))$. A rectangular hole whose size is $H \times H \times 150$ is created at the center of each network, where H ranges between 200 and 400 in different networks. Small holes other than the artificial hole might exist in regions where node density is low. Disconnected networks are discarded. For each D and H, we generate 30 networks to repeat the simulation. For each network, we select at most 5000 pairs of nodes as the sources and destinations. Since all protocols have the same path length when greedy forwarding is successful, we require that, in these selected pairs of nodes, the destination is a local-minimum of the source in order to create larger differences in the simulation results.

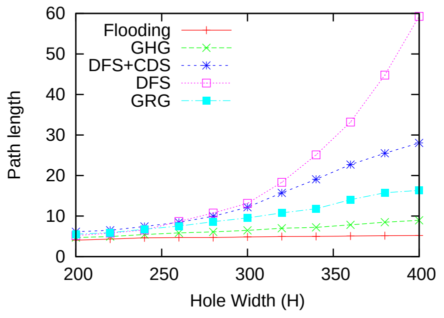
First, we compare GHG with Flooding, DFS, and DFS+CDS. Figures 8(a)-8(c) show that the path length of all protocols increases as the size of the hole increases (the increase in Flooding is the smallest). The performance of GHG is, on average, only 20% longer than the optimal path



(a) D=8.



(b) D=12.



(c) D=16.

Fig. 8. Comparison of routing performances.

length of Flooding and is at most 50% longer in the worst-case. Comparatively, DFS and DFS+CDS have a longer path length which in the worst case is about six and three times longer than GHG, respectively. The performance of DFS and DFS+CDS becomes worse as the network density increases. As the density increases, the percentage of nodes on the hulls (or border of the holes) decreases and the percentage of nodes inside the dense region of the network increases. Therefore, the simple combination of DFS and CDS is not sufficient to improve performance in dense networks, where the hull-based search in GHG is more efficient.

Second, we compare GHG with GRG. From Figures 8(a)-8(c), the performances of Flooding, GRG, and GHG all increase as network density increases. This is because the actual size of the holes decreases as the network density increases and the percentage of nodes on the hull decreases as the density increases. The performance of GRG is less efficient than that of GHG: GRG has a longer path length which in the worst-case is about two to four times longer than GHG. GRG also searches the target hull when recovering. The difference is that GRG performs a simple random walk search. GHG has better performance since (1) its hull-based DFS tries to send the message to the nodes closer to the destination to speed-up the recovery process; and (2) it tries not to repeat sending messages to each node.

C. *Summary of simulation*

To summarize the simulation results, our localized PUDT has low overhead: it requires each node to exchange just over 1-hop of information to calculate. The routing performance of GHG is shown to be, on average, only 20% longer than the optimal path and can be three times shorter than the DFS+CDS in dense network ($D = 16$) and four times shorter than GRG in sparse networks ($D = 8$).

VIII. Conclusion

In this paper, we propose some solutions for efficient geometric routing in 3D networks. We present the first 3D localized PUDT algorithm, hull recognition algorithm, and GHG, the first 3D analogue to face routing. Simulation results show that our PUDT algorithm is low in cost, and GHG is more efficient than DFS and GRG. We believe many problems in geometric routing in 2D networks can be redefined or extended to 3D networks based on our model, which include multicast, geocast, virtual coordinates, handling uncertain position information, and energy efficient routing.

Acknowledgement

This work was supported in part by NSF grants CNS 0422762, CNS 0434533, CNS 0531410, CCF 0545488, and CNS 0626240. Email: jie@cse.fau.edu.

References

[1] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with Guaranteed Delivery in Ad Hoc Wireless Networks. In *Proc. of Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, 1999.

[2] B. Karp and H.T. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In *Proc. of ACM MobiCom*, 2000.

[3] H. Frey and I. Stojmenovic. On Delivery Guarantees of Face and Combined Greedy-Face Routing in Ad Hoc and Sensor Networks. In *Proc. of ACM MobiCom*, 2006.

[4] F. Kuhn, R. Wattenhofer, and A. Zollinger. Worst-Case Optimal and Average-Case Efficient Geometric Ad-Hoc Routing. In *Proc. of ACM MobiHoc*, 2003.

[5] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger. Geometric Ad-Hoc Routing: Of Theory and Practice. In *Proc. of ACM PODC*, 2003.

[6] G. Toussaint. The Relative Neighborhood Graph of a Finite Planar Set. *Pattern Recognition*, 12(5):261268, 1980.

[7] K. Gabriel and M. Pearlman. A New Statistical Approach to Geographic Variation Analysis. *Systematic Zoology*, 18:259278, 1969.

[8] J. Gao, L.J. Guibas, J. Hershburger, L. Zhang, and A. Zhu. Geometric Spanner for Routing in Mobile Networks. In *Proc. of ACM MobiHoc*, 2001.

[9] X.-Y. Li, G. Calinescu, P.-J. Wan, and Y. Wang. Localized Delaunay Triangulation With Application in Wireless Ad Hoc Networks. In *Proc. of IEEE INFOCOM*, 2003.

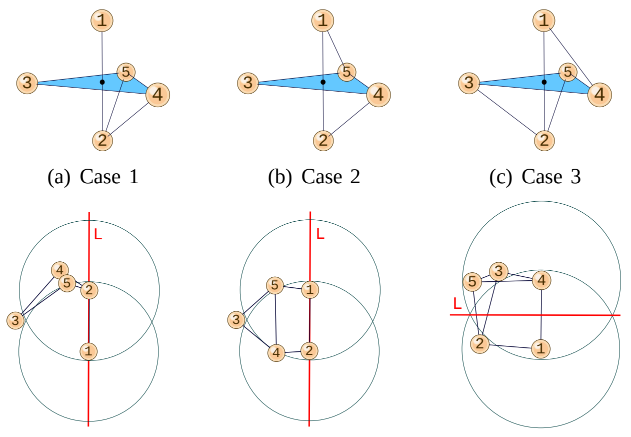
[10] R. Flury and R. Wattenhofer. Randomized 3D Geographic Routing. In *Proc. of IEEE INFOCOM*, 2008.

[11] I. Stojmenovic, M. Russell, and B. Vukojevic. Depth First Search and Location Based Localized Routing and QoS Routing in Wireless Networks. *Computers and Informatics*, 21(2):149–165, 2002.

[12] A. Rao, C. Papadimitriou, S. Shenker, and I. Stoica. Geographical Routing Without Location Information. In *Proc. of ACM MobiCom*, 2003.

[13] A. Caruso, A. Urpi, S. Chessa, and S. De. GPS-free Coordinate Assignment and Routing in Wireless Sensor Networks. In *Proc. of IEEE INFOCOM*, 2005.

[14] B. Leong, B. Liskov, and R. Morris. Geographic Routing Without Planarization. In *Proc. of NSDI*, 2006.



(a) Case 1 (b) Case 2 (c) Case 3

(d) Case 1 (e) Case 2 (f) Case 3

Fig. 9. Illustrations for the proof of Theorem 2.

[15] R. Kleinberg. Geographic Routing Using Hyperbolic Space. In *Proc. of IEEE INFOCOM*, 2007.

[16] Y.-J. Kim, R. Govindan, B. Karp, and S. Shenker. Geographic Routing Made Practical. In *Proc. of NSDI*, 2005.

[17] Q. Fang, J. Gao, L. J. Guibas, V. de Silva, and L. Zhang. GLIDER: Gradient Landmark-Based Distributed Routing for Sensor Networks. In *Proc. of IEEE INFOCOM*, 2005.

[18] S. Funke and N. Milosavljevi. Guaranteed-delivery Geographic Routing Under Uncertain Node Locations. In *Proc. of IEEE INFOCOM*, 2007.

[19] S. Datta, I. Stojmenovic, and J. Wu. Internal Node and Shortcut Based Routing with Guaranteed Delivery in Wireless Networks. *Cluster Computing, Special Issue on Mobile Ad Hoc Networks*, April 2002.

APPENDIX

Proof of Theorem 1: If two balls intersect at three points, then each ball is divided into two domes by the intersecting plane determined by the three points. For each ball, one of the domes is inside the other ball and the other dome is outside the other ball.

Case 1: As shown in Figure 3(a), we will show that if p_2 is outside $ball(p_1, p_3, p_4, p_5)$, then p_1 is outside ball $ball(p_2, p_3, p_4, p_5)$. Since the intersection u of $edge(p_1, p_2)$ and $\Delta(p_3, p_4, p_5)$ is inside $ball(p_1, p_3, p_4, p_5)$, $edge(p_1, p_2)$ has an intersection v with $ball(p_1, p_3, p_4, p_5)$. Since p_1 and v are on different sides of $plane(p_3, p_4, p_5)$, they are on two different domes of $ball(p_1, p_3, p_4, p_5)$ separated by $plane(p_3, p_4, p_5)$. Since v is inside $T(p_2, p_3, p_4, p_5)$, $dome(p_3, p_4, p_5, v)$ is inside $ball(p_2, p_3, p_4, p_5)$. Thus, $dome(p_3, p_4, p_5, p_1)$ is outside $ball(p_2, p_3, p_4, p_5)$ which follows that p_1 is outside $ball(p_2, p_3, p_4, p_5)$.

Case 2: As shown in Figure 3(b), we will show that if p_2 is inside $ball(p_1, p_3, p_4, p_5)$, then p_1 is inside $ball(p_2, p_3, p_4, p_5)$. Since p_2 is inside $ball(p_1, p_3, p_4, p_5)$, $line(p_1, p_2)$ intersects with $ball(p_1, p_3, p_4, p_5)$ at v, and p_1 and v are on different sides of $plane(p_3, p_4, p_5)$. It follows that p_1 and v are on different domes of $ball(p_1, p_3, p_4, p_5)$ separated by $plane(p_3, p_4, p_5)$. As p_2 is inside $T(p_3, p_4, p_5, v)$, it is impossible for v to be inside $ball(p_2, p_3, p_4, p_5)$; otherwise p_2 is inside $ball(p_2, p_3, p_4, p_5)$ instead of on the ball. Therefore, $dome(p_3, p_4, p_5, v)$ is outside $ball(p_2, p_3, p_4, p_5)$,

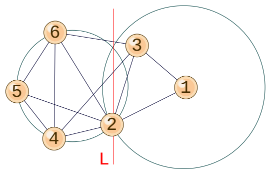


Fig. 10. Illustration for the proof of Theorem 3.

$dome(p_1, p_3, p_4, p_5)$ is inside $ball(p_2, p_3, p_4, p_5)$, and finally p_1 is inside $ball(p_2, p_3, p_4, p_5)$. ■

Proof of Theorem 2: We will show that the opposite cases below are impossible which are illustrated in Figures 9(a)-9(f). We ignore the symmetric cases. If neither p_1 nor p_2 connects to both p_3, p_4, and p_5, we can assume p_1 does not connect to p_3 without loss of generality. If p_2 does not connect to p_3, then either both p_4 and p_5 do not connect to p_1 (case 1), or p_4 does not connect to p_1 and p_5 does not connect to p_2 (case 2). If p_2 connects to p_3, then p_2 cannot connect to p_4 and p_1 cannot connect to p_5 (case 3).

Case 1: As shown in Figures 9(a) and 9(d), p_3 connects to neither p_1 nor p_2, and p_4 and p_5 do not connect to p_1. Let L be a plane which contains p_1 and p_2 and which is perpendicular to $\Delta(p_1, p_2, p_3)$. p_4, p_5, and p_3 must be on the same side of L. Therefore, $edge(p_1, p_2)$ cannot intersect $\Delta(p_3, p_4, p_5)$.

Case 2: As shown in Figures 9(b) and 9(e), p_3 connects to neither p_1 nor p_2, p_5 does not connect to p_2, and p_4 does not connect to p_1. Let L be a plane which contains p_1 and p_2 and which is perpendicular to $\Delta(p_1, p_2, p_3)$. p_4, p_5, and p_3 must be on the same side of L. Therefore, $edge(p_1, p_2)$ cannot intersect $\Delta(p_3, p_4, p_5)$.

Case 3: As shown in Figures 9(c) and 9(f), p_1 does not connect to p_3 and p_5, and p_4 does not connect to p_2. Let L be the intersection plane of the unit balls centered at p_1 and p_4 respectively. p_3 and p_5 must be above L and p_2 must be under L. Therefore, $edge(p_1, p_2)$ cannot intersect $\Delta(p_3, p_4, p_5)$. ■

Proof of Theorem 3: Illustrated in Figure 10, the theorem can be rephrased as: if one of the edges in $\Delta(p_1, p_2, p_3)$ intersects another $\Delta(p_4, p_5, p_6)$, then any vertex in $\Delta(p_1, p_2, p_3)$ is at most 2 hops away from any vertex in $\Delta(p_4, p_5, p_6)$.

Symmetrically, we only need to prove that p_1 is at most 2 hops away from any vertex in $\Delta(p_4, p_5, p_6)$. If it is $edge(p_1, p_2)$ or $edge(p_1, p_2)$ that is invalidated by $\Delta(p_4, p_5, p_6)$, p_1 is at most 2 hops from any vertex in $\Delta(p_4, p_5, p_6)$. This follows directly from Theorem 2.

We need to prove that if $edge(p_2, p_3)$ is invalidated by $\Delta(p_4, p_5, p_6)$, then p_1 directly connects to at least one vertex in $\Delta(p_4, p_5, p_6)$. This must be true. As shown in Figure 10, let $|p_1, p_2| > |p_1, p_3|$, we can see that p_3 is outside of $ball(p_4, p_5, p_6, p_2)$ since $edge(p_2, p_3)$ is invalidated by $\Delta(p_4, p_5, p_6)$. By assumption, all vertexes of $\Delta(p_4, p_5, p_6)$ are outside of the ball centered at p_1 with radius equal to $|p_1, p_2|$. Therefore, all vertexes in $\Delta(p_4, p_5, p_6)$ are to the left of the intersecting plane L of the two balls, while all vertexes in $\Delta(p_1, p_2, p_3)$ are on or to the right of L. This contradicts the fact that $edge(p_2, p_3)$ intersects $\Delta(p_4, p_5, p_6)$. ■