Proof GridCast Forms the Delaunay Triangulation

INSERT AUTHORS HERE

started June 12, 2000, halted June 15, 2000, restarted with new proof June 22, 2000, this version compiled July 5, 2000

1 Introduction

This is a proof that the GridCast protocol forms the Delaunay triangulation.

We will discuss the current state of the GridCast protocol as a graph; this graph is the logical topology.

When we discuss the algorithm *halting*, we mean there exists a future where all of our assumptions hold and no more modification to the graph take place.

2 Scope

We do not try to prove that the GridCast protocol forms the Delaunay triangulation under all conditions; we limit the scope of the proof by making a few assumptions.

We assume a set of nodes start operating and attempt to join the Delaunay triangulation and neither attempt to leave nor stop performing operations.

We assume that no messages are lost. We allow messages to be sped-up or slowed-down to arrive in any order, but messages must arrive in a finite amount of time. We refer to $t_{deliver}$ as the maximum time taken to deliver a message. We also assume that onnections never timeout — that is, that $t_{timeout} > t_{heartbeat} + t_{deliver}$.

The protocol assumes that every node begins knowing its own coordinates. For this proof, we further assume that no two coordinates are the same and that no set of four coordinates inscribe the same circle. The later assumption can also be stated as: no set of four coordinates are equidistant from the same point. (The point would be the center of the circle.)

These are all of our assumptions. Our assumptions about coordinates concern cases where the Delaunay triangulation is either undefined or not a proper triangulation; these are common assumptions in Delaunay triangulation proofs. The other assumptions eliminate the possibility of a node removing a neighbor; after the completion of this proof, we will consider two special cases where a neighbors are removed: when a node leaves the topology and when a node stops operating.

3 Outline of Proof

Having assumed away all situations which cause a neighbor to be removed, the only remaining operation which modifies the graph is the add neighbor operation. Since the algorithm is asynchronous, our proof must accommodate these operations being done in any order. To do this, we begin by defining *progress* in terms of a binary relation that determines which of two graphs is closer to a Delaunay triangulation. We then show that GridCast's add neighbor operation *progresses*, that is, always produces a graph which is closer to the Delaunay triangulation. Thus, any order of add neighbor operations will always proceed towards a Delaunay triangulation.

To complete the proof we must prove two addition theorems: (1) that the series of add neighbor operations does not halt until it forms the Delaunay triangulation, and (2) that it forms the Delaunay triangulation in a finite amount of time.

We begin our proof of (1) by proving a series of theorems of the form "if property P does not hold, an insert neighbor operation occurs". An example of a property is P_2: all edges are bidirectional. We can conclude from each of these theorems that when the algorithm halts, property P holds. We then use these properties to show that the graph must be a triangulation and then that the only triangulation the algorithm can form is the Delaunay triangulation.

We prove (2) by showing that it takes at most a constant number of messages to cause an add neighbor and that a node can only add another node as a neighbor once. This means there can be at most $n^2 - n$ add neighbor operations, each taking place in a constant amount of time.

Since every operation progresses towards the Delaunay triangulation, and the algorithm does not halt performing operations until it reaches the Delaunay triangulation, and the algorithm always always reaches the Delaunay triangulation in a finite amount of time, we can conclude that the algorithm always forms the Delaunay triangulation.

4 Definition of Progress

An operation progresses if, for every application of the operation, the graph after the operation is closer to a Delaunay triangulation than the graph before the operation. We define a graph Z to be at least as close to a Delaunay triangulation as a graph Y (over the same set of nodes), if every node in Z has a perceived Voronoi region which is a subset of the node's perceived Voronoi region in Y. A graph Z is closer to a Delaunay triangulation than Y if Z is *at least as close* and the perceived Voronoi region of at least one node in Z is a proper subset of the node's perceived Voronoi region in Y.

We define the Perceived Voronoi Region of a node n in a graph Z to be the Voronoi region of n in a graph consisting of n and all nodes connected by edges out-going from n in Z. (See Figure 1) In a Delaunay triangulation, the perceived Voronoi region of a node is the same as its Voronoi region.

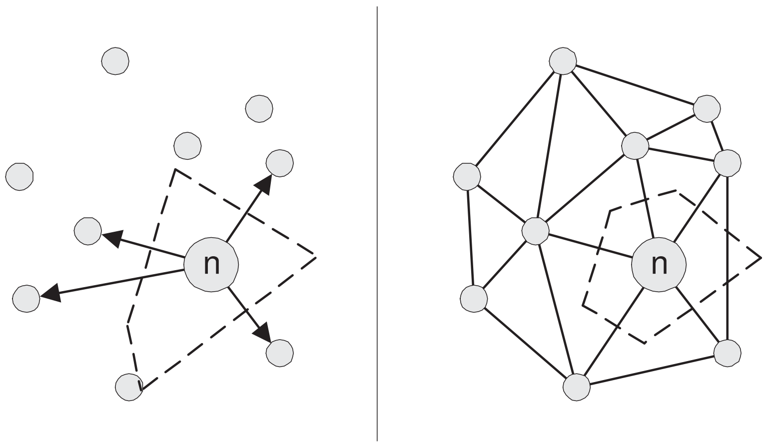

Figure 1: Perceived Voronoi Region of a node n in two graphs. We define the perceived Voronoi region of a node to be the the Voronoi region of the node and all nodes it is connected to by out-bound edges. The graph on the right demonstrates that in a Delaunay triangulation, a node's perceived Voronoi region is equivalent to its Voronoi region.

5 Insertion of a New Neighbor Progresses

GridCast performs a number of operations, but only one modifies the graph: adding a new neighbor. *NOTE: Removing a neighbor does modify the graph but we have assumed away any event that would cause a remove neighbor operation (Goodbye messages and timeouts).* We will show that the add neighbor operation progresses because the node adding a neighbor shrinks its own perceived Voronoi region and all other all nodes' perceived Voronoi regions remain unmodified.

To show that adding a neighbor shrinks the node's perceived Voronoi region, we must use the neighbor table invariant. The invariant is that in a Delaunay triangulation of just the node and its neighbors, there must be an edge connecting the node to every neighbor. Another way to state this is: the perceived Voronoi region of the node must have a border for each neighbor.

A new neighbor is added if and only if, in a Delaunay triangulation of the node, its neighbors, and the new neighbor, the new neighbor is connected to the node by an edge. That is, in this local Delaunay triangulation, the new neighbor's Voronoi region touches that of the node.

We now employ a property of Voronoi regions: the Voronoi region of any node in a set S must be a subset of the Voronoi region of the same node in a set T if S is a subset of T. Using this and the fact that the set of only the old neighbors is a subset of the set of the old neighbors plus the new neighbor, we can conclude that the perceived Voronoi region of the node with the new neighbor must be a subset of the old perceived Voronoi region.

We must now show that the new perceived Voronoi region is a proper subset of the old perceived Voronoi region. To do this, we must show that there exist at least one element in the old region which is not in the new one. The perpendicular bisector of the new neighbor and the node must intersect at least one edge forming the boundry of the node's old perceived Voronoi region. The endpoints of this intersected edge cannot lie on the perpendicular bisector, because they are already equidistant to three vertices and lying on the perpendicular bisector would make them equidistant to a fourth, violating our assumption that no four points inscribe a circle. Thus, for the edge to intersect the perpendicular bisector, at least one endpoint must lie on each side. The endpoint which lies on the side closer to the new neighbor is closer to the new neighbor than the node, and will not be part of the node's new perceived Voronoi region. Since this endpoint was in the old region but not the new, we can conclude that the new perceived Voronoi region is a proper subset of the old perceived Voronoi region.

Since all other node's neighbor tables are not modified by the add neighbor operation, we can conclude that their perceived Voronoi regions remain the same. Thus, all the perceived Voronoi regions of graph after the add neighbor operation are subsets of the perceived Voronoi regions before the operation, and a the perceived Voronoi region of the node adding the neighbor is a proper subset, and so conclude that the add neighbor operation always progresses.

6 Properties

We will now prove four lemmas. Each lemma refers to a property and has the form where if a property does not hold, an add neighbor operation must occur. From these lemmas, we can conclude that, when the algorithm halts, the graph has each of the properties. The properties are:

1. All nodes, except the Leader, have a connection to a node with greater coordinates.
2. All edges are bidirectional.
3. Neighbors have the same perceived Voronoi region border between them.
4. Adjacent neighbors are connected, if the angle between then is less than π radians.

6.1 All nodes, except the Leader, have a connection to a node with greater coordinates

We will prove that if a node, not the Leader, does not have a connection to a node with greater coordinates, an add neighbor operation takes place. We define Coordinates (x_1, y_1) to be greater than (x_2, y_2) if $y_1 > y_2$ or if $y_1 = y_2$ and $x_1 > x_2$. Note that the coordinates of two different nodes cannot be equal, because we assumed that duplicate coordiantes do not exist.

After every node has sent a message to the server, the server stores the coordinates of the Leader, the node with the greatest coordinates.

Consider a node, a, that does not have a connection to a node with a greater coordinate. a sends an AmILeader message to the server every $t_{heartbeat}$ seconds and the server replies with a LeaderIs message containing the physical address and coordinates of the Leader.

If a is not the Leader, a will always add the Leader as a neighbor. There are two cases.

The first case is if the Leader's y-coordinate is greater than a's. a's perceived Voronoi region must contain the point at a's x-coordinate with an arbitrarily large y-coordinate. The Leader is closer to this point than a, so in the local Delaunay triangulation, the Leader's Voronoi region will touch a's Voronoi region, and thus a will add the Leader as a neighbor.

The second case is if the Leader's y-coordinate is the same as a's, but the Leader has a greater x-coordinate. In this case, we consider the point at the Leader's x-coordinate and an arbitrarily large y-coordinate. This point must lie in a's perceived Voronoi region. Once again, the Leader is closer to the point than a, so in the local Delaunay triangulation, the Leader's Voronoi region will touch a's Voronoi region, and a will add the Leader as a neighbor.

So, if any node, not the Leader, does not have a connection to a node with a greater coordinate, an add neighbor operation occurs. From this we can conclude that when the algorithm halts, all nodes in the resulting graph, except the Leader, will have a connection to a node with greater coordinates.

A corollary of this is that there exists a path from every node to the Leader. This path consists of always taking a connection to a node with a greater coordinate.

6.2 All edges are bidirectional

Consider a pair of nodes, a and b, joined by a unidirectional edge — a has b as a neighbor but b does not have a as a neighbor. We wish to show that for any situation of this case, and add neighbor operation will take place.

Every $t_{heartbeat}$ seconds, a will send a Hello to b. If b should have a as a neighbor, a is inserted as a new neighbor.

If b should not have a as a neighbor, then, in the local Delaunay triangulation of b, b's neighbors, and a, there must not be an edge between a and b. This can only be the case if, in this local Delaunay triangulation, the Voronoi regions of a and b do not touch.

Now consider the border of b's Voronoi regions, in the local Delaunay triangulation, that is closest to the perpendicular bisector of a and b. Because Voronoi regions are convex, the closest part of b's Voronoi region is either a single vertex or a single edge.

Consider the case it is a vertex, and call that vertex v. v is equidistant from b and two of b's neighbors — let's call them c and d. (See Figure 2.) In a Delaunay triangulation of a, b, c, and d, the Voronoi regions of a and b do not touch, because v is still the closest point in b's Voronoi region and it does not reach a's. (A result of Voronoi regions being convex and that none of b's neighbors can lie on the line through a and b.)

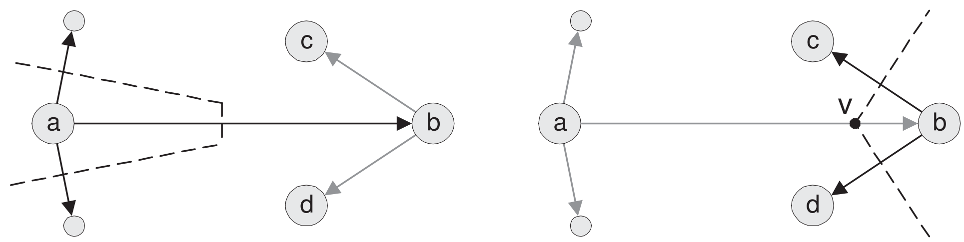

Figure 2: An example of a unidirectional edge. The left shows a's perceived Voronoi region, which contains a border for the node b. On the right is b's perceived Voronoi region, which does not contain a border for a.

The only way that a could have b as its neighbor is because it does not know about both of c and d. When b receives a Hello from a, it sends a GoAway message in reply, including in the message information on nodes c and d. (That c and d are included will be proved shortly.) a will receive the GoAway message and determine that c or d or both should be its neighbors and send it (them) Hello messages. c or d or both will respond with either a Hello or GoAway message(s) and be added as a new neighbor by a.

Now, we shall prove that c and d are included in the GoAway message from b to a. c and d are on either side of the line passing through a and b, otherwise v would not be the closest vertex. Also, c and d must be adjacent neighbors in b's neighbor table, otherwise their borders would not share the vertex v. Because c and d are adjacented and lie on either side of the line from a to b, they must be the neighbors of b that lie CW and CCW of a with respect to b. Thus, we can conclude that if the closest part of b's Voronoi region to the perpendicular bisector is a vertex, a neighbor is added.

Now consider the case that the closest part of b's Voronoi region to the perpendicular bisector is an edge. Let's call the neighbor that causes that edge e. For the closest part to be an edge rather than a vertex, the edge must be parallel to the perpendicular bisector of a and b. It can only be parallel if e lies on the line through a and b. e must lie between a and b for b not to have a as a neighbor.

The only way that a could have b as its neighbor is because it does not know about e. When b receives a Hello from a, it sends a GoAway message in reply, including in the message information on node e, which lies straight on of a with respect to b. a will receive the GoAway message and determine that e should be its neighbors and send it a Hello message. e will respond with either a Hello or GoAway message(s) and be added as a new neighbor by a.

Thus, if an edge is unidirectional an add neighbor operation must occur. We conclude that the resulting graph, the graph when the algorithm halts, must contain only bidirectional edges.

6.3 Neighbors have the Same Perceived Voronoi Region Border between Them

We know that the resulting graph will be bidirectional. Consider a pair of nodes, a and b, connected by a bidirectional edge, that is, having each other as neighbors. To be mutual neighbors, a border of a and b's perceived Voronoi regions must lie on the perpendicular bisector of a and b. Without loss of generality, assume there exists a point, p, on the perpendicular bisector which lies inside a's perceived Voronoi region but outside of b's. We will prove that if p exists, an add neighbor operation must occur.

Consider a point, q, which lies on the perpendicular bisector of a and b such that q is the point closest to p inside b's perceived Voronoi region. (It is not necessary for q to lie inside a's perceived Voronoi region.) (See Figure 3.) q is an endpoint of the border between a and b in b's perceived Voronoi region. Being an endpoint, q must be equidistant from a, b, and a third node, c, which is a neighbor of b. c must be a neighbor of b which lies adjactently CW or CCW of a, because their borders in b's perceived Voronoi region share the common endpoint q.

Because p lies on the perpendicular bisector of a and b, it is equidistant from both a and b. p must also be closer to c than b or else p would lie inside b's perceived Voronoi region. Since $d(a,p) = d(b,p)$ and $d(c,p) < d(b,p)$ we know that $d(c,p) < d(a,p)$, that is that c is closer to p than a.

When b sends a Hello message to a (which must occur every $t_{heartbeat}$ seconds) it includes information on node c, because it lies adjacently CW or CCW to a in b's neighbor table. When a receives the message, it will determine that c should be its neighbor, because a's old perceived Voronoi region included p, but in the new local Delaunay triangulation, p lies inside c's Voronoi region, implying that c's region must now border a's. After determining that c should be its neighbor, a sends a message to c, and on receiving a reply inserts c as a new neighbor.

When a pair of connected nodes disagree on the mutual border of their preceived Voronoi regions, an add neighbor operation must take place. Thus, the algorithm will only halt when connected nodes agree on the mutual borders of their Voronoi regions.

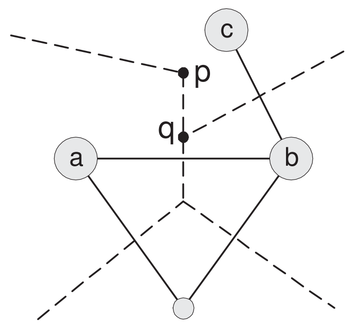

Figure 3: Nodes a and b are connected by an edge. Each node's perceived Voronoi region borders the perpendicular bisector, but there exists a point p on the bisector that lies inside a's region and not in b's.

6.4 Adjacent Neighbors are Connected to Each Other

Consider a node a with two neighbors b and c. Assume without loss of generality that the CW angle $\angle bac$ is less than π radians. Assume b and c are adjactent, that is: there does not exist d, a neighbor of a, where the CW angles $abs(\angle bad) + abs(\angle dac) = abs(\angle bac)$. We will prove that, when the algorithm halts, b and c must have a edge between them.

Let's call p the vertex on a's perceived Voronoi region at the intersection of the borders caused by b and c. b must have p as the endpoint to the border of its perceived Voronoi region caused by a or an add neighbor operation must be done. b can only have p as the endpoint if c is its neighbor, otherwise at least 4 nodes lie on a circle centered at p. The same argument applies for c having b as a neighbor.

Thus, an add neighbor operation occurs unless adjacent neighbors are connected to each other. Thus in the resulting graph, all adjacent neighbors where the angle between them is less than π radians, will be connected.

7 The graph is Straight-Edge Planar

We will show, using the lemmas proved above, that assuming both that the algorithm has halted and that the graph is not straight-edge planar leads to a contradiction. From this, we can conclude that when the algorithm halts, the graph is straight-edge planar.

A graph is straight-edge planar if there does not exist an intersection of straight edges. Assume the algorithm has halted and there exist an intersection. Of all intersections, select the intersection with the greatest coordinates. One endpoint of each edge must have greater coordinates than the other endpoint of the edge. Label the two nodes greater than the other end a and b, and the other ends of the intersecting edges c and d, respectively. (See Figure 4.)

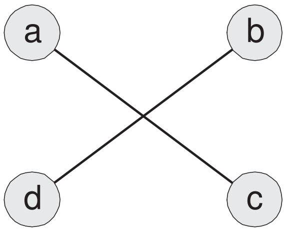

Figure 4: An intersection of two edges. Nodes a has greater coordinates than c, and b has greater coordinates than d.

Consider the case if a and b are connected by an edge. For this to be the case, the nodes a, b, and c must not lie on a striaght line, nor can the nodes a, b, and d. a must be adjacent to d in b's neighbor table. If it is not, assume there exists a neighbor of b called e that lies between a and d, adjacent to d. Either the connection from b to e or the connection from e to d (required by the adjacent neighbor lemma) must intersect the edge $\bar{ac}$ at a greater coordinate, violating our assumption that the intersection of $\bar{ac}$ and $\bar{bd}$ is the intersection with the greatest coordinates. Since it must be the case that a and d are adjacent neighbors of b, by the immediately preceeding property, we can conclude that there is a connection from a to d. Using the same logic we can conclude there is an edge from b to c.

The resulting graph is shown in figure 5. We will now show that this graph cannot exist and therefore the case that a and b are connected cannot exist.

For a to have the edges it has in figure 5, there must exist edges from a to all of b, c, and d in a's local Delaunay triangulation. For this to occur, c must lie outside the circle inscribed by a, d, and b. But for b to have a connection to c, c must lie inside the circle inscribed by a, d, and b. Contradiction. Thus, the graph show in figure 5 cannot exist and a cannot have a connection to b.

a cannot have a direct connection to b, but there must exist a path from a to b through the Leader. We will see that this fact will lead to a contradiction.

Let a' be the neighbor of a adjacent to c going CCW around a. Let a'' be the neighbor of a' adjacent to a going CCW around a'. Let a''', a'''', ... be defined similarly. Let b', b'', ... be defined in the same way, except going CW.

The CCW angle between c and a' must be at least π radians or there would exist a connection between c and a' which intersects $\bar{bd}$ or some connection in a path from b to the Leader — creating an intersection with greater coordinates than the $\bar{ac}$ and $\bar{bd}$ intersection, which we've assumed to be the intersection with the greatest coordinates.

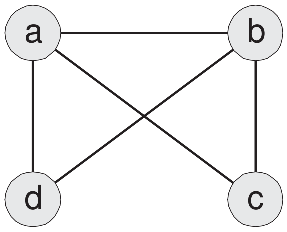

Figure 5: Case where a and b are connected by an edge. By the adjacent neighbor property, edges $\bar{ad}$ and $\bar{ad}$ must exist. This eventually leads to a contradiction.

The CCW angle between a and a'' must be at least *pi* radians, or a would have a'' as a neighbor, and a' would not be c's adjacent neighbor CCW around a. This argument holds for all further angles in the path c, a, a', a'', ...

Similar arguments hold for the path d, b, b', b'', ... Figure 6 shows an image of the paths.

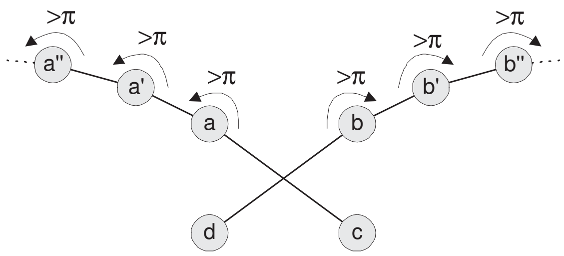

Figure 6: The paths a, a', a'', ... and b, b', b'', ...

We now come to two cases: The special case is where all the labeled points lie on a single line, and the normal case is where they do not.

If all the labeled points lie on a straight, then we consider the lowest node in common between the paths a, a', a'', ... and b, b', b'', ... There must exist a node in common, because there exists a path from each node to the Leader. Let's name the lowest node in common e. e must have separate connections to the next lower node in both the path a, a', a'', ... and the path b, b', b'', ... In this case, e has connections to two different nodes which both lie on a straight line through e. By the neighbor table invariant, this cannot exist. Therefore, the special case leads to a contradiction.

Now, we cover the normal case. The paths a, a', a'', ... and b, b', b'', ... do not all lie on a straight line. Once they separate, these paths will never cross but grow wider and wider apart. They cannot "loop around underneath" because, by a previously proved property, each node, except the Leader, must have a connection to a node with greater coordinates. But also by that property, we proved that by taking a path of connection to nodes with greater coordinates, that path will always progress to the Leader. Thus, we have a contradiction between that the paths will never cross and that the paths will meet at the Leader.

We initally assumed that the algorithm had halted and the graph was not straight edge planar. Based on properties of the graph when the algorithm halts, we found that all cases lead to a contradiction. Therefore, our assumption must be wrong and it must be the case that the algorithm has not halted. We can then conclude that when the algorithm halts, the graph will be striaght-edge planar.

7.1 The Graph is a Triangulation

Since we know the resulting graph must be planar, we can now talk about the graph's faces. A planar straight-edge graph is a triangulation if every face, except the convex hull, has 3 edges.

Consider a face with more than 3 edges. The angle between any adjacent vertices must be at least π radians or an insert neighbor operation occurs. If a face is made entirely of internal angles greater than or equal to π radians, and the graph is connected, that face must be the convex hull.

Therefore every face that has more than 3 sides must either be the convex hull or an insert neighbor operation occurs. We can conclude that when the algorithm halts, the graph is a triangulation.

8 The Only Possible Triangulation is the Delaunay Triangulation

So far, we have proved that when GridCast halts, the graph will be a triangulation. Now, using a proof done by Sibson in 1977, we will show that the only possible triangulation is the Delaunay triangulation.

Sibson proved that the Delaunay triangulation and the locally equiangular triangulation were the same graph. Sibson defined locally equiangular by:

> If two triangles in the triangulation share a common edge, they define a quadrilateral with that common edge as [a] diagonal. If that quadrilateral is strictly convex (that is, each vertex is an extremal point of it) the replacement of the chosen diagonal by the alternative one must not increase the minimum of the six angles in the two triangles making up the quadrilateral, and this must hold for all such stricly convex quadilaterals.

In the resulting triangulation, consider two triangles, $\triangle abc$ and $\triangle abd$, where the quadrilateral $\Box acbd$ is strictly convex. a has connections to b, c, and d. This means that a, b, c, and d were present in a's local Delaunay triangulation. In that Delaunay triangulation, a had a connection to b. The connection $\bar{ab}$ must be the correct connection for local equiangularity because it was selected in a's local Delaunay triangulation which included the nodes a, b, c, and d.

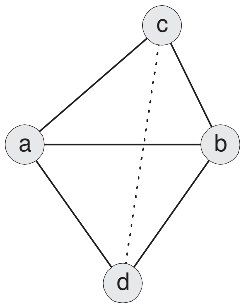

Figure 7: Two triangles, $\triangle abc$ and $\triangle abd$, where the quadrilateral $\Box acbd$ is strictly convex. A graph is locally equiangular if, for each such pair of triangles, the minimum internal angle is not decreased by selecting the alternate diagonal of the quadrilateral (dotted line).

So the local Delaunay triangulations enforce local equiangularity on the graph, and, by Sibson's proof, a triangulation which is locally equiangular, is the Delaunay triangulation.

9 Conclusion of Proof

We have shown that the add neighbor operation always modifies the graph to make it closer to a Delaunay triangulation. We have also shown, that add neighbor operations will continue to be done until the graph is the Delaunay triangulation. We now need only show that the algorithm forms the Delaunay triangulation in a finite number of operations; this is to prevent argument that the algorithm only asymptotically forms the Deluanay triangulation.

We now show that a node can only add another node as a neighbor once. Consider a node a which might add b as a neighbor for a second time. If b is currently a neighbor, it will not be added again. If b was a neighbor in the past and is not now, the Voronoi region of a, in the local Delaunay triangulation, was made at some time not to touch the perpendicular bisector of a and b. Recall that both the Voronoi region of a in the local Delaunay triangulation is the same as a's perceived Voronoi region and that, because the add neighbor operation progresses, the perceived Voronoi region of a can only get smaller. Thus, if a's

Voronoi region did not touch the perpendicular bisector of a and b in the past, it will never touch it, and b will never be added as a neighbor. Thus, a node can only add another node as a neighbor once.

Since a node can only add another node as a neighbor once, and we have at most $n^2 - n$ pairs of node, we know the Delaunay triangulation will be formed after at most $n^2 - n$ add neighbor operations. By examination the lemmas, we can conclude that an add neighbor operation will occur after at most a constant number of messages. Thus, in any situation complying with our assumptions, the GridCast algorithm will form the Delaunay triangulation in finite time.

Quod Erat Demonstrandum

10 Removing a Neighbor

In the proof, we assumed a neighbor was never removed from a node's neighbor table. Here, we make an argument that if a node does attempt to leave or stops performing operations, the logical topology will form the Delaunay triangulation of the remaining nodes.

When a node attempts to leave or stops operating, it ceases to send any Hello or GoAway messages. After roughly $t_{timeout}$ seconds, all of its neighbors will remove it from their neighbor tables. At this time, the node will be removed from the logical topology. By our preceding proof, the remaining nodes will form the Delaunay triangulation.

If the leaving or failing node happened to be the Leader, after roughly $t_{timeout}$ seconds, the server will remove it as the Leader. The Leader of the remaining nodes will have removed the leaving or failing node from its neighborhood and contacted the server. From then on, the server has the correct address of the current Leader and the Delaunay triangulation will form.

Thus, given sufficient time, the algorithm will form a Delaunay triangulation of the currently active nodes.