

Georouting and Delta-gathering: Efficient Data Propagation Techniques for GeoSensor Networks

Dina Goldin, Mingjun Song, Ayferi Kutlu, Huayan Gao, Hardik Dave

Dept. of Computer Science and Engineering
University of Connecticut
Storrs, CT 06269, USA

ABSTRACT

We consider the issue of *query and data propagation* in the context of geosensor networks over geo-aware sensors. In such networks, techniques for efficient propagation of queries and data play a significant role in reducing energy consumption.

Georouting is a new technique for the broadcasting of *localized data and queries* in geo-aware sensor networks; it makes use of the existing query routing tree, and does not involve the creation of any additional communication channels. In addition to localized broadcasting, georouting is useful for (non-localized) broadcasting *spatial data*, greatly reducing the amount of communication, and hence energy consumption, during broadcasts. We demonstrate its effectiveness empirically, having implemented this technique.

In addition to broadcasting queries and data to the sensors, we consider *data gathering*, where data is being transmitted from the sensors back towards the central processor. *Delta-gathering* is a new technique for reducing the amount of communication during data gathering.

Finally, we apply our delta-gathering approach toward the problem of sensor data *visualization*. We present *sensor terrains* as a preferable alternative to isoline-based visualization (*contour maps*) for this problem.

1 INTRODUCTION

Sensor networks can be embedded in a variety of geographic environments, such as high-rise buildings, airports, highway stretches, or even the ocean. They enable the monitoring of these environments for a wide variety of applications, from security to biological. For many of the anticipated applications, the ability to query sensor networks in an *ad hoc* fashion is key to their usefulness. Rather than re-engineering the network for every task, as is commonly done now, *ad hoc querying* allows the same network to process any of a broad class

of queries, by expressing these queries in some query language. In essence, the network appears to the user as a single distributed agent whose job it is to observe the environment wherein it is embedded, and to interact with the user about its observations.

Unlike traditional database applications, where spatial considerations are often irrelevant (except as expressed by traditional attributes such as *address* or *zip code*), it is believed that most applications of sensor networks, in such diverse fields as security, civil engineering, environmental engineering, or meteorology, will involve queries that combine *spatial data* with *streaming sensor data*. For this reason, we are focusing our investigation on a query system that combines a *spatial database* [26] with a *geo-aware sensor network* [11] SPASEN-QS for short. There are currently several research projects, including those at Berkeley [22, 23, 25] and Cornell [34, 35] dealing with query issues in sensor networks. However, we are not aware of any other projects that have focused on sensor network querying for spatial data.

As is common for the sensor network query setting, SPASEN-QS architecture involves a *central processor* which hosts the spatial data and provides a user interface to the query system. A *routing tree* is maintained over the sensors, whose root communicates directly with the central processor. All communication is therefore *vertical*, either down from the central processor towards the sensors (*broadcasting*, or *distributing*) or up from the individual sensors towards the central processor (*gathering*, or *collecting*).

Sensors are expected to run battery-powered and unattended for long periods of time, hence the need to minimize their *energy consumption*. Energy consumption therefore serves as the *optimization metric* for sensor network computations, analogous to *time and space complexity* in traditional computation.

Of the four types of sensor activities (*transmitting*, *sensing*, *receiving*, *computing*), the first is the most *expensive* in terms of energy consumption. Efficient techniques for the propagation of queries and data in sensor networks play a significant role in reducing energy consumption for sensor network computation.

In this paper, we consider the issue of query and data propagation in geosensor network query systems such as SPASEN-QS. *Georouting* and *Delta-gathering* are the two techniques we propose.

Georouting is a new technique for localized broadcasting of queries in geo-aware sensor networks; it makes use of the existing query routing tree, and does not involve the creation of any additional communication channels. Besides localized query broadcasting, georouting is also useful when broadcasting spatial data, greatly reducing the amount of communication, and hence energy consumption, during broadcasts. We have implemented georouting, and demonstrate its effectiveness empirically.

In addition to broadcasting queries and data to the sensors, we consider *data gathering*, where data movement is reversed towards the central data manager.

Delta-gathering is an new technique for reducing the amount of communication during data gathering. The goal of delta-gathering is to improve power consumption of the sensor network by reducing the amount of communication at the gathering phase. In the absence of a new value from some sensor, unless we know that the sensor is down, we assume that the value at this sensor has not appreciably changed since the last transmission, and is not worth transmitting. Note that this technique does not affect the *semantics* of the data, only the method of gathering.

We apply delta-gathering toward the problem of *sensor data visualization* via *sensor terrains*. Sensor terrains are a preferable alternative to isoline-based visualization [12]. They are represented by *triangulated irregular networks* (TINs). Visualization of sensor terrains is therefore a special case of *dynamic TIN generation*, a computational geometry problem for which we present a new incremental delta-based algorithm.

At any given time t , each sensor in the network corresponds to a point (x, y, z) , where (x, y) is the location of the sensor and z is its reading at time t . A *sensor terrain* is a surface which passes through all these sensor points. As the readings change, so does the sensor terrain; it is *dynamic*, more like a video than a static surface. There are several reasons to prefer sensor terrains to contours as the means of sensor data visualization: *more intuitive, less lossy, greater manipulability, easier updates*. These are discussed in section 3.

We represent sensor terrains by *triangulated irregular networks* (TINs) [7]; An alternative representation are NURBS [27]. For sensor data visualization, we must continuously regenerate the TIN corresponding to the dynamic sensor terrain. *Efficient dynamic TIN generation* is a new computational geometry problem for which we present an *incremental* $O(\log n)$ algorithm.

Given a sensor terrain, a contour map can be computed from it (but not vice versa). We therefore conclude by presenting a new efficient algorithm for dynamically generating isolines from the sensor terrain.

Outline. We discuss georouting in section 2, sensor terrains in section 3, and isoline extraction in section 4. We conclude in section 5.

2 GEOROUTING

In this section, we discuss *georouting*, a new technique for localized broadcasting of queries in geo-aware sensor networks. In addition to localized query broadcasting, georouting is also useful when broadcasting spatial data, greatly reducing the amount of communication during broadcasts. We demonstrate its effectiveness empirically, and show that the use of special trees customized for georouting do not offer significant advantages over the existing routing tree.

2.1 Localized Broadcasting

In geospatial sensor networks, the data or the queries to be broadcast are often *localized*, i.e. of relevance only to those sensors located within a specific geographic region. When the information to be broadcast is spatial, the geolocation of the sensor often determines whether this information is relevant to it. For example, if a query needs to initialize sensors that are located within a given region X , then this operation is not relevant to those sensors which fall outside X ; moreover, if all the sensors in a given subtree of the routing tree are outside of X , the information about X need not be routed to that subtree at all. Since communication consumes a large fraction of a sensor network's energy [33, 4], it is desirable to avoid unnecessary routing of spatial information.

Previous work on constraining the broadcasts to a geographic area include work in *geoaware routing* [14, 36], *directed diffusion* [13], rumor routing [1]. These algorithms were developed outside the sensor network querying context; they do not use a routing tree, relying on localized neighbor selection to efficiently route a packet to a destination. In contrast to these approaches, georouting relies on the existing *routing tree* for *all* communication. Specifically, it tags each node of the routing trees with *bounding box* information for itself and all its children. Furthermore, neither directed diffusion nor rumor routing make any use of geoinformation. Whereas the gradient information allows the localization of the sink node, messages in the opposite direction (from the sink) cannot be localized and involve a broadcast to all the nodes. *SRT trees* [22] have also been used for localized broadcasting, and are the most alike georouting trees. Both SRT and georouting trees involve decorating the existing query routing tree with additional information, without creating any additional communication channels. However, SRT trees store exactly one interval per attribute per node, whereas georouting trees store the intervals of each child as well. This results in much greater communication efficiency during localized broadcasts.

In addition, georouting is completely *decentralized*; the route is computed in-network rather than at the central processor. This is accomplished by augmenting the routing tree to make it geo-aware: at each internal node, the spatial bounding box of each child is stored; this bounding box is used during the routing to minimize unnecessary communication. We discuss the details of this algorithm in the next section.

2.2 Georouting Tree

Routing trees are more attractive for sensor network querying than in the standard network setting, due to the following three points of contrast between these settings:

- Normally, the sensor nodes serve strictly to route messages, with no in-network processing. In SNQ, there is in-network processing performed at the sensors to optimize query evaluation. Hence, SNQ nodes need to choose a single parent when routing data towards the sink, rather than send the same message to multiple candidate parents.
- Normally, the *sink* node, towards which the message is routed, changes often and a single tree routed at the sink cannot be maintained for long. In SNQ, a fixed root is assumed, which serves as the sink throughout the continuous evaluation of the query.
- While conversations in regular sensor networks between a source and a sink are short-lived (just long enough to send all the packets), sensor network queries are long-lived. They can perform monitoring functions over days if not months, during which time we must collect data continuously over the same path.

For the above reasons, a single routing tree that can be maintained over time, is the most suitable approach to routing in the case of SNQ.

Georouting trees augment routing tree architecture by maintaining at each sensor X a *bounding box* for each child Y of X , where a bounding box for Y encloses the geo-locations of all the sensors in the routing subtree rooted at Y . The bounding box of X is defined recursively as the maximum bounding rectangle of the bounding boxes for all of X 's children, and the bounding box for each leaf node is simply its geo-location coordinates.

The algorithm for building the georouting tree is described next, based on original routing tree algorithms in [22, 23].

Algorithm for building the georouting tree:

1. (Assign levels top-down.) We assign a level to each node according to its distance from the root, starting by assigning 0 to the root itself. Given a current node A at level k in the tree, any node B within A 's sensing range is assigned level $k + 1$ and added to the list of A 's *candidate children*, unless it has already been assigned level k or less. Note that a node may be the candidate child of several nodes, each of which will be its *candidate parent*.
2. (Select the parents and compute the bounding boxes bottom-up.) Starting from the leaf nodes, we select one parent for each node, out of its list of candidate parents. We always select the geographically nearest node as the parent. Once a node's parent is chosen, we remove this node from the candidate children list of all other candidate parents.
3. (Assign the bounding box.) This operation is also done recursively, at the same time as step 2 (parent selection). First, assign the bounding box of

all leaf nodes to be their coordinates points and then go up to the root, calculate the bounding box of each node as the minimum rectangle which includes the bounding boxes of all its children. Store the bounding boxes of the children in the parents.

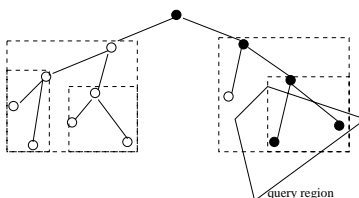


Figure 1: Message broadcast in georouting tree

After building the georouting tree, the bounding box information at each internal node is used to *filter out* queries; the query is only transmitted to those children whose bounding boxes overlap with it. This is illustrated in figure 1. In this figure, the query region is on the right, and the bounding boxes are shown in dashed lines; the sensors where the query was routed are filled in, while the ones where the query was filtered out are white.

2.3 Georouting Tree Maintenance

Although in our setting we assume that the sensor nodes are not mobile, we cannot assume that the routing tree will stay constant over the duration of a query. This is due to the inherently dynamic nature of sensor networks, involving node failures, new nodes joining the network, etc. In this section, we analyze the communication cost of georouting tree updates. We do *not* consider here the costs incurred by the maintenance of the routing tree itself, but only on the additional costs needed to properly maintain the the bounding box information associated with the georouting tree.

Whenever a node joins or leaves the network, the geourouting tree needs to be updated; the update operations are *insert* and *delete*, respectively. For each operation, the bounding box of the node's parent needs to be recomputed. If the parameters of the parent's bounding box are changed, the parent's parent also has to be recomputed, and so on. Furthermore, if a non-leaf node fails, its children have to find new parents whose bounding boxes must be recomputed in a similar fashion.

In the best case, when a leaf node fails and its parents' bounding box is not affected, no messages may be needed to "repair" the tree. As soon as the parent node detects that it has not heard from its child for a period of time, it will remove that child's bounding box from its own without any messages

involved. This is due to the fact that a georouting tree node stores all of its children's bounding boxes locally. (For more information on how parents may detect the loss of a child, we refer to [23]). However, for an insert operation, there is at least one message involved, since the location of the new node must be communicated to its parent.

Let the parameter k represent the communication cost, for a random node s , of repairing all its ancestors in case of s 's failure; $0 \leq s \leq d$, the depth of the tree. If the node to be deleted has children, the total communication costs are greater than k : the failure not only affects s 's ancestors, but also the future ancestors of its children, who now need to select new parents. Each child needs at least one message to transmit its location to its new parent, plus k possible messages to propagate that change. The cost for each child is therefore the same as in case of *insert*, i.e. $k + 1$. The total cost for a deletion is therefore $k + c(k + 1)$, where c is the number of children of a failed node; the total cost for an insertion is $k + 1$.

To evaluate the communication cost of georouting tree updates, we performed an experiment to measure the following:

When a random sensor node s is removed from the georouting tree, what is the average number of messages needed to update the tree?

This corresponds to $k + c(k + 1)$ in the above analysis.

Our experimental setting consisted of 1000 sensors with randomly assigned locations in a 100×100 area; the sensing range varied from 10 to 50, in steps of 5. After creating a georouting tree with a given sensing range, we simulated failure of a randomly chosen node by removing it from the tree, and performed a tree update, counting the number of messages. This number was averaged over many trials, to obtain the average total cost of deletion in a georouting tree.

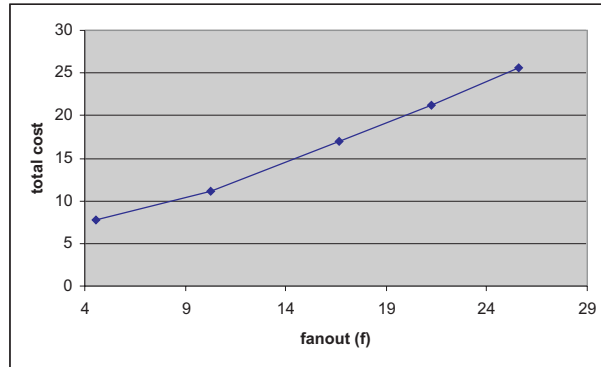


Figure 2: The cost of deletion in a georouting tree.

Figure 2 plots this cost against the *fanout* of the tree, i.e the average number of children per internal node. We achieved higher fanouts by increasing the range while keeping the number of sensors fixed.

We conclude this section by noting that, in order to obtain communication savings from a georouting tree, it must be the case that tree updates do not occur too frequently. Specifically, if the expected cost of an update is c_u and the expected savings per epoch are c_s , then updates should occur on the average less than once per c_u/c_s epochs. We expect that this will be the case for many applications.

2.4 Experimental Results For Georouting

Having analyzed the costs associated with maintaining the georouting tree, we now consider the communication savings associated with georouting. In this section, we discuss an experiment that we have performed to access the performance of georouting, when compared either with SRT trees or with regular broadcasting. We report very significant savings, when compared with either of the other methods.

After choosing a fixed range of $(0, 100)$ in both x and y directions as the coordinate space of our “world”, we randomly generated 1000 pairs of values in this range to simulate the positions of sensors. We then constructed a georouting tree over these sensors, with the root in the center of the world. Figure 3, generated automatically by our simulation, shows the georouting tree we obtained; here, the sensing range is set at 10 units.

We then simulated 500 localized broadcasts over this sensor network. For each broadcast, a rectangle was used to approximate the spatial region of interest (*query box*); this query box was generated randomly and propagated down the georouting tree. Figure 3 shows one such query box on the left; the paths involved in this broadcast are shown with thicker lines. Note that not all of these paths lead into the query box; some of them lead to nodes outside the query box, whose bounding boxes overlap the query box.

For each broadcast, the number of *hops* was measured and plotted against the number of sensors in the query box; figure 4 shows the resulting plot.

Analysis. We define *georouting efficiency* as the ratio between the minimum number of necessary hops from the root to all sensors in the query box and the number of hops used in georouting. We calculated that over 500 queries, the average number of necessary hops was 192, whereas the average number of actual hops was 229. Therefore, the efficiency is:

$$192/229 * 100\% = 84\%.$$

We ran exactly the same set of experiments using an SRT tree instead of a georouting tree. That is, each node only stored its own bounding box and not

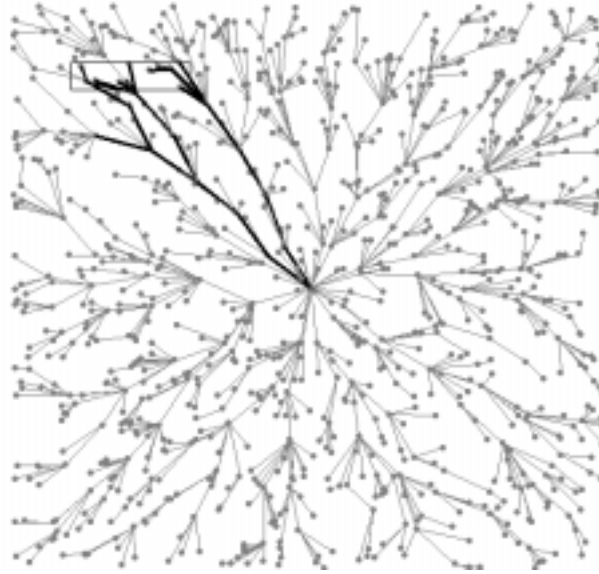


Figure 3: Georouting tree for our simulation

the ones for its children. As a result, the average number of hops was 305, and the efficiency is much lower:

$$192/305 * 100\% = 63\%.$$

The above analysis measures how far georouting is from optimal routing. We can also compare georouting to regular tree routing, and measure what percentage of hops was saved. Regular tree routing would always result in 999 hops (one for every edge in the routing tree), whereas the average number of hops for our system was 229. Therefore, the percentage of hops saved is:

$$(999-229)/999 * 100\% = 77\%$$

Again, this is a significant improvement over the results for SRT routing:

$$(999-305)/999 * 100\% = 69\%$$

Furthermore, this saving can be compared with the cost of georouting tree updates in case of node failure or a new node joining the network. While that cost depends of the fanout (section 2.3), it is clear from our experiments that the savings with even a single broadcast of a localized query are greater than the cost of multiple updates to the tree.

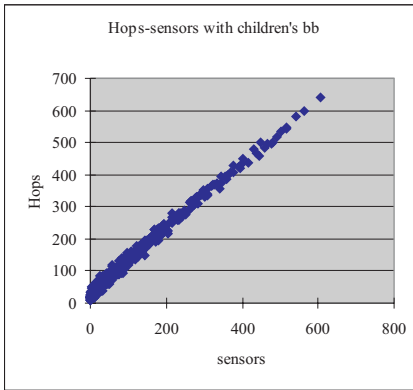


Figure 4: Simulation results

2.5 Selective Filtering During Broadcasts

In this section, we discuss application of georouting to spatial data broadcasts; in this case, the benefits of georouting apply even when the broadcast is not localized.

When the data being broadcast is a spatial relation, consisting of many spatial features each with its own geographic extent, only a subset of this relation may be relevant to any given sensor node for its computation. When the broadcast is not localized, simple *boolean filtering*, that decides whether to transmit the data to this sensor or not, does not reduce the amount of communication involved in the broadcast. Instead, we can use *selective filtering*, that decides how much of the data to transmit, if any.

To perform selective filtering in georouting trees, we compute the intersection of the sensor's bounding box and the bounding boxes of the spatial features that are candidates for transmission; only those features that intersect the sensor's box are transmitted. This is illustrated in figure 5.

3 SENSOR TERRAINS

In this section, we discuss *delta-gathering*, a technique for reducing communication during data gathering. We then apply our delta-gathering approach toward the problem of sensor data *visualization*. We present *sensor terrains* as an important alternative to isoline-based visualization (*contour maps*).

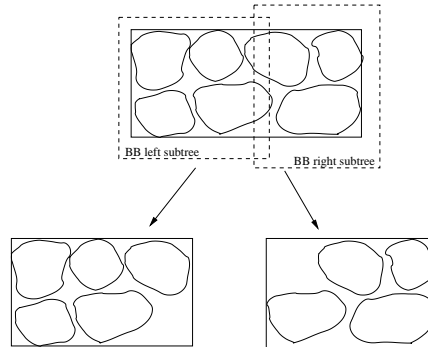


Figure 5: Selective filtering of spatial data in georouting tree

3.1 Delta-gathering

For many types of sensor readings, such as *temperature* or *pressure*, there is very little change in value from one epoch to the next. Rather than transmit the readings of all sensors at all times, we only need to transmit readings when there has been sufficient change. In this section, we introduce a new technique to accomplish this, called *delta-gathering*.

Delta-gathering is not to be confused with *delta compression* [23, 29], a related technique. In *delta compression*, we transmit a new value only when the change from the last transmitted value is above some threshold. Delta compression is performed *explicitly*, by specifying the threshold and storing the old value for comparison. This can be done either directly in the query (TinyDB) or with a built-in function (CQL):

TinyDB query with delta compression:

```
SELECT light
FROM buf, sensors
WHERE |s.light - buf.light| > t
OUTPUT INTO buf
SAMPLE PERIOD 1s
```

CQL query with delta compression:

```
SELECT Istream(delta_compr(light))
FROM Sensors
WHERE location = 'NEST-1012'
```

As a result of delta compression, the number of data elements in the stream is reduced. For example, the adjacent values in the output streams of the above queries are guaranteed to differ by more than the tolerance value.

Our new alternate approach, *delta-gathering*, does not involve the difference operator. Instead we are only interested in those values which represent “crossing a threshold”.

Delta-gathering:

Let J be the set of *threshold* values. Let x be the last transmitted value, and y be the current sensor reading; w.l.o.g., assume that $y > x$. y is transmitted only if the interval $(x, y]$ (which excludes x but includes y) contains some value in J .

For example, if the thresholds J consist of multiples of 1, and the latest transmitted value was 2.3, then only the last value in the following sequence will be transmitted: 2.5, 2.7, 2.9, 3.1. Note that $3.1 - 2.3 = 0.8$, which is less than 1.

The goal of delta-gathering is to improve power consumption of the sensor network by reducing the amount of communication at the gathering phase. In the absence of a new value from some sensor, unless we know that the sensor is down, we assume that the value at this sensor has not appreciably changed since the last transmission, and is not worth transmitting.

Unlike delta compression, this technique does not affect the *semantics* of the data, only the method of gathering.

The data is not compressed; we acknowledge that the untransmitted reading exists and should be part of the data, but we assume that the last transmitted value provides a sufficient substitute for it. This assumption is important for sensor data mining applications such as data visualization, discussed next. When visualizing the data, we will continue displaying the latest known reading for every sensor, until we are notified that it has changed.

3.2 3D Visualization Of Sensor Readings

Good visualization of the streaming data produced in sensor networks will enable better monitoring effect of sensitive environmental parameters such as temperature, providing people capacity to respond to alarming changes and make instant decisions. Visualization with *isolines* has been considered so far [12]; we have chosen to use *sensor terrains* instead.

We represent a *sensor terrain* as a *triangulated irregular network* (TIN), which is a set of contiguous triangles without overlap. Its vertices are 3D points (x, y, z) where (x, y) is the location of a sensor and z is the reading at that sensor. The TIN representation is popular in *terrain mapping* [7] because of its capacity to represent terrains over irregularly scattered data points, such as the case here.

There are several reasons to prefer sensor terrains to contours as the means of sensor data visualization:

- *more intuitive*: 3D surfaces are cognitively easier than contour maps; for example, differences in height are directly recognizable whereas in isolines, values have to be interpreted
- *less lossy*: we can extract a contour map from the sensor terrain, but not vice-versa
- *greater manipulability*: graphic manipulations of sensor terrains, such as rotations or changes to shading, can further enhance our understanding of the data; this is not possible with isolines
- *easier updates* (for 2D TINs): if one sensor changes value, then only the z -coordinate of that point changes; by contrast the contour map requires more change

An alternative representation to TINs for terrains over irregularly scattered data points is NURBS [27]. This representation is more time consuming to generate and maintain. Another advantage of TINs is the ease of shading, and of extracting isoline information. To be precise, in sensor networks we have a *dynamic* version of TINs and NURBS, where the z values are continuously changing. As the sensor readings change, so does the terrain – it is more like a video than a static surface.

3.3 Dynamic TINs: Overview

There are three basic algorithms for constructing the triangulated representation of a sensor terrain [37]:

- *divide-and-conquer* [10] divides the original data sets into disjoint subsets and solves the subproblem recursively;
- *sweeping* [6] constructs valid Delaunay edges by sweeping the points upward one at a time;
- *greedy insertion* [10] inserts one site at a time into the triangulation and updates the triangulation by iteratively replacing the invalidated edges.

Based on whether the triangulation algorithm makes use of the z values (rather than just x and y), the algorithms are classified as *3D* (also known as *data-dependent*) or *2D* (also known as *data-independent*). In the *2D* case, the triangulation depends only on the sensor locations and not on their readings; in the *3D* case, it depends on the readings as well.

In the dynamic setting like ours, we assume that the TIN has already been computed, with one of the methods above; instead, we are concerned with *updates* to the TIN. There are three types of updates:

1. *modify value*: corresponds to a change in sensor reading
2. *insert vertex*: corresponds to a sensor joining the network
3. *delete vertex*: corresponds to a sensor leaving the network

The difference between 2D and 3D TINs is clearest in the case of the first type of update, *modify*; we are assuming *delta-gathering* (section 3.1), so presumably the reading has crossed a threshold. In the 2D case, we only need to modify the z attribute of one vertex; the triangulation stays the same. By contrast, in the 3D case the triangulation may change.

All updates to the sensor network are placed into an *update queue* at the central processor. They are processed one at a time, to maintain a dynamic TIN whose geometry visualizes the sensor terrain. To maintain the dynamic TIN in real time, two assumptions must be made. First, we assume that the number of updates per epoch is small. This assumption is made feasible by applying *delta-gathering*. Second, we assume that each update is computed very quickly, i.e. with time complexity $O(\log n)$, where n is the size of the network. In the next section, we discuss the algorithms that make it possible.

3.4 Efficient Updating Of TINs

In case of sensor networks, where the updates we must display the surface dynamically and in real time as the updates stream in. Therefore, we found 2D triangulation preferable for sensor networks; the triangulation is precomputed and fixed, until a new sensor needs to be added. For adding new sensors, we use the greedy insertion triangulation algorithm.

In this section, we describe the insertion algorithm for the TIN representation of sensor terrains; the *delete* operation is handled in a similar fashion. This algorithm is based on the algorithm for incremental site (vertex) insertion that is part of the *greedy insertion* triangulation algorithm for constructing a 2D TIN, found in [10].

Insert. Our *insert* algorithm for 2D triangulation closely follows the logic from [10]. Assuming that S is the new vertex to be inserted, it consists of the following steps:

1. **Locate** the triangle T where the vertex S will be located.
2. **Connect** the vertex S with each vertex of the triangle T .
3. **Initialize** the *list of suspect edges* to contain all the edges of T .
4. Remove a *suspect* edge from the list and **test** to determine whether it is *valid*.
5. If invalid, **replace** it with its *alternate*, adding new suspect edges to the list.
6. *Repeat* the last two steps while there are still suspect edges.

In [10], the invalid edges are identified with the *inCircle test*), which dictates that no vertex can be within the circumcircle of any triangle to which it does not belong.

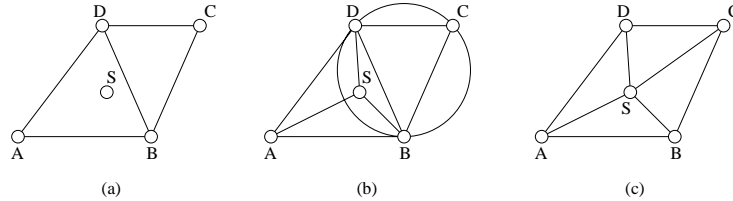


Figure 6: Incremental TIN update in 3 steps

Example. In figure 6 (a), S is the new site to be inserted, and we find that it lies inside the triangle ABD . In figure 6 (b), we connect S to these vertices and run the *inCircle test* for edges AB , AD and BD . We discover that the edge BD is invalid because S is located inside the circumcircle of BCD . In figure 6 (c), BD is replaced by SC . Note that we are not done. Now, BC and CD have become suspect and need to be checked; this procedure is repeated until all invalid edges are removed.

Bounded Change Propagation. As described above, the worse-case performance for *insert* is $O(n)$, due to *change propagation*: all the edges in the triangulation might need to be tested for validity. To ensure $O(\log n)$ performance, we adapted a *bounded change propagation* strategy: for each update, the maximum number of tested edges is bound at $c \log n$, where c is a constant defined outside our algorithm. With this strategy, the triangulation is no longer *correct* in all cases; hence, the dynamic TIN maintained by our system is *approximate* rather than *exact*. Note that our algorithm is *adaptive*: by increasing c , we can better approximate the correct TIN.

3.5 Simulation Of Sensor Terrain Update

We used a sensor terrain of 257 sensors, with coordinates whose x values were randomly distributed in a $[0, 9600]$ range and y values in a $[0, 10115]$ range (this range represented the UConn campus). For our sensor reading, we used actual data for the geographic terrain around the UConn campus, where the sensor readings represent the local height, which is from 0 to 420 feet, when adjusted.

Figure 7 shows the shaded TIN (a) before and (b) after a sensor in the lower left quadrant changed its value, from 350 to 149.49. One can clearly see the difference in the shape of the two sensor terrains.

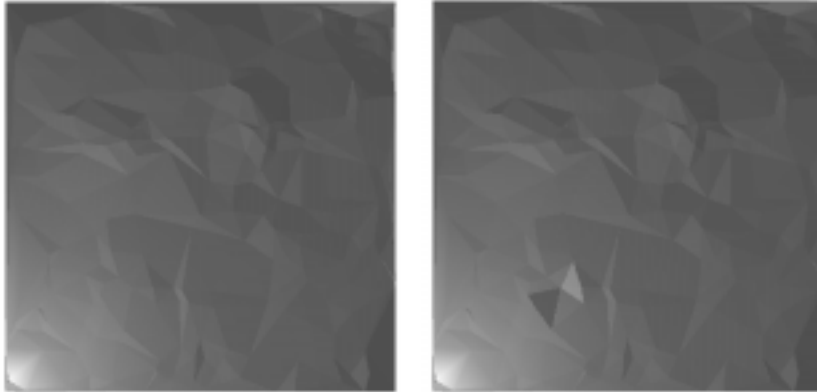


Figure 7: TIN update example: shaded image (a) before and (b) after update

4 DYNAMIC ISOLINE EXTRACTION FROM SENSOR TERRAINS

In section 3, we have presented *sensor terrains* as an important alternative to isoline-based visualization (*contour maps*). We have also shown how to maintain a dynamic sensor terrain by incremental updates. In this section, we discuss how to build and maintain a *dynamic contour map* from the dynamic sensor terrain.

We assume that the segments comprising the isolines in the contour map have been computed once from the TIN representing the sensor terrain. Our focus is on *updates* to the TIN, discussed in section 3.3, which necessitate updating the contour map accordingly. The goal is to maintain the TIN and the isolines in real time, for real-time *visualization* of the sensor network. One can imagine the contour map displayed together with the sensor terrain; both of them move on the screen to portray the current state of the sensor network.

For our algorithm, we assume that we can assess the triangles and vertices of the TIN in constant time. We are also assuming *delta-gathering* (section 3.1), so the vertices are only updated when their z value crosses some threshold. It is probably advisable if the set of thresholds for delta-gathering includes the isoline heights of the contour map that is being computed.

We will first present *interval trees*, a data structure that plays a central role in isoline extraction. Given a TIN, the interval tree is computed from this TIN; isoline segments are then computed from the interval tree.

4.1 Interval Trees

Every edge e in a TIN has a z -span, which is an interval indicating the minimum and maximum z values in e . Suppose the two end-points of some edge are e_0, e_1 , and their height values are r_0, r_1 respectively, where $r_0 \leq r_1$. The z -span for the edge would be $[r_0, r_1]$.

Let Z be the set of all the z -spans of a given TIN. Then, the *interval tree* over this TIN is a binary tree whose nodes are labeled with the following two attributes:

- some *split value* s
- the subset of Z consisting of those intervals that overlap s

Interval trees obey the following properties:

1. Given a node X with split value s , a z -span I of the form (a, b) is in the interval list of X if and only if $a \leq s \leq b$
2. If node Y is a left (right) child of node X , then the split value at Y is smaller (larger) than the split value at X .
3. If the tree has n nodes, then the depth of the tree is $O(\log n)$.

Our algorithm to extract an interval tree from a TIN is similar to the one in [17]; the major difference is that they have an interval for every *triangle* rather than *edge*. We found edges more convenient for our dynamic implementation.

Figure 8 (a) gives an example of a TIN; figure 8 (b) shows the corresponding interval tree. The lists of intervals are displayed twice, sorted first by start point and then by end.

4.2 Updating The Interval Tree After Change To Sensor Reading

A change to the value of any sensor in the network will affect the triangulation, and hence the set of its z -spans. The interval tree needs to be updated accordingly, so it continues to satisfy the three properties listed in section 4.1.

To update the interval tree, two operations may need to be performed:

1. *update the interval lists*: without changing the *split values* at any of the tree nodes, we modify the interval lists so the first property of interval trees is satisfied
2. *rotate*: without changing the attributes at any nodes, we rotate the interval tree to decrease its height

During the first step above, a new leaf node may have to be added if there are intervals that do not belong to the lists of any of the current nodes. Also, a node will be deleted if its list of intervals is empty.

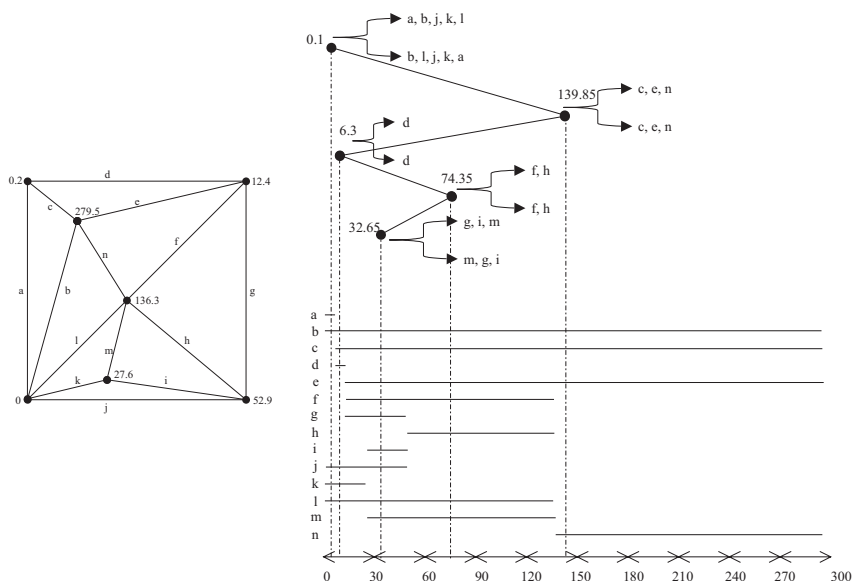


Figure 8: TIN (a) and corresponding tree (b)

Without going into the details of this step, we illustrate it in figure 9, where the sensor reading for the left middle sensor (figure 8 (a)) has changed from 136.3 to 170.4. This figure shows this changes the set of z -spans, and correspondingly the interval tree (before rebalancing). After changing, there are no longer any intervals that lie completely to the left of the root's split value 74.35. There is also a new leaf on the right, whose split value is 224.95. The time complexity of step 1 is $O(\log n)$, where n is the size of the interval tree.

Clearly, the tree in figure 9 is unbalanced. Figure 10 shows the same tree after a rebalancing (step 2). We use the AVL rebalancing scheme [31] for our interval tree updates, to obtain the overall time complexity of $O(\log n)$ for our algorithm.

Note that we can *defer* the rebalancing of the tree. That is, we assume that there exists a predetermined constant c such that step 2 is done only once out of every c times that step 1 is done. If the size of the interval tree is initially n , then the time complexity of AVL tree rebalancing after c updates is $O(c(\log(c + n)))$ [21].

Figure 11 shows the isolines, computed for the sensor terrain in figure 7 (a), then updated when a sensor in the lower left quadrant was changed from 350 to 149.49. The thick lines represents isoline values of 200 and 300, respectively. The change to the isoline contours is clearly visible.

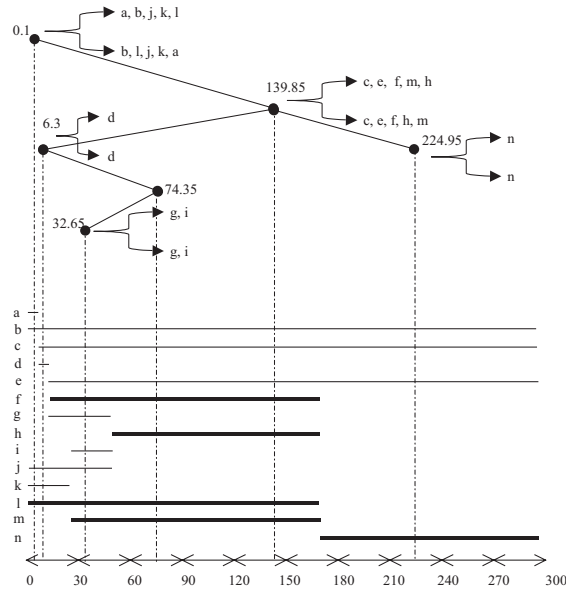


Figure 9: The interval tree after a change of value.

5 CONCLUSION AND FUTURE WORK

We have considered the issue of *query and data propagation* for geosensor network query systems, including our own system SPASEN-SQ. In such systems, techniques for efficient propagation of queries and data play a significant role in reducing energy consumption.

Georouting is a new technique for the broadcasting of *localized data and queries* in geo-aware sensor networks; it makes use of the existing query routing tree, and does not involve the creation of any additional communication channels. In addition to localized broadcasting, georouting is useful for (non-localized) broadcasting *spatial data*, greatly reducing the amount of communication, and hence energy consumption, during broadcasts. We demonstrated its effectiveness empirically, having implemented this technique.

In addition to broadcasting queries and data to the sensors, we considered *data gathering*, where data is being transmitted from the sensors back towards the central processor. *Delta-gathering* is a new technique for reducing the amount of communication during data gathering. We noted that unlike *delta compression*, a related technique, delta-gathering does not affect the *semantics* of the data, only the method of gathering.

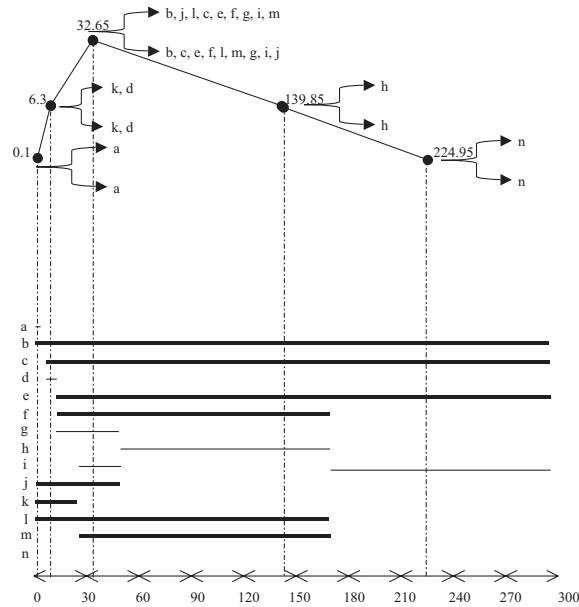


Figure 10: The interval tree after rebalancing.

Finally, we applied delta-gathering toward the problem of *sensor data visualization via sensor terrains*. Sensor terrains are a preferable alternative to isoline-based visualization (*contour maps*) for this problem. Sensor terrains are represented by *triangulated irregular networks* (TINs). Visualization of sensor terrains is therefore a special case of *dynamic TIN generation*, a computational geometry problem for which we present a new incremental delta-based algorithm.

Future work includes a real-time interactive sensor terrain and isoline visualization tool which relies on delta-gathering, built into SPASEN-SQ. We also plan to study *in-network* algorithms for the problems discussed above.

References

- [1] Braginsky, D. and Estrin, D., Rumor Routing Algorithm For Sensor Networks, *In Proc. First ACM Int'l Workshop on Sensor Networks and Applications (WSNA)*, Atlanta, GA, Sep. 2002.
- [2] Bertino, E., Guerrini, G. and Merlo, I., Trigger Inheritance and Overriding in an Active Object Database System, *IEEE Transactions on Knowledge and Data Engineering*, 12:4, pp. 588–608, 2000.

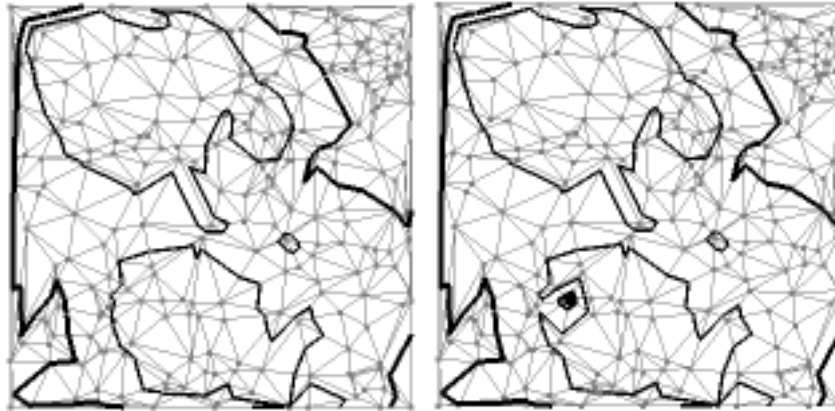


Figure 11: TIN update example: isolines (a) before and (b) after update

- [3] Cerpa, A. et al., Habitat monitoring: Application driver for wireless communications technology, *ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, Costa Rica, April 2001.
- [4] Chang, J-H. and Tassiulas, L., Energy Conserving Routing in Wireless Ad-hoc Networks, In *Proc. IEEE Infocom*, pp. 22-31, TelAviv, Israel, March 2000.
- [5] Elmasri, R. and Navathe, S., *Fundamentals of Database Systems*. Addison-Wesley, New York 2000.
- [6] Fortune, S., A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153-174, 1987.
- [7] De Floriani, L., Puppo, E. and Magillo, P., Applications of computational geometry to geographical information systems, Chapter 7 in *Handbook of Computational Geometry*, J.R. Sack, J. Urrutia (Editors), Elsevier Science, pp.333-388, 1999.
- [8] Garland, M. and Heckbert, P.S., Fast polygonal approximation of terrains and height fields, Technical Report CMU-CS-95-181, Carnegie Mellon University, 1995.
- [9] Gehani, N. and Jagadish, H.V., Ode as an Active Database: Constraints and Triggers, *Proc. 17th Int'l Conference on Very Large Databases*, 1991.
- [10] Guibas, L. and Stolfi, J., Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams, *ACM Transactions on Graphics*, 4(2):75-123, 1985.
- [11] Heidemann, J. and Bulusu, N., Using Geospatial Information in Sensor Networks, In *Proceedings of the Computer Sciences and Telecommunications Board (CSTB) Workshop on the Intersection of Geospatial Information and Information Technology*, Arlington, VA. October 1-2, 2001.

- [12] Hellerstein, J.M. et al., Beyond Average: Towards Sophisticated Sensing with Queries, *2nd Int'l Workshop on Information Processing in Sensor Networks (IPSN '03)*, March 2003.
- [13] Intanagonwiwat, C., Govindan, R. and Estrin, D., Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks, In *Proc. Sixth Annual International Conference on Mobile Computing and Networks*, August 2000, Boston, Massachusetts.
- [14] Karp, B. and Kung, H.T., Greedy Perimeter Stateless Routing for Wireless Networks, in *Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2000)*, Boston, MA, August, 2000, pp. 243-254
- [15] Kuper, G., Libkin, L. and Paredaens, J. (Eds.), *Constraint Databases*. Springer-Verlag Berlin Heidelberg, 2000.
- [16] Kulik, J., Rabiner, W. and Balakrishnan, H., Adaptive Protocols for Information Dissemination in Wireless Sensor Networks, *Proc. 5th Int'l Conf. on Mobile Computing and Networking*, Seattle, WA, 1999.
- [17] Van Kreveld, M., Efficient methods for isoline extraction from a digital elevation model based on triangulated irregular networks, In *Proc. Sixth Int'l Symposium on Spatial Data Handling*, pp.835-847, 1994.
- [18] Kung, H.T., Efficient Location Tracking using Sensor networks, *Proc. 2003 IEEE Wireless Communications and Networking Conference*.
- [19] Leach, G., Improving worst-case optimal Delaunay triangulation algorithms, In *Proc. 4th Canadian Conference on Computational Geometry*, 1992.
- [20] Li, Q. et al., Reactive Behavior in Self-reconfiguring Sensor Networks, *ACM MobiCom 2002*, September 2002.
- [21] Larsen, K.S., Soisalon-Soininen, E. and Widmayer, P., Relaxed Balance through Standard Rotations, *Workshop on Algorithms and Data Structures*, 1997
- [22] Madden, S.R. et al., *TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks*, *OSDI*, December 2002.
- [23] Madden, S.R. et al., The Design of an Acquisitional Query Processor for Sensor Networks, *SIGMOD*, June 2003, San Diego, CA.
- [24] McErlan, D. and Narayanan, S., Distributed Detection and Tracking in Sensor Networks, *36th Asilomar Conference on Signals, Systems and Computers*, 2002.
- [25] Madden, S.R. et al., Supporting Aggregate Queries Over Ad-Hoc Wireless Sensor Networks, *Workshop on Mobile Computing and Systems Applications*, 2002.
- [26] Rigaux, P., Scholl, M. and Voisard, A., *Spatial Databases* Morgan Kaufmann, 2001
- [27] Song, M., Goldin, D.Q. and Peng, T., NURBS Surface Interpolation for Terrain Modeling, to appear in *Proceedings of ASPRS/MAPPS 2003 Conference on Terrain Data*, October 2003, North Charleston, South Carolina.
- [28] Silberschatz, A., Korth, H. and Sudarshan, S., *Database System Concepts*. McGraw-Hill, New York, 2002.

- [29] The STREAM query repository, Stanford University.
- [30] Woo, A. and Culler, D.E., A Transmission Control Scheme for Media Access in Sensor Networks, *Proc. 7th Int'l Conf. on Mobile Computing and Networking*, Rome, Italy, July 2001.
- [31] Weiss, M.A., *Data Structures and Algorithm Analysis in C*, Addison-Wesley, 1997
- [32] Adjue-Winoto, W. et al., The design and implementation of an intentional naming system, In *ACM SOSP*, December 1999.
- [33] Xu, Y. and Heidemann, J., Geography-informed Energy Conservation for Ad Hoc Routing, *Proc. 7th Int'l Conf. on Mobile Computing and Networking*, Rome, Italy, July 2001.
- [34] Yao, Y. and Gehrke, J., The Cougar Approach to In-Network Query Processing in Sensor Networks, In *SIGMOD Record*, September 2002
- [35] Yao, Y. and Gehrke, J., Query Processing in Sensor Networks, *CIDR 2003*, January 2003.
- [36] Yu, Y., Govindan, R. and Estrin, D., Geographical and Energy Aware Routing: A Recursive Data Dissemination Protocol for Wireless Sensor Networks, UCLA Technical Report UCLA/CSD-TR-01-0023, May 2001.
- [37] Peter Su, Efficient parallel algorithms for closest point problems, Ph.D thesis, Dartmouth College, New Hampshire, 1994.