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

May 30, 2003

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 [RSV01] with streaming sensor data [HB01].

For this reason, we are focusing our investigation on a query system that combines a spatial database [RSV01] with a geo-aware sensor network, SPASEN-QS for short. There are currently several research projects, including those at Berkeley [MFHH02, MFHH03, MSFC02] and Cornell [YG02, YG03] 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. In addition to 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 can 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 [HHMS03]. 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 4.

We represent sensor terrains by triangulated irregular networks (TINs) [FPM99]; NURBS are an alternative representation [SGP03]. 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.** Triggers play a significant role in SPASEN-SQ, for multiple reasons. We discuss triggers and their role in geosensor networks in section 2. Then, we discuss georouting in section 3, sensor terrains in section 4, and isoline extraction in section 5. We conclude in section 6.

## 2 Triggers

In this section, we discuss triggers and events, which play an important role in SPASEN-SQ.

### 2.1 Triggers in active databases

In the relational database context, a trigger is a statement that the system executes automatically as a side effect of a database update. Triggers are often used to notify human operator that something unexpected has occurred, by raising an error flag or generating an action that creates visual notification. For example, in a relational company employee database, inserting an extra payment of salary to an employee will trigger some action notifying the payroll manager of what has happened.

Triggers are also useful in automating certain functionality of the system. They often work as pre-packaged rules, to keep the database consistent and uniform [KLP00]. For example, a trigger could update each employee’s payroll account on December 31st to account for holiday bonus received.

Relational databases where triggers play a significant role are known as active [BGM00, GJ91]. The word “active” refers to the capability of these databases to automatically react to events which change the state of the system, such as database updates or specific values of the clock.

### 2.2 Triggers in sensor networks

In sensor networks, triggers can be defined analogously, as statements that are executed automatically when there is change to the state of the system. However, unlike the closed-box behavior of static relational databases, sensor networks are open, and their state is affected by changes to the physical environment wherein the system is embedded. As a result of these changes, events can occur without any user-initiated
action, and outside of user control. For this reason, triggers and events are expected to play a larger role in sensor network query systems such as SPASEN-SQ than they do for relational database systems.

Discussion of triggers and events for sensor networks can be found in [BE02, HHMS03, KV03, MFHH03]. In general, there is disagreement about the precise meaning of these terms; often, the distinction between events and triggers is unclear. In SPASEN-SQ, we consider an event as state change of interest, and a trigger as a rule specified within the system, to be activated by an event.

In the sensor network context, we assume continuous queries, which are constantly reevaluated as new sensor readings stream in. One common type of event for sensor network queries is query-driven; this type of event occurs when the query result assumes a specific value, or satisfies a specific constraint. An example is the ROOM-HOT event, when the temperature in some room of a building reaches over 80 degrees Fahrenheit. Another type of an event is interrupt-driven; it occurs when an external interrupt is reported within a system. An example is the BIRD-HOME event, when a bird returning to its nest activates a pressure gauge under the nest [MFHH03].

Triggers in sensor network query systems are typically used to initiate, terminate, or modify some continuous query. This allows them to play a strategic role in minimizing the energy consumption of the system. For example, given two continuous queries \( /D5/BD \) and \( /D5/BE \) where the latter is more detailed and more expensive, triggers can make sure that \( /D5/BD \) is run instead of \( /D5/BE \) unless there is interesting data to be reported, whereupon they switch.

Typically, events within geosensor networks are associated with a specific geographic location. In sensor network query systems such as SPASEN-SQ, these events often trigger localized queries, to be performed over a specific subarea of the network, or otherwise cause broadcasts of localized data of interest to only a subset of sensor nodes. In the next section, we discuss georouting, a technique for minimizing communication in localized broadcasts.

3 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.

3.1 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 geo-location 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 [XH01, CT00], 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* [KK00, YGE01], *directed diffusion*, and rumor routing [BE02]. is another form of localized broadcasting where the sensors are not geo-aware. These algorithms, developed outside the sensor network querying context, do not use a routing tree, relying on localized neighbour selection to efficiently route a packet to a destination. Furthermore, neither directed diffusion nor rumour routing are geoaware, so they cannot take advantage of sensor location.

In contrast to these approaches, georouting relies on the existing *routing tree* for all communication, and does not involve the creation of any additional communication channels. *Routing trees* [MFHH02, WSBL99] are a common technique for broadcasting information through a distributed network of sensors, and for collecting data from the sensors; the root of the routing tree connects this network to the server (central processor). A number of routing protocols [WC01, IGE00, KRB99] is suitable for building routing trees.

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.

### 3.2 Georouting tree

*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 [MFHH02, MFHH03].

**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 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](image)

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.

### 3.3 Experimental results for georouting

We now describe an experiment that we have performed to access possible communication savings with georouting.

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. Figure 2, generated automatically by our simulation, shows the georouting tree we obtained; here, the origin \((0,0)\) acts as the root node, and the sensing range is set at 10 units.

Finally, we 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 2 one such query box on the right; the hops for this broadcast are shown with thicker lines. For each broadcast, the number of hops was measured and plotted against the number of sensors in the query box; figure 3 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 for georouting. We calculated that over 500 queries, the average number of necessary hops was 118, and the average number of actual hops was 137. Therefore, the efficiency is:

\[
\frac{118}{137} \times 100
\]
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 137. Therefore, the percentage of hops saved is:

\[(999-137)/999 \times 100\]

This is the same number as before, which we believe to be a coincidence.

3.4 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 4.

We conclude our discussion of georouting by noting the similarity between georouting trees and semantic routing trees (SRTs) [MFHH03] used for in-network aggregation [MFHH02]. In both trees, the internal
nodes store intervals of values ($x$— and $y$—coordinates in our case), from minimum to maximum; these intervals are used to determine if a given new value needs to be propagated any further. However, there are major differences:

- the direction of propagation for SRT is the reverse of ours
- in SRT, there is only one interval being stored per node
- as discussed above, georouting trees can perform selective filtering whereas SRTs are only capable of boolean filtering
4 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).

4.1 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 [MFHH03, STREAM], 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.

4.2 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 [HHMS03]; 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 [FPM99] 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 [SGP03]. 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.
4.3 Dynamic TINs: Overview

There are three basic algorithms for constructing the triangulated representation of a sensor terrain [FPM99]:

- **divide-and-conquer** [GS85] divides the original data sets into disjoint subsets and solves the subproblem recursively;
- **sweepline** [For87] constructs valid Delaunay edges by sweeping the points upward one at a time;
- **greedy insertion** [GS85] 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 and the resulting triangulation are classified as 3D (also known as *data-dependent* triangulation) 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 (either 2D or 3D) 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 leaving the network
3. **delete vertex**: corresponds to a sensor joining 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 4.1), 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.

4.4 Efficient updating of dynamic 3D TINs

In this section, we describe the algorithms for *modify* and *insert*, two of the incremental update operations for the 3D TIN representation of sensor terrains; the *delete* operation is handled in a similar fashion. Our algorithms are based on the algorithm for incremental site (vertex) insertion that is part of the *greedy insertion* triangulation algorithm for constructing a 3D TIN 4.3, found in [GS85].

**Insert.** Our *insert* algorithm for 3D (data-independent) triangulation closely follows the logic from [GS85]. 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 [GS85], 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. Our algorithm replaces the inCircle test by an error comparison from [GH95]. In this case, an edge is considered invalid if its alternate has a lower error, as in the example below.

**Example.** In figure 5, $S$ is the new site to be inserted, and we find that it lies inside the triangle $ABC$. In figure 6, we connect $S$ to these vertices and run the inCircle test for edges $AB$, $AD$ and $BD$. We discover

![Figure 5: Incremental TIN updates: insert, step 1](image1)

![Figure 6: Incremental TIN updates: insert, step 2](image2)

![Figure 7: Incremental TIN updates: insert, step 3](image3)
that the edge $BD$ is invalid because the triangulation $SCD$ and $SBC$ yields a lower error than triangulation $SBD$ and $BCD$. In figure 7, $BD$ 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.

**Modify.** The modify algorithm is very similar to the insert algorithm above. Only step 2 is changed:

1. **Locate** the polygon $T$ where the vertex $S$ is located.
2. **Change** the $z$ value of the vertex $S$.
3. the rest is the same as before...

Note that $T$ might not be a triangle any more (if other updates took place since $S$ was inserted) so the number of suspect edges in the initial list may be larger than three.

**Bounded Change Propagation.** As described above, the worse-case performance for insert and modify if $O(n)$, due to change propagation: all the edges in the triangulation might need to be tested for validness. To ensure $O(\log n)$ performance, we propose 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.

### 4.5 Simulation of sensor terrain update

We did a simulation using nine sensors with the following readings:

<table>
<thead>
<tr>
<th>SensorID</th>
<th>$x$</th>
<th>$y$</th>
<th>Reading</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>320</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>310</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>2</td>
<td>340</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
<td>390</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>3</td>
<td>360</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>7</td>
<td>520</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>9</td>
<td>670</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>8</td>
<td>530 → 630</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>9</td>
<td>620</td>
</tr>
</tbody>
</table>

When sensor 8 changes its reading from 530 to 630, the sensor terrain changes. Figure 8 shows the shaded TIN before this change (the scene is rotated for watching convenience).

Figure 9 shows the same TIN after the change to sensor 8. The difference in the shape of the two sensor terrains is very clear.

### 5 Dynamic isoline extraction from sensor terrains

In section 4, 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.
Just as for sensor terrains, we assume that the segments comprising the isolines in the contour map have been computed once from the TIN representing the sensor terrain; an algorithm to extract a contour map from a TIN can be found in [Kre94]. Our focus is on updates to the TIN, discussed in section 4.3, which necessitate updating the contour map accordingly. The goal is to maintain the TIN and the isolines in real time, so they reflect the real-time changes to the sensor network. One can imagine the contour map displayed together with the sensor terrain as part of the same data visualization; both of them move on the screen to show 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 4.1), 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.
5.1 Interval Trees

Every triangle \( T \) in a TIN has a \( z \)-span, which in an interval indicating the minimum and maximum \( z \) values in \( T \). It can be computed in constant time, from \( T \)’s vertices. 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 \( O(\log n) \).

An algorithm to extract an interval tree from a TIN can be found in [Kre94]. We will not repeat it here.

![Figure 10: An example of a TIN.](image)

Figure 10 gives an example of a TIN; figure 11 show the corresponding interval tree. The lists of intervals are displayed twice, sorted first by start point and then by end.

5.2 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 5.1.

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.

Without going into the details of this step, we illustrate it in figure 12, where the sensor reading for the left middle sensor (figure 10) has changed from 5 to 8. This figure shows this changes the set of z-spans, and correspondingly the interval tree (before rebalancing). There are no longer any intervals that lie completely to the left of the root’s split value 7.5, so there are no nodes there. There is also a new leaf on the right, whose split value is 8.5. The time complexity of step 1 is $O(\log n)$, where $n$ is the size of the interval tree.

Clearly, the tree in figure 12 is unbalanced. Figure 13 shows the same tree after a rebalancing (step 2). We use the AVL rebalancing scheme [Wei97] 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 $O(c(\log(c + n)))$ [LSW97].

6 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.

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

Figure 13: The interval tree after rebalancing.


Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks

Greedy Perimeter Stateless Routing for Wireless Networks

Karp, B. and Kung, H.T., Greedy Perimeter Stateless Routing for Wireless Networks


Philippe Rigaux, Michel Scholl, Agnes Voisard. Spatial Databases Morgan Kaufmann, 2001


The STREAM query repository, Stanford University.


Yong Yao and Johannes Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks. In SIGMOD Record, September 2002