Efficient Processing of Nearest Surrounder Query for 3D Geospatial Data

Jaehwa Chung\textsuperscript{1} and Daewon Lee\textsuperscript{2*},
\textsuperscript{1} Dept. of Computer Science, Korea National Open University, Korea,
\textsuperscript{2} Division of General Education, Seokyeong University, Korea,
\textsuperscript{1}jaehwachung@knou.ac.kr, \textsuperscript{2}daelee@skuniv.ac.kr

Abstract

The Nearest Surrounder (NS) query is to find a set of all the visible objects that are not obstructed by other objects around the query location. Formally, for a given query $Q$ and dataset $D$, NS($Q$) returns a set of tuples, $\forall \in \{O, \theta, \varphi\}$ if and only if (i) $O \in R$ and $O$ is visible to $Q$, and (ii) $O \in (R \setminus \{O\})$, $\text{dist}(Q, O)$ at angular range $[\theta, \varphi]$, where $i \neq j$. Using the R-tree index, NS query processing algorithm is based on the not only distance bounding properties but also angle-based bounding properties. Although NS query has a wide spectrum of applications, such as surveillance and augmented reality services, the existing work is not able to effectively support the 3D geospatial environments. Motivated by the weaknesses, we suggest a solution, termed 3D Nearest Surrounder (3DNS) query, to maintain NS query result from a dataset of 3D geospatial objects. In this paper, we propose heuristics for 3DNS query based on R-tree spatial index without pre-computing the visible region.

Keywords: Nearest Surrounder, 3D Geospatial Data, Spatial Query

1. Introduction

Over the past decades, miniaturization of computing devices and the applications with communication technologies change the way consuming information. Especially, the advancement of assistive technologies, such as GPS and gyroscope sensor, and seamless communication among wireless networks (e.g., 4G and WiMax) endows mobility to those computing devices. Based on those technologies, spatial query processing techniques have been researched that they return the geographically related information around the given location from the location dataset. Thus, users are able to obtain information with respect to their locations using the spatial queries in the form of the Location Based Services (LBSs). Among the various types of spatial queries, the Nearest Surrounder (NS) query \cite{1} is to find a set of all the visible objects that are not occluded by other objects from the query location. Formally, for a given query $Q$ and dataset $R$, NS($Q$) returns a set of tuples, $\eta \in \{O, \theta, \varphi\}$ if and only if (i) $O \in R$ and $O$ is visible to $Q$, and (ii) $O \in (R \setminus \{O\})$, $\text{dist}(Q, O)$ at angular range $[\theta, \varphi]$, where $i \neq j$.

Although the NS query provides a wide spectrum of applications, such as the augmented reality services on mobile devices, it is restricted to be implemented as the real services since the existing work \cite{1-3} on NS queries considers only two dimensional spaces. Figure 1 illustrates an example of NS query in 2D spaces and 3D spaces. The conventional NS query returns the query result including the $O, O_s, O_\alpha, O_\beta, O_\gamma$ and $O$, as visible objects as shown in Figure 1 (a). However, when we consider the three dimensional spaces, objects $O$ and $O_s$ can be visible depend on their heights and they are

\textsuperscript{*}Corresponding Author
visible over the objects $o_i$ and $o_j$, respectively as shown in Figure 1 (b). The visibility and its viewable range for 3D objects cannot be determined by the conventional NS query.

Motivated by the weaknesses, we propose a novel solution, termed 3D Nearest Surrounder (3DNS) query that finds every visible objects in 3D spaces. As depicted in Figure 1 (b), the search range of a 3DNS query draws half spherical spaces. In this paper, we propose an effective solution for 3DNS query based on R-tree spatial index without pre-computing the visible region. To the best of our knowledge, this paper is an initiative work that makes NS query pragmatic in 3D spaces.

Our contributions are summarized as follows:

- Address the issues of NS queries and provide an efficient solution with 3D geospatial objects dataset.
- Define the angular measurements in 3D spaces using the spherical coordinate system and revise the prune heuristics to cooperate with NS query processing algorithm for promoting NS query execution.
- Evaluate and analyze 3DNS query-processing algorithms with the various environmental settings. For the accurate performance evaluation, we varied object cardinality and volume of object.

The remainder of this paper is organized as follows. Section 2 argues the previous researches on spatial queries domain related to visibility. Section 3 presents the preliminaries for 3DNS query and proposes the query processing algorithm. Section 4 evaluates the experimental performance of the 3DNS query against the conventional. Section 5 summarizes our work and concludes this paper.

2. Related Work

The notion of visibility has been intensively researched in the field of image rendering and computational geometry. Such as ray shooting query [4] can be applied to complete 3DNS query. The effectiveness of ray shooting query is elaborated in Section 4. In the field of spatial database, Lee. K., et al., [1] proposed the notion of NS query initially. NS query processing algorithm is based on the R-tree index. For a given visible object $O_i$, the viewable angle for $O_i$ is denoted as $[\theta^s, \theta^e]$ in terms of the polar coordination system, where $\theta^s$ and $\theta^e$ are the start and end viewable angles of $O_i$ respectively. In order to avoid sequential comparisons for the visibility among the objects in 2D spaces, Lee. K., et al., defined the angular-based distance metric $\text{madist}(O_i, \alpha)$ which returns a distance from a query to object $O_i$ at angle $\alpha$. This metric function is adopted to apply pruning heuristics to decrease the latency of executing NS query as shown in Figure 2 (a). The search algorithm traverses the index from the root to leaf node and exploits the index nodes or objects using the priority queue in best-first search manner. While traversing the index, NS query algorithm avoids visiting unnecessary nodes by setting the lower-bound distance of visible of objects that uses the proposed $\text{min_madist}$ (upper-bound) and
min\_madist (lower-bound), where min\_madist and max\_madist metrics refer to the minimum and maximum values of madist function in given angular range respectively. Lee, K. et al., [2] also extends NS query which tracks the NS query result for a given time interval in a moving object environment. After initial NS query processing, it manages the safe regions for each disjointed angular bound of entries (close angle) and empty space (open angle) as shown in Figure 2 (b). Chung, J. et al., [3] proposed a server-client framework for NS query that tracks the NS results of a query trajectory. In framework, once a server processes a full NS query processing, a client caches the NS result.

![Figure 2. NS Query Processing](image)

3. Query Processing in 3D Geospatial Environment

Handling 3D objects requires the additional considerations and algorithmic revisions of NS query processing. NS query in 3D spaces searches every visible object in the spherical area. However considering applicable services of NS query, we establish two assumptions in the environments. First, a query is represented as a point and placed on the bottom of 3D spaces. That is, for give query \( q = \{q_x, q_y, q_z\} \), \( q_z \) should be 0. Second, every object in dataset \( R \) is approximated in a cube and placed from the bottom of spaces as same as a query. We name the area that satisfies these two assumptions the geospatial environments, and the objects resigned in this space are called geospatial objects.

In this Section, we propose 3DNS query scheme in 3D geospatial environments. The preliminaries and theories to measuring the viewable angles for 3D geospatial objects are explained in Section 3.1 and Section 3.2 and revised NS query processing algorithm are proposed in Section 3.3

3.1. Affiliations

In 2D spaces, the viewable angular range for an object is represented as two disjoint angle values, and the viewable angular range denotes. Thus, for two given objects, \( \alpha \) and \( \beta \), and then \( \alpha, \beta \) are determined to be invisible to \( Q \), where \( \alpha \) is a start(end) angle of the object. We term this invisibility condition. Unlike 2D spaces, visibility of cannot be determined until the height of \( \beta \) is considered in 3D spaces, even though and satisfy the invisibility condition. In order to present the angular range for a 3D geospatial object, we adopt the spherical coordination system. We expand the viewable angular range presentation into a tuple that refers the object’s viewable angular range on xy-plane in terms of the polar coordination system. In addition, the inclination angle which is measured from a fixed reference direction on xy-plane as shown in Figure 3 (a). With respect to the definition of the viewable angular range for objects in 3D geospatial spaces, 3DNS query is defined as follows:

**Definition 1 (3D Nearest Surrounder Query)** Given dataset \( R \) and a query point \( Q \), 3D nearest surrounder query of \( Q \), 3DNS(\( Q \)), returns a set of tuple \( \eta : (\theta, \phi) \in G \) satisfies that:
Given an object \( Q \) in space, the inclination angle refers to the minimum inclination angle \( \phi \) of \( Q \) at angular range \((\theta, \phi)\), where

\[
\phi = \arctan\left( \frac{z_\rho}{\sqrt{(x_Q - x_\rho)^2 + (y_Q - y_\rho)^2}} \right) = \arctan\left( \frac{z_\rho}{\text{madist}(Q, \rho, \theta)} \right)
\]

, where \( \theta \) is the angle formed between \( Q \) and \( \rho \) on xy-plane in polar coordinate system.

Regarding the equation (1), the value of \( \phi \) is not static for every point on an object since the \text{madist} is varied according to the coordinate values of points on xy-plane as illustrated in Figure 1 (b). In this figure, the inclination angle \( \phi \) is minimized when \text{madist} is maximized and \( \phi \) is maximized, when \text{madist} is minimized. Therefore, we define two different angles for a geospatial object, \( \phi_{\text{min}} \) and \( \phi_{\text{max}} \) to represent the inclination angular range as shown in Figure 3 (c).

**Definition 2** \((\phi_{\text{min}}, \phi_{\text{max}})\) Given an object \( O \) and a query \( Q \), \( \phi_{\text{min}} \) and \( \phi_{\text{max}} \) refer the minimum and maximum inclination angles of \( O \) for \( Q \), and defined by the following equations:

\[
\phi_{\text{min}} = \frac{O^{\text{height}}}{\max_{\text{madist}}(Q, O, \left[\theta_{\text{min}}, \theta_{\text{max}}\right])}
\]

\[
\phi_{\text{max}} = \frac{O^{\text{height}}}{\min_{\text{madist}}(Q, O, \left[\theta_{\text{min}}, \theta_{\text{max}}\right])}
\]

### 3.2. Invisible Region Formulation

For a visible object, it produces an invisible region (grayed region) for a query as illustrated in Figure 4 (a). If a geospatial object is totally involved in an invisible region generated from any visible objects, then the object is marked as ‘invisible’ and excluded from the Viewable List (VL).

**Heuristic 1**: Suppose that current visible list, \( \text{VL} = \{O_1, O_2, \ldots, O_n\} \). Given an object, \( O \), for invisibility condition, if \( \exists O \in \text{VL} \) such that \( \theta^+ > \theta^- \) and \( \theta^+ < \theta^- \) and \( \text{madist}(\epsilon, Q) < \text{madist}(O, Q) \), and \( O, \phi_{\text{min}} < \epsilon, \phi_{\text{max}} \), then \( \epsilon \) is excluded from the query results.

The volume of the **Invisible Region** (IR) grows as much as the point on xy-plane is far from the visible object. Especially, the height of this region, \( H^{\text{IR}} \), at the point, \( \rho \), and angle \( \alpha \) is given as follows:

\[
H^{\text{IR}} = \frac{O^{\text{height}}}{\text{madist}(Q, O, \alpha) \cdot \text{dist}(O, \rho)}
\]

Regarding the equation (2), if the \( H^{\text{IR}} \) is higher than the tallest object among the object dataset, then it implicates that no object can be visible at query location within this region.
formed beyond the point \( p \). \( p \) is the point where \( H^e \) becomes same with the height of the tallest object in dataset. We term this area the **Secluded Region** (SR) of a visible object where any object in the dataset should not be included in VL of a given query point and the distance from \( Q \) to the point \( p \), \( dist(Q,p) \), term the **Secluded Distance** (SD). As illustrated in Figure 4 (a), the object \( O_j \) is included in the secluded region of \( O_i \), then \( O_j \) is excluded from the 3DNS result without the visibility examination.

The contour of SR boundary for a visible object \( O_i \), is determined by the inclination angular range \( \theta_i^- \sim \theta_i^+ \), where \( \theta_i^- \) and \( \theta_i^+ \) are the upper and lower bounds of the inclination range, respectively. However, representing a duel parabolic curve and determining the containment of an object with the curvy region requires additional computation cost. To simplify the boundary of an SR, we defined the new SR for a given visible objects with the lower boundary of genuine SR boundary where inclination angle become the minimum. This boundary is determined with a single SD point where the value of \( \text{mindist} \) becomes minimum in equation (2). The formal definition of the SD is given in Definition 3.

**Figure 4. Invisible Region Formulation**

**Definition 3 (Secluded Distance)** For a given visible object, \( O_i \in VL \), a secluded distance of \( O_i \), \( O_i,SD \), is the distance from a query to a point where the height of an SR becomes equal to that of the tallest object in dataset. The SD defined as follows:

\[
O_i,SD = (\text{max}_i \text{mindist}(Q,O_i) \cdot H_{max}) / O_i^{\text{right}}
\]

where \( H_{max} \) is the height of the tallest object in \( R \). \( SR \) indicates that if \( [\theta_i^- \sim \theta_i^+] \subseteq [\theta_j^- \sim \theta_j^+] \) and \( \text{max}_i \text{mindist}(Q,O_i,[\theta_i^- \sim \theta_i^+]) < \text{max}_j \text{mindist}(Q,O_j,[\theta_j^- \sim \theta_j^+]) \) for given objects \( O_i \) and \( O_j \), then \( O_j \) is never visible to a query.

According to the concept of SR and SD, we are able to devise another heuristic and utilize it for the early-pruning scheme that the algorithm excluded objects located in an SR before the visibility comparison.

**Heuristic 2:** Given an object, \( \varepsilon \in R \), if \( \exists O_j \in VL \) such that \( \theta_j^- > \theta_e^- \) and \( \theta_j^+ < \theta_e^+ \) and \( \text{mindist}(\varepsilon,Q) > O_j,SD \), then \( \varepsilon \) is excluded from the query’s VL.

### 3.3. 3DNS Query Processing

In this section, we explained the 3DNS query processing algorithm with R-tree index which is based on the viewable angle range notation that was defined in section 3.1. This algorithm is also able to work with other indexes except R-tree, such as k-d-tree [5] and quadtree [6] with the minor revision. The proposing query processing algorithm of 3DNS basically follows that of the NS query. In this regard, we applied the devised novel
heuristics in Section 3.2 to accelerate the query execution. Algorithm 1 details the steps of 3DNS query processing. In lines 1–3 a priority queue, \( \text{PriQ} \), is initialized with the root of the R-tree, and an angle, \( \alpha \), indicates the examined end angle and set 0 initially. Before starting to sample the viewable angle range for each object, every element in the root node is inserted into the \( \text{PriQ} \). Algorithm 1 continues to examine each entry (an object or an index node) in \( \text{PriQ} \) until the \( \text{PriQ} \) is empty. In line 5, the Lookahead function finds an entry \( \varepsilon \) which has the smallest start viewable angle according to xy-plane in the \( \text{PriQ} \). In lines 6–7, it looks for visible object (\( \eta \)) in the current \( \text{VL} \), \( Y \), which shares viewable angular range of \( \varepsilon \). If \( \text{mindist}(Q, \varepsilon) > \eta \text{SR} \), then \( \varepsilon \) is necessarily invisible. Otherwise \( \varepsilon \) might be visible. Then \( \varepsilon \) is inserted into \( Y \) in case of \( \varepsilon \) is an object, or \( \varepsilon \) is exploited and every sub-node in \( \varepsilon \) is inserted into \( \text{PriQ} \) in lines 9 ~ 12. If \( \varepsilon \) does not have any common viewable angular ranges, then it examine visibility of each edges of \( \varepsilon \) with each edges of \( \eta \in Y \), and update \( \Theta \) and \( \Phi \) of \( \varepsilon \) and \( \eta \) according to the invisibility condition in lines 14 ~ 22. In line 23, \( \alpha \), which indicates the max angle has been swept, is updated. This process is repeated until \( \text{PriQ} \) is empty. Finally, the 3DNS result (\( Y \)) returns in line 24.

Algorithm 1. 3DNS Query Processing Algorithm

<p>| In: a query ( Q ), root node ( N ) of R-tree |</p>
<table>
<thead>
<tr>
<th>Out 3DNS query result ( Y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Let ( \text{PriQ} ) and ( \alpha ) be the priority queue and max angle that has been swept.</td>
</tr>
<tr>
<td>2: Let ( \varepsilon ) be a node pointer.</td>
</tr>
<tr>
<td>3: Insert all subnodes in ( N ) into the ( \text{PriQ} );</td>
</tr>
<tr>
<td>4: while ( \text{PriQ} ) is not empty do</td>
</tr>
<tr>
<td>5: ( \varepsilon ) ← Lookahead(( Q ), ( \text{PriQ} ), ([0, \alpha]));</td>
</tr>
<tr>
<td>6: if ( \eta \leftarrow \text{getConservative}(Y, [0, \alpha]) ) then</td>
</tr>
<tr>
<td>7: if ( \eta SD &lt; \text{mindist}(Q, \varepsilon) ) then skip; /* Heuristic 1 */</td>
</tr>
<tr>
<td>8: else</td>
</tr>
<tr>
<td>9: if ( \varepsilon ) is an object then</td>
</tr>
<tr>
<td>10: ( Y \leftarrow \langle \varepsilon, [\phi^+, \phi^-]) [\phi^+, \phi^-) \rangle };</td>
</tr>
<tr>
<td>11: else</td>
</tr>
<tr>
<td>12: access ( \varepsilon ) and put all its objects to ( \text{PriQ} );</td>
</tr>
<tr>
<td>13: else</td>
</tr>
<tr>
<td>14: foreach ( \eta \in Y ) do</td>
</tr>
<tr>
<td>15: Let ( [\phi^+, \phi^-) ) be the common angular range of ( \varepsilon ) and ( \eta );</td>
</tr>
<tr>
<td>16: if ( [\phi^+, \phi^-) ) is not empty and ( \text{mindist} ) of ( \varepsilon ) is smaller than that of ( \eta ) then</td>
</tr>
<tr>
<td>17: if ( [\phi^+, \phi^-) ) and ( [\phi^+, \phi^-) ) then /* Heuristic 2 */</td>
</tr>
<tr>
<td>18: delete ( \eta ) from ( Y );</td>
</tr>
<tr>
<td>19: else</td>
</tr>
<tr>
<td>20: update ( \eta ) with ( [\phi^+, \phi^-) ) and ( [\phi^+, \phi^-) )</td>
</tr>
<tr>
<td>21: else</td>
</tr>
<tr>
<td>22: ( Y \leftarrow \langle \varepsilon, [\phi^+, \phi^-) \rangle \rangle );</td>
</tr>
<tr>
<td>23: ( \alpha \leftarrow \text{max}(\alpha, \theta^+);</td>
</tr>
<tr>
<td>24: return ( Y );</td>
</tr>
</tbody>
</table>

4. Evaluation

We performed extensive experiments and evaluated the results to prove the efficiency of the proposed techniques. We varied the object cardinality and volume of object in the
simulation settings for a precise comparison the previous NS query algorithm to the proposed algorithm.

4.1. Experiments Environments

Our simulation environment consists of four components in advance of performance analysis to prove the superiority of 3DNS query to conventional NS query in 3D geospatial environment. First, GSTD [7,8] was adopted to generate query trajectory data randomly. The synthetic datasets generated by GSTD uniformly. Second, simulation code was programmed with Eclipse IDE 6.2 using JDK 6.1. Finally, Gnuplot 4.4.3 [9] was used as a visualization tool for both query and dataset. Every object dataset is indexed using R*-tree [10]. We adopted the main memory-based R*-tree [11], which maintains every node in main memory, for fast query processing. The server was equipped with an Intel i7 CPU and 8GB memory. Table 1 details the experiment parameters during the evaluation. Bolded words denote the default setting values.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object Cardinality</td>
<td>0.01K, 0.1K, 1K, 10K, 100K</td>
</tr>
<tr>
<td>Object Volume</td>
<td>0.1, 0.3, 0.5, 0.7, 0.9</td>
</tr>
<tr>
<td>Query Cardinality</td>
<td>1K</td>
</tr>
</tbody>
</table>

4.2. Experiments Environments

To observe the efficiency of the proposed algorithm, the performance of each experiment is measured by the elapsed time (CPU sec.), the number of nodes accessed in R*-tree index. A thousand randomly generated queries are executed in each experiment and the average values are produced. Experiments are designed to compare the effectiveness of heuristics. In every result graph, 3DNS+FF, 3DNS+TF and 3DNS+TT indicate the setting that no heuristic, the first heuristic and both two heuristics are applied respectively. 3DNS+FF is the expanded version NS query algorithm that (i) it follows the NS algorithm in 2D spaces and (ii) it investigates the visibilities for the objects using the ray shooting query scheme. The ray shooting technique [4] detects the invisible objects by edges of objects are projected to a query on the projecting lines which are placed on the circumscribed sphere of the determined 3D spaces. In this paper, shooting queries are evaluated for each sample angles at 0.01° for 3DNS+FF setting.

Impact of Object Cardinality: This experiment set is organized to examine how object cardinality (n) affects 3DNS query algorithm against that of NS query, in terms of CPU time and the accessed index nodes. The cardinality of objects varies from 10 to 100K. The volume of object is set 0.6 maximal in fixed square [0, 1000]×[0, 1000] space. Figure 5 (a) depicts the experimental results in elapsed time to complete a query. It is observed that an object rarely obstructs another object in sparse object distribution (n = 0.01K) and every object is visible to query in most cases. The elapsed times are extremely shorter than other experimental result. This results in that the two proposed heuristics are also effective. In period n varies from 1K to 10K, the ratio of the heuristics applied increases to complete a query. Meanwhile, the count of 3DNS result also increases that the latencies of 3DNS+TF and 3DNS+TT increase for handling result. Both schemes show approximately 48%~94% and 53%~95% better performances than 3DNS+FF scheme in terms of CPU time. Unexpectedly, the performance of all three scheme is improved in case of n=100K. It can be analyzed that the 3DNS query is affected by the density of objects in the given spaces. This result comes from that the
closer objects produces the larger invisible region, and increases the ratio of invisible 
condition applied.

Figure 5 (b) depicts the number of accessed nodes to complete 3DNS queries. The 
numbers of node accessed by 3DNS+TF and 3DNS+TT algorithms are approximately 
14%–46%, 42%–69% lower than that of 3DNS algorithms due to the invisibility 
condition and the SR scheme. However, the number of nodes accessed does not decrease 
as much as the CPU time due to the nature of progressive exploring search space in all 
three schemes.

![Figure 5. Impact of Object Cardinality (n)](image1)

**Figure 5. Impact of Object Cardinality (n)**

**Impact of Object Volume:** The second set of experiments evaluates the performance of 
3DNS algorithms by varying the volume of objects (l) from 0.1 to 0.9. The cardinality 
of objects is set 10K in default. Figure 6 illustrates the results for these settings in terms of 
CPU time and the number of nodes accessed. The 3DNS algorithm is rigidly affected by 
the volume of objects as shown in Figure 6 (a). Same as the setting n = 100K in Figure 5 
(a), the larger volumes of objects devote for producing larger invisible region and SR. 
However, 3DNS+FF was not differed for all area compared to 3DNS+TF and 3DNS+TT 
in CPU time. This results in the massive number of ray shooting queries to generate 
3DNS result. Besides 3DNS+FF algorithm, a small object volume deteriorates the effect 
of 3DNS algorithm and the heuristics. In overall, 3DNS+TF and 3DNS+TT outperform 
3DNS+FF about 88%–96% and 93%–97% in terms of CPU time.

Figure 6 (b) illustrates the number of nodes accessed from R*-tree index for 
completing a 3DNS query. The results are shown in similar with CPU time, but the ratios 
are not steep as much as CPU time. Approximated 2%–57% and 28%–43% lower nodes 
are accessed for 3DNS+TF and 3DNS+TT than that of 3DNS+FF. This result shows that 
the ray shooting queries requires excessive computing resources for examine the 
visibility conditions of 3D geospatial objects.
5. Conclusion

In this paper, we propose the novel type of a spatial query, termed 3DNS query that returns a set of every visible geospatial objects around the query location in 3D geospatial environments. We first define the inclination angles, termed $\phi_{\text{min}}$ and $\phi_{\text{max}}$, that indicate the minimum and maximum inclination angles for 3D geospatial objects. Using the angular range representation, we propose the 3DNS query algorithm which utilizes two heuristics, invisibility condition and secluded region to avoid sequential comparison for visibility among objects in dataset. Through the extensive experiments, we evaluated the efficiency of the 3DNS query processing algorithm and the result indicates that the proposed scheme outperforms the conventional schemes.

Acknowledgments

This Research was supported by Seokyeong University in 2013.

References


Authors

Jaehwa Chung, is an assistant professor at Dept. of Computer Science in Korea National Open University. He received M.S. and Ph.D. degrees at Dept. of Computer Science Education in Korea University, Korea. His research interests include spatial query and index, spatio-temporal database, mobile data management, location-based services, Spark, WSNs and mobile data mining.

Daewon Lee, he received his B.S. in division of Electricity and Electronic Engineering from Soonchunhyang University, Asan, ChungNam, Korea in 2001. He received his M.E. and Ph.D. degrees in Computer Science Education from Korea University, Seoul, Korea in 2003 and 2009. He is currently a full time lecturer in the Division of General Education at SeoKyeong University in Korea. His
research interests are in IoT, Mobile computing, Distributed computing, Cloud computing, and Fault-tolerant systems.