Geoinformatica
DOI 10.1007/s10707-013-0199-6

Large-scale geo-tagged video indexing and queries

He Ma · Sakire Arslan Ay · Roger Zimmermann · Seon Ho Kim

Received: 14 December 2012 / Revised: 8 August 2013 / Accepted: 28 November 2013

Abstract With the wide spread of smartphones, a large number of user-generated videos are produced everyday. The embedded sensors, e.g., GPS and the digital compass, make it possible that videos are accessed based on their geo-properties. In our previous work, we have created a framework for integrated, sensor-rich video acquisition (with one instantiation implemented in the form of smartphone applications) which associates a continuous stream of location and viewing direction information with the collected videos, hence allowing them to be expressed and manipulated as spatio-temporal objects. These sensor meta-data are considerably smaller in size compared to the visual content and are helpful in effectively and efficiently searching for geo-tagged videos in large-scale repositories. In this study, we propose a novel three-level grid-based index structure and introduce a number of related query types, including typical spatial queries and ones based on bounded radius and viewing direction restriction. These two criteria are important in many video applications and we demonstrate the importance with a real-world dataset. Moreover, experimental results on a large-scale synthetic dataset show that our approach can provide a

H. Ma (✉) · R. Zimmermann
School of Computing, National University of Singapore, Singapore 117417, Singapore
e-mail: mahe@comp.nus.edu.sg; he.ma@nusri.cn

R. Zimmermann
e-mail: rogerz@comp.nus.edu.sg

S. Arslan Ay
School of Electrical Engineering & Computer Science, Washington State University,
355 Spokane St. Pullman, Washington, DC 99164-2752, USA
e-mail: sakire.arslanay@wsu.edu

S. H. Kim
Integrated Media Systems Center, University of Southern California, Los Angeles, CA 90089, USA
e-mail: seonkim@usc.edu

H. Ma
Center of Interactive Media and Software Development, National University of Singapore (Suzhou)
Research Institute, 377 Lin Quan Street, Suzhou Industrial Park, Jiang Su 215123,
People's Republic of China

Published online: 28 December 2013

significant speed improvements of at least 30 %, considering a mix of queries, compared to a multi-dimensional R-tree implementation.

Keywords Geo-taggeed video search · Grid-based index · Meta-data generation · Query processing

1 Introduction

With the recent developments in the video capture technology, a large number of user-generated video (UGV) are produced on a daily basis. For example, the smartphones, which are carried by users all the time, have become extremely popular in capturing and sharing online videos due to their handiness, enhanced quality of images and wireless bandwidth. YouTube [23] has indicated that by the end of July 2013, over six billion hours of videos are watched each month and 100 hours of video are uploaded every minute. 25 % of global YouTube views come from mobile devices. According to another study from Cisco [5], the overall mobile data traffic reached 885 petabytes per month at the end of 2012, 51 percent of which is on mobile video. It is forecast that mobile video traffic will grow at a Compound Annual Growth Rate (CAGR) of 75 percent between 2012 and 2017 and reach at 7.4 exabytes per month by 2017. In the presence of a huge size video depository such as YouTube, effectively and efficiently searching such a repository for meaningful results is still a challenging problem. Current video search techniques that annotate videos based on the visual content are struggling to achieve satisfactory results in online UGVs, particularly in accuracy and scalability.

Alternatively, the embedded sensors (e.g., GPS and compass units) have been cost-efficiently deployed on video cameras. Consequently, the related meta-data of videos, especially geographical properties of video scenes can easily be collected during video capturing. This association of video scenes and their geographic meta-data has raised interesting research topics in the multimedia community, for example, the captured sensor meta-data can be utilized to aid in modeling, indexing and searching of geo-tagged videos at the high semantic level preferred by humans.

In our earlier work [1], we proposed to model the coverage region (i.e., field of view, and FOV for short) of video frames as a pie-shaped geometric area described by the geospatial sensor data, such as camera location, viewing direction and visible distance. This approach treats the visual content as a series of spatial objects. Compared to visual content, the meta-data occupies much less space, which makes searching among large scale of videos practical. Consequently, the challenging video search problem is transformed into a known spatial data selection problem. The objective is then to index the spatial objects and to search videos based on their geographic properties. Our previous study [15] demonstrated the effectiveness of geographic sensor meta-data for searching a huge amount of videos.

For a practical implementation of search engine with a large amount of geo-tagged videos and their associated geospatial meta-data, there remain some other critical issues to be resolved. For example, the performance of searching sensor meta-data should efficiently handle large video depositories (such as YouTube). Therefore, it is essential to develop a study of high performance index structure which can effectively harness the captured geospatial meta-data. The widely used index structures, i.e., the R-tree [10] (and/or its variations [3, 22]) has been the chosen structure to index geometric figures. However, its performance deteriorates as the number of figures indexed increases greatly. Assuming all

videos in a huge depository are represented using sensor meta-data, i.e., streams of geospatial objects, a R-tree may suffer significantly to provide enough search performance due to its increased heights.

Furthermore, from the semantic perspective, in searching videos through their geographic coverage, distance and direction are two important criteria that can help to improve query functionality. Not every view from any direction or distance to a location is distinctive or attractive to users [24]. For example, people are searching for a video that records a large building (e.g., the Marina Bay Sands Hotel). Some people would like to see the panoramic view while others prefer to discover a specific part of the building. In such application, searching videos based on distance helps to accelerate the query processing and retrieve meaningful results to end-users. However, due to the large geographical coverage region of the FOV, indexing FOVs with a R-tree or its variations can only provide overlap calculation but not prune any unnecessary search on-the-fly if the query is with distance restriction. Moreover, searching for videos captured from a specific direction is helpful in applications such as event reviews and video summaries. Another possible application can be to automatically extract panoramic images of a building from a video and use these images to construct the 3D model of the building. With the viewing direction information of the camera, we can select the smallest subsets from the images, which are with complete coverage but least redundant details, to finish the target. The R-tree can support this query functionality by adding one more dimension, but its performance will deteriorate.

The above drawbacks with the R-tree based structures on searching large-scale geo-tagged videos raise the question that it is essential to develop a study of high performance index structure which can effectively harness the captured geospatial meta-data. An important observation is that the geo-space is bounded while the number of videos is almost un-bounded. Based on this observation, we propose a new three-level grid-based index structure for geo-tagged video search by fully utilizing their geographic properties. Specifically, this index structure is created to allow efficient access of FOVs based on their distance to the query location and the cameras viewing direction. Based on these criteria, we introduce a number of related query types which support better human perception of images in resulting videos, including typical spatial queries (i.e., point queries, range queries, and kNN queries) and the queries with bounded radius or direction restriction.

Among these, the most unique query types proposed in this work is *Nearest Video Segments query* (k-NVS). This query retrieves the k closest video segments that show a given query point. k-NVS query can significantly enhance human perception and decision in identifying requested video images, especially when search results return a large number of videos in a highly populated area. Moreover, the query can additionally specify a bounded radius range to get the closest video segments that show the query point from a distance within a given radius range. Alternatively, the query may specify a certain viewing direction to specifically retrieve the k closest segments that show the query point from that direction, which is critical in human perception of objects. Similarly, k-NVS can serve as a useful feature for video browsing applications. For example, on a map-based interface the videos that show important landmarks from the user's viewing point can be quickly retrieved as the user navigates by issuing continuous k-NVS queries.

In the remaining sections of the manuscript we describe our geo-tagged video indexing and searching approach, and report on an extensive experimental study with a synthetic dataset. The results we have obtained illustrate that the three-level grid index structure

supports new geospatial video query features. It efficiently scales to large datasets and significantly speeds up the query processing (compared to the R-tree) for finding the related video segments, especially for queries with direction. The rest of this paper is organized as follows. Section 2 provides the background information and summarizes the related work. Section 3 details the proposed data structure. Section 4 introduces the new query types and details the query processing algorithms. Section 5 reports the results on the performance evaluations of the proposed algorithms. Finally, Section 6 concludes the paper.

2 Background and related work

2.1 Modeling of camera viewable scene

The *camera viewable scene* is what a camera in geo-space captures. This region is referred to as *camera field-of-view* (FOV in short) with the shape of a pie-slice [1]. The FOV coverage in 2D space can be defined with four parameters: the camera location P, the camera viewing direction vector $\overrightarrow{d}$, viewable angle α, and maximum visible distance R_V (see Fig. 1). The location P of camera is the $< latitude, longitude >$ coordinate reading from a positioning device (e.g., GPS and/or Cricket coordinates [20]). The camera viewing direction vector $\overrightarrow{d}$ is acquired from a digital compass. We use θ to represent its value with respect to the North. The camera viewable angle (α) is calculated based on the camera and lens properties for the current zoom level [8]. The visible distance R_V is the maximum distance at which a large object within the camera's FOV can be recognized. Then, the camera viewable scene at time t is denoted by the tuple $FOV\,(P\,\langle lat, lng\rangle, \theta, \alpha, R_V, t)$. These geospatial meta-data can be obtained from the embedded sensors during the video recording.

2.2 Related work

Associating geo-location and camera orientation information for video retrieval has become an active topic. Research [25] used the 3D geo-data to help on video mosaicing for environment monitoring. Hwang et al. [11] and Kim et al. [12] proposed a mapping between the 3D world and the videos by linking the objects to the video frames in which they appear. Their work used GPS location and camera orientation to build links between video frames and world objects. Liu et al. [14] presented a sensor enhanced video annotation system (referred to as SEVA). Navarrete and Blat [16] utilized geographic information to segment and index video. Our prior work [1] proposed a viewable scene model to link the video content and sensor information. However, none of the above methods addresses indexing and searching

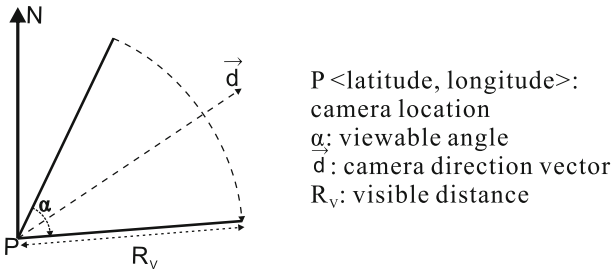


Fig. 1 Illustration of the field-of-view (FOV) model in 2D space

issues based on the geo-properties of the videos, on the large scale, especially for processing queries with distance and direction restrictions.

Our approach represents each video frame as a spatial object. There exist two categories of spatial data indexing methods: *data-driven structures* and *space-driven structures* [21]. The R-tree family (including R-tree [10], R^+-tree [22], R^*-tree [3]) belongs to the category of data-driven structures. Guttman [10] proposed the R-tree, which is a dynamic tree data structure, as an extension of the ubiquitous B-tree in multi-dimensional space, for spatial data indexing. Each node in the R-tree is represented as a bounding rectangle. To access a node of the indexed collection, one typically follows a path from the root down to one or several leaves, testing each bounding rectangle at each level for either containment or overlap. However, these methods are designed mainly for supporting efficient query processing when the construction and the maintenance of the data structure is computationally expensive. The space-driven structures include methods such as the grid file [17], quadtree [7], and Voronoi diagram [19]. Recent researches use either the grid structure [4], the skip quadtree [6] or Voronoi diagram [18] to process multiple types of queries. These structures split the cell when the maximum capacity is reached. The difference is that each time the grid file splits the cell into two while quadtree splits it into four. Moreover, indexing using Vonoroi diagram keeps safety region for each object so as to quickly process kNN query. However, these data structures consider spatial objects as points or small rectangles, and none of them are appropriate to index our FOV model. The reason is that the FOV model has a large coverage region and consecutive FOVs have large overlap region. This results in the difficulty in partitioning the space while guaranteeing each FOV only belongs to one cell without cell overlapping.

Our prior work [13] proposed a vector-based approximation model to efficiently index and search videos based on the FOV model. It mapped an FOV to two individual points in two 2D subspaces using a space transformation. This model works well on supporting the geospatial video query features, such as point query with direction and bounded distance between the query point and camera position. However, it does not investigate query optimization issues. The vector model works effectively for basic query types, such as point and range query, however does not support the k-NVS query. Moreover, there was no consideration in scalability. Next, we will introduce the proposed three-level index structure.

3 Grid based indexing of camera viewable scenes

We present our design of the memory-based grid structure for indexing the geographic coverage area of each camera viewable scene. The grids can cover the whole Earth but only those are recorded by videos are indexed. The proposed structure constructs a three-level index, where the first level indexes the video FOVs according to location, the second level indexes them based on the distance to the overlapping cell, and the third level builds an index based on FOV viewing direction. That is, the FOV is indexed by any first level cell if its coverage region overlaps with the cell. If the FOV overlaps with the first-level cell, the distance between the camera location P and the center of the first-level cell is then indexed in the second-level cell by which subcell P locates. The proposed three-level index structure is illustrated in Fig. 2. The collections of cells at the first, second, and third level are denoted by $C_{\ell 1}$, $C_{\ell 2}$, and $C_{\ell 3}$, respectively, and s is a system parameter indicating the number of subcells within a row or a column. Note that, each level of the index structure stores only the ID numbers of the FOVs for the efficient search of the video scenes. The actual FOV meta-data (i.e., P, θ, α, R_V and t values) are stored in a MySQL database where the

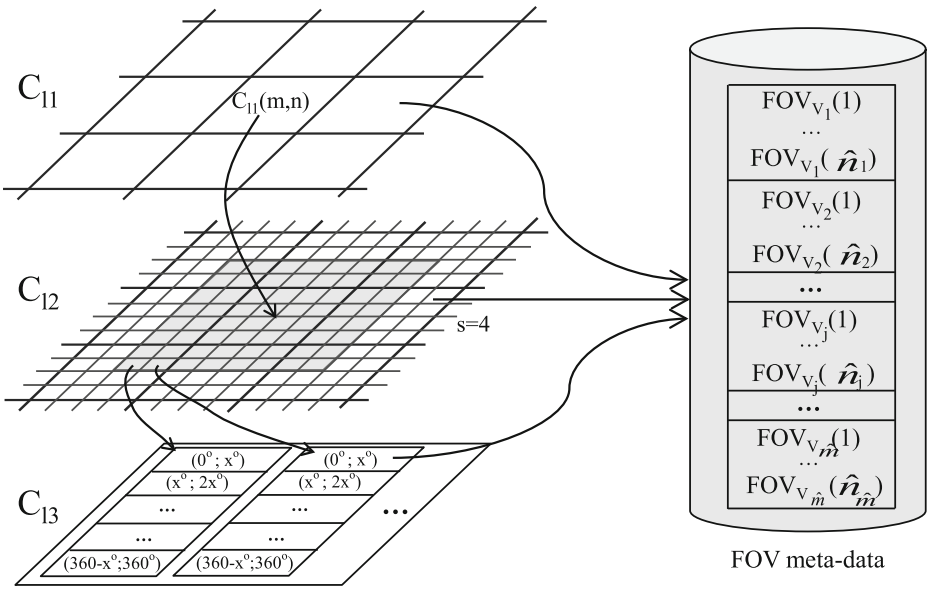
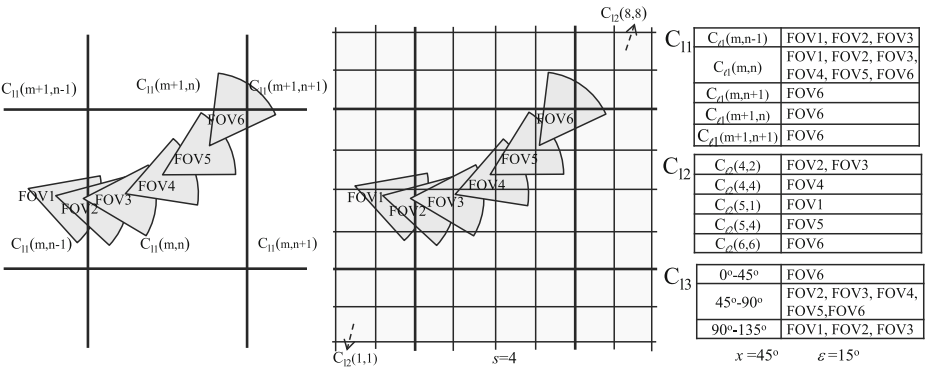


Fig. 2 The three-level grid data structure

meta-data for a particular FOV can be efficiently retrieved through its ID number. Figure 3 illustrates the index construction with an example of a short video file. In Fig. 3c, only the index entries for the example video file are listed.

The first level organizes the embedding geo-space, which covers all the regions that geo-tagged videos are recording, as a uniform coarse grid. The space is partitioned into a regular grid of $M \times N$ cells, where each grid cell is an $\delta \times \delta$ square area, and δ is a system parameter that defines the cell size of the grid. A specific cell in the first-level grid index is denoted by $C_{\ell 1}(row, column)$ (assuming the cells are ordered from the bottom left corner of the space). The 2D geographical coverages of the FOVs are indexed in this coarse grid structure. Specifically, FOVs are mapped to the grid cells that overlap with their coverage areas and each grid cell maintains the IDs of the overlapping FOVs. In Fig. 3c, the set of FOVs that overlap with the first-level cells are listed in the upper table.

The second-level grid index organizes the overlapping FOVs at each first-level cell based on the distance between the FOV camera locations and the center of the cell. To construct the second-level grid, each $C_{\ell 1}$ cell is further divided into $s \times s$ subcells of size $\left(\frac{\delta}{s} \times \frac{\delta}{s}\right)$, where each subcell is denoted by $C_{\ell 2}(f, g)$ (see Fig. 2). s is a system parameter and defines how fine the second-level grid index is. For each first level grid cell $C_{\ell 1}(m, n)$, we maintain the range of the second-level subcells, covering the region in and around $C_{\ell 1}(m, n)$ and containing all the FOVs that overlap with the cell $C_{\ell 1}(m, n)$. In Fig. 2, the shaded region at $C_{\ell 2}$ shows the range of $C_{\ell 2}$ subcells corresponding to the first-level cell $C_{\ell 1}(m, n)$. Note that the FOVs whose camera locations are at most R_V away from cell $C_{\ell 1}(m, n)$, will also be included in that range. In the example shown in Fig. 3, the second-level range for $C_{\ell 1}(m, n)$ includes all subcells $C_{\ell 2}(1, 1)$ through $C_{\ell 2}(8, 8)$. While the first-level cells hold the list of the FOVs whose viewable scene areas overlap with the cell, the second-level subcells hold the list of FOVs whose camera locations are within those subcells. For example in Fig. 3c, the table in the middle lists the non-empty second-level subcells and the set of FOV IDs



(a) First-level grid. (b) Second-level grid. (c) Index tables.

Fig. 3 Illustration of Index construction. **a** First-level grid. **b** Second-level grid. **c** Index tables

assigned to them. In order to retrieve the FOVs closest to a particular query point in the cell $C_{\ell 1}(m, n)$, first, the second-level cell $C_{\ell 2}(f, g)$ where the query point resides is obtained, and then the FOV IDs in and around subcell $C_{\ell 2}(f, g)$ are retrieved. The second-level index enables the efficient retrieval of closest FOVs in the execution of queries with bounded radius restriction, e.g., a k-NVS (k Nearest Video Segments) query.

The first- and second-level grid cells hold the location and distance information only, therefore cannot fully utilize the collected sensor meta-data, such as direction. Direction can be an important cue in retrieving the most relevant video results when the videos showing the query location from a certain viewpoint are of higher interest. To support the directional queries we construct a third-level in the index structure that organizes the FOVs based on the viewing direction. The 360 ° angle is divided into x ° intervals in clockwise direction, starting from the North (0 °). We assume an error margin of $\pm\varepsilon°$ around the FOV orientation angle $\theta°$. Each FOV is assigned to one or two of the view angle intervals with which its orientation angle margin ($\theta° \pm \varepsilon°$) overlaps. The value of ε can be customized based on the application. In Fig. 3c, the lower table lists the third-level index entries for the example video for $x = 45°$ and $\varepsilon = 15°$.

For a video collection with about 2.95 million FOVs, the index size for the three-level index structure is measured as 1.9 GB. As the dataset size gets larger the index size grows linearly. For example, for datasets with 3.9 million and 5.4 million FOVs, the index size is measured as 2.5 GB and 3.3 GB, respectively. In our experiments in Section 5, we report the results for a dataset of 5.4 million FOVs. Next we will describe the query processing for various query types.

4 Query processing

We represent the coverage of a video clip as a series of FOVs where each FOV corresponds to a spatial object. Therefore the problem of video search is transformed into finding the spatial objects in the database that satisfy the query conditions. In searching video meta-data, unlike a typical spatial query (i.e., point queries, range queries and kNN queries), the query may enforce additional application specific parameters. For example, it may search with a range restriction for the distance of the camera location from the query point, which is interpreted as the *query with bounded radius*. Or the query may ask only for the videos that

show the query location from a certain viewpoint, then it may restrict the FOV direction to a certain angle range around the specified viewing direction, which is interpreted as the *query with direction*. In this section we introduce several new spatial query types for searching camera viewable scenes. We will formulate these query types in Section 4.1. All the queries work at the FOV level. In Section 4.2 we will provide the details about the query processing and present the algorithms of the proposed queries.

4.1 Query definitions

Let $FOV_{v_j} = \{FOV_{v_j}(i), i = 1, 2, ..., \widehat{n_j}\}$ be the set of FOV objects for video v_j and let $\mathbb{FOV} = \{FOV_{v_j}, j = 1, 2, ..., \widehat{m}\}$ be the set of all FOVs for a collection of $\widehat{m}$ videos. Given $\mathbb{FOV}$, a query q returns a set of video segments $\{VS_{v_j}(s, e)\}$, where $VS_{v_j}(s, e) = \left\{FOV_{v_j}(i),\ s \leq i \leq e\right\}$ is a segment of video v_j which includes all the FOVs between $FOV_{v_j}(s)$ and $FOV_{v_j}(e)$, where i stands for the ith frame, and $< s, e >$. denotes the starting and ending frame of a video segment respectively.

Definition 1 *Point Query with Bounded Radius* (PQ-R):
Given a query point q in geo-space and a radius range from MIN_R to MAX_R, the PQ-R query retrieves all video segments that overlap with q and whose camera locations are at least MIN_R and at most MAX_R away from q, i.e.,

$$\begin{aligned} &PQ\text{-}R(q, MIN_R, MAX_R) : \\ &q \times \mathbb{FOV} \rightarrow \Big\{VS_{v_j}(s, e),\ \text{where } \forall j\ \forall i\ s \leq i \leq e \text{ such that } FOV_{v_j}(i) \cap q \neq \varnothing\ , \\ &\qquad \text{and } MIN_R \leq dist(P(FOV_{v_j}(i)), q) \leq MAX_R\Big\}\ , \end{aligned}$$

where P returns the camera location of an FOV and function $dist$ calculates the distance between these two points. Here, we define $\cap$ as the geographic overlap between the two factors.

Definition 2 *Point Query with Direction* (PQ-D):
Given a query point q in geo-space and viewing direction β, the PQ-D query retrieves all FOVs that overlap with q and that were taken when the camera was pointing towards β with respect to the North. The PQ-D query exploits the camera's bearing to retrieve the video frames that show the query point from a particular viewing direction. Since slight variations in the viewing direction does not significantly alter the human perception, using only a precise direction value β may not be practical in video search. Therefore a small angle margin ε around the query view direction is introduced, and the query searches for the video segments whose directions are between $\beta - \varepsilon$ and $\beta + \varepsilon$.

$$\begin{aligned} &PQ\text{-}D(q, \beta) : \\ &q \times \mathbb{FOV} \rightarrow \Big\{VS_{v_j}(s, e),\ \text{where } \forall j\ \forall i\ s \leq i \leq e \text{ such that } FOV_{v_j}(i) \cap q \neq \varnothing\ , \\ &\qquad \text{and } \beta - \varepsilon \leq D(FOV_{v_j}(i)) \leq \beta + \varepsilon\Big\}\ , \end{aligned}$$

where D returns an FOV's camera direction angle with respect to North.

Definition 3 *Range Query with Bounded Radius* (RQ-R):
Given a rectangular region q_r in geo-space and a radius range from MIN_R to MAX_R, the RQ-R query retrieves all video segments that overlap with q_r and whose camera locations are at least MIN_R and at most MAX_R away from the border of q_r. RQ-R definition is very similar to PQ-R query, therefore we omit further details here.

Definition 4 *Range Query with Direction*(RQ-D):
Given a rectangular region q_r in geo-space and a viewing direction β, the RQ-D query retrieves all video segments that overlap with region q_r and that show it with direction interval between $\beta - \varepsilon$ and $\beta + \varepsilon$. We also omit the details for RQ-D query, as the definition is similar to PQ-D query.

Definition 5 *k-Nearest Video Segments Query* (k-NVS):
Given a query point q in geo-space, the k-NVS query retrieves the closest k video segments that show the query point q. The returned video segments are ordered from closest to the farthest based on their distance to q.

$$
\begin{aligned}
&k\text{-}NVS(q,k):\\
&q \times \mathbb{FOV} \rightarrow \{(VS_{v_j}(s_1,e_1),..,VS_{v_j}(s_k,e_k)) \text{ where } \forall s_t, e_t (t=1,..,k),\\
&\qquad \text{and } \forall j \forall i \; s_t \le i \le e_t, \text{ such that } FOV_{v_j}(i) \cap q \neq \varnothing\\
&\qquad dist(VS_{v_j}(s_t,e_t),q) \le dist(VS_{v_j}(s_{t+1},e_{t+1}),q)\big\},
\end{aligned}
$$

The function $dist$ calculates the minimum distance between the camera locations of a video segment and the query point.

Definition 6 *k-Nearest Video Segments Query with bounded Radius* (k-NVS-R):
The k-NVS-R query is similar to the k-NVS and PQ-R queries. Give a query point q in geo-space and a radius range from MIN_R to MAX_R, the k-NVS-R query retrieves the closest$k$ video segments that show the query point q from a distance between MIN_R to MAX_R. Similar to the k-NVS query, the returned video segments are ordered from the closest to the farthest based on their distance to q.

Definition 7 *k-Nearest Video Segments Query with Direction* (k-NVS-D):
The k-NVS-D query is also similar to the k-NVS and PQ-D queries. Given a query point q in geo-space and a viewing direction β, the k-NVS-D query retrieves the closest k video segments that show the query point q with the direction β.

4.2 Algorithm design

The query processing is performed in two major steps. In the first step, the FOVs (i.e., the video frames) that satisfy the query conditions in the set $\mathbb{FOV}$ are retrieved. The returned FOVs are grouped according to the video files that they belong to. And in the second step, the groups of adjacent FOVs from the same videos are post processed to retrieve as the video segments in the query results. We argue that, the length of the resulting video segments should be larger than a certain threshold length for visual usefulness. For some query types, such as RQ-R and RQ-D queries, the number of consecutive FOVs that match the query requirements is usually large enough to form a reasonable length video segment, therefore this post processing step is straightforward. However, for more restricted queries such as k-NVS query, often the formed video segments may contain only a few FOVs. Therefore this post processing step may add additional video frames to the video segments according to the requirements of the search application.

Next we will further elaborate on these two major steps of the query processing. In Section 4.2.1, we will describe the retrieval of the FOVs that match the query requirements for each of the proposed query types. In Section 4.2.2, we will describe a simple approach for the post processing of the retrieved FOVs to form the resulting video segments.

4.2.1 Query processing: retrieval of matching FOVs

In this section, we will present the algorithms for running the proposed query types on our three-level grid structure. We will describe these queries under three groups: Point query (PQ-R and PQ-D), Range query (RQ-R and RQ-D) and k-NVS query (k-NVS and k-NVS-D). Within each query group, we will further elaborate on the direction and bounded radius queries.

We retrieve the FOVs that match the query requirements in two steps: a *filter step* followed by a *refinement step*. First, in the filter step, we search the three-level index structure starting from the first level and then moving down to the second and third level, if needed. The set of FOVs resulting set from the filter step are referred as the *candidate set*. In the refinement step, an exhaustive method is applied to check whether an FOV actually satisfies the query conditions.

Point Query The video segments, that show a certain object of interest at a specific location, can be retrieved through the point query. When the object size is small, it would be preferred to retrieve the close-up views of the object, with a reasonable size for better visual perception. The PQ-R query searches the video frames with a certain radius range restriction for the distance of the camera locations from the query point, according to the required level of details in the video. Additionally, the camera viewing direction when the query object appears in the video can be an important factor for the image perception of the observer. For example, an object's images from a certain viewing direction (e.g., the frontal view, when the object is viewed from the North) can be of higher importance. The PQ-D query can exploit the collected camera directions for querying video segments when the camera is pointing towards the requested direction (e.g., the North).

Algorithm 1: Point query with bounded radius (PQ-R) and direction (PQ-D).

Input: query type: q_type (PQ-R, or PQ-D), query point: $q\langle lat, lng\rangle$,
(for PQ-R) min and max radius: MIN_R, MAX_R,
(for PQ-D) viewing direction : β
Output: vector $segments$ $\langle v_j, VS(s_t, e_t)\rangle$

```
1  C_ℓ1 = getCellID(q);                                   /* First-level cell */
   /* Point Query with bounded Radius                                        */
2  if q_type is PQ-R then
3  |  C_ℓ2 = getSubCellID(q);                            /* Second-level cell */
4  |  subCellsInR = applyRadius(q, C_ℓ2, MIN_R, MAX_R);
5  |  candidateFOVs = fetchData(subCellsInR);
6  end
   /* Point Query with Direction                                             */
7  if q_type is PQ-D then
8  |  C_ℓ3 = getDirCellID(C_ℓ1,β,ε);                     /* Third-level cell */
9  |  candidateFOVs = fetchData(C_ℓ3);
10 end
11 res = refinementStep(candidateFOVs);
12 segments = getVideoSeg(res,q_type);
13 return segments
```

Algorithm 1 formalizes the query execution for point queries PQ-R and PQ-D. When processing the point query, we first calculate the first-level cell ID $C_{\ell 1}$ where the query point is located. For a typical point query (PQ), the candidate FOVs would include all FOVs indexed at the cell $C_{\ell 1}$. For the *Point Query with bounded Radius* (PQ-R), we additionally apply the distance condition given by the radius range (MIN_R, MAX_R). The function

applyRadius reduces the search area for the candidate FOVs in the second-level index by eliminating the subcells outside of the radius range (see Algorithm 2). In function *applyRadius*, we first retrieve the $C_{\ell 2}$ subcell where query point q is located. Then we find out all the second-level subcells around $C_{\ell 2}$, which are within distance range MIN_R to MAX_R from the query point. For example, in Fig. 4, according to the minimum (MIN_R) and maximum (MAX_R) distance conditions, only the FOVs located between the two dot circles will be returned. Since this function works on subcell level, it takes all the subcells that overlap with the region between two circles (i.e., the shadow region) into account. In this example, both video frames $FOV1$ and $FOV2$ overlap with q. However, since the location of $FOV2$ is outside of the shadow region, it won't be returned by the function *applyRadius*, and therefore $FOV2$ will not be included in the candidate set. For the *Point Query with Direction* (PQ-D), we check the third–level index cell to find cells that cover the query angle range given by ($\beta-\varepsilon$, $\beta+\varepsilon$). We return the FOVs indexed in the cells $\{C_{\ell 3}(h_1), ..., C_{\ell 3}(h_2)\}$ where $\beta - \varepsilon$ falls into the angle range of $C_{\ell 3}(h_1)$ and $\beta+\varepsilon$ falls into the angle range of $C_{\ell 3}(h_2)$. As an example, let us assume that the 360° viewing angle range is divided into $x = 45°$ intervals in the third-level index. When $\beta = 0°$ (i.e., North) and $\varepsilon = 5°$, we would retrieve the FOVs in the third-level cells $C_{\ell 3}(7)$ and $C_{\ell 3}(0)$ as the candidate FOVs.

After the candidate FOVs are retrieved, we run the refinement step (through the function *refinementStep*) to get the actually matching FOVs (See Algorithm 3). For each FOV in the candidate set we check whether the FOV overlaps with q. In the refinement step of PQ-R query, we also check whether the distance between the camera location and the query point is within radius range (MIN_R, MAX_R). While for PQ-D query, we check whether the viewing direction of the camera falls into the angle range ($\beta - \varepsilon$, $\beta+\varepsilon$). These FOVs, along with their video file IDs (v_j) are stored in the vector res.

The last step in the point query processing is the generating of resulting video segments from the retrieved FOVs. The function $getSegments$ organizes the group of consecutive FOVs from the same video as video segments $VS_{v_j}(s_t, e_t)$, where s_t is the starting FOV ID

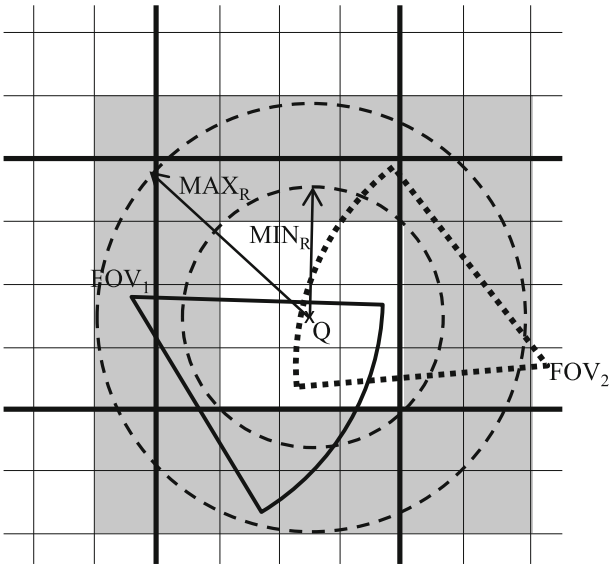


Fig. 4 Illustration of the function *applyRadius*

and e_t is the ending FOV ID for the segment. The details of the *getSegments* function is explained in Section 4.2.2.

Algorithm 2: applyRadius()

Input: query point: $q\langle lat, lng\rangle$,
Minimum and maximum bounded radius: MIN_R, MAX_R,
first–level cell: $C_{\ell 1}$, second–level cell: $C_{\ell 2}$
Output: set of second-level cells: *checkRadius*

```
distClose = compMinDist(q, C_ℓ2);
while distClose ≤ MAX_R do
    /* Find out the minimum distance between the q and C_ℓ2 */
    distFar = compMaxDist(q, C_ℓ2);
    if distFar ≥ MIN_R then
        checkRadius.add(getCellsAtDist(q,distClose));
    end
    distClose += GRIDSIZE/s;
end
return checkRadius
```

Algorithm 3: refinementStep()

Input: query type: *q_type* (PQ-R, or PQ-D)
FOV candidate set: vector *candidateFOVs*,
(for PQ-R) min and max radius: MIN_R, MAX_R
Output: vector *res* $\langle v_j, FOV.id\rangle$

```
for all the FOVs in the candidateFOVs do
    if q_type is PQ-R then
        distP2P = dist(q, FOV.P);          /* distance between two points */
        if distP2P ≥ MIN_R AND distP2P ≤ MAX_R then
            if pointInFOV(q, FOV) then
                res.push( ⟨FOV.v_j, FOV.id⟩ );
            end
        end
    end
    if q_type is PQ-D then
        if FOV.θ ≥ β − ε AND FOV.θ ≤ β + ε then
            if pointInFOV(q, FOV) then
                res.push( ⟨FOV.v_j, FOV.id⟩ );
            end
        end
    end
end
```

Range Query When the search application asks for the videos that show a large region in geo-space, rather than a point location, it may issue a range query. The queried region is estimated with a bounding rectangle. Similar to the PQ-R query, the closeness to the query region, therefore the level of details in the video, can be customized through the RQ-R query. Additionally, the RQ-D query retrieves videos of the query region from different view points.

In the range query processing, a naive approach is to access only to the first-level index to get the candidate FOVs. Since the first-level grid cells are larger, each FOV appears only in a few $C_{\ell 1}$ cells. When the overlap area between the $C_{\ell 1}$ cell and the query rectangle is large, using the first-level index is more efficient, since the duplicate FOV IDs in the candidate set is minimized. On the other hand, if the query rectangle overlaps with a small percentage of

the $C_{\ell 1}$ cell, the retrieved candidate set will have many false positives due to FOVs covering parts of the $C_{\ell 1}$ cell but not the query region. Therefore, in our range query processing algorithm, we use a hybrid approach where we try to cover the query region with a mixture of $C_{\ell 1}$ and $C_{\ell 2}$ cells. We try to minimize the uncovered regions in cells (i.e., minimizing the false positives) and at the same time, we also minimize the duplicate FOV IDs in the candidate set, by using as many $C_{\ell 1}$ cells as possible. The goal is to reduce the size of the candidate set, so that the time required to process and sort the FOVs in the refinement step is minimized.

Algorithm 4 formalizes the query execution for the range queries RQ-R and RQ-D. In Algorithm 4 we first find out which cells will be accessed from the first-level and second-level indexes. Among the $C_{\ell 1}$ cells that overlap with q_r, we choose the cells whose overlap areas are larger than a certain threshold value ϕ (e.g., the overlap area is 40 % of that of the $C_{\ell 1}$ cell). If the overlap area is less than ϕ, we cover the overlap region with the $C_{\ell 2}$ subcells. Recall that the second-level subcells hold the list of the FOVs whose camera locations are within those subcells. Therefore, to retrieve the candidate FOVs from a $C_{\ell 2}(m, n)$ subcell, we need to search for the neighboring subcells around it, and find out the FOVs in those subcells which overlap with the $C_{\ell 2}(m, n)$. After finding out the cells and subcells that we would retrieve the candidate FOVs from, the rest of the query processing is similar to PQ-R and PQ-D queries.

k-NVS Query Typical kNN queries consider only the distance between a query point and the objects in the database. In our geo-tagged video search system, we consider not only the distance between the query point and camera location in the database, but also the visibility of the query point from the camera location. Here, we propose the *k-Nearest Video Segments* query as, "For a given point q in geo-space, find the k nearest video segments that overlap with q". Taking Fig. 5 as an example, the camera locations of the video segment V_1 are closer to the query point q than that of V_2. Due to the camera's location and viewing direction, the FOVs of V_1 cannot cover q while the FOVs of V_2 can. In typical kNN queries, V_1 will be selected before V_2 because V_1 is closer to q. However, in the k-NVS query, V_2 will be selected as the nearest neighbor instead of V_1 because of the visibility. The k-NVS query can be utilized in various video search applications to continuously retrieve

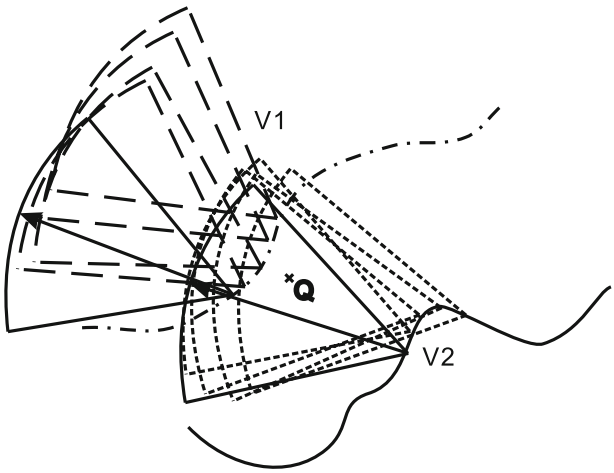


Fig. 5 Illustration of k-NVS query

the most related videos that show a frequently updated query point. Additional radius range and viewing direction requirements can be added to the query through the k-NVS-R and k-NVS-D queries.

Algorithm 4: Range query with bounded radius (RQ-R) and direction (RQ-D).

```
   Input: query type: q_type (RQ-R, or RQ-D)
   query rectangle: q_r⟨lat1, lng1; lat2, lng2⟩,
   (for RQ-R) min and max radius: MIN_R, MAX_R,
   (for RQ-D) viewing direction : β
   Output: vector segments ⟨v_j, VS(s_t, e_t)⟩
 1 C_ℓ1 = getCellID(q_r)
   /* Range Query with bounded Radius                                   */
 2 if q_type is RQ-R then
 3     cellsInR = applyRadius(q_r, C_ℓ1, MIN_R, MAX_R);
 4     for each cell C_ℓ1(m, n) in cellsInR do
 5         overlapArea = compOverlap(C_ℓ1(m, n),q_r); /* Compute the overlap area */
 6         if overlapArea ≥ φ then
 7             candidateFOVs.append(fetchData(C_ℓ1(m, n)));
 8         end
 9         else
10             subCellsInR = applyRadius(overlapArea,C_ℓ2,MIN_R,MAX_R);
11             candidateFOVs.append(fetchData(subCellsInR));
12         end
13     end
14 end
   /* Range Query with Direction                                        */
15 if q_type is RQ-D then
16     for each cell C_ℓ1(m, n) in cellsInR do
17         overlapArea = compOverlap(C_ℓ1(m, n),q_r);
18         if overlapArea ≥ φ then
19             C_ℓ3 = getDirCellID(C_ℓ1(m, n),β,ε);
20         end
21         else
22             subCells = applyRadius(overlapArea,C_ℓ2,0,R_V);
23             C_ℓ3 = getDirCellID(subCells,β,ε);
24         end
25         candidateFOVs.append(fetchData(C_ℓ3));
26     end
27 end
28 res = refinementStep(candidateFOVs);
29 segments = getVideoSeg(res,q_type);
30 return segments
```

Algorithm 5 formulates the k-NVS query processing. We first retrieve the $C_{\ell 2}$ cell where the query point is located and, similar to the PQ-R query, we find out the neighboring subcells around $C_{\ell 2}$ from which the FOVs can see q. For the k-NVS-R query, the search range around the $C_{\ell 2}$ cell is (MIN_R, MAX_R) whereas for the k-NVS and k-NVS-D queries search range is (0, R_V). For the the k-NVS queries, we need to return only the closest k video segments. Therefore, in order to find the candidate FOVs, we gradually search the neighboring subcells in the search range, starting with the closest subcells. As shown in Algorithm 5, we first retrieve the candidate FOVs in the subcells closest to $C_{\ell 2}$ (within distance 0 or MIN_R). And at each round we increase the search distance by δ/s and retrieve the FOVs in the next group of cells within the enlarged distance (δ/s is the size of a second-level subcell). We apply the refinement step on these candidate FOVs and store them in

priority queue, in which the FOVs are sorted based on their distance to q in ascending order. The refinement steps for the k-NVS-R and k-NVS-D queries are similar to PQ-R and PQ-D queries. After each round of candidate retrieval, the candidate FOVs are organized as videos segments, i.e., the consecutive FOVs from the same video file are grouped together. The search for candidate FOVs ends either when the number of video segments reaches k or when there are no more subcells that need to be checked. The output of the algorithm is the list of the retrieved video segments, ordered from closest to the farthest.

Algorithm 5: k-Nearest Video Segments Queries: k-NVS, k-NVS-R, and k-NVS-D

Input: query point: $q\langle lat, lng\rangle$, number of output video segments: k,
(for k-NVS-R) min and max radius: MIN_R, MAX_R,
(for k-NVS-D) viewing direction : β,
Output: vector *segments* $\langle VS_{v_j}(s_t, e_t)\rangle$

$C_{\ell 1}$ = getCellID(q), $C_{\ell 2} = getSubCellID(q)$;
priority_queue $sortedFOVs\ \langle\ FOVID$, distance to $q\ \rangle = \emptyset$;
if *q_type is k-NVS-R* **then**
  $subCellsInR$=applyRadius(q,$C_{\ell 2}$,MIN_R,MAX_R);
end
else
  $subCellsInR$ = applyRadius(q, $C_{\ell 2}$, 0, R_V);
end
i=0; distClose=δ/s;
while *not enough FOVs AND nextSubCells= getNeighbors(q,subCellsInR,i++) is not empty* **do**
  $candidateFOVs$ = fetchData($nextSubCells$);
  for *all the FOVs in the candidateFOVs* **do**
    $distP2P$ = dist(q, FOV);
    if *q_type is k-NVS* **then**
      if *pointInFOV(q, FOV)* **then**
        $sortedFOVs$.push($\langle FOVID, distP2P\rangle$);
      end
    end
    if *q_type is k-NVS-R* **then**
      if $distP2P \geq MIN_R$ *AND* $distP2P \leq MAX_R$ *AND pointInFOV(q, FOV)* **then**
        $sortedFOVs$.push($< FOVID, distP2P >$);
      end
    end
    if *q_type is k-NVS-D* **then**
      if $FOV.\theta \geq \beta - \varepsilon$ *AND* $FOV.\theta \geq \beta + \varepsilon$ **then**
        $sortedFOVs$.push($< FOVID, distP2P >$);
      end
    end
  end
  while $sortedFOVs.top() \leq distClose$ *AND* $numsegments < k$ **do**
    $topFOV = sortedFOVs$.pop();
    if *isNewSegment(topFOV,res)* **then**
      $numsegments$++;
    end
    res.push($topFOV$);
  end
  i++; $distClose += \delta/s$;
end
$segments$ = getVideoSeg(res,q_type);
return *segments*

4.2.2 *Query processing: returning video segments*

As explained in Section 4.2.1, in query processing after retrieving the FOVs that satisfy the query requirements, the groups of adjacent FOVs from the same videos are returned as the resulting video segments. The length of the returned segments may vary extensively for different query types. For example for the range query, when the query region expands to a large area, the number of consecutive FOVs that overlap with the query region is usually large. However for more selective queries, such as k-NVS query, the length of an individual segment can be as short as a few seconds. In Table 1, we report the number of FOVs returned from the k-NVS query and the number of video segments that these FOVs form for different values of k. The average segment length for $k = 20$ is around 3 seconds, with a maximum segment length of 20 seconds. As the k value increases, the segment lengths also get longer. Practically, for visual clarity, the length of the resulting video segments should be larger than a certain threshold length. Depending on the requirements of the video search application, the query processing unit should customize the creation of the returned segments.

In our current implementation, for the point and range queries, the returned FOVs are post-processed to find out the consecutive FOVs that form the segments. If two separate segments of the same video file are only a few seconds apart, they are merged and returned as a single segment. For the k-NVS query, the video segments are formed simultaneously as the closest FOVs are retrieved. For each video segment the video ID, the starting and ending FOV IDs and the segments distance to the query point are maintained, i.e., $\langle v_j, s_t, e_t, dist \rangle$. When the next closest FOV is retrieved, if it is adjacent to one of the existing segments it is merged with it, otherwise a new segment is formed. The s_t, e_t, and $dist$ values are updated accordingly. For example, in our current configuration of the experiments, we set that the returned segments should be at least 20 seconds long. Therefore the short segments are expanded to 20 seconds. The segment's starting and ending offsets are adjusted so that the $dist$ value for the segment is minimized.

5 Experimental evaluation

In this section, we elaborate on the experiments carried out on two datasets: one small set of real-world videos and sensor meta-data, and one large synthetic dataset. We use the real-

Table 1 Statistics for k-NVS queries with different k values

k	# of FOVs	# of Segments	Segment Max Length
20	109,847	35,391	20
50	212,746	56,664	50
100	291,957	72,179	71
150	318,504	77,096	89
200	326,110	78,523	89
300	327,541	78,796	89

world dataset to show the importance of the bounded radius and direction restrictions on displaying videos, and the synthetic dataset to demonstrate the scalability of the proposed structure for large-scale applications.

5.1 Experiments with a real-world dataset

We collected 1200 geo-tagged videos (representing about 38.55 hours) all around Singapore. The lengths of videos vary from less than one minute to 18.45 minutes. To show how the bounded radius and direction restriction affects the video searching results, we presented two representative query results on two landmarks: Q1 targets at the Merlion (a small statue) while Q2 targets the Marina Bay Sands (a tall and wide building).

5.1.1 Importance of bounded radius

Figure 6 shows the sampling frames from the resulting videos that answering Q1 and Q2 with various distances from the query locations. All the frames shown in Fig. 6a actually capture Q1 in the scene, but only the first one from a closed position less than 50 meters away from Q1 displays a clear view. Due to the large distance, although Q1 appears in the other three frames, it is meaningless to end-users who want to see the the Merlion. The situation is quite different when processing Q2 (shown in Fig. 6b). The frames captured from closed positions only show parts of the building, while users can have a panoramic view of the whole building from far away. Consequently, typical spatial queries on video search might sometimes not satisfy users' demands, and it is helpful to have spatial queries with bounded radius. For example, displaying small objects from a closed position, while displaying large buildings from a far-away location can help to show meaningful videos to users.

(a) Sampling frames from Q1.

(b) Sampling frames from Q2.

Fig. 6 Sampling frames from the video searching results with various distances. **a** Sampling frames from Q1. **b** Sampling frames from Q2

5.1.2 Importance of viewing direction

The sampling frames in Fig. 7 are extracted from the same video and all of them capture the Merlion from different directions. The first two frames record Q1 from the best place while the last one records it from the back, which is not desired by users. Thus, direction restriction is also an important factor for video searching on displaying best results.

5.2 Experiments on a synthetic dataset

Due to the difficulty of collecting large set of real videos associating with meta-data, a synthetic dataset that simulate the movements of cameras was used to test the performance of the grid-based index structure with large-scale data.

5.2.1 Synthetic data generation

The dataset for moving camera trajectories was generated with positions inside a 75 km $\times$ 75 km region in the geo-space using the Georeferenced Synthetic Meta-data Generator [2]. The generated synthetic meta-data exhibit equivalent characteristics to the real world data. The camera's viewable angle is 60° and the maximum visible distance is 250 m [1]. In the experiments, we chose 100 randomly-distributed center points within the area and generate 5, 500 moving cameras around these center points. Hence each one of the cameras is traced for 1, 000 seconds, with snapshot of one frame per second, due to the sampling rate of GPS and the compass. (Note that the digital compass can achieve 30 or 40 readings per second but GPS can only get one sample per second. Thus, we only collect one snapshot per second). Therefore we have a dataset with about 5.4 million video frames. The center points are randomly distributed in the experiment region, which are used as the initial positions of the camera location. Subsequently, the cameras start to move inside the region under a maximum speed limit, as well as a viewing direction rotation limit. The speed of the cameras, and the position of center points, affect the final distribution of the frames. Faster movement causes the frames distributed uniformly throughout the region in contrast to slower movement. To simulate real-world case, we set the maximum speed of moving cameras as 60 km/h, with the average speed as 20 km/h. Besides the speed limit, we also set the camera's maximum rotation limit as 30° per second, which guarantees that the camera rotates smoothly and not jump from one direction to another, the same as what people do when they are capturing videos. With restriction to these limitations, unexpected data (e.g., the object's speed is larger than the speed threshold, viewing direction rotates over the rotation limit and etc.) are thrown away from the dataset. The parameters used are summarized in Table 2.

Fig. 7 Sampling frames from the video searching results with various directions on Q1

Table 2 Parameters of the synthetic dataset

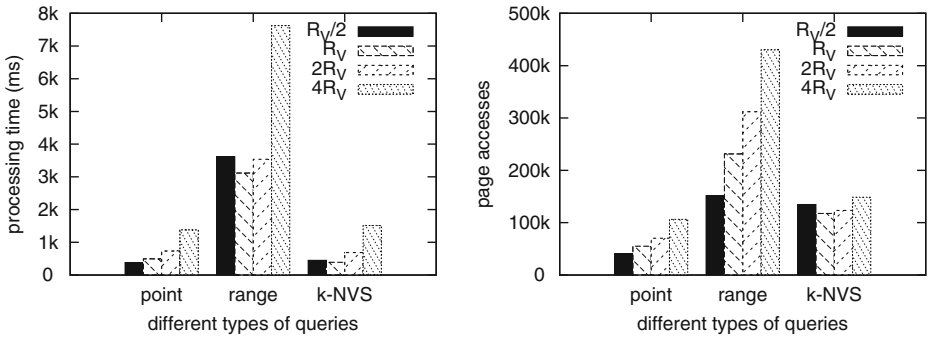
Parameter	Value
# of Center points	100
Speed Limit	60 km/h
Average Speed	20 km/h
Rotation Limit	30 °/s
# of Cameras	5500
# of Snapshots	1000
# of FOVs	5405051
viewable angle of FOV (α)	60 °
visible distance of FOV (R_V)	250 m

5.2.2 Experimental settings

For all the experiments we constructed a local MySQL database and stored all the FOV meta-data, as well as the index structure tables. All the experiments were conducted on a server with two quad core Intel(R) Xeon(R) $X5450$ 3.0GHz CPUs and 16GB memory running under Linux RedHat 2.6.18. All the comparisons used in the experiments are based the geo-coordinates (latitude and longitude). The experiment results reported here show the cumulative number of FOVs returned for $10,000$ randomly generated queries within the experiment region. In our experiments, we mainly measure the Processing Time (PT for short) and the number of Page Access (PA for short). The PT includes the total amount of time for searching for the candidate set through the index structure in the filter step and the time for using the exhaustive method to process overlap calculation in the refinement step. We assume that even if the index structure was in memory, when we access to it, we count the PA as it is on disk. Additionally, we set the page cache size as one page large. Therefore, when there is no page hit in the cache, the PA will be increased by one. This also helps to analyze the performance if the index structure is disk-based instead of memory-based. In our experiments, we try to fully utilize the space inside each one page by storing as many nodes as possible for both the grid-based approach and R-tree. The page has empty space only when there exists no exact match between the page space and the node size.

In the next two experiments, we process the typical queries without any distance or direction condition as preliminary experiments to decide the basic parameters: the value of the grid size δ and the overlap threshold ϕ. In the following experiments, if not specifying, the default value of k is 20, and query rectangle size is 250 m $\times$ 250 m. When generating the moving objects, the maximum viewable distance (R_V) of the camera is set as 250 m. As shown in Fig. 8, grid with size equalling to $R_V/2$ or R_V performs better than larger sizes. The performance of grid-based index structure with size of R_V is better in some cases while worse in others compared to that of $R_V/2$. Since our structure is mainly designed for k-NVS query and the value of the PT and the PA to process k-NVS query is minimum when δ equals to R_V, we thus choose δ equalling to R_V as the optimized configuration.

As well as the grid size, the overlap threshold ϕ for range query also affects the performance of the grid-based index structure. As presented in Section 3, both the first-($C_{\ell 1}$) and second-level($C_{\ell 2}$) indices are loaded into memory. To decide the value of ϕ, we ran a series of typical range queries without distance and direction conditions. As shown in Fig. 9, the PA of grid-based index structure is smaller than that of R-tree when ϕ is smaller than 40 %.



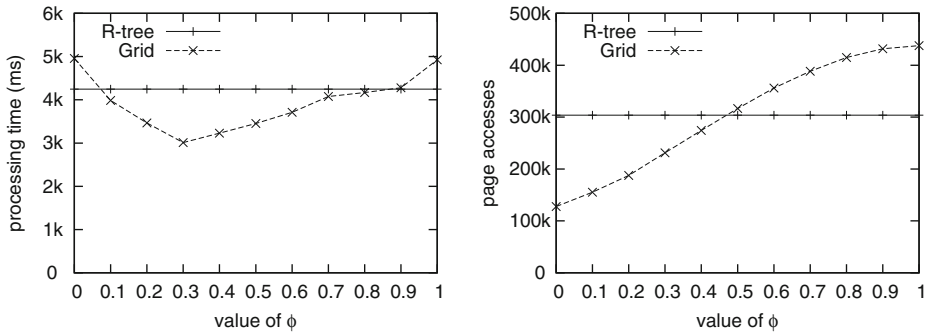
(a) Processing time. (b) Page accesses.

Fig. 8 Effect of grid size. **a** Processing time. **b** Page accesses

Moreover, the grid approach is faster than R-tree for most of the cases, and we achieve the fastest performance at value of 30 %. Consequently, ϕ is chosen as 30 %. The parameters used in the experiment are summarized in Table 3.

5.2.3 Comparison

The R-tree is one of the basic index structures for spatial data which is widely used. In our experiments, we insert the Minimum Bounding Rectangle (MBR for short) of all FOVs into R-tree and process all types of queries based on R-tree [9] implemented by Melinda Green for comparison. To the best of our knowledge, this implementation achieves the best performance compared to others. We use Eq. 1 to calculate the MBR of an FOV with geo-coordinates. The parameter σ_x and σ_y denotes the factor of converting distance to geo-coordinate difference in the x-axis or y-axis directions, respectively. The query procedure is to search for all the FOVs whose MBRs overlap with the query input in the filter step and hence to use the exhaustive method to calculate the actual overlap in the refinement step.



(a) Processing time. (b) Page accesses.

Fig. 9 Effect of the overlap threshold ϕ on range query performance. **a** Processing time. **b** Page accesses

Table 3 Experiment parameters and their values

Parameter	Value
Page Size	4, 096
Page Cache Size	4, 096
Non-Leaf Node Size(R-tree)	64
Leaf Node Size (R-tree)	36
Non-Leaf Node Size (R-tree with viewing direction as a dimension)	80
Leaf Node Size (R-tree with viewing direction as a dimension)	52
$C_{\ell 1}$ Node Size	68
$C_{\ell 2}$ Node Size	36
$C_{\ell 3}$ Node Size	4
FOV Meta-data Size	32
Grid Size δ	250 m
s	4
Angle Error Margin ε	15 °
Overlap Threshold ϕ	30 %

Consequently, some of the parameters (e.g., value of k for k-NVS query, distance condition, etc.) have no effect on PA for R-tree.

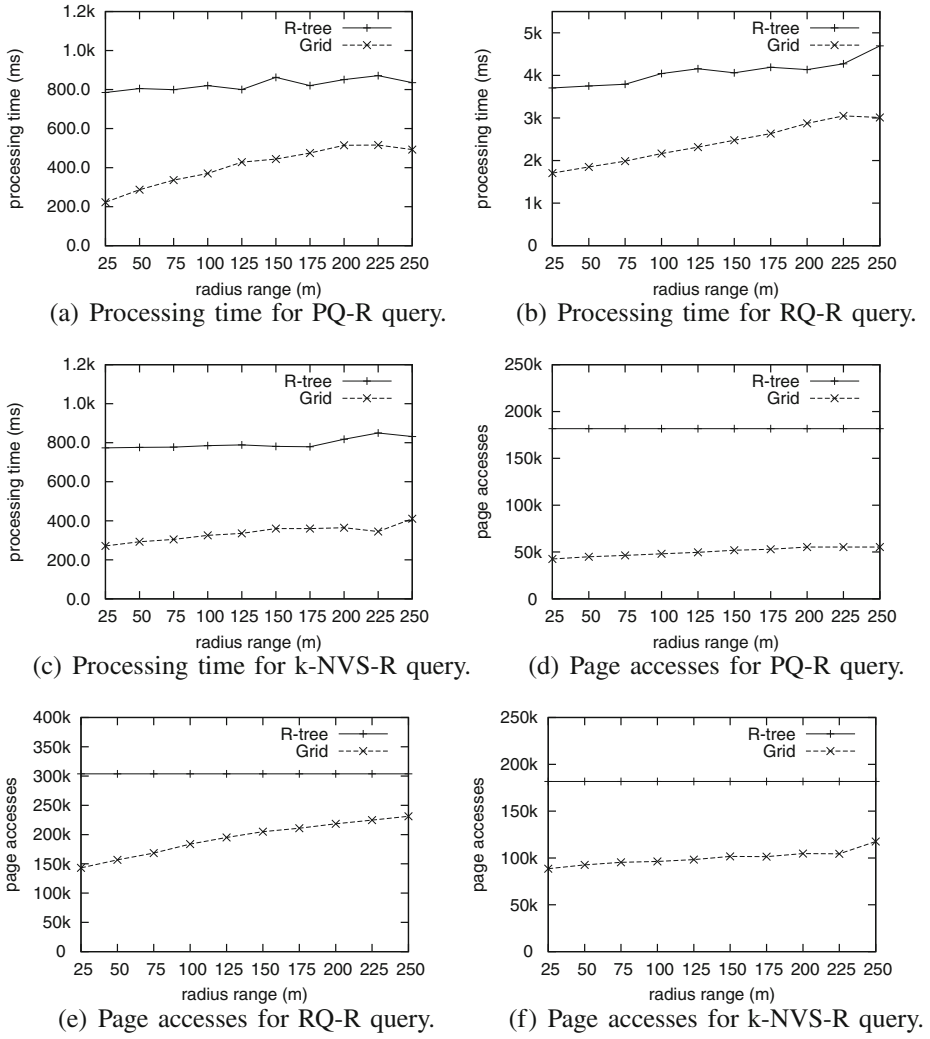
$$MBR.left = min(lng, lng \pm R_V \times \sin(\theta \pm \alpha/2)/\sigma_x) \quad (1)$$

$$MBR.right = max(lng, lng \pm R_V \times \sin(\theta \pm \alpha/2)/\sigma_x) \quad (2)$$

$$MBR.bottom = min(lat, lat \pm R_V \times \cos(\theta \pm \alpha/2)/\sigma_y) \quad (3)$$

$$MBR.ceil = max(lat, lat \pm R_V \times \cos(\theta \pm \alpha/2)/\sigma_y) \quad (4)$$

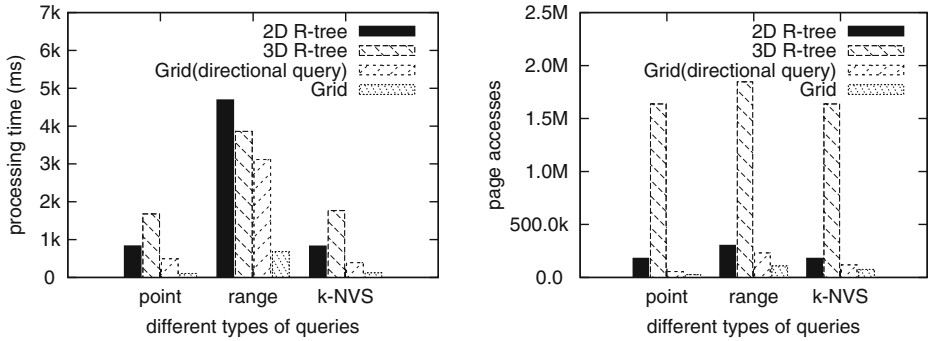
Effect of distance condition We study the effect of the distance condition by varying the radius range from 25 m to 250 m. For each one of the radius range, we start from the minimum distance condition MIN_R equalling to 0 m until the maximum distance condition MAX_R reaching 250 m. For example, when the radius range is equal to $25m$, the value of the tuple $\langle MIN_R, MAX_R \rangle$ can be one of the followings: $\{\langle 0, 25\rangle, \langle 25, 50\rangle, \langle 50, 75\rangle, ... \langle 200, 225\rangle, \langle 225, 250\rangle\}$. While the radius range is equal to 225 m, the value of $\langle MIN_R, MAX_R \rangle$ can only be either $\langle 0, 225\rangle$ or $\langle 25, 250\rangle$. The results shown in Fig. 10 are the averages of the different radius ranges. Since the R_V of an FOV is set as 250 m when generating the synthetic data, the last point with radius range of 250 m is the result of processing queries without distance condition. Figure 10a, b and c illustrate the PT of PQ-R query, RQ-R query and k-NVS-R query respectively, while Fig. 10d, e and f illustrate the PA for each type of query. In general, the performance of our grid-based index structure works better than R-tree on both the PT and the PA. Figure 10a and b show that the PT for radius range of 250 m is a little shorter than that of 225 m. The reason is that all the subcells in the second-level index $C_{\ell 2}$ needs to be checked for large radius range and this costs extra PT compared to queries without distance condition. As shown in Fig. 10d, e and f, the PA using the R-tree remains the same because the R-tree finds out all the FOVs whose MBRs overlap with the query in the filter step, regardless of the radius range. It cannot prune unnecessary search based on the radius range. The PA of the grid-based structure grows as the radius range becomes larger but is still smaller than that of the R-tree even when the radius range reaches the largest number.



(a) Processing time for PQ-R query.
(b) Processing time for RQ-R query.
(c) Processing time for k-NVS-R query.
(d) Page accesses for PQ-R query.
(e) Page accesses for RQ-R query.
(f) Page accesses for k-NVS-R query.

Fig. 10 Effect of distance condition. **a** Processing time for PQ-R query. **b** Processing time for RQ-R query. **c** Processing time for k-NVS-R query. **d** Page accesses for PQ-R query. **e** Page accesses for RQ-R query. **f** Page accesses for k-NVS-R query

Effect of direction condition We proceed to evaluate the efficiency of our grid-based index structure with directional queries. In this experiment, the query datasets are the same as those used in queries without direction condition, except the additional viewing direction constraint. As presented in Table 3, the angle margin ε in this experiment is 15°. The $2D$ and $3D$ R-trees used in Fig. 11 denote the R-tree for processing queries without direction condition and queries with direction condition, respectively. Figure 11a shows that, in the processing of PQ-D query and k-NVS-D query, the PT of the R-tree is almost two times of that without direction condition. The reason for this is that searching one more dimension in the R-tree slows down the performance of the R-tree. The situation is different for RQ-D query because of less number of candidates obtained from the filter step so that the

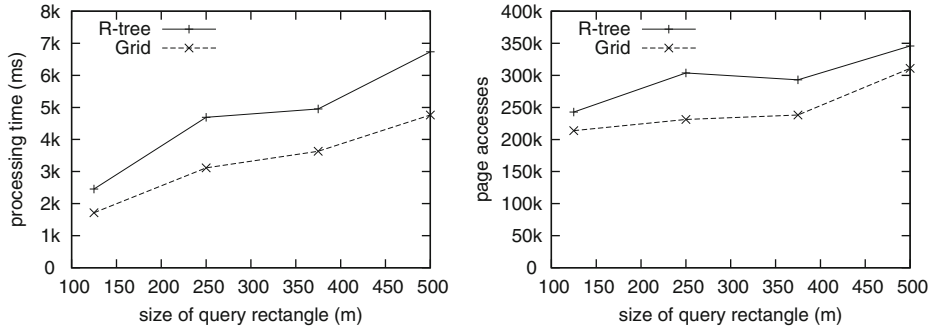


(a) Processing time. (b) Page accesses.

Fig. 11 Effect of direction condition. **a** Processing time. **b** Page accesses

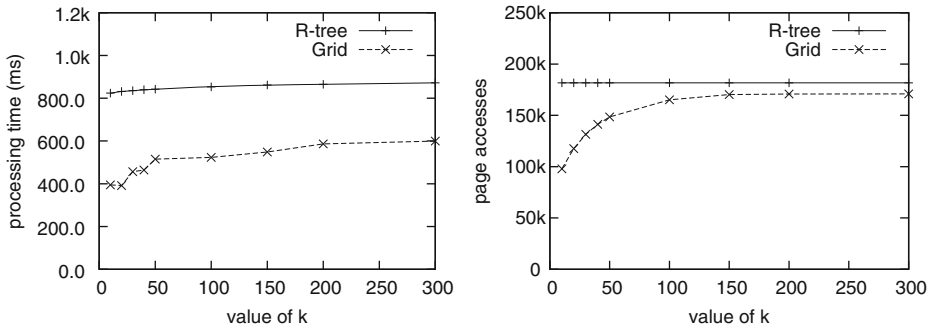
refinement step costs less time. On the contrary, the grid-based approach directly accesses the third-level ($C_{\ell 3}$) cell to narrow down the search for a small amount of meta-data within a short time. Figure 11b shows that the PA in the R-tree for processing queries with direction is over eight times larger than the typical ones while the grid-based approach shrinks to about half. Because most of the PA is to memory pages, the difference in the PT which is not that large as the PA. Comparing queries with and without direction condition between the R-tree and the grid-based approach, our algorithm significantly improves the performance for directional queries.

Effect of query rectangle size We next study the effect of the query rectangle size to range query. The query rectangle size varies from 125 m to 500 m, which is from half to two times of the grid size δ. Larger area contains more number of videos and thus leads to longer processing time and more number of accesses. As expected, the result in Fig. 12 shows that the PT and the PA increases with the query rectangle size. From Fig. 12a, PT increases slower using the grid-based approach, which means that our approach performs even faster for large query area than the R-tree. However, Fig. 12b shows that as the query area grows, the difference in number of the PA between these two methods gets larger when the query rectangle size increases from 125 m to 250 m, and then becomes smaller after the query rectangle size is larger than 250 m, which the maximum visible distance of the camera.



(a) Processing time. (b) Page accesses.

Fig. 12 Effect of rectangle size on range query performance. **a** Processing time. **b** Page accesses



(a) Processing time. (b) Page accesses.

Fig. 13 Effect of value of k on k-NVS query performance. **a** Processing time. **b** Page accesses

Therefore, when users are interested in what happened at special places or small regions, e.g., an area with size 500 m $\times$ 500 m, our grid-based approach outperforms better than the R-tree.

Effect of k value To test the performance of the grid-based approach with different values of k for k-NVS query, we calculate the PT and the PA using the same query points. The results in this experiment are discrete FOVs (not video segments). Figure 13a shows the comparison in the PT and Fig. 13b shows the comparison in the PA. As k increases, the PT increases for the grid-based index at the beginning and keeps nearly unchanged when k is larger than 200, which is closed to the maximum number of FOVs found in PQ. The PT for R-tree is almost the same with different k values because all the results are found and sorted once. When k is larger than 150, the PA for the grid-based approach is almost the same since the searching radius is enlarged to the maximum according to the design of the structure. From the gap in Fig. 13a and b between the R-tree and our approach, we can infer that even if the dataset is large and k is big, the grid-based index structure performs better than the R-tree.

6 Conclusions

In this study we proposed a novel three-level grid-based index structure and a number of related query types that facilitate application access to such augmented, large-scale video repositories. Experiments on a real-world dataset show the importance of the queries with bounded radius and viewing direction restriction. The experimental results with a large-scale synthetic dataset show that this structure can significantly speed up the query processing, especially for directional queries, compared to the typical spatial data index structure R-tree. The grid-based approach successfully supports new geospatial video query types such as queries with bounded radius or queries with direction restriction. We also demonstrate how to form the resulting video segments from the video frames retrieved.

References

1. Arslan Ay S, Zimmermann R, Kim S (2008) Viewable scene modeling for Geospatial video search. ACMMM, pp 309–318
2. Arslan Ay S, Zimmermann R, Kim SH (2010) Generating Synthetic Meta-data for Georeferenced Video Management. In: SIGSPATIAL GIS international conference on advances in geographic information systems. ACM, pp 280–289
3. Beckmann N, Kriegel H, Schneider R, Seeger B (1990) The R*-tree: an efficient and robust access method for points and rectangles. In: ACM international conference on management of data. SIGMOD, pp 322–331
4. Chon H, Agrawal D, Abbadi A (2003) Range and KNN query processing for moving objects in grid model. Mob Netw Appl 8(4):401–412
5. Cisco (2013) Cisco visual networking index: global mobile data traffic forecast update, 2012–2017. http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-.520862.pdf
6. Eppstein D, Goodrich M, Sun J (2005) The skip Quadtree: a simple dynamic data structure for multidimensional data. In: Annual symposium on computational geometry
7. Finkel R, Bentley J (1974) Quad Trees: a data structure for retrieval on composite keys. Acta Informatica 4(1):1–9
8. Graham CH, Bartlett NR, Brown JL, Hsia Y, Mueller CC, Riggs LA (1965) Vision and visual perception
9. Green M (2010) R-Tree, Templated C++ Implementation. http://superliminal.com/sources/RTreeTemplate.zip
10. Guttman A (1984) R-Trees: a dynamic index structure for spatial searching. In: ACM international conference on management of data. SIGMOD, pp 47–57
11. Hwang TH, Choi KH, Joo IH, Lee JH (2003) MPEG-7 Metadata for Video-based GIS Applications. In: IEEE international geoscience and remote sensing symposium, vol 6, pp 3641–3643
12. Kim KH, Kim SS, Lee SH, Park JH, Lee JH (2003) The interactive geographic video. In: IEEE international geoscience and remote sensing symposium, vol 1. IGARSS, pp 59–61
13. Kim S, Arslan Ay S, Yu B, Zimmermann R (2010) Vector model in support of versatile georeferenced video search. In: SIGMM conference on multimedia systems. ACM
14. Liu X, Corner M, Shenoy P (2005) SEVA: sensor-enhanced video annotation. In: ACM international conference on multimedia. SIGMM, pp 618–627
15. Ma H, Arslan Ay S, Zimmermann R, Kim SH (2012) A grid-based index and queries for large-scale geo-tagged video collections. In: 17th international conference, DASFAA workshops. SIM^3, pp 16–228
16. Navarrete T, Blat J (2002) VideoGIS: segmenting and indexing video based on geographic information. In: Conference on geographic information science. AGILE, pp 1–9
17. Nievergelt J, Hinterberger H, Sevcik K (1984) The grid file: an adaptable, symmetric multikey file structure. ACM Trans Database Syst (TODS) 9(1):38–71
18. Nutanong S, Zhang R, Tanin E, Kulik L (2008) The V*-Diagram: a query-dependent approach to moving KNN queries. Proc VLDB Endowment 1(1):1095–1106
19. Okabe A (2000) Spatial tessellations: concepts and applications of voronoi diagrams. Wiley
20. Priyantha NB, Chakraborty A, Balakrishnan H (2000) The cricket location-support system. In: ACM international conference on mobile computing and networking. MobiCom, pp 32–43
21. Rigaux P, Scholl M, Voisard A (2001) Spatial databases with application to GIS, Morgan Kaufmann
22. Roussopoulos N, Faloutsos C, Timos S (1987) The R^+-tree: a dynamic index for multi-dimensional objects. In: VLDB International Conference on Very Large Databases, pp 507–518
23. YouTube (2013) YouTube press statistics. http://www.youtube.com/t/press_statistics
24. Yu FX, Ji R, Chang S-F (2011) Active query sensing for mobile location search. In: The 19th ACM international conference on multimedia. ACM, pp 3–12
25. Zhu Z, Riseman E, Hanson A, Schultz H (2005) An efficient method for geo-referenced video mosaicing for environmental monitoring. In: Machine vision and applications, vol 16. Springer, pp 203–216

He Ma is a Ph.D. student with the Department of Computer Science at the National University of Singapore (NUS). He received his B.S. degree in computer science and technology from Northeastern University, P.R.China in 2008. His research interests are on geo-tagged videos management, uncertainty of sensor data.

Sakire Arslan Ay is a clinical assistant professor with the School of Electrical Engineering and Computer Science at the Washington State University (WSU). Previous to her position at WSU, she had worked as a Research Fellow at the National University of Singapore (NUS) under the supervision of Dr. Roger Zimmermann. She received her M.S. and Ph.D. degrees from the University of Southern California (USC) in 2005 and 2010, and her B.S. degree from Bogazici University in 1999. Dr. Arslan Ay's research interests are in the areas of GIS, databases, geospatial video management, sensor-rich video, and mobile video management.

Roger Zimmermann is an associate professor with the Department of Computer Science at the National University of Singapore (NUS) where he is also a deputy director with the Interactive and Digital Media Institute (IDMI) and a co-director with the Centre of Social Media Innovations for Communities (COSMIC). He received his Ph.D. degree from USC in 1998. Among his research interests are distributed and peer-to-peer systems, collaborative environments, streaming media architectures, geospatial video management, and mobile location-based services. He has co-authored a book, five patents and more than a hundred-forty conference publications, journal articles and book chapters in the areas of multimedia and information management. He is an Associate Editor of the ACM Computers in Entertainment magazine and the ACM Transactions on Multimedia Computing, Communications and Applications journal.

Seon Ho Kim is a computer scientist currently working in the Integrated Media Systems Center (IMSC) at the University of Southern California. Before joining IMSC, he had worked at the University of Denver and the University of the District of Columbia as a faculty member for eleven years since he received his Ph.D. in Computer Science from the University of Southern California in 1999. He also received his BS degree in Electronic Engineering from the Yonsei University, Seoul, Korea in 1986, and M.S. in Electrical Engineering from the University of Southern California in 1994. Dr. Kim's primary research interests include multimedia servers, storage systems, databases, GIS, and mobile media applications.