Metadata Organization and Query Optimization for Large-scale Geo-tagged Video Collections

He Ma, Sakire Arslan Ay
School of Computing
National University of Singapore
Singapore 117417
[mahe, dcssakir]@comp.nus.edu.sg

Roger Zimmermann
School of Computing
National University of Singapore
Singapore 117417
rogerz@comp.nus.edu.sg

Seon Ho Kim
Integrated Media Systems Center
University of Southern California
Los Angeles, CA 90089
seonkim@usc.edu

ABSTRACT

Currently a large number of user-generated videos are produced on a daily basis. It is further increasingly common to combine videos with a variety of meta-data that increase their usefulness. In our prior 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 direction information with the acquired videos, hence allowing them to be expressed and manipulated as spatio-temporal objects. In this study we propose a novel multi-level grid-index and a number of related query types that facilitate application access to such augmented, large-scale video repositories. Specifically our grid-index is designed to allow fast access based on a bounded radius and viewing direction – two criteria that are important in many applications that use videos. We present performance results with a comparison to a multi-dimensional R-tree implementation and show that our approach can provide significant speed improvements of at least 30%, considering a mix of queries.

1. INTRODUCTION

Due to the recent developments in the video capture technology, a large number of user-generated videos are produced everyday. 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. Moreover, a number of sensors (e.g., GPS and compass units) have been cost-efficiently deployed on video cameras. Consequently, some critical meta-data, especially geographical properties of video scenes can be captured while being recorded. This association of video scenes and their geographic meta-data has raised interesting research issues, for example, the captured sensor meta-data can be utilized to aid in indexing and searching of geo-tagged videos at the high semantic level preferred by humans.

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 user-generated online videos, particularly in accuracy and scalability. Alternatively, utilizing related meta-data of videos, especially geographical properties from various sensors, has been introduced and received attention from the multimedia community. Compared to visual content, the meta-data occupies much less space, which makes searching among large scale of videos practical. Furthermore, utilizing these meta-data helps easily access to individual frames instead of viewing the whole video clip.

In our earlier work [3], we proposed to model the viewable scene area (i.e., field of view) of video frames as a pie-shaped geometric figure using geospatial sensor data, such as camera location and direction. This approach converts the content of video into a series of spatial objects. Consequently, the challenging video search problem is transformed into a known spatial data selection problem. The objective then is to index the spatial objects and to search videos based on their geographic properties. Our previous study 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). So, there should be a study of high performance index structure which can effectively harness the captured geospatial meta-data. To our knowledge, there has been no such study for geo-tagged video search.

R-tree [10] (and/or its variance [21, 5]) has been the choice of index structure to index geometric figures. However, its performance deteriorates as the number of figures indexed increases greatly. For example, YouTube archived 13 million hours of videos in a single year (2010) and ever growing in even faster way [1]. Assuming all videos in a huge depository are represented using sensor meta-data, i.e., streams of geospatial objects, R-tree may suffer significantly to provide enough search performance due to its increased heights.

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 multi-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. In searching videos through their geographic coverages, distance and direction are two important criteria that can help to improve query functionality. We introduce a number of related query types which support better human perception of images in resulting videos.

One of the 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. An example application that can utilize k-NVS is automatically building panoramic (360 degree) view of a point-of-interest (i.e., a query point). The application needs to search for the nearest videos that look at the query point from different viewing directions and that are within a certain distance from it. 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 users 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 synthetic datasets. 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 area is referred to as *camera field-of-view* (FOV in short) with a shape of pie-slice [3]. The FOV coverage in 2D space can be defined with four parameters: camera location P, camera orientation vector $\overrightarrow{d}$, viewable angle α, and maximum visible distance R_V (see Figure 1). The location P of camera is the $<$ *latitude*, *longitude* $>$ coordinate read from a positioning device (e.g., GPS). The camera viewing direction vector $\overrightarrow{d}$ is obtained based on the orientation angle (θ), which can be acquired from a digital compass. The camera viewable angle (α) is calculated based on the camera and lens properties for the current zoom level [9]. 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 $FOV\left(P\left\langle lat, lng\right\rangle, \theta, \alpha, R_V\right)$. These P, θ, α, and R_V values, i.e., the geospatial meta-data, can be acquired from the attached sensors during the video capture.

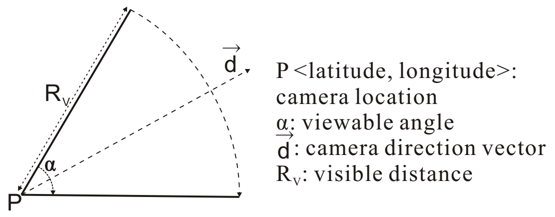

R_V
$\overrightarrow{d}$
α
P
P <latitude, longitude>: camera location
α: viewable angle
$\overrightarrow{d}$: camera direction vector
R_V: visible distance

Figure 1: Illustration of the FOV model in 2D

2.2 Related Work

Associating geo-location and camera orientation information for video retrieval has become an active topic. Researches [22, 18] associating geographic information always helped on video applications. 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 [15] utilized geographic information to segment and index video. Our prior work [3] proposed a viewable scene model to link the video content and sensor information. However, none of the above methods addresses indexing and searching issues, on the large scale.

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* [20]. The R-tree family (including R-tree [10], R$^+$-tree [21], R*-tree [5]) 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 [16], quadtree [8], and Voronoi diagram [19]. Recent researches use either the grid structure [6], the skip quadtree [7] or Voronoi diagram [17] to process multiple types of queries. However, these data structures consider spatial objects as points or small rectangles, and none of them are appropriate to index our FOV model.

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. 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 coverage area of the camera viewable scenes. 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 direction. The proposed three-level index structure is illustrated in Figure. 2. The collections of cells at the first, second, and third levels are denoted by $C_{\ell 1}$, $C_{\ell 2}$, and $C_{\ell 3}$, respectively. 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, and R_V values) are stored in a MySQL database where the 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 Figure. 3 (c), only the index entries for the example video file are listed.

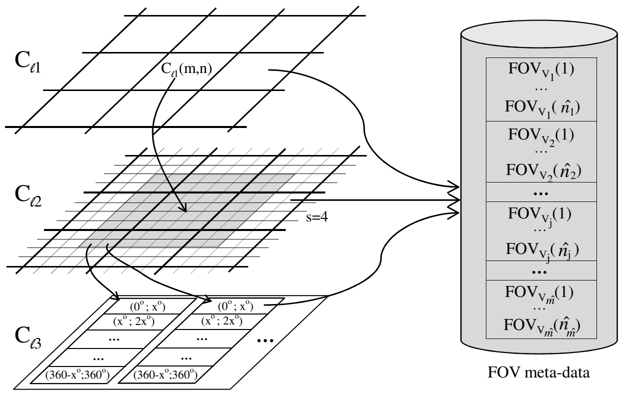


Figure 2: Three-level grid data structure.

The first level organizes the embedding geo-space 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 Figure. 3, the set of FOVs that overlap with the first-level cells are listed in the first table.

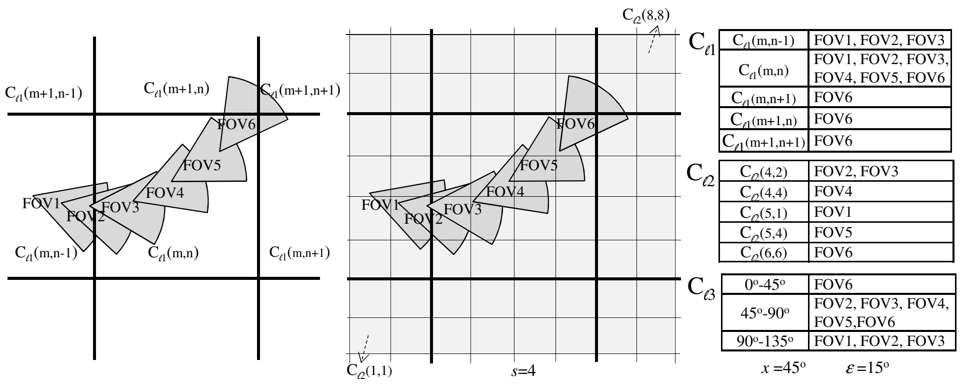
The second-level grid index organizes the overlapping FOVs at each first-level cell based on the distance between the FOV camera locations and the center of the cell. To construct the second-level grid, each $C_{\ell 1}$ cell is further divided into $s \times s$ subcells of size $\left(\frac{\delta}{s} \times \frac{\delta}{s}\right)$, where each subcell is denoted by $C_{\ell 2}(f, g)$ (see Figure. 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 Figure. 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 Figure. 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 Figure. 3, the second table lists the non-empty second-level subcells and the set of FOV IDs 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 k-NVS (k Nearest Video Segments) queries.

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 that its orientation angle margin $(\theta° \pm \varepsilon°)$ overlaps with. ε value can be customized based on the application. In Figure. 3, the third table lists the third-level index entries for the example video for x=45 ° and ε=15 °.

For a video collection with about 2.95 million FOVs, the index size for the three-level index structure is measured as 1.9GB. 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 2.5GB 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 FOV_{v_j} 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 general spatial query, 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.



$C_{\ell 1}$	$C_{\ell 1}$(m,n-1)	FOV1, FOV2, FOV3
	$C_{\ell 1}$(m,n)	FOV1, FOV2, FOV3, FOV4, FOV5, FOV6
	$C_{\ell 1}$(m,n+1)	FOV6
	$C_{\ell 1}$(m+1,n)	FOV6
	$C_{\ell 1}$(m+1,n+1)	FOV6
$C_{\ell 2}$	$C_{\ell 2}$(4,2)	FOV2, FOV3
	$C_{\ell 2}$(4,4)	FOV4
	$C_{\ell 2}$(5,1)	FOV1
	$C_{\ell 2}$(5,4)	FOV5
	$C_{\ell 2}$(6,6)	FOV6
$C_{\ell 3}$	0°-45°	FOV6
	45°-90°	FOV2, FOV3, FOV4, FOV5,FOV6
	90°-135°	FOV1, FOV2, FOV3

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

Figure 3: Index construction example.

In Section 4.3 a motivated example will be presented to show the possible application of these stated query types.

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 $VS_{v_j}(s, e)$=$\{FOV_{v_j}(i),\ s \le i \le e\}$ is a segment of video v_j which includes all the FOVs between $FOV_{v_j}(s)$ and $FOV_{v_j}(e)$.

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.,
PQ-R(q,MIN_R,MAX_R) :

$$q \times \mathbb{FOV} \rightarrow \{VS_{v_j}(s, e),\ \text{where } \forall j\ \forall i\ s \le i \le e,\ \text{such that } FOV_{v_j}(i) \cap q \ne \varnothing \text{ and } MIN_R \le dist(P(FOV_{v_j}(i)), q) \le MAX_R\},$$

where P returns the camera location of an FOV and function *dist* calculates the distance between two points.

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 cameras' 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 $\beta + \varepsilon$.
PQ-D(q,β):

$$q \times \mathbb{FOV} \rightarrow \{VS_{v_j}(s, e),\ \text{where } \forall j\ \forall i\ s \le i \le e,\ \text{such that } FOV_{v_j}(i) \cap q \ne \varnothing \text{ and } \beta - \varepsilon \le D(FOV_{v_j}(i)) \le \beta + \varepsilon\},$$

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-R):
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 β.

Definition 5 *k-Nearest Video Segments Query* (k-NVS):
Given a query point q in geo-space, the k-NVS 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.
k-NVS(q,k):

$$q \times \mathbb{FOV} \rightarrow \{(VS_{v_j}(s_1, e_1), .., VS_{v_j}(s_k, e_k)),\ \text{where } \forall s_t, e_t (t = 1, .., k) \text{ and } \forall j \forall i\ s_t \le i \le e_t,\ \text{such that } FOV_{v_j}(i) \cap q \ne \varnothing \text{ and } dist(VS_{v_j}(s_t, e_t), q) \le dist(VS_{v_j}(s_{t+1}, e_{t+1}), q)\},$$

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. Given 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. We refer to the set of FOVs resulting set from the filter step as the *candidate set*. In the refinement step, an exhaustive method is carried out to check whether an FOV actually satisfies the query conditions.

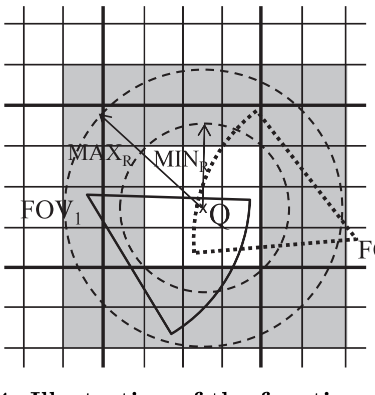


Figure 4: Illustration of the function *applyRadius*

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., 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 : $\beta$
   Output: vector segments $\langle v_j, VS(s_t, e_t)\rangle$
 1 $C_{\ell 1}$ = getCellID($q$);          /* First-level cell */
   /* Point Query with bounded Radius                    */
 2 if $q\_type$ is PQ-R then
 3     $C_{\ell 2}$ = getSubCellID($q$);  /* Second-level cell */
 4     $subCellsInR$ = applyRadius($q$, $C_{\ell 2}$, $MIN_R$,
       $MAX_R$);
 5     $candidateFOVs$ = fetchData($subCellsInR$);
 6     $res$ = refinementStep($candidateFOVs$);
 7 end
   /* Point Query with Direction                         */
 8 if $q\_type$ is PQ-D then
 9     $C_{\ell 3}$ = getDirCellID($C_{\ell 1}$,$\beta$,$\varepsilon$); /* Third-level cell
       */
10     $candidateFOVs$ = fetchData($C_{\ell 3}$);
11     $res$ = refinementStep($candidateFOVs$);
12 end
13 $segments$ = getVideoSeg($res$,$q\_type$);
14 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 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 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. Since this function works on subcell level, it takes all the subcells in the shadow region into account. For example, in Figure 4, according to the minimum (MIN_R) and maximum (MAX_R) distance conditions, only the FOVs locating between the two dot circles will be returned. 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 (β-ε, β+ε). We return the FOVs indexed in the cells $\{C_{\ell 3}(h_1), ..., C_{\ell 3}(h_2)\}$ where β-ε falls into the angle range of $C_{\ell 3}(h_1)$ and β+ε 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 $x = 45^\circ$ intervals in the third-level index. When $\beta = 0^\circ$ (i.e., North) and $\varepsilon = 5^\circ$, we would retrieve the FOVs in the third-level cells $C_{\ell 3}(7)$ and $C_{\ell 3}(0)$ as the candidate FOVs.

Algorithm 2: applyRadius()

```
Input: query point: $q\langle lat, lng\rangle$,
min and max radius: $MIN_R$, $MAX_R$,
first–level cell: $C_{\ell 1}$, second–level cell: $C_{\ell 2}$
Output: set of second-level cells: $checkRadius$
1  $distClose$ = compMinDist($q$, $C_{\ell 2}$);
2  while $distClose \le MAX_R$ do
3      $distFar$ = compMaxDist($q$, $C_{\ell 2}$);
4      if $distFar \ge MIN_R$ then
5          $checkRadius$.add(getCellsAtDist($q$,$distClose$));
6      end
7      $distClose$ += $GRIDSIZE/s$;
8  end
9  return $checkRadius$
```

After the candidate FOVs are retrieved, we run the refinement step (through the function *refinementStep*) to get the actually matching FOVs. 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 (β-ε, β+ε). These FOVs, along with their video file ids (v_j) are stored in the vector *res*.

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 $res$ $\langle v_j, FOV.id\rangle$
1  for all the FOVs in the candidateFOVs do
2      if $q\_type$ is PQ-R then
3          $distP2P$ = dist($q$, $FOV$.P);
4          if $distP2P \ge MIN_R$ AND $distP2P \le MAX_R$
           then
5              if pointInFOV(q, FOV) then
6                  res.push( $\langle FOV.v_j, FOV.id\rangle$ );
7              end
8          end
9      end
10     if $q\_type$ is PQ-D then
11         if $FOV.\theta \ge \beta - \varepsilon$ AND $FOV.\theta \le \beta + \varepsilon$ then
12             if pointInFOV(q, FOV) then
13                 res.push( $\langle FOV.v_j, FOV.id\rangle$ );
14             end
15         end
16     end
17 end
```

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

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.

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: $q_r\langle lat1, lng1; lat2, lng2\rangle$,
(for RQ-R) min and max radius: $MIN_R$, $MAX_R$,
(for RQ-D) viewing direction : $\beta$
Output: vector $segments$ $\langle v_j, VS(s_t, e_t)\rangle$
1  $C_{\ell 1}$ = getCellID($q_r$)
   /* Range Query with bounded Radius */
2  if $q\_type$ is RQ-R then
3      $cellsInR$ = applyRadius($q_r$, $C_{\ell 1}$, $MIN_R$, $MAX_R$);
4      for each cell $C_{\ell 1}(m, n)$ in cellsInR do
5          $overlapArea$ = compOverlap($C_{\ell 1}(m, n)$,$q_r$);
6          if $overlapArea \ge \phi$ then
7              $candidateFOVs$.append(fetchData($C_{\ell 1}(m, n)$));
8          end
9          else
10             $subCellsInR$ =
               applyRadius($overlapArea$,$C_{\ell 2}$,$MIN_R$,$MAX_R$);
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 $C_{\ell 1}(m, n)$ in cellsInR do
17         $overlapArea$ = compOverlap($C_{\ell 1}(m, n)$,$q_r$);
18         if $overlapArea \ge \phi$ then
19             $C_{\ell 3}$ = getDirCellID($C_{\ell 1}(m, n)$,$\beta$,$\varepsilon$);
20         end
21         else
22             $subCells$ =
               applyRadius($overlapArea$,$C_{\ell 2}$,0,$R_V$);
23             $C_{\ell 3}$ = getDirCellID($subCells$,$\beta$,$\varepsilon$);
24         end
25         $candidateFOVs$.append(fetchData($C_{\ell 3}$));
26     end
27 end
28 $res$ = refinementStep($candidateFOVs$);
29 $segments$ = getVideoSeg($res$,$q\_type$);
30 return $segments$
```

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. 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.

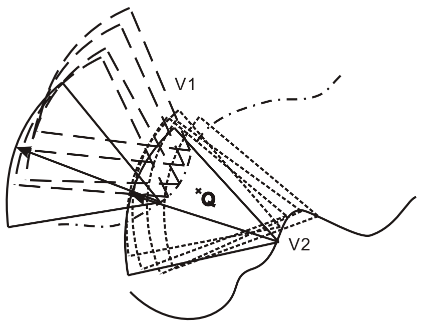


Figure 5: Illustration of k-NVS query

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 cameras 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 nearest k video segments that overlap with q." Taking Figure 5 as an example, the camera locations of the video segment V_1 at time t is 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 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 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 : $\beta$,
Output: vector segments $\langle VS_{v_j}(s_t, e_t)\rangle$
1  $C_{\ell 1}$ = getCellID($q$), $C_{\ell 2}$ = getSubCellID($q$);
2  priority_queue sortedFOVs ⟨ FOVID, distance to $q$ ⟩
   = $\emptyset$;
3  if q_type is k-NVS-R then
4      subCellsInR=applyRadius($q$,$C_{\ell 2}$,$MIN_R$,$MAX_R$);
5  end
6  else
7      subCellsInR = applyRadius($q$, $C_{\ell 2}$, 0, $R_V$);
8  end
9  i=0; distClose=$\delta/s$;
10 while not enough FOVs AND nextSubCells=
   getNeighbors(q,subCellsInR,i++) is not empty do
11     candidateFOVs = fetchData(nextSubCells);
12     for all the FOVs in the candidateFOVs do
13         distP2P = dist($q$, FOV);
14         if q_type is k-NVS then
15             if pointInFOV(q, FOV) then
16                 sortedFOVs.push(⟨FOVID, distP2P⟩);
17             end
18         end
19         if q_type is k-NVS-R then
20             if distP2P $\geq MIN_R$ AND
               distP2P $\leq MAX_R$ AND pointInFOV(q,
               FOV) then
21                 sortedFOVs.push(<
                   FOVID, distP2P >);
22             end
23         end
24         if q_type is k-NVS-D then
25             if FOV.$\theta \geq \beta - \varepsilon$ AND FOV.$\theta \geq \beta + \varepsilon$ then
26                 sortedFOVs.push(<
                   FOVID, distP2P >);
27             end
28         end
29     end
30     while sortedFOVs.top() $\leq$ distClose AND
       numsegments $< k$ do
31         topFOV = sortedFOVs.pop();
32         if isNewSegment(topFOV,res) then
33             numsegments++;
34         end
35         res.push(topFOV);
36     end
37     i++; distClose+ = $\delta/s$;
38 end
39 segments = getVideoSeg(res,q_type);
40 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.

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.

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

k	# of FOVs	# of Segments	Segment Max Length
20	109847	35391	20
50	212746	56664	50
100	291957	72179	71
150	318504	77096	89
200	326110	78523	89
300	327541	78796	89

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, we assume 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.

4.3 Motivated Example

With the development of digital cameras and mobile devices, people always want to search for videos that show what is happening during a special event. In this case, an event reviewing system could be used. People always capture their own videos from a special location and upload to video searching engines (such as Youtube). Hence, users may suffer from reviewing all these videos to know what happened. Therefore, automatically generating video covering both detail and the whole perspective would be helpful for users to know the event. In such kinds of applications, queries with restriction of either the viewing direction or bounded radius can be applied for generating videos. For example, users want to search for what was happening at Marina Bay area during National Day in Singapore this year. In this case, we can firstly search for all the videos capturing that area at that special time. After that, video segments are split and classified according to different viewing directions and distance ranges from the query location. Based on these video segments, a reviewing video can be generated with high semantics.

5. EXPERIMENTAL EVALUATION

5.1 Synthetic Data Generation

Due to the difficulty of collecting large set of real videos, synthetic dataset for moving cameras with positions inside a $75km \times 75km$ region in the geo-space is used to test the performance of the grid-based index structure. We generate camera trajectories using the Georeferenced Synthetic Meta-data Generator [4]. 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 $250m$ [3]. In the experiments, we chose 100 randomly-distributed center points within the area and generate 5500 moving cameras around these center points. Hence each one of the cameras is traced for 1000 seconds, with snapshot of one frame per second, due to the sampling rate of GPS and the compass. 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 which is the contrast to slower movement. To simulate real-world case, we set the maximum speed of moving cameras as $60km/h$, with the average speed as $20km/h$. 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.

Table 2: Parameters of the Synthetic Dataset

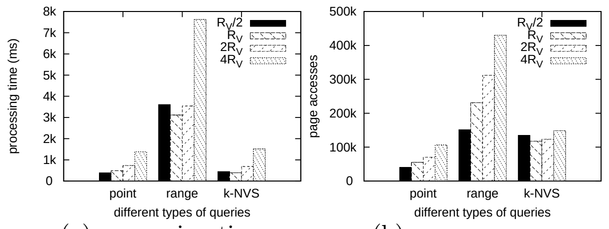
Parameter	Value
# of Center points	100
Speed Limit	60km/h
Average Speed	20km/h
Rotation Limit	30 °/s
# of Cameras	5500
# of Snapshots	1000
# of FOVs	5405051
viewable angle of FOV (α)	60 °
visible distance of FOV (R_V)	250m

5.2 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 10000 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). 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 PA as it is on disk. Additionally, we set the buffer size as one page large. Therefore, when there is no page hit in the buffer, 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.

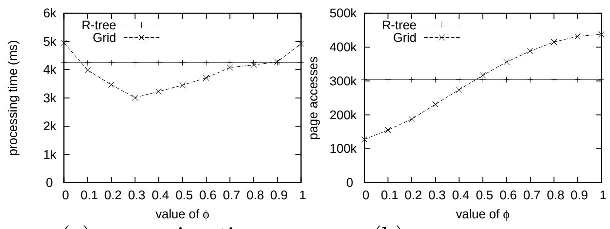
In the next two experiments, we process the typical queries without any distance or direction condition as preliminary experiments to decide the basic parameters: the value of the grid size δ and the overlap threshold ϕ. In the following experiments, if not specifying, the default value of k is 20, and query rectangle size is $250m \times 250m$. When generating the moving objects, the maximum viewable distance(R_V) of the camera is set as $250m$. As shown in Figure 6, grid with size equalling to $R_V/2$ or R_V performs better than greater 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, we set δ equalling to R_V.

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 4.2.1, 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 Figure 7, PA of grid-based index structure is smaller than that of R-tree when ϕ is smaller than 40%. 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.



(a) processing time (b) page accesses

Figure 6: Effect of grid size



(a) processing time (b) page accesses

Figure 7: Effect of ϕ on range query performance

Table 3: Experiment Parameters and Their Values

Parameter	Value
Page Size	4096
Cache Size	4096
Non-Leaf Node Size($2D$ R-tree)	64
Leaf Node Size ($2D$ R-tree)	36
Non-Leaf Node Size ($3D$ R-tree)	80
Leaf Node Size ($3D$ R-tree)	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	$250m$
s	4
Angle Error Margin ε	15 °
Overlap Threshold ϕ	30%

5.3 Comparison

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 [2] implemented by Melinda Green for comparison. To our best knowledge, this implementation achieves the best performance compared to others. We use Equation 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. 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.

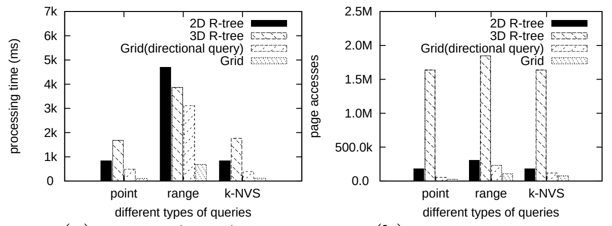
$$\begin{aligned} MBR.left &= min(lng, lng \pm R_V \times \sin(\theta \pm \alpha/2)/\sigma_x) \\ MBR.right &= max(lng, lng \pm R_V \times \sin(\theta \pm \alpha/2)/\sigma_x) \\ MBR.bottom &= min(lat, lat \pm R_V \times \cos(\theta \pm \alpha/2)/\sigma_y) \\ MBR.ceil &= max(lat, lat \pm R_V \times \cos(\theta \pm \alpha/2)/\sigma_y) \end{aligned} \quad (1)$$

5.3.1 *Effect of distance condition*

We study the effect of the distance condition by varying the radius range from $25m$ to $250m$. For each one of the radius range, we start from the minimum distance condition MIN_R equalling to $0m$ until the maximum distance condition MAX_R reaching $250m$. The results shown in Figure 8 are the averages of the different radius ranges. Since the R_V of an FOV is set as $250m$ when generating the synthetic data, the last point with radius range of $250m$ is the result of processing queries without distance condition. Figures 8 (a), (b) and (c) illustrate the processing time of PQ-R query, RQ-R query and k-NVS-R query respectively, while Figures 8 (d), (e) and (f) illustrate PA for each type of query. In general, the performance of our grid-based index structure works better than R-tree on both PT and PA. Figures 8 (a) and (b) show that PT for radius range of $250m$ is a little shorter than that of $225m$. 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 Figures 8 (d), (e) and (f), PA remains the same because R-tree finds out all the FOVs whose MBRs overlap with the query in the filter step, regardless of the radius range. The PA grows as the radius range becomes larger.

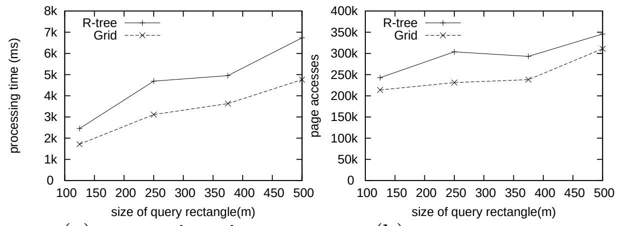
5.3.2 *Effect of direction condition*

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 Figure 9 denote the R-tree for processing queries without direction condition and queries with direction condition, respectively. Figure 9 (a) shows that, in the processing of PQ-D query and k-NVS-D query, PT of R-tree is almost two times of that without direction condition. The reason for this is that searching one more dimension in R-tree slows down the performance of R-tree. The situation is different for RQ-D query because of less number of candidates obtained from the filter step so that the 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 9 (b) shows that PA in 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 PA is to memory pages, the difference in PT which is not that large as PA. Comparing queries with and without direction condition between R-tree and the grid-based approach, our algorithm significantly improves the performance for directional queries.



(a) processing time (b) page accesses

Figure 9: Effect of direction condition



(a) processing time (b) page accesses

Figure 10: Effect of rectangle size on range query performance

5.3.3 *Effect of query rectangle size*

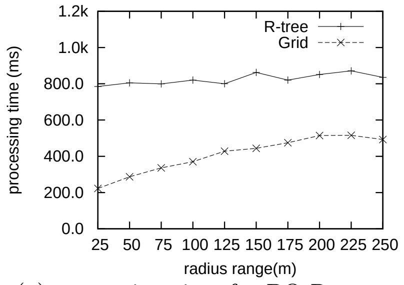
We next study the effect of the query rectangle size to range query. The query rectangle size varies from $125m$ to $500m$, which is from half to two times of the grid size δ. Larger area contains more number of videos and thus leads to longer processing time and more number of accesses. As expected, the result in Figure 10 shows that PT and PA increases with the query rectangle size. From Figure 10(a), PT increases slower using the grid-based approach, which means that our approach performs even faster for large query area than R-tree. However, Figure 10(b) shows that as the query area grows, the difference in number of page accesses between these two methods gets smaller. Since users are always interested in what happened at special places or small regions, our grid-based approach is better than R-tree.

5.3.4 *Effect of* k *value*

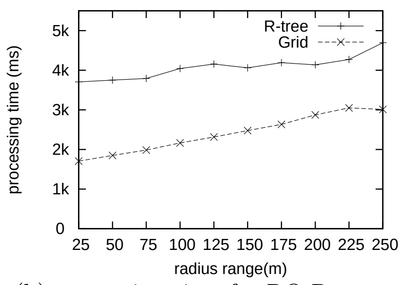
To test the performance of the grid-based approach with different values of k for k-NVS query, we calculate PT and PA using the same query points. The results in this experiment are discrete FOVs (not video segments). Figure 11(a) shows the comparison in PT and Figure 11(b) shows the comparison in PA. As k increases, 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. 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, 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 Figures 11(a) 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 R-tree.

6. CONCLUSIONS

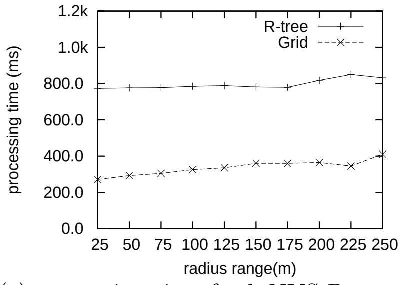
In this study we proposed a novel multi-level grid-based index structure and a number of related query types that



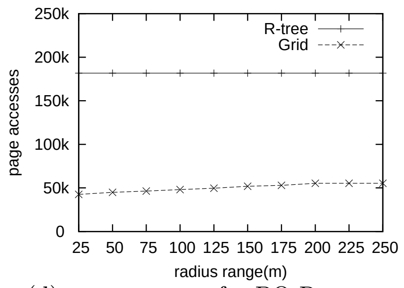
(a) processing time for PQ-R query



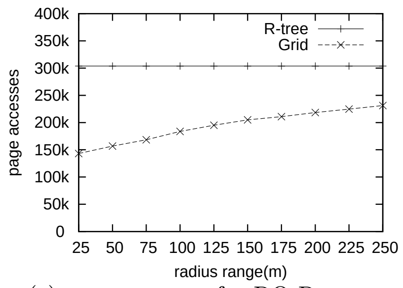
(b) processing time for RQ-R query



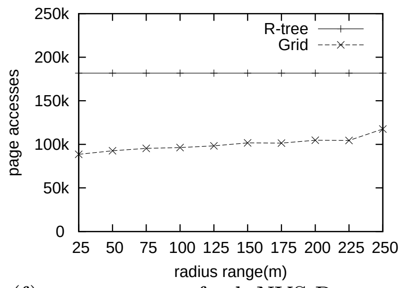
(c) processing time for k-NVS-R query



(d) page accesses for PQ-R query

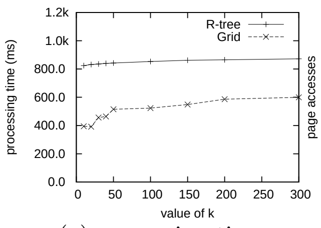


(e) page accesses for RQ-R query

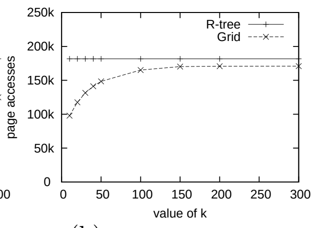


(f) page accesses for k-NVS-R query

Figure 8: Effect of distance condition



(a) processing time



(b) page accesses

Figure 11: Effect of value of k on k-NVS query performance

facilitate application access to such augmented, large-scale video repositories. Our experimental results 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.

7. REFERENCES

[1] `http://www.youtube.com/t/press_statistics`.

[2] `http://superliminal.com/sources/RTreeTemplate.zip`.

[3] S. Arslan Ay, R. Zimmermann, and S. Kim. Viewable Scene Modeling for Geospatial Video Search. In *ACM Int'l Conference on Multimedia*, pages 309–318, 2008.

[4] S. Arslan Ay, R. Zimmermann, and S. H. Kim. Generating Synthetic Meta-data for Georeferenced Video Management. In *ACM SIGSPATIAL Int'l Conference on Advances in Geographic Information Systems (GIS)*, 2010.

[5] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *ACM SIGMOD Int'l Conference on Management of Data*, pages 322–331, 1990.

[6] H. Chon, D. Agrawal, and A. Abbadi. Range and KNN Query Processing for Moving Objects in Grid Model. *Mobile Networks and Applications*, 8(4):401–412, 2003.

[7] D. Eppstein, M. Goodrich, and J. Sun. The Skip Quadtree: A Simple Dynamic Data Structure for Multidimensional Data. In *Annual Symposium on Computational Geometry*, 2005.

[8] R. Finkel and J. Bentley. Quad Trees: A Data Structure for Retrieval on Composite Keys. *Acta informatica*, 4(1):1–9, 1974.

[9] C. Graham. Vision and Visual Perception. 1965.

[10] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *ACM SIGMOD Int'l Conference on Management of Data*, 1984.

[11] T. Hwang, K. Choi, I. Joo, and J. Lee. MPEG-7 Metadata for Video-based GIS Applications. In *IEEE Int'l Geoscience and Remote Sensing Symposium*, 2004.

[12] K. Kim, S. Kim, S. Lee, J. Park, and J. Lee. The Interactive Geographic Video. In *IEEE Int'l Geoscience and Remote Sensing Symposium (IGARSS)*, volume 1, pages 59–61, 2003.

[13] S. Kim, S. Arslan Ay, B. Yu, and R. Zimmermann. Vector Model in Support of Versatile Georeferenced Video Search. In *ACM Int'l Conference on Multimedia*, 2010.

[14] X. Liu, M. Corner, and P. Shenoy. SEVA: Sensor-Enhanced Video Annotation. In *ACM Int'l Conference on Multimedia*, pages 618–627, 2005.

[15] T. Navarrete and J. Blat. VideoGIS: Segmenting and Indexing Video Based on Geographic Information. In *5th AGILE Conference on Geographic Information Science*, pages 1–9, 2002.

[16] J. Nievergelt, H. Hinterberger, and K. Sevcik. The Grid File: An Adaptable, Symmetric Multikey File Structure. *ACM Transactions on Database Systems (TODS)*, 9(1):38–71, 1984.

[17] S. Nutanong, R. Zhang, E. Tanin, and L. Kulik. The V*-Diagram: A Query-dependent Approach to Moving KNN Queries. *Proceedings of the VLDB Endowment*, 1(1):1095–1106, 2008.

[18] N. O'Connor, T. Duffy, C. Gurrin, H. Lee, D. Sadlier, A. Smeaton, and K. Zhang. A Content-Based Retrieval System for UAV-like Video and Associated Metadata. In *Airborne Intelligence, Surveillance, Reconnaissance (ISR) Systems and Applications*. SPIE, 2008.

[19] A. Okabe. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons Inc, 2000.

[20] P. Rigaux, M. Scholl, and A. Voisard. Spatial Databases with Application to GIS. *SIGMOD Record*, 32(4):111, 2003.

[21] N. Roussopoulos, C. Faloutsos, and S. Timos. The R$^+$-tree: A Dynamic Index for Multi-dimensional Objects. In *Int'l Conference on Very Large Databases (VLDB)*, pages 507–518, 1987.

[22] Z. Zhu, E. Riseman, A. Hanson, and H. Schultz. An Efficient Method for Geo-referenced Video Mosaicing for Environmental Monitoring. *Machine Vision and Applications*, 16(4):203–216, 2005.