Knowledge-Based Systems 61 (2014) 1–12

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

GeoTree: Using spatial information for georeferenced video search ☆

Youngwoo Kim, Jinha Kim, Hwanjo Yu *

Department of Computer Science and Engineering, Pohang University of Science and Technology (POSTECH), Pohang, Republic of Korea

ARTICLE INFO

Article history:
Received 13 August 2013
Received in revised form 29 January 2014
Accepted 29 January 2014
Available online 17 February 2014

Keywords:
Georeferencing
Geotagging
Video search
Spatial indexing
R-tree

ABSTRACT

With the rapid popularization of video recording devices, more multimedia content is available to the public. However, current video search engines rely on textual data such as video titles, annotations, and text around the video. Video recording devices such as cameras, smartphones and car blackboxes are nowadays equipped with GPS sensors and the ability to capture videos with spatiotemporal information such as time, location, and camera direction. We call such videos *georeferenced videos*. This paper proposes an efficient spatial indexing method, called GeoTree, which facilitates rapid searching of georeferenced videos. In particular, we propose a new data structure, called MBTR (Minimum Bounding *Tilted* Rectangle) to efficiently store the areas of moving scenes in the tree. We also propose algorithms for building MBTRs from georeferenced videos and algorithms for efficiently processing point and range queries on GeoTree. The results of experiments conducted on real georeferenced video data show that, compared to previous indexing methods for georeferenced video search, GeoTree substantially reduces index size and also improves search speed for georeferenced video data. An online demo of the system is available at "http://dm.postech.ac.kr/geosearch".

1. Introduction

With the rapid popularization of video recording devices, more multimedia content is available to the public. Both media made by professionals and those made by ordinary users called UCC (User-Created Contents) comprise a large portion of the web. However, unlike the search for textual data which is effectively executed by existing search engines, there is yet no effective method of searching for video. While search engines utilize document contents to identify related documents, identifying related video requires an understanding of video contents, which is a very demanding and complex process. Thus, current video search engines rely on textual data such as video titles, annotations, and text around the video.[1,2]

Video capture devices such as cameras, smartphones, and car blackboxes are nowadays equipped with GPS sensors and the ability to capture videos with spatiotemporal information such as time, location, and camera direction. We call such videos *georeferenced videos*. For some applications, such spatiotemporal information plays key roles in the querying of georeferenced videos. For example, some people may want to find videos of a specific event that occurred at a particular time and in a particular location, e.g., videos of a traffic accident captured by car blackboxes, videos of a goal scene in a soccer game captured by users, or videos of a concert with celebrities captured by users. Also, some compound applications, such as path recommendation or tour guide program annotated with videos, utilize spatial information to search for relevant videos. Note that, although many devices can capture the geo-data with videos, YouTube does not tag the geo-data on videos, as it does not utilize it for search. The search system must inherently support tagging and utilizing the geo-data. There are several sites doing that[3,4] including our own search system.[5]

This paper proposes an efficient spatial indexing method, called GeoTree, which enables rapid searching of georeferenced videos. In particular, we propose a new data structure, called MBTR (Minimum Bounding *Tilted* Rectangle) to efficiently store the areas of moving scenes in the tree. While traditional MBR (Minimum Bounding Rectangle) is popularly used to describe an area in spatial indexes such as R-Tree, MBR is not suitable for describing the

☆ This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (No. 2012M3C4A7033344). This work was also supported by IT Consilience Creative Program of MKE and NIPA (C1515-1121-0003).

* Corresponding author. Tel.: +82 1041187006.
E-mail addresses: ywkim10@postech.ac.kr (Y. Kim), goldbar@postech.ac.kr (J. Kim), hwanjoyu@postech.ac.kr (H. Yu).

[1] http://www.youtube.com.
[2] http://video.search.yahoo.com.
[3] http://www.you4wd.com.
[4] http://geovid.org.
[5] GeoSearch: http://dm.postech.ac.kr/geosearch.

http://dx.doi.org/10.1016/j.knosys.2014.01.026
0950-7051/

areas of moving scenes in which location and direction are continuously changing. On the other hand, MBTR is designed to describe an area (or scenes) of moving location and direction. GeoTree specifically adopts MBTR in the leaf nodes of R-Tree to efficiently store meaningful parts of moving scenes, which leads to substantial reductions in index size. MBTR also enables efficient pruning of unpromising parts of videos and thus saves query processing time.

There have been several approaches to the problem of search in georeferenced videos. SA Ay et al. [4] proposed a model to represent a camera viewable scene and basic search mechanisms for it. They expressed the camera viewable scene using four parameters – location, direction, viewable angle, and object distance. They also proposed metrics for evaluating and ranking the relevance of videos along with a search algorithm [2]. Our index adopts their scene model but substantially improves the search efficiency. Related works are further discussed in Section 2.

Our major contributions are summarized as follows.

- We propose a new data structure, MBTR, to efficiently store the areas of moving scenes, and develop algorithms for building MBTRs from georeferenced videos to construct a GeoTree, a kind of R-Tree with MBTRs in the leaf nodes.
- We develop efficient algorithms for processing point and range queries on GeoTree.
- We demonstrate the effectiveness of GeoTree by performing experiments on real georeferenced video data. We compare our search methods with the previous georeferenced video search algorithm [2] and an R-Tree-based algorithm. GeoTree substantially reduces index size and also improves search speed for georeferenced video data. An online demo of the system is available at "http://dm.postech.ac.kr/geosearch".

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 presents our proposed indexing methods for georeferenced video. Section 4 presents experimental results. Section 5 concludes this paper.

2. Related work

This section discusses related work on georeferenced multimedia search (Section 2.1) and then reviews methods for spatial indexing (Section 2.2).

2.1. Georeferenced multimedia search

There are several approaches to the use of location information in image search. Toyama et al. [21] used metadata in image search and developed a database for it. They enabled spatial image search by recording images with longitudinal and latitudinal coordinates and timestamps. Several commercial websites, such as Flickr[6] and Woophy,[7] also provide georeferenced image search. They focus on showing static images of the locations in a map and support simple query such as nearest neighbor query. Google Street View [1] provides streams of street view images of their own. We can reference these images by camera's location and viewing direction.

Trajectory indexing methods were developed in order to efficiently store and search moving objects [15,8]. Our indexing method for supporting moving *scenes* can be considered an extension of trajectory indexing which takes into consideration camera directions as well as the location of the cameras.

Research associated with "trajectory-based video indexing" [7,17] and "spatial indexing for video" [20] have been published in the computer vision community. However, the problem they address is different from ours; their goal is to index the relative location of objects in the video, not the viewable scenes. They also use video content retrieval methods such as object segmentation and motion tracking to detect objects, while we use metadata to search georeferenced videos.

A viewable scene model for enabling georeferenced video search has been recently proposed by Ay et al. [2,4]. Based on the viewable scene model, they developed methods for supporting point and range queries using MBR-based filtering. (We compare this method as a baseline method in our experiments in Section 4.) Their scene model is explained in Section 3.1. Our method builds an index GeoTree based on their viewable scene model.

They also developed a *relevance* video search method based on the viewable scene model, and proposed metrics to measure the relevance of video to the given range query [2]. The metrics compute the relevance scores based on the size of the overlapping area. Since computing the exact size of the overlapping area is computationally expensive, they approximate it using grids and histograms. However, their approach requires vast storage to keep all the videos stored in grids, and also the accuracy is compromised due to the running of the query on the gridded data instead of the original data. Moreover, the overlapping area-based relevance may not reflect the true relevance implied in the user's query; user's relevance may be more related to parts that are overlapped and the distance to the overlapping area. Inducing a good relevance function is itself a nontrivial research problem.

Ay et al. also proposed a method for generating synthetic metadata for georeferenced video search [3]. Since it is often difficult to obtain a large set of georeferenced video data, this method can be used for evaluating new methods for georeferenced video search.

2.2. Spatial indexing

There are numerous works on spatial indexing and query processing in the database community. Many of the indexing structures are based on R-Tree [9,12,19]. Several works focus on querying location trajectories of moving objects. A typical example of an object trajectory query is "for a given set of trajectories of moving objects, find trajectories that intersect with the given query range." The common approach to this problem is to divide the trajectory into segments and insert them into an index structure such as an R-Tree or other R-Tree family [13,18]. To utilize the continuity of object trajectory and enable temporal query, several effective indexing structures such as STR-Tree and TB-tree were proposed [14]. These works mostly focus on dealing with temporal dimension and complex queries, rather than dealing with complex spatial data such as viewable scenes.

Spatiotemporal trajectory compressions for saving data storage and enabling fast query processing are also being researched [11,8]. They preprocess trajectory data in order to efficiently store and search trajectories. However, their methods only return approximate results. Their methods also focus on the data of points or location trajectories that are not directly applicable to viewable scenes. There are also works that focus on indexing range data or region data that consider static rectangular objects [6,10].

No effective indexing structure has yet been proposed that supports exact queries on camera viewable scenes of moving locations and directions.

3. GeoTree: Spatial indexing for georeferenced video search

This section presents our proposed indexing method, GeoTree, which enables an efficient search of georeferenced videos. We first discuss some preliminaries (Section 3.1) and give an overview of

[6] http://www.flickr.com.
[7] http://www.woophy.com.

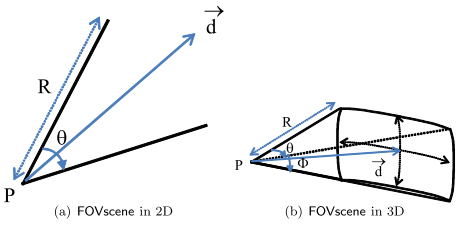


Fig. 1. Examples of FOVscene in 2D and 3D spaces.

our indexing method (Section 3.3). We then present algorithms for building the index (Section 3.4) and processing point and range queries on the index (Section 3.5).

3.1. Preliminary: Viewable scene model

The viewable scene changes, as the camera moves or rotates. The area captured by the camera is referred to as the Field-Of-View scene (FOVscene) [4]. As illustrated in Fig. 1(a), in a 2D space, four parameters are required to express a single FOVscene: (1) location of the camera P, (2) direction of the camera $\vec{d}$, (3) viewable angle of the camera θ, and (4) visible distance R.

$$\text{FOVscene}(P, \vec{d}, \theta, R) \tag{1}$$

Camera location P is represented as a (latitude, longitude) pair that can be obtained using a GPS sensor. Camera direction $\vec{d}$ can be obtained using a compass sensor. Camera viewable angle θ and visible distance R can be obtained from the camera lens' property and zoom level. As assumed in [4,2], we also fixed θ and R by assuming the use of static camera lens. (Some video recording devices such as smartphones and car blackboxes use static camera lenses.)

The camera view or the viewable scene information at a particular moment is entirely expressed by an FOVscene. An FOVscene corresponds to a frame in the video. A georeferenced video can be represented by a sequence of FOVscenes. We use the term FOVstream to denote a sequence of FOVscenes. As done in [4,2], we restrict FOVscene to a 2D space, however it can be extended to a 3D space in a straightforward manner. Altitude and pitch should be considered in the 3D model (Fig. 1(b)). but they do not alter the nature of the FOVscene model.

3.2. Limitation of MBR

The MBR (Minimum Bounding Rectangle) structure, which has been traditionally used for spatial indexing in R-tree, is mainly used for indexing point objects. However, since MBR indexes areas based on rectangles, it is inefficient to index the areas of moving scenes which are shaped like trajectories. Fig. 2 illustrates such a problem of MBR in indexing moving scenes. Fig. 2(a) used two MBRs to cover the area of moving scenes: the query point (a red circle) is evaluated as positive (false positive) since it is inside the MBRs although it is not really covered by the moving scenes. To reduce the false positive, Fig. 2(b) used 13 MBRs, one for each scene, to cover the area of moving scenes. However, in this case, we have to evaluate many MBRs to identify the final scenes, which will require more memory space to maintain the structure and takes more time to reach the leaf nodes of the R-tree. The MBTR structure is more appropriate for indexing moving scenes, as its shape is more flexible and not having to be parallel to axis. As Fig. 2(c) shows, two MBTR sufficiently cover the area of moving scene without generating false positive on the query point (a red circle).

3.3. GeoTree: Overview

To enable an efficient indexing of georeferenced video search, videos must be recorded with the metadata of $(P, \vec{d}, \theta, R)$ for each FOVscene. Video recording devices such as car blackboxes and smartphones are nowadays equipped with sensors that have the ability to record metadata with videos. An index, GeoTree, is constructed on the videos with the metadata. When a point or range query is submitted by users, it is processed on the GeoTree by finding the FOVstreams that overlap the query area. The processes of index construction and query processing are summarized below. We detail technical challenges and propose algorithms for each process in Sections 3.4 and 3.5 respectively.

- **Index construction:** GeoTree uses Minimum Bounding *Tilted* Rectangle (MBTR) instead of MBR (Minimum Bounding Rectangle) to represent an area of moving scenes in the index. Building MBTRs requires modeling moving scenes piece-wise linearly such that each MBTR covers as large an area as possible while producing as small a number of false positive areas as possible. Thus, our index construction in Section 3.4 discusses (1) algorithms for identifying *markup* FOVscenes that disconnect moving scenes into linear pieces of FOVstreams and (2) algorithms to construct an MBTR from each FOVstream. GeoTree is a kind of R-Tree that uses the MBTR s in the leaf nodes.
- **Query processing:** A point or range query is processed in a similar fashion to R-Tree in the nonleaf nodes, as the nonleaf nodes of GeoTree are also described by MBRs. Once the query reaches a leaf node of GeoTree, the query area is checked to see if it overlaps the MBTR; if it does, the corresponding FOVscenes are retrieved from the MBTR. Thus, our query processing method in Section 3.5 discusses checking and retrieving algorithms on the MBTR.

3.4. GeoTree: Index construction

To build a GeoTree, moving scenes must be represented as linear pieces of an FOVstream, and an MBTR is built for each linear piece of FOVstream. Note that, unlike indexing trajectories, an MBTR must take camera direction $\vec{d}$ as well as location P into consideration

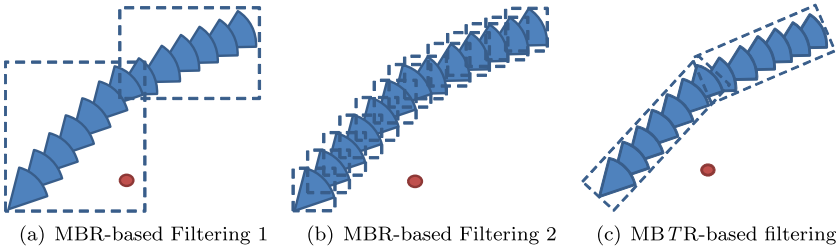


Fig. 2. Examples of MBR (Minimum Bounding Rectangle) and MBTR (Minimum Bounding *Tilted* Rectangle). Red circle is the querying area. Blue circular sector is a viewable scene. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

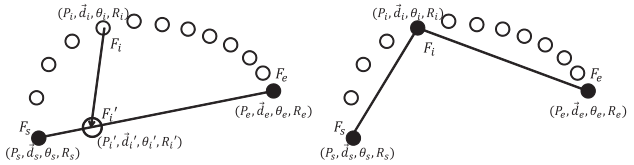


Fig. 3. Finding Markup FOVscenes. The circles are FOVscenes, and F_s and F_e are the starting and ending FOVscenes, respectively. In (a), F_i' is the estimated F_i according to the regression line. In (b), F_i becomes a new markup FOVscene.

in order to minimally cover the area of the moving scenes. Therefore, we first identify *markup* FOVscenes from moving scenes, while considering the changing patterns of P and $\vec{d}$. Markup FOVscenes are those at the edge of an MBR, such that the FOVscenes between markup FOVscenes are moving linearly in terms of P and constant in terms of $\vec{d}$, and thus they can be estimated from the markup FOVscenes. Since P is moving linearly and $\vec{d}$ is constant within an MBR, we can check if a query area is located within the MBR by simply retrieving the markup FOVscenes. (Algorithms for checking and retrieving FOVscene within an MBR are discussed in Section 3.5.)

3.4.1. Identifying markup FOVscenes

The key idea underlying the finding of markup FOVscenes is to find the point at which scenes (i.e., P and $\vec{d}$) are sharply changing. We extend the idea of the trajectory compression algorithm [11] to moving scenes. Given an FOVstream, we assume that the scenes are moving perfectly linearly from the start to the end in terms of P. In such a case, when we draw a linear regression between the start and end points, the regression line will perfectly fit to the FOVstream. If not, the actual scene in the middle will depart from the regression line. (Fig. 3(a) shows an example.)

Algorithm 1. FindingMarkupFOVScene($F,\ \epsilon_P,\ \epsilon_{\vec{d}}$)

```
1 F: an FOVstream
2 ε_P, ε_d: threshold for errors of P and d
3 S1 = MarkupFOVScene_P(F, s, e, ε_P)
4 S2 = MarkupFOVScene_d(F, s, e, ε_d)
5 return S1 ∪ S2
```

Algorithm 2. MarkupFOVScene_P($F,\ s,\ e,\ \epsilon_P$)

```
1 F: an FOVstream
2 s: starting index of F
3 e: ending index of F
4 ε_P: threshold
5 for i ← s to e do
6   P'_i = Regression (F_s, F_e, F_i)
7   dist = distance(P'_i, P_i)
8   if dist >dist_max then
9     dist_max ← dist
10    peak ← i
11  end
12 end
13 if dist_max > ε_P then
14  list1 = MarkupFOVScene_P(F, s, peak, ε_P)
15  list2 = MarkupFOVScene_P(F, peak, e, ε_P)
16  return list1 + list2
17 else
18  return [F_s, F_e]
19 end
```

Algorithm 3. Regression($F_s,\ F_e,\ F_i$)

```
1 F: an FOVstream
2 t_s and P_s: timestamp and location of F_s
3 t_e and P_e: timestamp and location of F_e
4 t_i: timestamp of F_i
5 Δe = t_e − t_s
6 Δi = t_i − t_s
7 P'_i = P_s + (Δi/Δe)(P_e − P_s)
8 return P'_i
```

Algorithm 4. MarkupFOVScene_d($F,\ s,\ e,\ \epsilon_{\vec{d}}$)

```
1 F: an FOVstream
2 s: starting index of F
3 e: ending index of F
4 ε_d: threshold
5 for i ← s to e do
6   // Get the mean d of two extremes in the interval
7   d'_i = MiddleValue(F, s, i)
8   dist = distance(d'_i, d_i)
9   if dist > ε_d then
10    list1 = [F_s, F_i]
11    list2 = MarkupFOVScene_d(F, i, e, ε_d)
12    return list1 + list2
13  end
14 end
15 return [F_s, F_e]
```

From this idea, we draw a regression line from the start to the end point of FOVstream. For each scene F_i between F_s (starting scene) and F_e (ending scene), we find the corresponding scene F_i' on the regression line as illustrated in Fig. 3(a). (F_i' has the same timestamp as F_i.) We then compute the distance between F_i and F_i'. If the distance is greater than a threshold ϵ, we divide the FOVstream based on F_i such that F_i will be the ending scene of the first division and also the starting scene of the second division, as illustrated in Fig. 3(b). F_i then becomes the markup FOVscene. We repeat this process recursively until the maximum distance between actual scenes and their linearly approximated scenes is lower than ϵ.

Given a real scene F_i, a linearly approximated scene $F\prime_i$ is found using the timestamps of scenes as follows.

$$\Delta e = t_e - t_s \Delta i = t_i - t_s P_i' = P_s + \frac{\Delta i}{\Delta e}(P_e - P_s) \tag{2}$$

where t_s and t_e are the timestamps of the starting and ending scenes, respectively, t_i is the timestamp of FOVscene F_i that we want to estimate, and P_i' is the estimated location of F_i.

Next, we find markup FOVscenes in terms of $\vec{d}$. Since $\vec{d}$ is assumed to be constant within an MBR, we check to see if $\vec{d}$ has moved more than a threshold $\epsilon_{\vec{d}}$ from the starting FOVscene. If it has, we create a markup FOVscene. Once markup FOVscenes are collected for $\vec{d}$, the markup FOVscenes are unioned with those collected for P.

Algorithm 1 describes the procedure for detecting markup FOVscenes. It calls the functions MarkupFOVScene_P (Algorithm 2) and MarkupFOVScene_d (Algorithm 4) and returns the set union of their results. The user-defined thresholds for P and $\vec{d}$ are ϵ_P and $\epsilon_{\vec{d}}$ respectively. Algorithm 2 finds markup FOVscenes for P using a

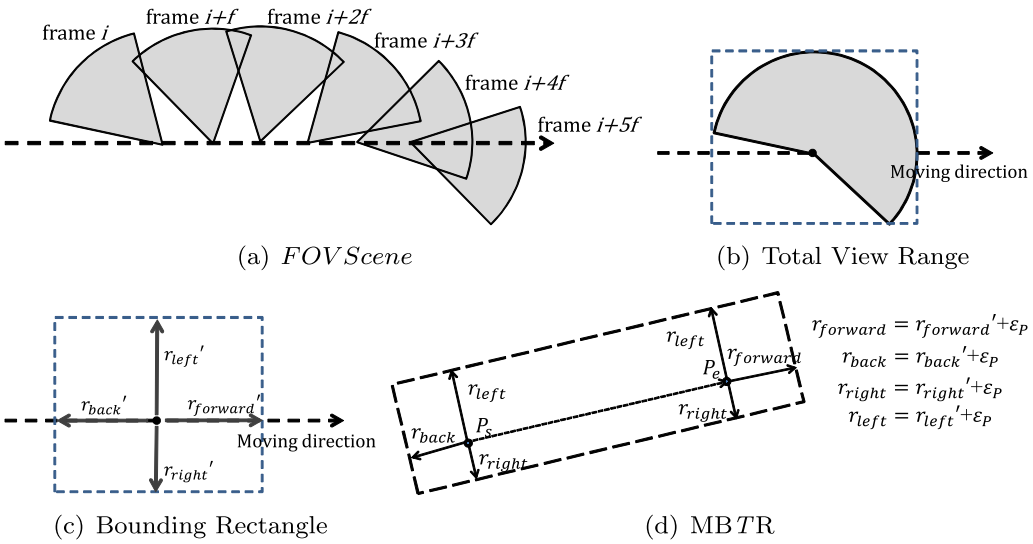


Fig. 4. MBTR construction.

linear regression; line 6 finds the location P_i' of scene $F_{i'}$, corresponding to F_i in terms of the timestamp (Algorithm 3). If its distance from P_i is greater than ϵ_P (lines 7 and 13), the FOVstream is divided based on P_i and calls itself with divided FOVstreams (lines 14 and 15).

Algorithm 4 finds markup FOVscenes for $\vec{d}$; it first computes the middle $\vec{d}$ value of the interval, which is the mean of the maximum and minimum $\vec{d}$ in the interval between the starting FOVscene and the current FOVscene (line 6). If the distance from the current $\vec{d}$ and the middle $\vec{d}$ is greater than the threshold (lines 7 and 8), FOVstream is divided by the current FOVscene, and the current FOVscene becomes a new markup FOVscene (line 9) and also the starting FOVscene of the remaining FOVstream (line 10).

3.4.2. Constructing MBTR from FOVstream

Once a set of markup FOVscenes are found, we construct MBTRs from pairs of adjacent markup FOVscenes. An MBTR must cover all FOVscenes between a pair of adjacent markup FOVscenes. We draw a tilted rectangular boundary for the FOVscenes, and expand this boundary by ϵ_P because ϵ_P was the error threshold for P such that the locations of FOVscenes within an MBTR can deviate from the regression line by as large a value as ϵ_P.

Fig. 4 illustrates the MBTR construction process. Each FOVscene has the shape of a circular sector. If we shift all the FOVscenes in the interval to the same location, they accumulate to form a larger circular sector. Fig. 4(b) gives an example of an accumulated circular sector. Note that the angle of the larger circular sector will not be larger than a predetermined maximum, $\theta + 2 \times \epsilon_{\vec{d}}$, because $\epsilon_{\vec{d}}$ was the error threshold for $\vec{d}$ within an MBTR. We then construct a bounding rectangle for the circular sector and expand the bounding rectangle by ϵ_P in each direction. The expanded bounding rectangle is called the E-MBR (Expected-MBR). Unlike MBR, which is parallel to the coordinate axis, E-MBR must be parallel to the moving direction. While traditional MBR requires height and width as well as location, E-MBR requires four widths and location. These four widths correspond to the four directions from the location of FOVscene. They are denoted as $r_{forward}$, $r_{backward}$, r_{left}, r_{right} respectively.

A single E-MBR covers a single FOVscene. However, all the E-MBRs within an MBTR have the same width and their location changes linearly. As a result, we only need to store the parameters of one E-MBR for one MBTR. As shown in Fig. 4(d), MBTR is simply a long tilted rectangle represented by a trajectory segment and four widths. MBTR parameters are summarized in Table 1. P_s and P_e are the locations of the first and last FOVscenes of an MBTR, respectively.

3.5. GeoTree: Query processing

This section explains query processing algorithms for point and range queries on GeoTree. A query is processed in a similar fashion to R-Tree in nonleaf nodes, as nonleaf nodes of GeoTree are also built using MBRs just as in R-Tree. Once a query reaches a leaf node in the GeoTree, it is processed in two steps – (1) MBTR filtering and (2) MBTR lookup. MBTR filtering ascertains if the query can overlap the MBTR. If MBTR filtering returns *false*, this means that there is no FOVscene in the MBTR that overlaps the query. However, if MBTR filtering returns *true*, MBTR lookup is called to compute and return an expected subsequence of FOVscenes that overlap the query. We will discuss the processing algorithms for point queries in Section 3.5.1, then discuss those for range queries in Section 3.5.3.

3.5.1. Point query processing

Algorithm 5. PointQueryInMBTR (q,MBTR)

1 P_s: Starting point of location trajectory
2 P_e: Ending point of location trajectory
3 q: Query Point
4 n: Number of FOVs in MBTR.
5 L: List of matching FOVs.
6 l: Distance between P_s, P_e
7 $r_{left}, r_{right}, r_{forward}, r_{back}$: MBTR parameters
8 B: Boundary for MBTR
9 {B is tilted rectangle given four r values}
10 **if** *pointPolygonIntersect(q,B)* **then**
11 D = projectedDistance(P_s, P_e, q)
12 $i = \lceil (D - r_{forward}) \times \frac{n-1}{l} \rceil$
13 $j = \lfloor (D + r_{back}) \times \frac{n-1}{l} \rfloor$
14 **for** $k \leftarrow i$ to j **do**
15 **if** *pointFOVIntersect*(q, F_k) **then**
16 addList(L, F_k)
17 **end**
18 **end**
19 **end**
20 return ***L***

Table 1
MBTR parameters.

P_s	Starting point of location trajectory
P_e	Ending point of location trajectory
r_{left}	Maximum distance to left sides of moving direction
r_{right}	Maximum distance to right sides of moving direction
$r_{forward}$	Maximum distance to forward of moving direction
r_{back}	Maximum distance to backward of moving direction

Table 2
Subroutines.

pointPolygonIntersect(q, P)	Returns *true* if point q overlaps with polygon P
pointFOVIntersect(q, F)	Returns *true* if point q overlaps with FOVScene F
polygonFOVIntersect(P, F)	Returns *true* if polygon P overlaps with FOVScene F
getIntersections(P_1, P_2)	Returns all intersection points between polygon P_1 and P_2
addList(L, F_k)	Add frame id of F into L
projectedDistance(P_s, P_e, q)	Distance of q' from P_s, where q' is projection of q onto $\overrightarrow{P_sP_e}$. It is given by $\frac{\overrightarrow{P_sq} \cdot \overrightarrow{P_sP_e}}{\lvert\overrightarrow{P_sP_e}\rvert}$

When a query arrives at a leaf node of GeoTree that contains an MBTR, MBTR filtering is called. MBTR filtering for a point query is straightforward: The query point is checked to determine if it is located inside the boundary of MBTR, which are described by the parameters in Table 1, using the algorithm for checking the overlap between a point and a polygon. Line 10 of Algorithm 5 corresponds to MBTR filtering.

Once MBTR filtering returns *true*, MBTR lookup returns a subsequence of FOVscenes that is expected to overlap with the query. Since an MBTR typically contains numerous FOVscenes, it would be inefficient to scan all the FOVscenes in the MBTR to find the overlapping FOVscenes. We directly compute, using the MBTR structure, a sequence of FOVscenes that is expected to contain the query point. Fig. 5(a) depicts an example: A query point, represented by a triangle, is inside the MBTR, but neither the first FOVscene nor the last FOVscene contains the query point. Fig. 5(b) shows the boundary of promising FOVscenes that could contain the query point.

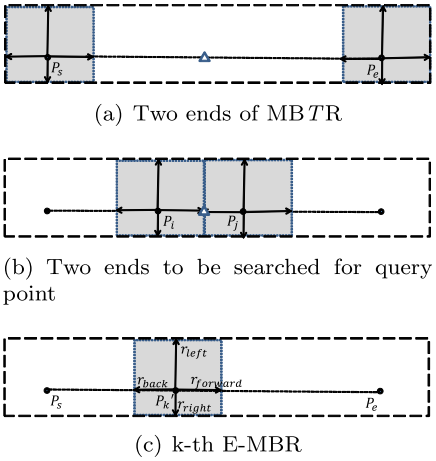
Since MBTR is a sequence of E-MBRs, and each E-MBR fully covers an individual FOVscene, we calculate the first and the last E-MBRs that overlap with the query point. The first index i and the last index j of the overlapping E-MBRs are computed using Eqs. (3) and (4).

$$i = \left\lceil (D - r_{forward}) \times \frac{n-1}{l} \right\rceil \tag{3}$$

$$j = \left\lfloor (D + r_{back}) \times \frac{n-1}{l} \right\rfloor \tag{4}$$

where k is the frame number of FOVscene counted from P_s, l is the length of $\overline{P_sP_e}$, and n is the number of FOVscenes in the MBTR. D is the distance from P_s to the point of query q projected onto the line $\overline{P_sP_e}$. D can be negative if q is projected onto the line outside $\overline{P_sP_e}$. Once i and j are computed, we only need to scan the FOVscenes from F_i to F_j to retrieve the FOVscenes containing the query point within the MBTR.

We will now justify Eqs. (3) and (4). Fig. 5(c) shows an example of E-MBR. There exists one E-MBR for each FOVscene, and each E-MBR bounds its corresponding FOVscene's range. Let P_k be the exact location of the kth FOVscene F_k and P'_k be the location of the corresponding E-MBR. Note that P'_k is located on $\overline{P_sP_e}$ and



(a) Two ends of MBTR

(b) Two ends to be searched for query point

(c) k-th E-MBR

Fig. 5. MBTR location index.

moves from P_s to P_e at a constant speed. The kth E-MBR, which bounds F_k, is given by a rectangle located at P'_k. This E-MBR has widths $r_{left}, r_{forward}, r_{right}, r_{back}$ from P'_k. If a query point turns out to be located inside the MBTR according to MBTR filtering, the query point exists within the width of r_{left} or r_{right} from $\overline{P_sP_e}$. Thus we only need to consider $r_{forward}$ and r_{back}. For a query point to be located inside the kth E-MBR, it has to be located between the backward boundary and the forward boundary of E-MBR. The distance from P_s to P'_k is $k \times \frac{l}{n-1}$. The backward boundary of the kth E-MBR is r_{back} from P'_k and the forward boundary is $r_{forward}$. Thus, a query point is located inside the kth E-MBR only if it satisfies Eq. (5).

$$k \times \frac{l}{n-1} - r_{back} \leqslant D \leqslant k \times \frac{l}{n-1} - r_{forward} \tag{5}$$

The first index i and the last index j of overlapping E-MBRs in Eqs. (3) and (4) are the lowest and highest index k, respectively, that satisfy Eq. (5).

3.5.2. Summary of query processing

In this section, we illustrate a simple running example of a point query processing in GeoTree A point query is processed in the following five steps summarized in Table 3.

1. The query point is tested whether it exists inside or outside of our data space.
2. The point is processed through the nonleaf nodes of R-Tree. After this, the candidate space is limited to an MBR which may contain several MBTRs Fig. 6(a).
3. The point is tested by the MBTR -Filtering, which corresponds to line 10 of Algorithm 5. After this, the corresponding MBTR is found 6(b).
4. The MBTR -Lookup is processed, which corresponds to lines 11–13 of Algorithm 5. Now the candidate space is shortened to a small slanted box 6(c).

Table 3
Change of search space during query processing.

Step	Candidate area (m)	
	Before test	After test
(1) Validity testing	$\infty * \infty$	10,000 × 10,000
(2) R-Tree pruning	10,000 × 10,000	100 × 100
(3) MBTR-filtering	100 × 100	20 × 120
(4) MBTR-lookup	20 × 120	20 × 40
(5) FOV to point comparison	20 × 40	Result

5. The FOVscenes in the slanted box are evaluated to return the query result, i.e., the final scenes that contain the query point. (line 15 of Algorithm 5).

3.5.3. Range query processing

Algorithm 6. RangeQueryInMBTR (q,MBTR)

```
1  P_s: Starting point of location trajectory
2  P_e: Ending point of location trajectory
3  Q: Query Range {Q is given by convex polygon}
4  n: Number of FOVs in MBTR.
5  L: List of matching FOVs.
6  l: Distance between P_s, P_e
7  B: Boundary for MBTR
8  {B is tilted rectangle given four r values}
9  C: Set of critical points to check
10 /* Collecting critical point */
11 C = getIntersections(B, Q)
12 for ∀v_Q ∈ Q do
13   if pointPolygonIntersect(v_Q, B) then
14     C = C ∪ v_Q
15   end
16 end
17 for ∀v_B ∈ B do
18   if pointPolygonIntersect(v_B, Q) then
19     C = C ∪ v_B
20   end
21 end
22 /* MBTR lookup */
23 if C ≠ φ then
24   D_min = min_{v∈C} projectedDistance(P_s, P_e, v)
25   D_max = max_{v∈C} projectedDistance(P_s, P_e, v)
26   i = ⌈(D_min − r_forward) × (n−1)/l⌉
27   j = ⌊(D_max + r_back) × (n−1)/l⌋
28   for k ← i to j do
29     if polygonFOVIntersect(Q, F_k) then
30       addList(L, F_k)
31     end
32   end
33 end
34 return L
```

We use a convex polygon to represent a range query, as other types of range queries can be represented as the sum of convex polygon queries. Like point query processing, range query is processed in the same way as R-Tree in the nonleaf nodes. Once the query reaches a leaf node, we perform MBTR filtering and MBTR lookup. Query processing inside the MBTR is summarized in Algorithm 6. Subroutines used in the algorithm are summarized in Table 2.

For MBTR filtering and MBTR lookup for range queries, we use *CriticalPoints*, which is a set of vertices that bounds the overlapping region between MBTR boundary and query polygon boundary Q. *CriticalPoints* are composed of three categories of points – (1) the set of points that the MBTR boundary and polygon Q's edges intersect, (2) polygon Q's vertices that are located inside the MBTR, and (3) MBTR vertices that are located inside polygon Q.

MBTR filtering corresponds to checking for the existence of *CriticalPoints*. If no *CriticalPoints* exists, it implies that there is no matching FOVscene in MBTR. If *CriticalPoints* do exist, they represent the boundary of the query range that we need to search inside the MBTR. The process of acquiring critical points corresponds to lines 11 through 21 of Algorithm 6.

MBTR lookup for range queries is similar to that for point queries except that the first index i and the last index j of the overlapping E-MBRs must be computed for *CriticalPoints*. Among *CriticalPoints*, we first find two end points, D_{min} and D_{max}, where D_{min} is the closest and D_{max} is the farthest point from P_s on the line $\overline{P_sP_e}$. Indices i and j are computed using equations that are similar to Eqs. (3) and (4) but D is replaced with D_{min} and D_{max} accordingly as follows.

$$i = \left\lceil (D_{min} - r_{forward}) \times \frac{n-1}{l} \right\rceil \tag{6}$$

$$j = \left\lfloor (D_{max} + r_{back}) \times \frac{n-1}{l} \right\rfloor \tag{7}$$

Accordingly, the process of MBTR lookup for range queries is described in lines 24 through 27 of Algorithm 6. After the interval of overlapping E-MBRs is determined, an exact FOVscene comparison is performed on this sequence of FOVscenes.

3.6. Noise filtering

In practice of video and meta-data recording, due to the pragmatic errors of magnetic sensors, the direction data often contains noise. Such noise could obstruct a smooth construction of MBTR in GeoTree. This section discusses the type of direction noise, its impacts in GeoTree construction, and how to remove such noise. The type of noise is following.

- **Latency noise:** When the camera drastically changes its direction, the sensor responds with latency producing inaccurate values.
- **Vibration noise:** Magnetic sensors sometimes produce small vibrations around true values. (Fig. 7(a) shows an example.)
- **Tremble noise:** Sensors sometimes produce sudden short trembles of the directions which are very different from real directions. For example, when the camera is fixed on a tripod and someone taps the tripod with his finger, magnetic sensors sometimes produce a big direction change (over 90 degree) even when the actual direction slightly changes (less than 10 degree). (Fig. 7(b) shows an example.)

Many effective noise filtering techniques were proposed [16]. We revised the *α-trimmed mean filter* method [5] to effectively remove both vibration noise and tremble noise. The original α-trimmed mean filter shows the characteristics of mean filter and median filter. Mean filter is effective for vibration noise and median filter is effective for tremble noise.

The original α-trimmed mean filter is used for data of scalar values and removes α portion of data from both high and low ends. In our application, the data is direction value represented by angles where $0^\circ = 360^\circ$, and there is no concept of high or low in direction. Thus we convert them to unit vectors in cartesian coordinate and remove $\alpha * 2$ portion of data which are farthest from normalized mean value. Vector v for direction $\vec{d}$ is given as follows.

$$v = (cos\theta, sin\theta) \tag{8}$$

where θ is angle between $\vec{d}$ and east direction.

Using the vector arithmetics, we compute the normalized mean vector.

$$C = \sum^{N} cos\theta$$

$$S = \sum^{N} sin\theta$$

$$\text{mean\_vector} = \left(\frac{C}{C^2 + S^2}, \frac{S}{C^2 + S^2} \right) \tag{9}$$

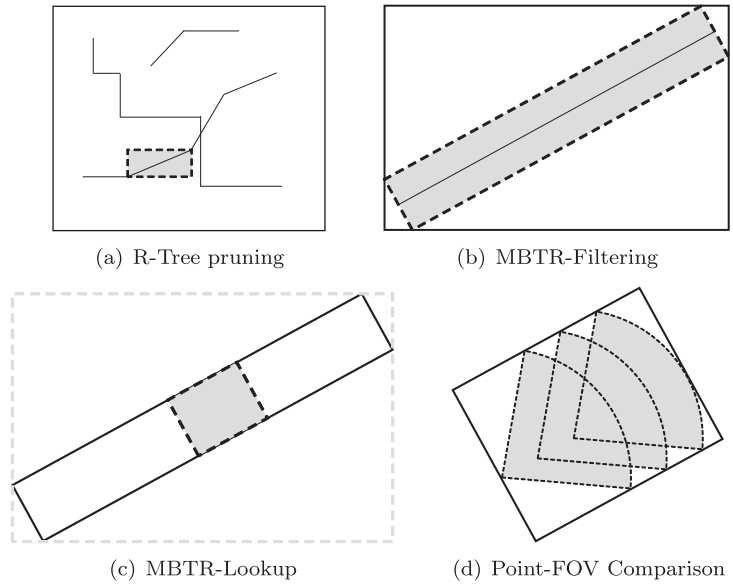


Fig. 6. Point query processing. Gray area depicts the pruned area after each step.

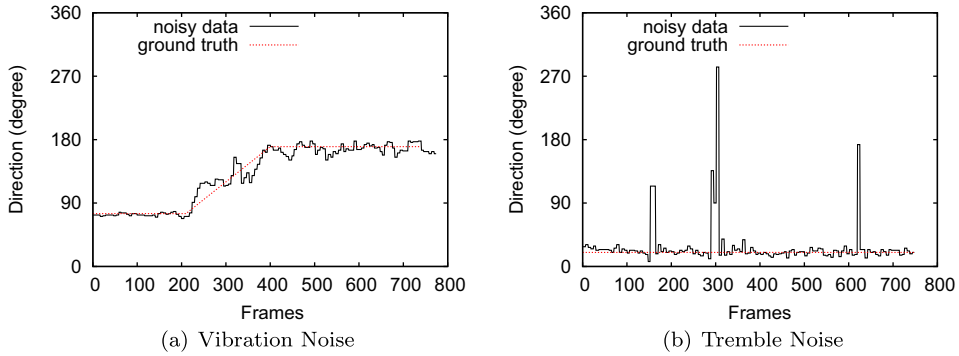


Fig. 7. Examples of noise.

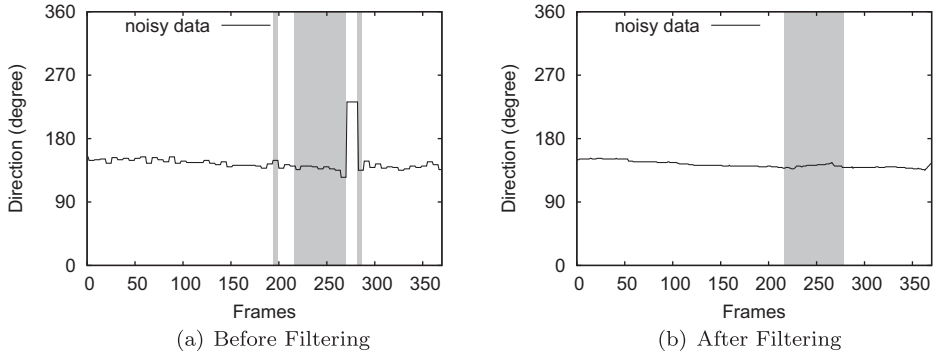
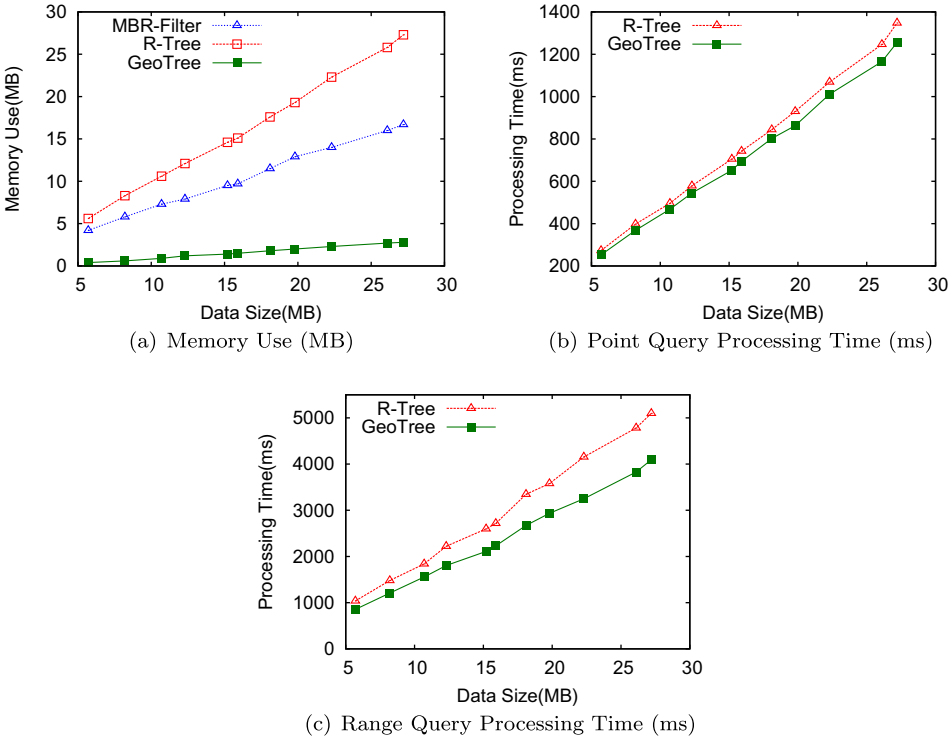


Fig. 8. Direction data and query response. Regions colored with gray represents the response to a query. Before filtering, the response interval appear in three fragments. After filtering, the response interval appear in a single fragment.

Then, we find m farthest vectors, i.e., outliers, from the normalized mean vector based on Euclidean distance in the cartesian coordinate space. The remaining $N-m$ vectors excluding the outliers are converted into angles.

In our experiments, we used filter of mask size $N=49$ and $\alpha=0.1$, which means we averaged 48 adjacent frame directions (24 before and 24 after) as well as its own frame directions to get the mean value. Then, we exclude m frames which are farthest from the mean value, where m is given by $N*2\alpha$. In our case, $N=49,\ \alpha=0.1$, so $m=9$.

Fig. 8 shows the effect of the filter. This graph shows the direction data and the query response in that interval. Before filtering, query response is affected by noise and the response interval appear in three fragments due to the extreme change of direction by tremble noise. After filtering, effect of tremble noise is minimized and the response came in a single fragment.



(a) Memory Use (MB) (b) Point Query Processing Time (ms)

(c) Range Query Processing Time (ms)

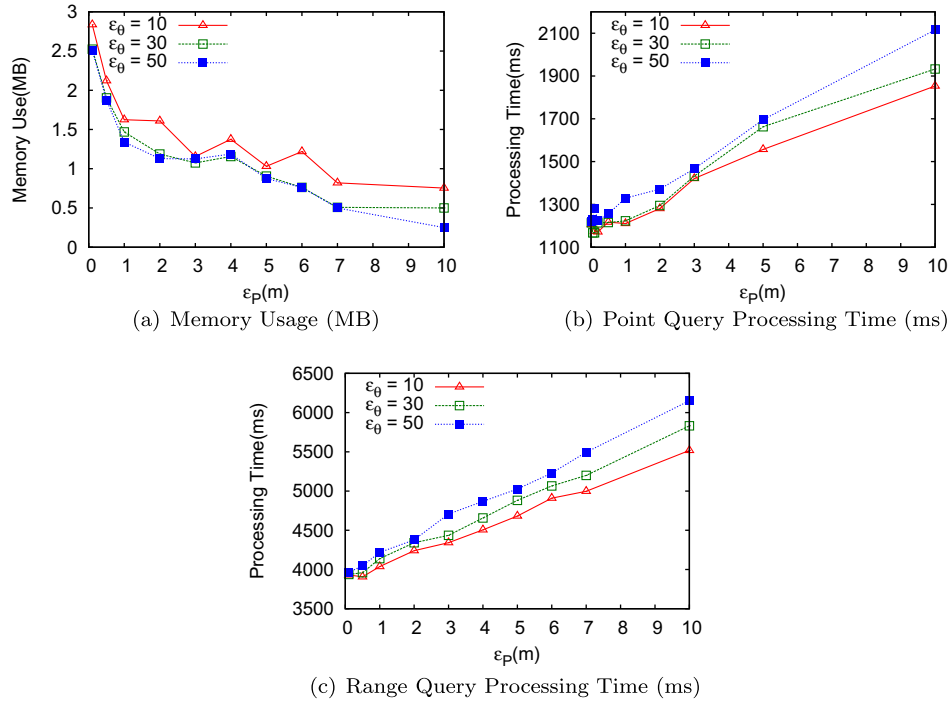
Fig. 9. Result of varying data sizes.

Table 4
Comparison of query processing time: mean (and standard deviation).

	MBR-filtering	R-Tree	GeoTree
Range query (ms)	307,461 (6272.1)	5668.7 (154.622)	4466.2 (104.322)
Point query (ms)	66,454 (1629.5)	1288.6 (32.368)	1210.5 (29.08)
Memory use (MB)	16.7	27.3	2.83

4. Experiments

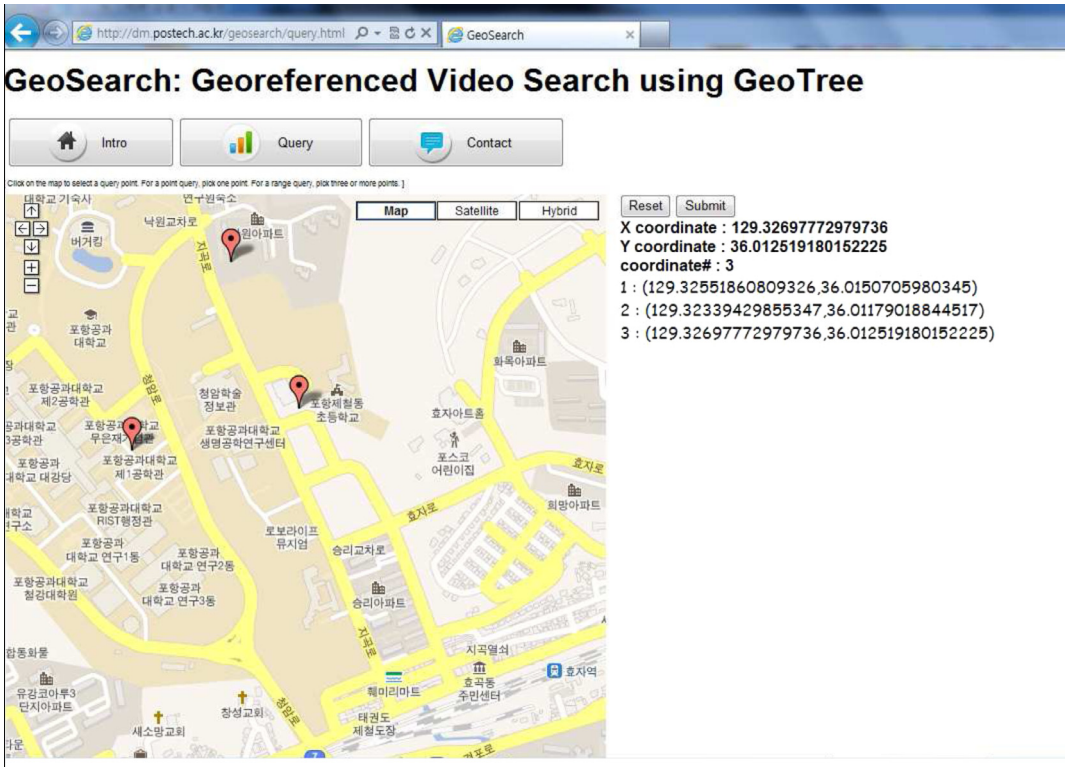
In this section, we evaluate the performance of GeoTree and compare it to R-Tree and MBR-Filter, i.e., the MBR-based filtering proposed in [2]. We evaluate the performance of processing point queries and range queries in terms of query processing time and also memory usage. All of the methods find exact results, and thus,



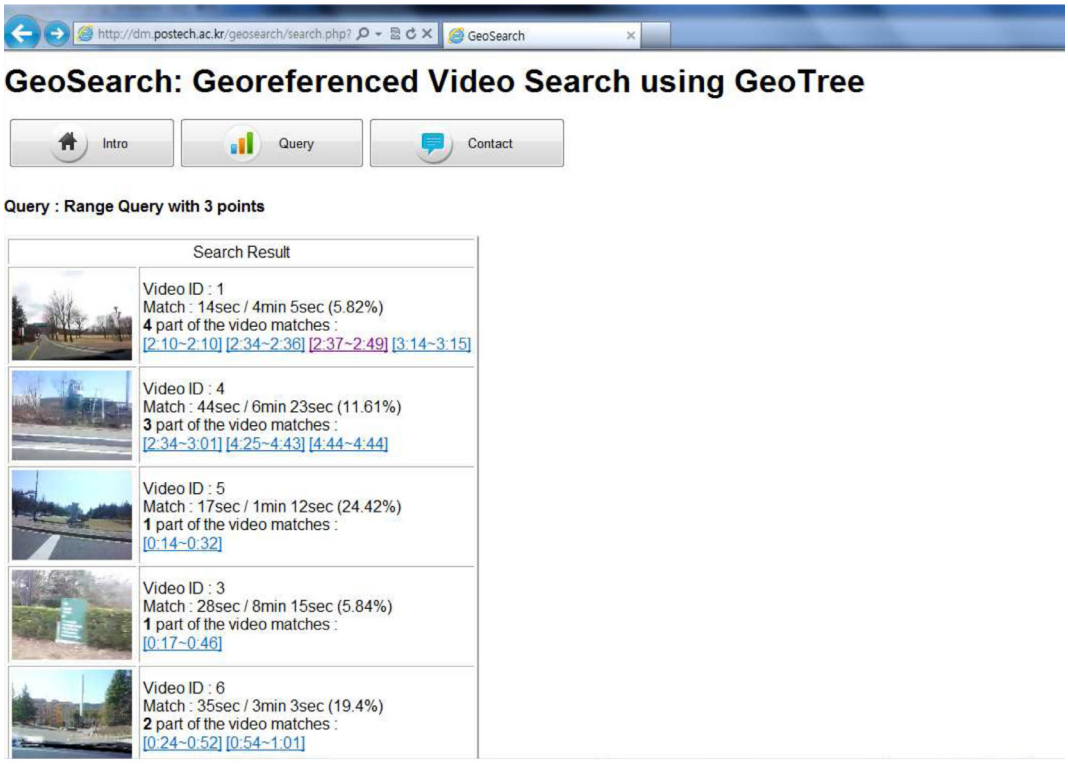
(a) Memory Usage (MB) (b) Point Query Processing Time (ms)

(c) Range Query Processing Time (ms)

Fig. 10. Result of varying parameter values.



(a) Querying Points



(b) Query Result

Fig. 11. Demonstration site.

the query results are identical. Therefour, we restrict our discussion to processing time and memory usage. Our experiments were done using a machine with the following specifications: dualcore CPU (2.4 GHz) and 4 GB of Memory.

4.1. DataSet

Experiments were performed on real FOVstream data sets. We captured real scenes using an Android mobile phone equipped

with a GPS receiver and a 3D compass. Since actual videos are represented as sequences of FOVscene metadata according to our scene model, we only collected metadata from the mobile phone. We ignored altitude and pitch angles from the 3D compass, as we are only interested in 2D. Thirty FOVscenes were captured every second. Latitudinal and longitudinal data were converted into meter metrics in the 2D plane. No zoom level change was simulated, thus viewable angle θ and visible distance R were kept constant and thus fixed to $\theta = 55^\circ$ and $R = 50$ m.

FOVscenes ranged in the area of 3.3 km long in the north–south and 3.8 km long in east–west. Test data corresponded to 11 streams with a total duration of 180 min. The number of FOVscenes was 325,313 in total. This number of streams corresponded to approximately 4.4 GB of typical video data. The sum of all trajectories' length was approximately 68 km.

For the leaf nodes of GeoTree with less than 5 FOVscenes, we simply used simple one-by-one search and did not perform MBTR filtering and lookup, because, when an MBTR contains just a few FOVscenes, scanning FOVscenes one-by-one runs faster due to some overhead of MBTR filtering and lookup.

4.2. Result summary

Table 4 summarizes the results of the query processing time for each method. For the range query test, we randomly generated 10,000 queries and measured the accumulated processing time for the queries. For the point query test, we randomly generated 100,000 queries and also measured the accumulated processing time for the queries. The processing times are averaged by 30 runs. Standard deviations are also shown – all of them show very small deviations.

For GeoTree, we first tuned the error threshold parameters on the dataset, and fixed the parameter values for all runs, these were. $\epsilon_P = 1$ m and $\epsilon_\theta = 10^\circ$.

MBR-Filter was one hundred times slower than the other tree-based searches. We omitted the naive sequential search results because it was even much slower than MBR-Filter. GeoTree gave a consistently faster query processing time than R-Tree. Although the query processing times of R-Tree and GeoTree were not substantially different, their memory usages were an order of magnitude different. As Fig. 9(a) shows, the performance difference became more significant, as data size increased.

It is because R-Tree does not utilize the continuity of the adjacent frames. In order to enable random access in R-Tree, it is necessary to put every single frame into a leaf node. Otherwise, it will result in significant increase in processing time. In contrast, leaf node of GeoTree covers some number of frames and MBTR structure enables the random access in the leaf node.

4.3. Result of varying data sizes

We compared the performance of indexing methods on various data sizes. In order to make datasets of varying sizes, we started from one FOVstream and added streams one-by-one. Fig. 9 shows the results. Both memory usage and query processing time increased, as data size increased. GeoTree outperformed R-Tree in every case. The processing time for MBR-Filter is not present, because it is too high to present in the same graphs.

4.4. Results of varying other parameters

The performance of GeoTree varies with the threshold parameter values. We investigated the performance change in GeoTree with respect to parameters ϵ_P and ϵ_θ. The results are illustrated in Fig. 10. There is a tradeoff relationship between memory usage and processing time. As the threshold parameter values increased, memory usage decreased while processing time increased. Larger threshold values created a smaller number of MBTRs, each with a larger number of FOVscenes. Memory usage primarily depends on the number of leaf nodes. We can save memory by setting large threshold values and creating a small number of MBTR s. Due to the MBTR lookup algorithm, a decrease in the number of MBTRs does not lead to a rapid increase in the processing time. Even in the MBTR with a large number of FOVscenes, MBTR lookup effectively filters out irrelevant FOVscenes during query processing. The query processing was still fast even with a small number of MBTRs.

4.5. Demonstration

We have built a demo system on a real map of the campus of POSTECH.[8] We captured various places of the campus and uploaded the videos to YouTube. We then built an online website, "http://dm.postech.ac.kr/geosearch", which can search corresponding video frames using point or range queries. In our online search website, a user submits a point or range query by mouse clicks on the map, and our system returns corresponding video frames using Tubechop.[9] Fig. 11(a) illustrates an example of a range query in our website. Fig. 11(b) shows the result in text and links to video views. Once the user clicks a link, our system plays corresponding video frames as shown in Fig. 11(c).

5. Conclusions

This paper proposed a new efficient indexing method, called GeoTree, which enables an efficient search of georeferenced video data. GeoTree adopts a new data structure, MBTR (Minimum Bounding *Tilted* Rectangle), in the leaf nodes of R-Tree, in order to efficiently store sequences of moving scenes and efficiently prune out unpromising parts of videos in query processing. Our experiments show that, compared to recent methods based on MBR pruning and R-Tree, GeoTree significantly reduces the index size and also the query processing time. Future work will include the generalizing of GeoTree to support searching for videos with adjustable viewable angles and distances.

References

[1] D. Anguelov, C. Dulong, D. Filip, C. Frueh, S. Lafon, R. Lyon, A. Ogale, L. Vincent, J. Weaver, Google street view: capturing the world at street level, Computer 43 (2010) 32–38.
[2] S. Arslan Ay, R. Zimmermann, S. Kim, Relevance ranking in georeferenced video search, Multimedia Syst. 16 (2010) 105–125. 10.1007/s00530-009-0177-x.
[3] S.A. Ay, S.H. Kim, R. Zimmermann, Generating synthetic meta-data for georeferenced video management, in: Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS '10, ACM, New York, NY, USA, 2010, pp. 280–289.
[4] S.A. Ay, R. Zimmermann, S.H. Kim, Viewable scene modeling for geospatial video search, in: Proceedings of the 16th ACM International Conference on Multimedia, MM '08, ACM, New York, NY, USA, 2008, pp. 309–318.
[5] J. Bednar, T. Watt, Alpha-trimmed means and their relationship to median filters, IEEE Trans. Acoust. Speech Signal Process. 32 (1) (1984) 145–153.
[6] M. Chaabouni, S.M. Chung, The point-range tree: a data structure for indexing intervals, in: Proceedings of the 1993 ACM Conference on Computer Science, CSC '93, New York, NY, USA, ACM, 1993, pp. 453–460.
[7] S. Dagtas, W. Al-Khatib, A. Ghafoor, R. Kashyap, Models for motion-based video indexing and retrieval, IEEE Trans. Image Process. 9 (1) (2000) 88–101.
[8] E. Frentzos, Y. Theodoridis, On the effect of trajectory compression in spatiotemporal querying, in: Proc. of East European Conf. Advances in Databases and Information Systems, Springer, Berlin/Heidelberg, 2007.
[9] A. Guttman, R-trees: a dynamic index structure for spatial searching, in: International Conference on Management of Data, ACM, 1984, pp. 47–57.
[10] H.-P. Kriegel, M. Ptke, T. Seidl, Interval sequences: an object-relational approach to manage spatial data, in: C. Jensen, M. Schneider, B. Seeger, V.

[8] http://www.postech.ac.kr.
[9] www.tubechop.com: a site that shows a segment of YouTube videos.

Tsotras (Eds.), Advances in Spatial and Temporal Databases, Lecture Notes in Computer Science, vol. 2121, Springer, Berlin/Heidelberg, pp. 481–501.
[11] N. Meratnia, R. de By, Spatiotemporal compression techniques for moving point objects, in: Proc. of Int. Conf. on Extending Database Technology (EDBT 2004), 2004.
[12] M.F. Mokbel, T.M. Ghanem, W.G. Aref, Spatio-temporal access methods, IEEE Data Eng. Bull. 26 (2003) 40–49.
[13] D. Pfoser, Indexing the trajectories of moving objects, IEEE Data Eng. Bull. 25 (2002) 3–9.
[14] D. Pfoser, C.S. Jensen, Y. Theodoridis, Novel approaches to the indexing of moving object trajectories, 2000, pp. 395–406.
[15] M. Potamias, K. Patroumpas, T. Sellis, Sampling trajectory streams with spatiotemporal criteria, in: 18th International Conference on Scientific and Statistical Database Management, 2006, pp. 275–284.
[16] A. Sabatini, Quaternion-based extended Kalman filter for determining orientation by inertial and magnetic sensing, IEEE Trans. Biomed. Eng. 53 (7) (2006) 1346–1356.
[17] E. Sahouria, A. Zakhor, A trajectory based video indexing system for street surveillance, in: IEEE Int. Conf. on Image Processing, 1999, pp. 24–28.
[18] S. Saltenis, C.S. Jensen, S.T. Leutenegger, M.A. Lopez, Indexing the positions of continuously moving objects, 2000.
[19] H. Samet, Spatial Data Structures, Addison-Wesley, 1995.
[20] K. Shearer, S. Venkatesh, D. Kieronska, Spatial indexing for video databases, J. Visual Commun. Image Represent. 7 (4) (1996) 325–335.
[21] K. Toyama, R. Logan, A. Roseway, Geographic location tags on digital images, in: Proceedings of the Eleventh ACM International Conference on Multimedia, MULTIMEDIA '03, ACM, New York, NY, USA, 2003, pp. 156–166.