GeoTree: Using Spatial Information for Georeferenced Video Search

Youngwoo Kim, Jinha Kim, Hwanjo Yu
{ywkim10,goldbar, hwanjoyu}@postech.ac.kr
March 2012
POSTECH-CSE-2012-03

Department of Computer Science & Engineering
Pohang University of Science and Technology
Republic of Korea, 790-784

Abstract

With the rapid popularization of video recording devices, more multimedia contents are available to 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 able 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 GeoTree which enables a fast search of georeferenced videos. In particular, we propose a new data structure MBTR (Minimum Bounding Tilted Rectangle) to efficiently store the areas of moving scenes in the tree. We propose algorithms for building MBTRs from georeferenced videos and algorithms for efficiently processing point and range queries on GeoTree. Experiment results on real georeferenced video data show that, compared to previous indexing methods for georeferenced video search, GeoTree substantially reduces the index size and improves the search speed for georeferenced video data.

1This work was partially supported by the Brain Korea 21 Project in 2012 and Mid-career Researcher Program through NRF grant funded by the MEST (No. KRF-2011-0016029). This work was also supported by IT Consilience Creative Program of MKE and NIPA (C1515-1121-0003).
2corresponding author
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 [3, 4].

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.

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 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 [8] 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 [6]. 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 [6] and an R-Tree-based algorithm. GeoTree substantially reduces index size and improves search speed for georeferenced video data.

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. [23] 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 [1] and Woophy [2], 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 [5] 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 [18, 11]. 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” [10, 19] and “spatial indexing for video” [22] 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. [6, 8]. 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 [6]. 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 non-trivial research problem.

Ay et al. also proposed a method for generating synthetic metadata for georeferenced video search [7]. 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 [12, 15, 21]. 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 [16, 20]. To utilize the continuity of object trajectory and enable temporal query, several effective indexing structures such as STR-Tree and TB-tree were proposed [17].
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 [14, 11]. 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 [9, 13].

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 our indexing method (Section 3.2). We then present algorithms for building the index (Section 3.3) and processing point and range queries on the index (Section 3.4).

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) [8]. As illustrated in Figure 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 $\theta$, and (4) visible distance $R$.

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

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 $\theta$ and visible distance $R$ can be obtained from the camera lens’ property and zoom level. As assumed in [8, 6], we also fixed $\theta$ 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 [8, 6], 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 (Figure 1(b)), but they do not alter the nature of the FOVscene model.
3.2 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 3.3 and 3.4 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. Figure 2 depicts examples of MBR and MBTR representing areas of moving scenes. MBTR is more suitable for representing an area of moving scenes; as video recording devices such as car blackboxes are often moving piece-wise linearly, MBTR produces less false positive areas. For example, when a query, i.e., the red circle in the figure, is submitted, MBR in Figure 2(a) includes the query area, thereby incurring a large area of false scanning of videos while MBTR (Figure 2(b)) efficiently prunes the videos.

  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 3.3 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 MBTRs 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 3.4 discusses checking and retrieving algorithms on the MBTR.

3.3 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 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 MBTR,
second division, as illustrated in Figure 3(b). The key idea underlying the finding of markup FOVscenes is that the scenes are moving perfectly linearly from the start to the end in terms of $P$. (Figure 3(a) shows an example.)

From this idea, we draw a regression line from the start to the end point of $F_i$. 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 Figure 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 $\epsilon$, 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 Figure 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 $\epsilon$.

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

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

Figure 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.

<table>
<thead>
<tr>
<th>Algorithm 1: FindingMarkupFOVScene($F$, $\epsilon_P$, $\epsilon_d$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 $F$ : an FOVstream</td>
</tr>
<tr>
<td>2 $\epsilon_P$, $\epsilon_d$: threshold for errors of $P$ and $\vec{d}$</td>
</tr>
<tr>
<td>3 $S_1 = \text{MarkupFOVScene}_P(F, s, e, \epsilon_P)$</td>
</tr>
<tr>
<td>4 $S_2 = \text{MarkupFOVScene}_d(F, s, e, \epsilon_d)$</td>
</tr>
<tr>
<td>5 return $S_1 \cup S_2$</td>
</tr>
</tbody>
</table>

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 MBTR, we can check if a query area is located within the MBTR by simply retrieving the markup FOVscenes. (Algorithms for checking and retrieving FOVscene within an MBTR are discussed in Section 3.4.)

3.3.1 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 [14] 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. (Figure 3(a) shows an example.)

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 Figure 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 $\epsilon$, 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 Figure 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 $\epsilon$.

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

$$
\Delta e = t_e - t_s
$$
$$
\Delta i = t_i - t_s
$$
$$
P_i' = P_s + \frac{\Delta i}{\Delta e} (P_e - P_s)
$$
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. $\epsilon_P$: threshold
5. for $i \leftarrow s$ to $e$ do
6. \quad $P'_i = \text{Regression}(F_s, F_e, F_i)$
7. \quad dist = distance($P'_i, P_i$)
8. \quad if dist > dist_max then
9. \quad \quad dist_max $\leftarrow$ dist
10. \quad \quad peak $\leftarrow$ $i$
11. \quad end
12. end
13. if dist_max > $\epsilon_P$ then
14. \quad list1 = MarkupFOVScene_P($F, s, peak, \epsilon_P$)
15. \quad list2 = MarkupFOVScene_P($F, peak, e, \epsilon_P$)
16. \quad return list1 + list2
17. else
18. \quad 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. $\Delta e = t_e - t_s$
6. $\Delta i = t_i - t_s$
7. $P'_i = P_s + \frac{\Delta i}{\Delta e}(P_e - P_s)$
8. return $P'_i$

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 MB7R, 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 $\epsilon_P$ and $\epsilon_{\vec{d}}$ respectively. Algorithm 2 finds markup FOVscenes for $P$ using a 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 $\epsilon_P$ (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
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. \(\epsilon\vec{d}\): threshold
5. for \(i \leftarrow s\) to \(e\) do
   6. // Get the mean \(\vec{d}\) of two extremes in the interval
   7. \(\vec{d}_i' = \text{MiddleValue}(F, s, i)\)
   8. \(\text{dist} = \text{distance}(\vec{d}_i', \vec{d}_i)\)
   9. if \(\text{dist} \geq \epsilon\vec{d}\) then
      10. \(\text{list1} = [F_s, F_i]\)
      11. \(\text{list2} = \text{MarkupFOVScene\(_d(F, i, e, \epsilon\vec{d})\)}\)
      12. return \(\text{list1} + \text{list2}\)
   13. end
14. end
15. return \([F_s, F_e]\)

\(P_s\) Starting point of location trajectory.
\(P_e\) Ending point of location trajectory.
\(r_{\text{left}}\) Maximum distance to left sides of moving direction
\(r_{\text{right}}\) Maximum distance to right sides of moving direction
\(r_{\text{forward}}\) Maximum distance to forward of moving direction
\(r_{\text{back}}\) Maximum distance to backward of moving direction

Table 1: MB7R Parameters

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).

3.3.2 Constructing MB7R from FOVstream

Once a set of markup FOVscenes are found, we construct MB7Rs from pairs of adjacent markup FOVscenes. An MB7R 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 \(\epsilon_P\) because \(\epsilon_P\) was the error threshold for \(P\) such that the locations of FOVscenes within an MB7R can deviate from the regression line by as large a value as \(\epsilon_P\).

Figure 4 illustrates the MB7R 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. Figure 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 MB7R. We then construct a bounding rectangle for the circular sector and expand the bounding rectangle by \(\epsilon_P\) 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 $r_{\text{forward}}$, $r_{\text{backward}}$, $r_{\text{left}}$, and $r_{\text{right}}$ 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 Figure 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.

3.4 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 3.4.1, then discuss those for range queries in Section 3.4.2.

3.4.1 Point Query Processing

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. Figure 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. Figure 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 Equations (3) and (4).

\[
(3) \quad i = \left\lceil (D - r_{\text{forward}}) \times \frac{n - 1}{l} \right\rceil
\]

\[
(4) \quad j = \left\lfloor (D + r_{\text{back}}) \times \frac{n - 1}{l} \right\rfloor
\]

where $k$ is the frame number of FOVscene counted from $P_s$, $l$ is the length of $P_sP_e$, 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 $P_sP_e$. $D$ can be negative if $q$ is projected onto the line outside $P_sP_e$. 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 Equations (3) and (4). Figure 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 k-th FOVscene $F_k$ and $P'_k$ be the location of the corresponding E-MBR. Note that $P'_k$ is located on $P_sP_e$ and moves from $P_s$ to $P_e$ at a constant speed. The k-th E-MBR, which bounds $F_k$, is given by a rectangle located at $P'_k$. This E-MBR has widths $r_{\text{left}}$, $r_{\text{forward}}$, $r_{\text{right}}$, $r_{\text{back}}$ 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 $r_{\text{left}}$ or $r_{\text{right}}$ from $P_sP_e$. Thus we only need to consider $r_{\text{forward}}$ and $r_{\text{back}}$. For a query point to be located inside the k-th 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 k-th E-MBR is $r_{\text{back}}$ from $P'_k$ and the forward boundary is $r_{\text{forward}}$. Thus, a query point is located inside the k-th E-MBR only if it satisfies Equation (5).

\[
(5) \quad k \times \frac{l}{n-1} - r_{\text{back}} \leq D \leq k \times \frac{l}{n-1} - r_{\text{forward}}
\]

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

### 3.4.2 Range Query Processing

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
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. $r_{\text{left}}, r_{\text{right}}, r_{\text{forward}}, r_{\text{back}}$: MBTR parameters
8. $B$: Boundary for MBTR
9. \{ $B$ is tilted rectangle given four $r$ values \}
10. if pointPolygonIntersect($q, B$) then
11. \hspace{1em} $D = \text{projectedDistance}(P_s, P_e, q)$
12. \hspace{1em} $i = \left\lceil (D - r_{\text{forward}}) \times \frac{n-1}{l} \right\rceil$
13. \hspace{1em} $j = \left\lfloor (D + r_{\text{back}}) \times \frac{n-1}{l} \right\rfloor$
14. \hspace{1em} for $k \leftarrow i$ to $j$ do
15. \hspace{2em} if pointFOVIntersect($q, F_k$) then
16. \hspace{3em} addList($L, F_k$)
17. \hspace{1em} end
18. \hspace{1em} end
19. return $L$

\begin{tabular}{|l|l|}
\hline
pointPolygonIntersect($q, P$) & Returns true if point $q$ overlaps with polygon $P$ \\
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 $P_sP_e$. It is given by \[ \frac{P_sq'.P_e}{P_sP_e} \] \\
\hline
\end{tabular}

Table 2: Subroutines

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, $D_{\text{min}}$ and $D_{\text{max}}$, where $D_{\text{min}}$ is the closest and $D_{\text{max}}$ is
the farthest point from $P_s$ on the line $P_sP_e$. Indices $i$ and $j$ are computed using equations that are similar to Equations (3) and (4) but $D$ is replaced with $D_{\text{min}}$ and $D_{\text{max}}$ accordingly as follows.

\begin{equation}
    i = \left\lceil \left( D_{\text{min}} - r_{\text{forward}} \right) \times \frac{n - 1}{l} \right\rceil
\end{equation}

\begin{equation}
    j = \left\lfloor \left( D_{\text{max}} + r_{\text{back}} \right) \times \frac{n - 1}{l} \right\rfloor
\end{equation}

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.

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

**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 FOVscen es 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 $\theta$ and visible distance $R$ were kept constant and thus fixed to $\theta = 55^\circ$ and $R = 50m$.

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

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

**Result Summary:** Table 3 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. $\epsilon_p = 1m$ 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 Figure 6(a) shows, the performance difference became more significant, as data size increased.

**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. Figure 6 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.

**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 $\epsilon_p$ and $\epsilon_\theta$. The results are illustrated in Figure 7. 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 FOVscen es. 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 MBTRs. 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 FOVscen es, MBTR lookup effectively filters out irrelevant FOVscen es during query processing. The query processing was still fast even with a small number of MBTRs.

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


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
   \{ $B$ is tilted rectangle given four $r$ values \}
8. $C$: Set of critical points to check
9. /* Collecting critical point */
10. $C = \text{getIntersections}(B, Q)$
11. for $\forall v_Q \in Q$ do
12.   if $\text{pointPolygonIntersect}(v_Q, B)$ then
13.     $C = C \cup v_Q$
14. end
15. end
16. for $\forall v_B \in B$ do
17.   if $\text{pointPolygonIntersect}(v_B, Q)$ then
18.     $C = C \cup v_B$
19. end
20. end
21. /* MBTR lookup */
22. if $C \neq \emptyset$ then
23.   $D_{\min} = \min_{v \in C} \text{projectedDistance}(P_s, P_e, v)$
24.   $D_{\max} = \max_{v \in C} \text{projectedDistance}(P_s, P_e, v)$
25.   $i = \left\lceil \left( D_{\min} - r_{\text{forward}} \right) \times \frac{n-1}{l} \right\rceil$
26.   $j = \left\lfloor \left( D_{\max} + r_{\text{back}} \right) \times \frac{n-1}{l} \right\rfloor$
27. for $k \leftarrow i$ to $j$ do
28.   if $\text{polygonFOVIntersect}(Q, F_k)$ then
29.     addList($L, F_k$)
30. end
31. end
32. end
33. return $L$
34

<table>
<thead>
<tr>
<th></th>
<th>MBR-Filtering</th>
<th>R-Tree</th>
<th>GeoTree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range Query (ms)</td>
<td>297,800 (4400)</td>
<td>6,039.4 (374.4)</td>
<td>4,554.2 (153.6)</td>
</tr>
<tr>
<td>Point Query (ms)</td>
<td>68,297 (492.3)</td>
<td>1,302.0 (17.28)</td>
<td>1,264.5 (67.75)</td>
</tr>
<tr>
<td>Memory Use (MB)</td>
<td>16.6</td>
<td>27.3</td>
<td>2.83</td>
</tr>
</tbody>
</table>

Table 3: Comparison of Query Processing Time: Mean (and Standard deviation)