Efficient Indexing and Querying of Geo-tagged Aerial Videos

Ying Lu, Cyrus Shahabi
Integrated Media Systems Center, University of Southern California, Los Angeles, CA 90089
{ylu720,shahabi}@usc.edu

ABSTRACT

Driven by the advances in control engineering, material science and sensor technologies, drones are becoming significantly prevalent in daily life (e.g., event coverage, tourism). Consequently, an unprecedented number of drone videos (or aerial videos) are recorded and consumed. In such a large repository, it is difficult to index and search aerial videos in an unstructured form. However, due to the rich sensor instrumentations of drones, aerial videos can be geo-tagged (e.g., GPS locations, drone rotation angles) at the acquisition time, providing an opportunity for efficient management of aerial videos by exploiting their corresponding spatial structures. Each aerial video frame can thus be represented as its spatial coverage, termed aerial Field-Of-View (aerial-FOV). This effectively converts a challenging aerial video management problem into a spatial database problem on aerial-FOVs. In this paper, we focus on efficient indexing and querying of aerial-FOVs.

Unfortunately, aerial-FOVs are shaped in irregular quadrilaterals, and this renders existing spatial indexes inefficient to index aerial-FOVs. Therefore, we propose a new index structure called TetraR-tree that effectively captures the geometric property of aerial-FOVs. Based on the TetraR-tree, we develop two novel search strategies to efficiently process point and range queries on aerial-FOVs. Our experiments using both real-world and large synthetic video datasets (over 30 years' worth of videos) demonstrate the scalability and efficiency of our proposed indexing and querying algorithms.

CCS CONCEPTS

• **Information systems → Multimedia databases**;

KEYWORDS

Aerial videos, Drone videos, Indexing, Querying, Geo-tagged

ACM Reference format:
Ying Lu, Cyrus Shahabi. 2017. Efficient Indexing and Querying of Geo-tagged Aerial Videos. In *Proceedings of SIGSPATIAL'17, Los Angeles Area, CA, USA, November 7–10, 2017,* 10 pages.
https://doi.org/10.1145/3139958.3140046

1 INTRODUCTION

Unmanned aerial vehicles (UAVs, a.k.a. drones) have been mainly used in military activities and public safety for decades. Recently, with advances in control engineering, material science and sensor technologies, drones have gained significant commercial momentum. The number of drones is consequently increasing dramatically. According to the statistic report of Digital Marketing Ramblings (DMR)[1] [2], about 0.7 million drones shipped in 2015 and the number is expected to increase up to 7 million by 2020. The estimated value of the drone industry is $3.3 billion in 2015 and is forecasted to rise up to $127 billion by 2020. Videos (termed "aerial videos") recorded by UAVs or drones are also correspondingly becoming prevalent. Aerial videos play an important role in daily life and they are used in the growing applications such as event coverage, tourism, transportation, rescues, agriculture and disaster responses. Consequently, organizing and searching aerial videos is becoming a critical problem.

However, these days, typical drones (e.g., Canada Drones [1]) almost always include rich sensors such as GPS, 3-axis accelerometers and gyroscopes, and their corresponding sensed data can be automatically collected during the video recording. To overcome the challenges of large-scale management of unstructured aerial videos, we leverage such drone sensors to represent the visible scenes with the spatial coverages of videos. Ideally, each aerial video frame can be modeled as the spatial extent of its coverage area, termed aerial Field-Of-View (aerial-FOV), using the camera location, drone rotation angles (i.e., azimuth, pitch and roll) and camera viewable angle [13, 23, 27]. In this paper, we tackle the challenges of large-scale aerial video data management by indexing and querying the corresponding aerial-FOVs.

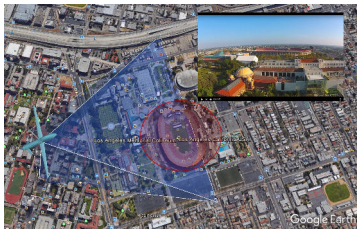


Figure 1: An example of range queries on aerial-FOVs

An aerial-FOV, the spatial coverage on the ground of an aerial video frame, is shaped as a quadrilateral, as shown in Fig. 1 (blue quadrilateral). Subsequently, an aerial video can be represented as a series of aerial-FOVs. For aerial-FOVs, we introduce two typical spatial queries: point and range queries. A point query finds all the aerial-FOVs that cover a user-specified query point (e.g., a person, a statue). A range query finds all the aerial-FOVs that overlap with a circular query range (e.g., a college campus, a park). Point and range queries are the main building blocks of many geospatial applications in various domains such as in urban planning and

SIGSPATIAL'17, November 7–10, 2017, Los Angeles Area, CA, USA

ACM ISBN 978-1-4503-5490-5/17/11…$15.00
https://doi.org/10.1145/3139958.3140046

[1] http://expandedramblings.com/index.php/drone-statistics/
[2] http://dronelife.com/2016/07/19/8-incredible-drone-industry-stats/

criminal investigation. For example, to investigate the disappearance of Yingying Zhang in UIUC [3], FBI needed to find all the videos that can cover the bus station where Yingying Zhang visited (point query) and all the videos that cover the areas around the bus station (range query). Similar queries were used to search crowdsourced videos of the Boston Marathon bombing. Fig. 1 illustrates the result of a circular range query that searches aerial-FOVs overlapping with Los Angeles Memorial Coliseum (red circle).

Indexing aerial-FOVs to answer such spatial queries poses challenges to existing spatial indexes due to the fact that aerial-FOVs are shaped in irregular quadrilaterals (see Figures 2 and 3). A typical baseline is R-tree [7], which can index aerial-FOVs by enclosing the coverage area (i.e., quadrilateral) of each aerial-FOV with its minimum bounding rectangle (MBR). However, R-tree suffers from large "dead spaces" (i.e., empty spaces that are covered by an index node but not overlap with any objects in the node), which results in many unnecessary index node accesses. Another baseline approach of indexing aerial-FOVs is extending OR-tree [17], which is the state-of-the-art index for ground-FOVs (i.e., coverages of videos recorded on the ground by (say) mobile phones). Different from ground-FOVs which are regular pie-shaped [5], an aerial-FOV is a quadrilateral with an arbitrary shape. Answering point and range queries for aerial-FOVs with OR-tree is complicated and computationally expensive (See Sec. 3 for details).

To overcome the shortcomings of the baselines, we propose a new index structure, called TetraR-tree, and two novel search strategies based on TetraR-tree for both point and range queries. Specifically, the contributions of this paper are listed below.

- We introduce the problem of indexing and querying on aerial videos by exploiting their spatial coverages, i.e., aerial-FOVs. To the best of our knowledge, this is the first work on aerial-FOV indexing and query processing.
- We propose a new index structure, called TetraR-tree. Different from R-tree which stores the MBR of aerial-FOVs, at each index node of TetraR-tree, we store four MBRs (tetra-corner-MBRs), each of which covers one of the four corner points of *all* the aerial-FOVs (i.e., quadrilaterals) in that index node. We also propose a heuristic during the index construction of a TetraR-tree that tries to minimize the alignment waste of the four corners of all the quadrilaterals in an index node (Sec. 4).
- We propose two novel search strategies based on TetraR-tree for both point and range queries. Based on the geometric properties of the tetra-corner-MBRs in a TetraR-tree index node, we can compute two convex hulls: 1) the smallest convex hull (called outer convex hull) that encloses all the aerial-FOVs in the node, and 2) the largest convex hull (called inner convex hull) where all the aerial-FOVs in the node can cover. Subsequently, based on the outer and inter inner convex hulls, we propose two new search strategies: pruning strategy (to effectively decide whether a node can be pruned or not) and total hit strategy (to effectively decide whether all the objects in a node are in the results or not) for both point and range queries (Sec. 5).
- To solve the fundamental geometric challenges in the computation of outer and inner convex hulls, we propose a filtering technique to accelerate the convex hull computation (Sec. 5.1.3).
- Our extensive experiments using a real-world dataset and a large synthetically generated dataset demonstrate the superiority of TetraR-tree with our proposed search algorithms over the baselines (R-tree and OR-tree) for both point and range queries by at least 70% in terms of query time and I/O cost. Note that on the large-scale generated dataset (more than 30 years' worth of aerial videos), our TetraR-tree can respond to both query types within a reasonable time (1 – 2 seconds). In addition, the results demonstrate the benefits of the two search strategies based on the outer and inner convex hulls as well as the two filtering techniques in the computations of the two convex hulls (Sec. 6).

[3] http://police.illinois.edu/search-updates/

2 PRELIMINARIES

We first introduce a spatial coverage model for aerial videos and then present aerial video queries based on their spatial coverages.

2.1 Spatial Model for Aerial Videos

These days, typical drones (e.g., Canada Drones [1]) almost always include rich sensors such as cameras, GPS, 3-axis accelerometers and 3-axis gyroscopes. Thus the geographic sensor metadata (e.g., camera locations and viewing directions) of drones can be automatically collected during the video recording.

Aerial videos can be represented as a sequence of video frames, and the field of view of each aerial video frame [13, 23, 27], referred as "aerial-FOV", is illustrated in Fig. 2. An aerial-FOV f is modeled as a 7-tuple Γ $(lat, lng, hgt, \theta_a, \theta_p, \theta_r, \alpha)$, in which, $< lat, lng, hgt >$ is the camera location: latitude, longitude and the height with respect to the ground, θ_a is the azimuth (yaw) angle rotating around the vertical axis (i.e., the angle from the north to azimuth direction), θ_p is the pitch angle rotating around the lateral axis (i.e., the angle from the direction toward to the earth to the pitch direction), θ_r is the roll angle rotating around the longitudinal axis (i.e., the angle from the parallel plane of the ground to the roll direction), and α is the drone camera visible angle.

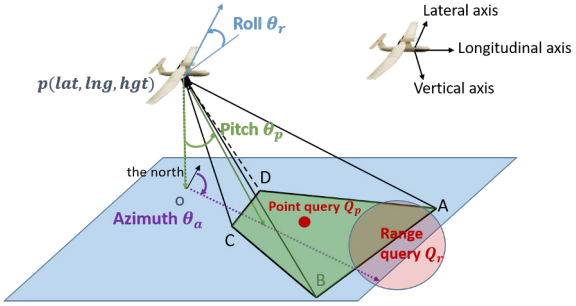
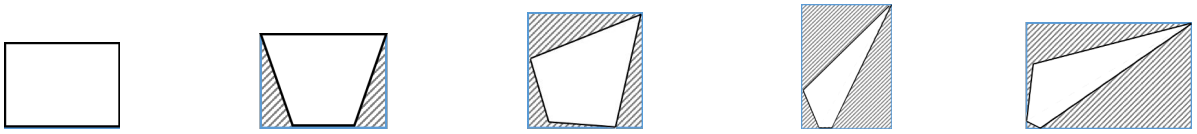


Figure 2: Aerial Field of View (Aerial-FOV) Model

As shown in Fig. 2, the spatial coverage on the ground of a drone video frame is shaped as a quadrilateral $\mathcal{P}$ which can be represented by its four vertices (or corner points) $\{A, B, C, D\}$. As most applications are typically interested in the video coverage area on the ground [13, 15, 27], in this paper, we focus on the spatial coverage on the ground of aerial videos and represent an aerial-FOV as a quadrilateral. The quadrilateral of an aerial-FOV $f(A, B, C, D)$ can be derived from the 7-tuple $\Gamma(lat, lng, hgt, \theta_a, \theta_p, \theta_r, \alpha)$ [15]. As the pitch and roll angles change, the shape of an aerial-FOV changes and can be any irregular quadrilateral, as shown in Fig. 3.



(a) Square $< 0°, 0°, 0° >$ (b) Trapezoidal $< 0°, 20°, 0° >$ (c) Kite $< 0°, 20°, 20° >$ (d) Irregular quadrilateral $< 0°, 50°, 10° >$ (e) Irregular quadrilateral $< 30°, 50°, 10° >$

Figure 3: Aerial-FOVs with various azimuth, pitch and roll angles. The vector is rotation angle vector $< \theta_a, \theta_p, \theta_r >$. Dashed areas denote the *"dead spaces"*.

2.2 Spatial Queries on Aerial-FOVs

For aerial-FOVs, we introduce two types of spatial queries: point and range queries. As illustrated in Fig. 2, 1) a point query Q_p finds aerial-FOVs that cover the query point; and 2) a range query Q_r finds aerial-FOVs that overlap with the query circle.

As we represent an aerial video database $\mathcal{V}$ as an aerial-FOV database $\mathcal{F}$, the problem of aerial video search is transformed into spatial queries on the aerial-FOV database $\mathcal{F}$. Given a query point $Q_p(q)$, a point query finds aerial-FOVs that cover the query point Q_p. It is formally defined as:

$$PointQ(Q_p, \mathcal{F}) \Longleftrightarrow \{f \in \mathcal{F} | Q_p \in f\} \quad (1)$$

Given a query circle $Q_r(q, r)$ with the center point q and the radius r, a range query finds aerial-FOVs that overlap with Q_r. It is formally defined as:

$$RangeQ(Q_r, \mathcal{F}) \Longleftrightarrow \{f \in \mathcal{F} | f \cap Q_r \neq \emptyset\} \quad (2)$$

3 BASELINE METHODS

3.1 R-trees

One baseline for indexing aerial-FOVs is using R-tree [7], which is one of the basic and widely used spatial index structures. To index aerial-FOVs efficiently with R-tree, we enclose each aerial-FOV (i.e., quadrilateral) with its MBR, as illustrated in Fig. 3. Given a point query Q_p (resp. a range query Q_r), for each visited index node N, R-tree decides whether to access the subtree of N through checking whether Q_p is covered by (resp. Q_r overlaps with) the MBR of N or not. Hence large "**dead spaces**" (i.e., empty spaces that are covered by the MBR of an R-tree node but not overlap with any objects under its subtree [7]) will produce many unnecessary index node accesses. R-tree will produce large dead spaces for indexing aerial-FOVs. For a ground-FOV, the dead space area is proved to be at most half of its MBR [17]. However, for an aerial-FOV, the dead space area ratio can be larger than 50% (e.g., Figures 3d and 3e).

3.2 OR-trees

Another baseline is extending OR-tree [16, 17], which is the state-of-the-art index for ground videos based on the spatial coverages.

We first review the spatial coverage model for ground videos. Each ground video frame is modeled as a field of view [5], referred as "ground-FOV", as shown in Fig. 4. In this paper, both aerial-FOVs and ground-FOVs are collectively called FOVs. A ground-FOV f is denoted as $(lat, lng, \theta, \alpha, R)$, where $<lat, lng>$ is the camera location, θ is the viewing orientation, α is the viewable angle, and R is the visible distance. During video recording using sensor-rich devices, for each video frame f, we can collect the camera location from GPS, and capture the camera view direction θ from the compass sensor automatically. Further, according to the camera lens properties and zoom level, we can calculate the viewable angle α of f [5].

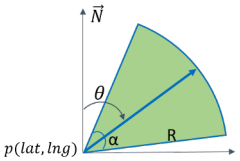


Figure 4: Ground-FOV model [5]

The main idea of OR-tree is indexing the camera locations, directions and visible distances of ground-FOVs individually. The straightforward extension of OR-tree for aerial-FOVs is indexing each dimension of the 7-tuple $\Gamma(lat, lng, hgt, \theta_a, \theta_p, \theta_r, \alpha)$, separately. Each index node includes a set of entries in the form of $(MBCube, MBOri, MinMax\alpha)$, where $MBCube = \{minLat, maxLat, minLng, maxLng, minHgt, maxHgt\}$ is the minimum bounding cube of the camera locations of the subtree aerial-FOVs, $MBOri = \{min\theta_a, max\theta_a, min\theta_p, max\theta_p, min\theta_r, man\theta_r\}$ is the minimum bounding viewing orientation of subtree aerial-FOVs, and $MinMax\alpha$ is the minimum and maximum visible angles of the subtree aerial-FOVs.

However, indexing aerial-FOVs with OR-trees results in poor performance for point and range queries due to low effectiveness of OR-tree's pruning strategy. The reason behind this is not trivial and we explain below in details. Different from ground-FOVs which are regular pie-shaped, an aerial-FOV is a quadrilateral with an arbitrary shape. It is complicated and computationally expensive to determine whether a query point (or a query range) is covered by (or overlap with) an aerial-FOV using its 7-tuple information. For example, for the aerial-FOV f in Fig. 2, to calculate its four corner points (A, B, C, D), we first need to compute four points (A', B', C', D') assuming the drone camera shooting vertically towards the ground, then rotate each of them based on the rotation angles $(\theta_a, \theta_p, \theta_r)$, and then project them on the ground. In order to decide whether the aerial-FOV f is covered by the point query $Q_p(q)$ (q is the query point), we need to examine whether the four angles of the vector $\overrightarrow{pq}$ to the four planes (i.e., pAB, pBC, pCD and pAD) are within the visible angle α. Hence, it is also complicated and computationally expensive to decide whether a point query is covered by an OR-tree node N or not using its bounding information $(MBCube, MBOri, MinMax\alpha)$.

An alternative spatial index for ground-FOVs is Grid-based Index, termed Grid [18]. Grid is a three-level grid-based index indexing ground-FOVs' viewable scenes, camera locations and view directions individually at different levels. However, experimental results in the study [17] showed that Grid performed worse than OR-trees as it stores ground-FOVs's information (e.g., locations and orientations) at different levels. Additionally, Grid performs poorly for skewed distribution of FOVs since the bucket occupancy of grid files rises very steeply for skewed distribution [10]. Therefore, we do not refer to Grid as our baseline for indexing aerial-FOVs.

4 TETRAR-TREE

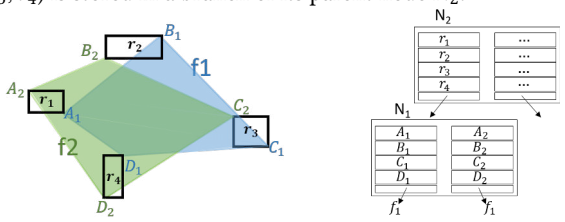
To overcome the drawbacks of R-tree [7] and OR-tree [17], we propose a new index structure for aerial-FOVs, called TetraR-tree. Unlike OR-tree that stores the sensor data (i.e., the 7-tuple $\Gamma(lat, lng, hgt, \theta_a, \theta_p, \theta_r, \alpha)$) directly for an aerial-FOV object f, TetraR-tree stores its corresponding quadrilateral.

In particular, for the leaf index nodes of a TetraR-tree, instead of storing the MBRs of aerial-FOVs like R-tree, we store the actual aerial-FOVs (or quadrilaterals). As such, each leaf index node N of a TetraR-tree contains a set of entries in the form of $(Oid, \mathcal{P})$, where Oid is the pointer to an aerial-FOV in the database; $\mathcal{P}$ is the set of four vertices of the quadrilateral. Each internal TetraR-tree node N contains a set of entries in the form of $(Ptr, \mathcal{R})$, where

- Ptr is the pointer to a child index node of N;
- $\mathcal{R}$ is a set of four corner-MBRs (aka, "tetra-corner-MBRs") in the form of $\{r_i | i \in [1,4]\}$. The four corner points of each quadrilateral $\mathcal{P}$ in the subtree rooted at Ptr are in the four corner-MBRs individually (see the definition in Eqn(3)).

$$N.\mathcal{R} = \{r_i \mid \forall \mathcal{P}\{p_1, p_2, p_3, p_4\} \in subtree(N), p_i \in r_i, i \in [1,4]\} \quad (3)$$

Fig. 5 shows an example of a TetraR-tree leaf node N_1 that includes two aerial-FOVs f_1 and f_2 and its tetra-corner-MBRs (r_1, r_2, r_3, r_4) is stored in a branch of its parent node N_2.



(a) aerial-FOV distribution (b) TetraR-tree node information

Figure 5: A TetraR-tree index node including two aerial-FOVs f_1 and f_2

While the framework of the TetraR-tree construction algorithm (Algorithm 1) is similar to that of the conventional R-tree, the optimization criteria in the procedures ChooseLeaf and Split are different. The procedure ChooseLeaf is to choose a leaf node to store a newly inserted aerial-FOV object. ChooseLeaf traverses the TetraR-tree from the root to a leaf node. When it visits an internal node N in the tree, it will choose the entry E of N with the least `Waste`, to be given in Eqn(5). The procedure Split is to split an index node N into two nodes when N overflows. We use the standard Quadratic Split algorithm [7] based on a new `Waste` function. We proceed to present our newly proposed optimization criteria `Waste` for TetraR-tree with four corner-MBRs in the index nodes.

During the index construction, one method is to follow the optimization heuristic of R-tree to minimize the enlarged area of the MBRs (i.e., the full coverages) of aerial-FOVs, and we refer to this method as "fullCoverage-based optimization". As shown in Fig. 6a, where $B\{b_1, b_2, b_3, b_4\}$ and $R\{r_1, r_2, r_3, r_4\}$ are two TetraR-tree index nodes, the fullCoverage-based optimization will minimize the enlarged area (the gray area) of the MBRs of B and R. However, this optimization cannot effectively group the TetraR-tree index nodes /objects together which have the similar viewable coverages. For example, in Fig. 6b, the two aerial-FOVs $R\{r_1, r_2, r_3, r_4\}$ and $G\{g_1, g_2, g_3, g_4\}$ have the same MBRs but their quadrilaterals have few overlap. However, R overlaps much more with $B\{b_1, b_2, b_3, b_4\}$

Algorithm 1: Insert (R: an old TetraR-tree, E: a new entry)

Output: The new TetraR-tree R after inserting E
1 $N \leftarrow$ ChooseLeaf(R, E);
2 Add E to node N;
3 **if** N *needs to be split* **then**
4 $\{N, N'\} \leftarrow$ Split(N, E);
5 **if** $N.isroot()$ **then**
6 Initialize a new node M;
7 M.append(N); M.append(N');
8 Store nodes M and N'; /* N is already stored */
9 $R.RootNode \leftarrow M$;
10 **else** AdjustTree($N.ParentNode$, N, N');
11 **else**
12 Store node N;
13 **if** $\neg N.isroot()$ **then** AdjustTree($N.ParentNode$, N, null) ;
Procedure ChooseLeaf(R, E)
14 $N \leftarrow R.RootNode$;
15 **while** N *is not leaf* **do**
16 $E' = \underset{E_i \in N}{argmax}\ Waste_{align}(E_i, E)$; /* Eqn(5) */
17 $N \leftarrow E'.Ptr$;
18 **return** N;
Procedure Split(N, E)
19 $E_1, E_2 = \underset{E_i, E_j \in N \cup \{E\}}{argmin}\ Waste_{align}(E_i, E_j)$; /* Eqn(5) */
20 **for** *each entry* E' *in* $N \cup \{E\}$, *where* $E' \neq E_1$, $E' \neq E_2$ **do**
21 **if** $Waste_{align}(E', E_1) \geq Waste_{align}(E', E_2)$ **then** Classify E' as Group 1 ;
22 **else** Classify E' as Group 2;
23 **return** Group 1 and Group 2;

although their MBRs are different. Consequently, we proceed to introduce a new optimization criteria, "alignment-based" waste, which considers the individual alignment wastes of the four corners of quadrilaterals. The intuition is that TetraR-tree index nodes / objects whose corner-MBRs / corner points are closer to each other have a higher probability to have the similar viewable coverages.

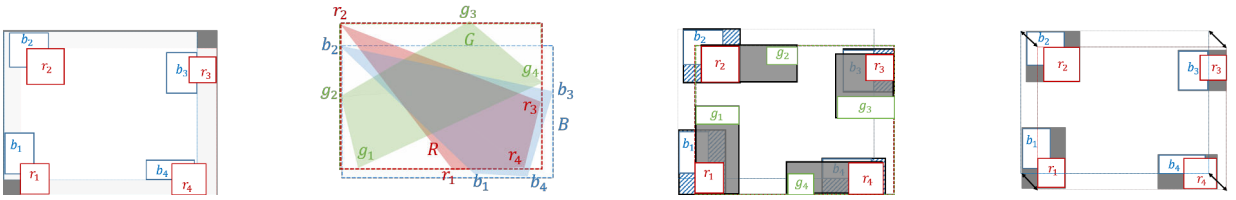
Given a TetraR-tree entry E with the tetra-corner-MBRs $\mathcal{R}\{r_i | i \in [1,4]\}$ and an aerial-FOV f with the quadrilateral $\mathcal{P}\{p_j | j \in [1,4]\}$, let $\Delta Area(r, p)$ be the enlarged space after enclosing a quadrilateral vertex p in f with an MBR r in E. The definition of $\Delta Area(r, p)$ is formulated in Eqn(4). And the alignment-based waste of inserting f into E, denoted by $Waste_{align}(E, f)$, is defined in Eqn(5). In Fig. 6c, the gray area is the alignment-based waste of the two nodes $R\{r_1, r_2, r_3, r_4\}$ and $G\{g_1, g_2, g_3, g_4\}$, and the blue dashed area is the alignment-based waste of R and the index node $B\{b_1, b_2, b_3, b_4\}$. Clearly $Waste_{align}(R, B) < Waste_{align}(R, G)$, thus with the alignment-based optimization, the index node R will be grouped together with B instead of G, assuming the fanout is 2.

$$\Delta Area(r, p) = Area(MBR(r, p)) - Area(r) \quad (4)$$

$$Waste_{align}(E, f) = \sum_{r \in E, p \in f} \Delta Area(r, p) \quad (5)$$

As there are four corner-MBRs in an index node E and four vertices in a quadrilateral f, there are $4! = 24$ permutations (i.e., compute the alignment-based waste 24 times). The exhaustive approach to optimize the alignment-based waste is enumerate all the possible combinations and select the one with the minimum waste. This approach, named exhAlignment, can find the minimum alignment-based waste but it is too expensive.

To reduce the index construction time, we propose a heuristic approach, named heurAlignment, to find the *near-optimal* alignment-based waste efficiently. Specifically, heurAlignment first calculates



(a) fullCoverage-based (b) fullCoverage-based (a bad case) (c) fullCoverage vs Alignment (d) heurAlignment-based

Figure 6: Various optimization mechanisms during TetraR-tree construction.

the minimum bounding rectangles $MBR(B)$, $MBR(R)$ for two TetraR-tree entries B and R. Then it aligns the four corners (i.e., upper-left, lower-left, lower-right and upper-right) of $MBR(B)$ and $MBR(R)$, as shown in Fig. 6d. Finally, for each corner, match the two corner-MBRs $b \in B$ and $r \in R$ that have not been matched and are closest to the corner among B and R, respectively. With this heuristic, heurAlignment calculates the alignment-based waste only once.

The deletion procedure of TetraR-tree is similar to that of the conventional R-tree. Except that the optimization criteria in the R-tree adjustment and merge are replaced with the new criteria we proposed for TetraR-tree.

5 QUERY PROCESSING

We proceed to present the query processing algorithms for point queries and range queries based on TetraR-tree.

5.1 Point Query on Aerial-FOVs

In this section, we develop an efficient algorithm to answer point queries. At the high-level, the algorithm descends from the TetraR-tree root in a branch-and-bound manner, progressively checking whether each visited aerial-FOV object / index node covers the query point. Subsequently, the key problem is to decide whether to prune an aerial-FOV object / index node, or to report the aerial-FOV / index node (all the aerial-FOVs in the node) to be result(s). In the following, before presenting the search algorithm, we first present an approach to identify whether an aerial-FOV covers the query point, and then exploit it to identify whether a TetraR-tree index node should be visited or not through two newly proposed search strategies: 1) pruning strategy and 2) total hit strategy.

5.1.1 Search Strategies for Point Queries. For an aerial-FOV object, we can apply the *even-odd rule* algorithm [24] described in Lemma 5.2 – a typical point-in-polygon algorithm – to identify whether the query point q is inside the aerial-FOV or not. Its complexity is $O(e)$, where e is the number of edges of the polygon.

Lemma 5.1 (even-odd rule [24]). *Given a polygon poly and a point q, let the* ***crossing number*** *cn(q, poly) be the times a ray starting from the point q with any fixed direction crosses the polygon poly boundary edges. It claims that q is inside poly if and only if the crossing number cn(q, poly) is odd, or outside if it is even.*

Lemma 5.2 (Point Query for an Aerial-FOV Object). *Given an aerial-FOV $f(\mathcal{P})$ and a query point q, q is inside of f if and only if the crossing number $cn(q, f)$ is odd, or outside if it is even.*

Proof. Lemma 5.2 is clearly true according to Lemma 5.1 as aerial-FOVs are quadrilateral-shaped, convex polygons. □

For example, in Fig. 7, query point q_1 is inside of the aerial-FOV $f\{A, B, C, D\}$ as its crossing number $cn(q_1, f)$ is odd whereas q_2 is outside as $cn(q_2, f)$ is even.

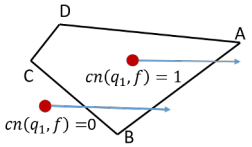


Figure 7: Even-odd rule algorithm for point query

Based on the lemmas above, we can develop two search strategies. In order to prune a TetraR-tree index node N or to report N to be result(s) without accessing the objects under the subtree of N, we first introduce the following several concepts.

Definition 5.3 (Possible Quadrilateral). Given a TetraR-tree node N, we call the **possible quadrilaterals** of N as a set of *abstract* quadrilaterals whose four vertices are respectively from the four corner-MBRs of N, i.e., $\{\mathcal{P}(p_1, p_2, p_3, p_4) \mid p_i \in N.MBR_i, i \in [1, 4]\}$.

Definition 5.4 (Outer Convex Hull). Given a TetraR-tree node N, we define the **outer convex hull** of N, denoted by $outerCH(N)$, as the smallest convex hull that encloses all the possible quadrilaterals of N, i.e., the union of all the possible quadrilaterals.

Definition 5.5 (Inner Convex Hull). Given a TetraR-tree node N, we define the **inner convex hull** of N, denoted by $innerCH(N)$, as the largest convex hull that is covered by all the possible quadrilaterals of N, i.e., the intersection of all the possible quadrilaterals.

The red convex hull in Fig. 8 illustrates the outer convex hull of a TetraR-tree index node N and the red convex hull in Fig. 9 shows its inner convex hull. It is non-trivial to efficiently compute the outer and inner convex hulls, and this will be covered in Sec. 5.1.3.

Based on the outer / inner convex hull definitions above, we propose two novel search strategies: pruning strategy in Lemma 5.6 and total hit strategy in Lemma 5.7, to examine whether an index node N is a result or not, without accessing the aerial-FOV objects in the subtree of N. For example, in Fig. 8, we can prune the index node N, which includes two aerial-FOVs f_1 and f_2, as the query point Q_p is outside of its outer convex hull. Note that, the query point is within the dead space (i.e., the dashed area) of the MBR of the two aerial-FOVs, and thus the index node needs to be accessed if we were indexing with an R-tree. We call an index node N a "total hit" if and only if all the objects under the subtree of N are results. If an index node N is a "total hit", then it is unnecessary to exhaustively check all the objects in N, so the processing cost can be significantly reduced. In Fig. 9, as the query point is inside of an index node N, we can claim N is a "total hit" and safely report both f_1 and f_2 to be results without checking whether they are results or not individually.

Lemma 5.6 (Pruning Strategy for Point Query). *Given a TetraR-tree index node N and a point query $Q_p(q)$, the index node N can be pruned if the query point q is outside of $outerCH(N)$.*

Proof. Lemma 5.6 holds as $outerCH(N)$ is the union of all the possible quadrilaterals of N. □

Lemma 5.7 (Hit Reporting Strategy for Point Query). *Given a TetraR-tree index node N and a point query $Q_p(q)$, we can report all the aerial-FOVs under the subtree of N are results if the query point q is inside of $innerCH(N)$.*

Proof. Lemma 5.7 holds as $innerCH(N)$ is the intersection of all the possible quadrilaterals of N. □

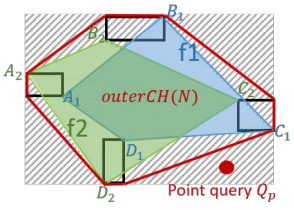


Figure 8: Pruning strategy with outer convex hull for point query.

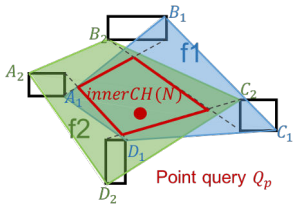


Figure 9: Total hit strategy with inner convex hull for point query.

5.1.2 Search Algorithm for Point Queries. Based on the two new search strategies, we develop an efficient algorithm to answer point queries (see Algorithm 2). The overview of the algorithm is to descend the TetraR-tree in a branch-and-bound manner, progressively applying the two strategies to answer the point query.

Algorithm 2: Point Query (T:TetraR-tree root, Q_p:query point)

```
   Output: All the aerial-FOV objects f ∈ results
1  Initialize a stack S with the root of the TetraR-tree T;
2  while ¬S.isEmpty() do
3      N ← S.pop();
4      outerCH ← getOuterCH(N);
5      if cn(Q_p, outerCH) is even then Prune N; /*Lemmas 5.1 and 5.6  */
6      else
7          innerCH ← getInnerCH(N);
8          if cn(Q_p, innerCH) is odd then
9              results.add(N.subtree()); /* Lemmas 5.1 and 5.7        */
10         else
11             for each child node ChildN of N do
12                 S.push(ChildN);

   Procedure getOuterCH(N)
13 P ← The outer-convex-dominate points of N; /* Lemma 5.9            */
14 return convexHull(P); /* Andrew's Monotone Chain Algorithm [3]      */
   Procedure getInnerCH(N)
15 if ∀r1, r2 ∈ {MBR_UL, MBR_LL, MBR_LR, MBR_UR}, r1 ∩ r2 ≠ ∅ then
16     return CH_UL ∩ CH_LL ∩ CH_LR ∩ CH_UR; /* Lemma 5.11              */
```

5.1.3 Convex Hull Computing. We proceed to discuss the geometric challenges in the computation of the outer and inner convex hulls. We utilize a study [20] that shows the calculation of outer convex hull for a set of regions (i.e., rectangles) can be transferred to the well-known convex hull problem [3, 25] on points, as described in Lemma 5.8. As shown in Fig. 10, to compute the outer convex hull of the tetra-corner-MBRs $\{r_1, r_2, r_3, r_4\}$, we compute the convex hull of the 16 vertices $\{r_i(p_j)|1 \le i \le 4, 1 \le j \le 4\}$. Andrew's Monotone Chain Algorithm [3], a typical solution to the convex hull problem, can compute the convex hull for a set of points with a complexity of $O(n \log n)$, where n is the number of points. To reduce the processing cost, we can reduce the number of points by removing the points that are unnecessary to be processed. Different from the traditional convex hull problem where points could be randomly distributed, in our tetra-corner-MBRs, points are from up-righted MBRs. Based on this property, we develop Lemma 5.9 to remove the "dominated points" of a TetraR-tree index node N (e.g., the black corner points in Fig. 10) to accelerate the outer convex hull computation of N. Lines 13 – 14 in Algorithm 2 present the pseudocode of the outer convex hull computation.

Theorem 5.8 (Outer Convex Hull Computing [20]). *Given a set of n rectangles $\mathcal{R}\{r_i\{p_1, p_2, p_3, p_4\}|1 \le i \le n\}$, the outer convex hull of $\mathcal{R}$ is the convex hull of all the corner points of all the rectangles, i.e., $\{r_i(p_j)|1 \le i \le 4, 1 \le j \le n\}$.*

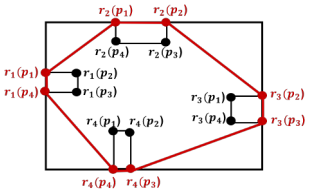


Figure 10: Illustration of the outer convex hull computing

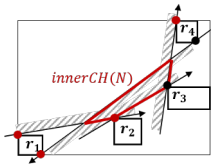


Figure 11: Inner convex hull: not constructed from non-dominated corner points

Lemma 5.9 (outer convex dominate points). *Given a TetraR-tree index node N with tetra-corner-MBRs $\mathcal{R}$, we say the corner points of a corner-MBR r in $\mathcal{R}$ are **boundary corner points** if they are on the boundary of the minimum bounding rectangle that encloses the four corner-MBRs $\mathcal{R}$, say r is a **boundary corner-MBR**, and suppose the other corner points of r are **dominated corner points**. We can conclude that the outer convex hull of N is the convex hull of its non-dominated corner points, i.e., the union of boundary corner points and the corner points of non-boundary corner-MBRs.*

Proof. For a boundary corner-MBR r in $\mathcal{R}$, its dominated corner points p will be "dominated" by its boundary corner points p', i.e., p is inside of the convex hull that passes p'; otherwise p will be the boundary of r. Hence Lemma 5.9 holds. □

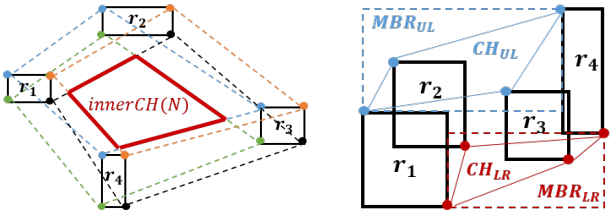
The calculation of the inner convex hull for a TetraR-tree index node N is more complicated as the vertices forming the inner convex hull are not necessary from the tetra-corner-MBRs of N and it is not straightforwardly constructed from the non-dominated corner points (see Fig. 11). The existing solution [20] calculates the inner convex hull through computing the intersection of four convex hulls, as described in Theorem 5.10. For example, in Fig. 12a, the inner convex hull is the intersection of the four convex hulls CH_{UL} (in blue), CH_{LL} (in green), CH_{LR} (in black) and CH_{UR} (in orange).

However, in some cases, there is no inner convex hull for a TetraR-tree node (e.g., Fig. 12a). To avoid the expensive computation for a non-existing inner convex hull, we propose a novel technique given in Lemma 5.11 to identify the case where no inner convex hull exists. For example, in Fig. 12a, there is no inner convex hull as the two minimum bounding rectangles MBR_{UL} (in blue) and MBR_{LR} (in red) do not overlap. The pseudocode of computing the inner convex hull is given in Lines 15 – 16 in Algorithm 2.

Theorem 5.10 (Inner Convex Computation [20]). *Given a set of n rectangles $\mathcal{R}$, let CH_{UL} (resp. CH_{LL}, CH_{LR}, CH_{UR}) be the convex hull of the the upper-left (resp. lower-left, lower-right, upper-right) vertices of all the rectangles in $\mathcal{R}$. The inner convex hull of $\mathcal{R}$ is the intersection of the four convex hulls CH_{UL}, CH_{LL}, CH_{LR} and CH_{UR}.*

Lemma 5.11 (Inner Convex Hull Pruning). *Given a set of n rectangles $\mathcal{R}$, let MBR_{UL} (resp. MBR_{LL}, MBR_{LR}, MBR_{UR}) be the minimum bounding rectangle that encloses the the upper-left (resp. lower-left, lower-right, upper-right) vertices of all the rectangles in $\mathcal{R}$. Among the four corner-MBRs, if there are two non-overlapping MBRs, then the inner convex hull of $\mathcal{R}$ does not exist.*

Proof. As the four convex hulls are enclosed in their corresponding MBRs (e.g., $CH_{UL} \subseteq MBR_{UL}$), if their MBRs do not overlap, then the convex hulls do not overlap, and thus Lemma 5.11 holds. □



(a) The inner convex consists of four convex hulls

(b) An example where no inner convex hull exists

Figure 12: Illustration on inner convex computation.

5.2 Range Query on Aerial-FOVs

We next present an efficient algorithm for processing range queries with TetraR-tree. The basic idea to answer range queries on aerial-FOVs is to transform the range query problem to the point query problem by enlarging the quadrilaterals of aerial-FOVs with the radius of the range query. As shown in Fig. 13, given a quadrilateral (or a polygon) $P\{A, B, C, D\}$, we denote *enlargedCov* as the convex after enlarging each edge (and vertex) of P with a radius r, denote *expanedPoly* as the polygon by expanding each edge of *enlargedCov* to enclose all of its rounded corners, and denote *trimmedPoly* as the polygon by trimming off the round corners of *enlargedCov*. Algorithm 3 depicts how to check whether a range query $Q_r(q, r)$ overlaps a quadrilateral P (i.e., an aerial-FOV or a polygon) or not. Specifically, we first check if the query center q is inside of *expandedPoly* (Line 2). If not, then return false. Otherwise we calculate *trimmedPoly* to see whether it covers q (Lines 4–5). If *trimmedPoly* covers q, then return true. Otherwise we perform the final check: examine the vertices p of the polygon P to see if one of them is within the query range (Line 7–8).

Similar to point queries, we can also design pruning and total hit strategies for range queries. To decide whether a TetraR-tree index node N overlaps with a range query $Q_r(q, r)$, we calculate the outer and inner convex hulls of N, *outerCH*(N) and *innerCH*(N), and then prune N if Q_r does not overlap with *outerCH*(N) and report N as a "total hit" if Q_r overlaps with *innerCH*(N) (use Algorithm 3). To process a range query with a TetraR-tree, we descend the TetraR-tree in the branch-and-bound manner, and repeatedly apply each of the search strategies to answer the range query.

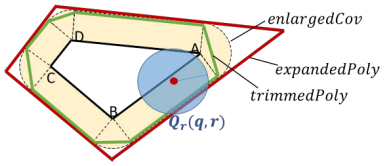


Figure 13: Illustration on how to transform a range query to a point query

Algorithm 3: Range overlap (P: a polygon, $Q_r(q, r)$: range query with query center q and radius r)

```
1 expandedPoly ← expanded convex of enlargedCov(P) ;
2 if cn (q, expandedPoly) is even then return false ;
3 else
4     trimmedPoly ← trimmed convex of enlargedCov(P) ;
5     if cn (q, trimmedPoly) is odd then return true ;
6     else
7         for each vertex p of the polygon P do
8             if EucliDist(q, p)>r then return false ;
9         return true
```

6 EXPERIMENTAL STUDIES

We conducted an experiments to evaluate the efficiency of our methods using two fundamental queries: point and range queries.

6.1 Experimental Methodology and Settings

Implemented Indexes. We implemented our proposed indexes and search algorithms: TetraR-tree with the full-coverage-based optimization (TetraR-fullCover), TetraR-tree with the exhaustive alignment-based optimization (TetraR-exhAlign), and TetraR-tree with the alignment-based heuristic (TetraR-heurAlign) for both point and range queries. In addition, we implemented two baselines for comparison: R-tree [7] and OR-tree [17]. For all these indexes each index node stored in one disk page. As TetraR-tree stores three MBRs more than R-tree, their fanouts are around three times less than that of R-tree. Therefore, we also implemented TetraR-heurAlign where the size of an index node is equal to four disk pages to allow it to have the same fanout as R-tree, and we refer to such index as TetraR-heurAlign*.

Datasets. We used two types of datasets: Real World (*RW*) [22] and Synthetically Generated (*Gen*) dataset as shown in Table 1. *RW* includes aerial videos that was recorded by Autonomous System Lab at Swiss Federal Institute of Technology Zurich with a solar-powered UAV (AtlantikSolar[4]). To evaluate the scalability of our solutions, we synthetically generated five *Gen* datasets with different data sizes in log scale from 0.1M (million) to 1B (billion) aerial-FOVs, see Table 2, using the mobile video generation algorithm [6]. This process simulates a situation where aerial-FOVs were uniformly distributed across a $16km \times 16km$ area around Los Angeles Downtown, assuming several drones being distributed in the area initially and then flying randomly at a speed of around 15 km/h and recording videos with cameras mounted to the drones. Unless specified, the dataset size of 10M aerial-FOVs is assumed in the reported experimental results.

Queries. For point queries, we randomly generated 10,000 query points within the dataset space. For range queries, we generated 5 query sets for each dataset (*RW* and *Gen*) by increasing the query radius from 0.2 to 1.0 km (kilometer) incremented by 0.2 km, and

[4]http://www.atlantiksolar.ethz.ch/

Table 1: Datasets

Statistics	*RW*	*Gen*
total # of aerial-FOVs	0.6 M	0.1 M~1 B
aerial-FOV# per second	0.2	1
total play time made by all the aerial-FOVs	81 hours	27.8 hours ~ 31.71 years
average camera moving speed (km/h)	35	15
average flying altitude (meter)	400	100
total covered area (km^2)	2.56	256

Table 2: Synthetically Generated *Gen* Datasets

aerial-FOV#	0.1 M	1 M	10 M	100 M	1 B
aerial-video#	100	1 K	10 K	100 K	1 M
total play time	27.8 hours	11.6 days	115.7 days	3.2 years	31.7 years

we use the query radius of 0.6 km by default. This is a reasonable variation range for aerial video queries since users are usually interested in videos in a small specified area (e.g., a building with 0.05 km radius, a college campus with 0.8 km radius). Each range query set contained 10,000 queries with the same query radius but the query centers are randomly distributed in the dataset area.

Setup and metrics. We implemented all the indexes on a server with Intel(R) Xeon(R) CPU E3 @3.50GHz, 6GB of RAM, 2000GB of hard disk, and used the page size of 4KB. We evaluated their performance based on disk-resident data. For the evaluation metrics, we report the average query time (end-to-end) and the average I/O cost (measured with $IOreduction$ in Eqn(6)) per query after executing 10,000 queries for each set. We measure the I/O cost of an index $\mathcal{A}$ with the reduction percentage over R-tree given Eqn(6), in which, $IOusage$ is the disk I/O usage (i.e., the actual number of disk reads and writes per second) monitored with the Linux iotop tool[5], which can provide the I/O usage for a specific process. Larger $IOreduction$ suggests more I/O cost reduction over R-tree.

$$IOreduction(\mathcal{A}) = \left(1 - \frac{IOusage(\mathcal{A}) * processTime(\mathcal{A})}{IOusage(\text{R-tree}) * processTime(\text{R-tree})}\right) * 100\% \quad (6)$$

6.2 Experimental results

6.2.1 Performance of Index Construction. The construction performance and space usage of the index structures for *RW* and *Gen* (1B) are reported in Tables 3 and 4. The TetraR-tree family requires about 2 times more space than R-tree and 30% more than OR-tree since TetraR-tree needs more space (four MBRs) in each index node, resulting in higher I/O costs than both R-tree and OR-tree. Moreover, the index construction process of TetraR-tree is more CPU intensive and hence it takes slightly longer than that of R-tree. However, if TetraR-exhAlign given that it is an exhaustive approach to optimize the index by enumerating all the possible combinations, the index construction time takes even longer, about 4 times more than R-tree and 3 times more than OR-tree. We also observe that TetraR-heurAlign* needs at most 30% more for its construction time than TetraR-heurAlign as more branches need to be processed for optimizing each index node even though it has less index nodes.

Table 3: Performance of indexing structures on *RW* Dataset

	R-tree	OR-tree	TetraR-tree			
			fullCover	exhAlign	heurAlign	heurAlign*
Fanout	203	71	53	53	53	203
Tree level#	3	4	4	4	4	3
Size (MB)	21.00	43.89	55.26	54.12	53.26	50.06
I/O cost	X	1.589X	2.117X	2.015X	1.919X	1.827X
Construct time (min)	1.07	1.24	1.13	5.13	1.30	1.69

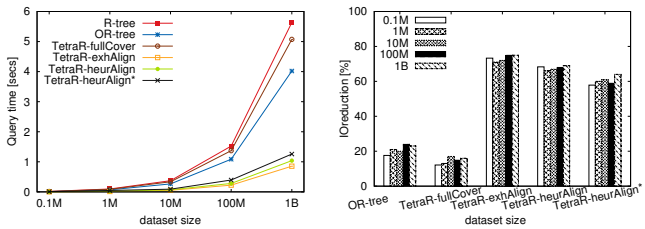
[5]https://linux.die.net/man/1/iotop

Table 4: Performance of indexing structures on *Gen* Dataset (1B)

	R-tree	OR-tree	TetraR-tree			
			fullCover	exhAlign	heurAlign	heurAlign*
Fanout	203	71	53	53	53	203
Tree level#	5	6	6	6	6	5
Size (GB)	22.98	40.23	60.04	58.01	57.30	53.79
I/O cost	X	1.657X	2.068X	2.254X	2.189X	1.929X
Construct time (hour)	3.47	4.87	4.26	17.09	5.38	6.97

6.2.2 Evaluation for Point Queries. In this set of experiments, we evaluated the performance of the indexes for point queries using the *Gen* dataset by varying the dataset size from 0.1 M to 1B. As shown in Fig. 14a, we can observe that: 1) all the TetraR-trees with the alignment-based optimization (i.e., TetraR-exhAlign, TetraR-heurAlign and TetraR-heurAlign*) significantly outperformed both R-tree and OR-tree in terms of query time. For example, TetraR-heurAlign took 80% less query time than R-tree and 70% less than OR-tree. This demonstrates the superiority of our alignment-based optimization in TetraR-tree. 2) TetraR-fullCover took less time than R-tree but more time than OR-tree as the fullCoverage-based optimization cannot optimize the actual waste of aerial-FOVs in a TetraR-tree index node. Thus, it performed worse than OR-tree which combines camera locations and viewing orientations [17]. TetraR-fullCover performed better than R-tree due to our proposed novel search strategies though they are based on the same optimization criteria. 3) Among all the indexes, TetraR-exhAlign performed the best for point queries as it can find the exact alignment-based waste. However, as mentioned before, it is too time consuming for the index construction. With the alignment-based heuristic, TetraR-heurAlign reduced the index construction time significantly (more than 65% reduction) and did not lose much (less than 20%) efficiency for the query processing. 4) TetraR-heurAlign* was marginally worse (took less than 20% query time) than TetraR-heurAlign since the outer and inner convex hulls of an index node will be slightly looser (i.e., more dead spaces) given more aerial-FOVs in the node. In addition, their construction times were also competitive, and thus it demonstrates that the performance of TetraR-tree is not very sensitive to the number of disk pages in an index node. Therefore, we omit TetraR-heurAlign* in the rest of the experiments.

Fig. 14b reports the average I/O cost reduction $IOreduction$ over R-tree. The overall performance improvement showed a similar trend as Fig. 14a. TetraR-tree with the alignment-based optimization provided at least 60% on I/O cost reduction as compared to R-tree. Additionally, one can observe that the I/O cost reductions of all indexes over R-tree are comparable for different the dataset sizes.



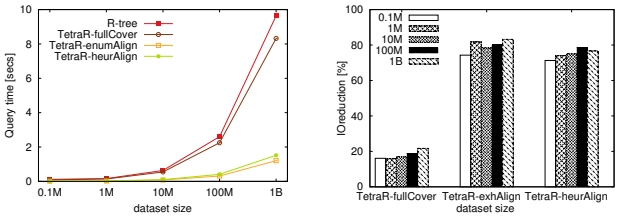
(a) Query time (b) I/O cost reduction

Figure 14: Point query on *Gen* Dataset

6.2.3 Evaluation for Range Queries. In this set of experiments we evaluated the performance of the indexes for range queries using the *Gen* dataset. We did not implement OR-tree for range queries since determining whether a query range overlaps with an OR-tree node,

in which all the aerial-FOVs are irregular quadrilateral-shaped, is too complicated and computationally expensive.

As depicted in Fig. 15, the overall performances of TetraR-exhAlign and TetraR-heurAlign were significant and similar to the results of point queries. Specifically, TetraR-heurAlign (resp. TetraR-exhAlign) took 84% (resp. 87%) less query time than R-tree and resulted in 75% (resp. 80%) I/O cost reduction over R-tree. Note that on the 1B aerial-FOVs (over 30 years' worth of aerial videos), our TetraR-trees with the alignment-based optimization took less than 2 seconds to answer range queries, which makes the corresponding application feels more interactive as compared to 8 – 10 seconds required by the other approaches.



(a) Query time (b) I/O cost reduction

Figure 15: Range query on *Gen* Dataset

Fig. 16 reports the impact of the query radius by varying the radius from 0.2km to 1.0km. It is obvious that the performances of TetraR-exhAlign and TetraR-heurAlign stayed relatively the same as the radius increase while those of R-tree and TetraR-fullCover worsen as the query radius grew. This is because the total hit strategy applied in our indexes can safely report all the objects in a TetraR-tree index node to be results if the node is a "total hit" without exhaustively checking all the objects in the node one by one, and thus yields a significant reduction of processing time even when the query radius grew (i.e., more query results).

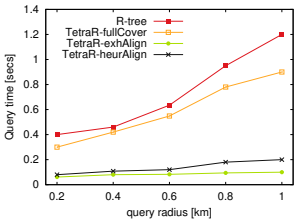
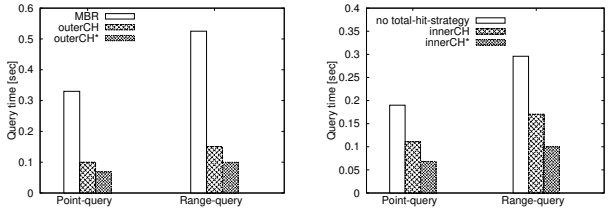


Figure 16: Vary query radius for range queries on *Gen* Dataset

6.2.4 Evaluation of Search Strategies. This set of experiments evaluate the effects of our proposed search strategies: pruning and total hit strategies. Recall that we applied the outer convex hull for the pruning strategy and applied inner convex hull for the total hit strategy. As introduced in Sec. 5.1.3, based on the existing outer convex hull computing method (outerCH) [20], we proposed an efficient approach (outerCH*) by filtering the dominated corner points. Based on the existing solution (innerCH) [20] to the inner convex hull computing, we proposed an optimized approach (innerCH*) by identifying non-existing inner convex hulls.

Fig. 17a shows the query times of TetraR-heurAlign by applying different pruning strategies (i.e., MBR, outerCH and outerCH*) for both point and range queries. Similar to R-tree, MBR-pruning uses the minimum bounding rectangle of the four corner-MBRs to filter out irrelevant objects or index nodes. In this set of experiments, the innerCH*-based total hit strategy is applied. Fig. 17a shows that with outerCH, the query time of TetraR-heurAlign was reduced significantly (70%) for both point and range queries comparing to the MBR pruning, which demonstrates the superiority of our pruning strategy. Additionally, if applying outerCH*, it provided better (upto 80%) reduction due to the benefit of the filtering of the dominated corner points before calculating the outer convex hulls.

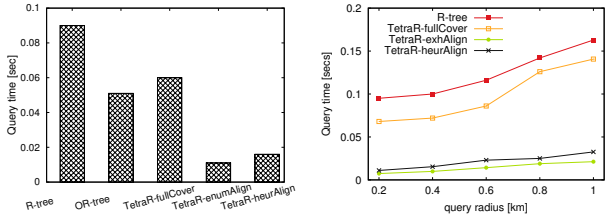
Fig. 17b illustrates the impact of total hit strategy for TetraR-heurAlign. Here, all the queries applied the outerCH*-based pruning strategy. We can observe that with innerCH, TetraR-heurAlign took 40% less query time than the case where no total hit strategy is applied. Meanwhile applying innerCH*, the query time was reduced by 60% for both point and range queries. This set of experiments show that total hit strategy can reduce the query time and applying the filtering technique to avoid computing a non-existing inner convex hull can reduce the processing time further.



(a) Pruning strategy (b) Total hit strategy

Figure 17: Evaluation on the search strategies

6.2.5 Evaluation on RW Dataset. To evaluate the effectiveness of our indexes and algorithms in a real-world setting, we also conducted a set of experiments using the real-world (*RW*) dataset. As shown in Fig. 18, the results show the same trends as the previus experiments with the *Gen* dataset for both point and range queries.



(a) Point query (b) Range query

Figure 18: Evaluation on *RW* Dataset

To summarize, the experimental results demonstrate that our proposed TetraR-tree with the alignment-based optimization and our search strategies based the outer/inner convex hulls consistently outperform the two baselines for both point and range queries.

7 RELATED WORK

We next review the related work on aerial videos and their indexing.

7.1 UAVs and Aerial Videos

UAVs have been widely studied in various academic communities, such as computer vision [11, 13] and robotics [4, 8]. In computer vision, researchers mainly focus on object detection and tracking from aerial videos/images for surveillance or traffic monitoring [11, 13]. In robotics, most studies focus on target tracking with UAV robots [8] or coverage path planning for UAV robots [4]. a path that passes over all points of an area of interest while avoiding obstacles. However, these studies focus on various applications of UAV videos or robots rather than indexing and querying of aerial videos.

7.2 Geo-tagged Video Indexing and Querying

There are many studies on video indexing and querying based on the geographic information of videos or images, and they can be categorized into two groups. The first group is indexing videos/images based on their camera locations. For example, Navarrete *et al.* [21] utilized R-tree to index the camera locations of videos. Toyama *et al.* [26] used grid files to index the camera location and time information. Irani *et al.* [9] and Alfarrarjeh *et al.* [2] discussed how to combine the spatial information (camera location) with the visual information into video indexing. The second group is indexing ground videos/images based on their spatial viewable coverages, i.e., representing geo-tagged videos as FOV objects. Our baseline indexes R-tree [5, 7] and OR-tree [17] as well as Grid [18] represent this group, and their drawbacks were analyzed in Sec. 3. Another recent index is GeoTree [12] (alternatively, GeoVideoIndex [14]) whose difference from R-tree is storing Minimum Bounding *Tilted* Rectangules (MB*T*R) in the leaf nodes. As GeoTree focused on dashcam videos recorded by cars driving on road networks, assuming camera shooting and moving directions do not change frequently, we did not refer to it as our baseline. However, all the indexes in both groups focused on ground videos.

There are few studies on aerial video indexing based on the spatial viewable coverages of videos. The most related one was proposed by Morse *et al.* [19] which indexed UAV videos by precomputing the "coverage quality map". The coverage quality map is generated by georegistering all the aerial videos to a terrain map. Specifically, each National Elevation Dataset (NED) point in a target area is associated with a set of video frames that cover the point and their corresponding quality values. The quality value of a video frame with respect to an NED point indicates how well the point can be visible in the video frame (i.e., considering the viewing distance, viewing angle, resolution, etc). The map is then used to index and search the aerial videos by indexing the NED points. Clearly, this precomputing-based approach requires large space and suffers from scalability problem for a large number of videos and a large target area (e.g., a city) and their approach is in fact orthogonal to our work. To the best of our knowledge, our paper is the first work on aerial video indexing and querying by representing their viewable coverages as quadrilateral-shaped aerial-FOVs.

8 CONCLUSION AND FUTURE WORK

In this paper, for the first time we represented aerial videos as a series of spatial objects, i.e., quadrilateral-shaped aerial-FOVs, and proposed a new index structure, called TetraR-tree that effectively captured the geometric property of aerial-FOVs. With TetraR-tree, we proposed two novel search strategies that employ the geographic properties of outer and inter convex hulls to expedite point and range queries on aerial videos. In future, we plan to extend this work in two directions. First, in addition to the geographic coverage, we intend to consider another important criteria, viewing direction, into the indexing and querying of aerial videos. For example, users may be interested in searching for aerial videos that cover a building viewed from a specific direction (e.g., the font side of the building). Second, we would like to design a hybrid index to support spatial queries on both aerial-FOVs and ground-FOVs efficiently.

9 ACKNOWLEDGEMENTS

This research has been funded in part by NSF grants IIS-1320149 and CNS-1461963, and USC Integrated Media Systems Center. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of any of the sponsors such as NSF.

REFERENCES

[1] 2017. Canada Drone. http://www.canadadrones.com/. (2017).
[2] A. Alfarrarjeh and C. Shahabi. 2017. Hybrid Indexes to Expedite Spatial-Visual Search. *CoRR* abs/1702.05200 (2017).
[3] A.M. Andrew. 1979. Another efficient algorithm for convex hulls in two dimensions. *Inform. Process. Lett.* 9, 5 (1979), 216 – 219.
[4] G. SC Avellar, G. AS Pereira, L. CA Pimenta, and P. Iscold. 2015. Multi-UAV Routing for Area Coverage and Remote Sensing with Minimum Time. *Sensors* 15, 11 (2015), 27783–27803.
[5] A. S. Ay, R. Zimmermann, and S. H. Kim. 2008. Viewable Scene Modeling for Geospatial Video Search. In *ACM Intl. Conf. on MM*. 309–318.
[6] S. A. Ay, S. H. Kim, and R. Zimmermann. 2010. Generating Synthetic Meta-data for Georeferenced Video Management. In *ACM SIGSPATIAL / GIS*. 280–289.
[7] A. Guttman. 1984. R-Trees: a dynamic index structure for spatial searching. In *SIGMOD*. 47–57.
[8] K. Hausman, J. Müller, A. Hariharan, N. Ayanian, and G. S. Sukhatme. 2014. Cooperative Control for Target Tracking with Onboard Sensing. In *the 14th Int'l Sym. on Exp. Rob., ISER*. 879–892.
[9] M. Irani and P. Anandan. 1998. Video indexing based on mosaic representations. *Proc. IEEE* 86, 5 (1998), 905–921.
[10] V. Jain and B. Shneiderman. 1994. Data Structures for Dynamic Queries: An Analytical and Experimental Evaluation. In *Proc. of the Workshop on Advanced Visual Interfaces*. 1–11.
[11] K. Kanistras, G. Martins, M. J. Rutherford, and K. P. Valavanis. 2015. *Survey of Unmanned Aerial Vehicles (UAVs) for Traffic Monitoring*. 2643–2666.
[12] Y. Kim, J. Kim, and H. Yu. 2014. Geotree: using spatial information for georeferenced video search. *Knowledge-based systems* 61 (2014), 1–12.
[13] R. Kumar, H. Sawhney, and S. Samarasekera. 2002. Aerial video surveillance and exploitation. *Proc. IEEE* 89, 10 (2002), 1518 – 1539.
[14] D. Lee, J. Oh, W.-K. Loh, and H. Yu. 2016. GeoVideoIndex: Indexing for georeferenced videos. *Information Sciences* 374 (2016), 210–223.
[15] S. Liu, H. Li, Y. Yuan, and W. Ding. 2014. *A Method for UAV Real-Time Image Simulation Based on Hierarchical Degradation Model*. 221–232.
[16] Y. Lu, C. Shahabi, and S. H. Kim. 2014. An Efficient Index Structure for Large-scale Geo-tagged Video Databases. In *ACM SIGSPATIAL / GIS*. 465–468.
[17] Y. Lu, C. Shahabi, and S. H. Kim. 2016. Efficient indexing and retrieval of large-scale geo-tagged video databases. *GeoInformatica* 20, 4 (2016), 829–857.
[18] H. Ma, S. Arslan Ay, R. Zimmermann, and S. H. Kim. 2014. Large-scale geo-tagged video indexing and queries. *GeoInformatica* 18, 4 (2014), 671–697.
[19] B. S Morse, C. H Engh, and M. A Goodrich. 2010. UAV video coverage quality maps and prioritized indexing for wilderness search and rescue. In *Human-Robot Interaction (HRI)*. IEEE, 227–234.
[20] T. Nagai, S. Yasutome, and N. Tokura. 1998. Convex hull problem with imprecise input. In *Japanese Conference on Discrete and Computational Geometry*. Springer, 207–219.
[21] T. Navarrete and J. Blat. 2006. A Semantic Approach for the Indexing and Retrieval of Geo-referenced Video. In *Proc. of the 1st Int'l Conf. on Semantic-Enhanced Multimedia Presentation Systems - Volume 228*. 88–100.
[22] P. Oettershagen, A. Melzer, T. Mantel, K. Rudin, T. Stastny, B. Wawrzacz, T. Hinzmann, S. Leutenegger, K. Alexis, and R. Siegwart. 2016. Design of small hand-launched solar-powered UAVs: From concept study to a multi-day world endurance record flight. *Journal of Field Robotics* (2016).
[23] H. S. Sawhney, A. Arpa, R. Kumar, S. Samarasekera, M. Aggarwal, S. Hsu, D. Nister, and K. Hanna. 2002. Video Flashlights: Real Time Rendering of Multiple Videos for Immersive Model Visualization. In *Proceedings of the 13th Eurographics Workshop on Rendering*. 157–168.
[24] M. Shimrat. 1962. Algorithm 112: Position of Point Relative to Polygon. *Commun. ACM* 5, 8 (Aug. 1962), 434–.
[25] S. Suri, K. Verbeek, and H. Yildiz. 2013. On the Most Likely Convex Hull of Uncertain Points. In *European Symposium on Algorithms*. 791–802.
[26] K. Toyama, R. Logan, and A. Roseway. 2003. Geographic Location Tags on Digital Images. In *ACM MM*. 156–166.
[27] Z. Zhu and A. R Hanson. 2006. Mosaic-based 3D scene representation and rendering. *Signal Processing: Image Communication* 21, 9 (2006), 739–754.