An Efficient Strategy for Spatio-temporal Data Indexing and Retrieval

Antonio d'Acierno[1], Marco Leone[2], Alessia Saggese[2] and Mario Vento[2]
[1]*Institute of Food Sciences, National Research Council, Avellino, Italy*
[2]*Department of Electronic and Information Engineering (DIEII), University of Salerno, Salerno, Italy*

Keywords: Spatio-temporal Queries, Spatio-temporal Data Indexing, Information Retrieval.

Abstract: Moving people's and objects' trajectories extracted from video sequences are increasingly assuming a key role for detecting anomalous events and for characterizing human behaviors. Among the key related issues, there is the need of efficiently storing a huge amount of 3D trajectories together with retrieval techniques sufficiently fast to allow a real-time extraction of trajectories satisfying spatio-temporal requirements. Unfortunately, while exist well established solutions for 2D trajectories, theoretical solutions proposed for 3D ones are not widely available in commercial and free spatially enabled DBMS; the paper thus presents a novel method for extending available 2D indexes to 3D data. In particular, starting from a redundant bi-dimensional indexing scheme recently introduced in (d'Acierno et al., 2011), we propose a new retrieval system that, while still using off-the-shelf solutions, avoids almost any redundancy in data to be handled; both the spatial complexity and the retrieval efficiency for time-interval queries have been significantly improved.

1 INTRODUCTION

In the past decades we have witnessed an increase in the number of acquisition cameras that represent a suitable solution for their relative low cost of maintenance and the possibility of installing them virtually everywhere. Once extracted moving objects' trajectories by means of video analytic algorithms, these data need to be efficiently *indexed* and properly *stored*, so to optimize the final *retrieval* step. Since information are usually stored in a database, we will refer to the retrieval phase as the *query processing* step. All the above mentioned steps are far from being simple tasks: one single video sequence lasting a few hours can contain thousands of objects of interest, with hundreds of thousands of spatio-temporal displacements.

With regard to the indexing phase, aimed at optimizing the retrieval operation, a widely adopted solution for bidimensional problems is represented by R-trees (Guttman, 1984), which hierarchically organize geometric data representing each object using its *Minimum Bounding Rectangle* (MBR). Starting from Guttman's pioneering paper, many other indexing schemes have been proposed, most of which optimize and extend R-trees to the 3D space.

(Pfoser et al., 2000), for instance, capture and index trajectory data by using STR-tree and TB-tree, while SEB-trees have been adopted by (Song and Roussopoulos, 2003) in order to segment trajectories with respect to space and time. Another indexing scheme based on R-trees, recently introduced by (Priyadarshini et al., 2011), is R k-d trie tree, which is able to reduce the time complexity. (Chakka et al., 2003) present a R*-tree based indexing scheme called SETI, which is demonstrated outperforming the three-dimensional R-tree; SETI's basic idea is to partition the space and use R*-tree on each of these partitions in order to build a sparse time index. (Park et al., 2010) present IsGrid, a grid-based indexing scheme, which provides better performance by avoiding some unnecessary visits while descending the indexing structure. Other indexing structures, proposed by (Zheng, 2011), are TPR-tree and an optimized version of it, namely FT-tree (Full Temporal tree). As for the low-level storage optimization, TrajStore (Cudre-Mauroux et al., 2010) aims at minimizing the number of disk accesses by co-locating on a disk block (or in a collection of near blocks) trajectory segments and by using an adaptive multi-level grid; thanks to this method, it is possible to retrieve the desired information by only reading a few blocks.

All the above approaches, even presenting efficient solutions from different perspectives, typically are not widely supported both in commercial and freely available products; for instance, PostGIS (Obe and Hsu, 2011), the well known extension of PostgreSQL DBMS for storing spatial data, while supporting three (and four)-dimensional data, does not support three-dimensional intersection and indexing operations. As a consequence, there is a strong interest in those methods which, even using off-the-shelf solutions, allow to solve the problem in the three-dimensional space.

In (d'Acierno et al., 2011), problems related to 3D data are solved by means of a redundant storing method that, at the extent of an increased spatial complexity, allows to index data using widely available bidimensional strategies. In this paper we propose a system that, even still using well-established bidimensional indexes, substantially avoids any redundancy in the stored data; the resulting querying time, moreover, is substantially decreased when compared to (d'Acierno et al., 2011) even thanks to a segmentation algorithm aimed at optimizing the use of the adopted indexes.

2 THE PROPOSED METHOD

A three dimensional trajectory is usually referred to as a sequence of spatio-temporal points:

$$T^i = < P_1^i, P_2^i, ..., P_N^i >$$

where the generic point $P_k^i = (x_k^i, y_k^i, t_k^i)$ represents the spatial location (x_k^i, y_k^i) of an object at the time instant t_k^i. From now on, we will use the line segments model (Pfoser et al., 2000), each segment being the line connecting two consecutive points.

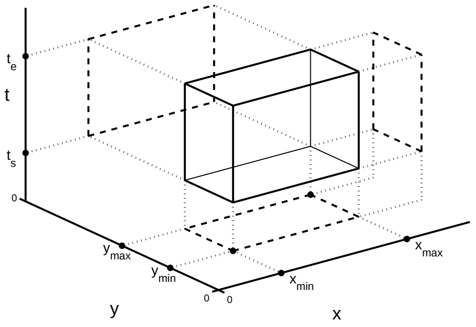


Figure 1: A query box representing a *TIQ*.

A trajectory-based time interval query (*TIQ*) aims at detecting all those objects' trajectories passing through a given spatial area in a given time interval. Here, we think to the area as a rectangle with coordinates (x_{min}, y_{min}) and (x_{max}, y_{max}) while $[t_s, t_e]$ are the starting and final time instants. According to this assumption, each *TIQ* can be associated to a query box *B* (Figure 1), which identifies the spatio-temporal volume possibly containing the trajectories.

To solve a *TIQ*, we have to verify, for each trajectory, if at least one of its segments intersects the query box. The intersection between a segment and a box can be verified by using a *clipping algorithm*, a well-known family of algorithms widely used for identifying the portion of an image which is either outside or inside a picture. One of the most efficient is the Cohen-Sutherland Line Clipping Algorithm (Newman and Sproull, 1979), which works, in its 3D formulation, by subdividing the plane into 27 regions by extending the faces of the query box.

However, despite its simplicity, the use of a clipping algorithm is not suited for handling large datasets: in fact, in the worst case, arising when a trajectory does not intersect the query box, all the trajectory' segments must be processed, making this approach unfeasible for a large amount of trajectory data. It means that, in real applications, it is necessary to make use of more efficient approaches, as the ones using suitable indexing strategies, known as spatial indexing. Spatial indexes allow to efficiently perform queries involving geometry data types such as points, lines and polygons; a query in this case represents a spatial relationship among these geometric entities.

In the 3D space, given a trajectory T^k and a query box *B*, it is straightforward to observe that, if T^k intersects *B*, then the projection of T^k on each coordinate plane also intersects the correspondent query box projection; this of course represents a necessary but not sufficient condition, as the opposite is clearly not true. Thus, if all projections of T^k intersect the correspondent box projections, we suggest to consider T^k as a candidate to be clipped in the 3D space.

According to the above considerations, in (d'Acierno et al., 2011), for each 3D trajectory T^k, we proposed to store three 2D trajectories obtained by projecting T^k on the *xy* plane (T_{xy}^k), on the *xt* plane (T_{xt}^k) and on the *yt* plane (T_{yt}^k). Given a box *B* representing the time interval query to be solved, we similarly considered B_{xy}, B_{xt} and B_{yt}. By using one of the available bidimensional indexes, it is possible to efficiently find, on each coordinate plane *kz*, the set:

$$\Theta_{kz} = \{T_{kz} : MBR(T_{kz}) \cap B_{kz} \neq \emptyset\} \qquad (1)$$

The set Θ of trajectories to be clipped in the 3D space is thus trivially defined as:

$$\Theta = \{T : T_{xy} \in \Theta_{xy} \wedge T_{xt} \in \Theta_{xt} \wedge \Theta_{yt} \in T_{yt}\} \qquad (2)$$

This strategy, while taking advantage of widely available efficient bidimensional indexes, still presents two weak points. First, for a *n* points trajectory, we need to redundantly store $6 \cdot n$ values ($2 \cdot n$ for each of the three coordinate planes). Another subtle crucial point is that the use of bidimensional indexes is not optimized: as a matter of fact, the MBR of each projected trajectory can easily span a great percentage of the whole area.

It is possible to observe that, for a given trajectory T^k, rather than storing the three different trajectory projections in each coordinate plane, we can store T^k as the original sequence of points in the 3D space, and separately maintain three different bidimensional MBRs: $MBR_{xy}(T^k)$, $MBR_{xt}(T^k)$ and $MBR_{yt}(T^k)$. $MBR_{xy}(T^k)$ (respectively $MBR_{xt}(T^k)$ and $MBR_{yt}(T^k)$) is obtained by projecting on the xy (respectively xt and yt) plane the 3D MBR of T^k.

It is worth noting that the redundancy introduced by the three MBR projections is not dependent on the number of points in the trajectory and, therefore, has only a marginal impact on the spatial complexity, since it only requires the storage of six pairs of points.

Assuming such a scheme, on each 2D plane we find the trajectories intersecting the corresponding 2D query box in a very efficient manner by using one of the available 2D indexes. Let Γ_{xy}, Γ_{xt} and Γ_{yt} be three sets of trajectories, each one defined as:

$$\Gamma_{kz} = \{T : MBR_{kz}(T) \cap B_{kz} \neq \emptyset\} \tag{3}$$

where, as usual, B_{xy}, B_{xt} and B_{yt} are the projections of the 3D query box B. The set Θ of the trajectories candidate to be clipped in the 3D space is therefore now defined as:

$$\Theta = \Gamma_{xy} \cap \Gamma_{xt} \cap \Gamma_{yt} \tag{4}$$

It should be clear at this point that the entire system performance will strongly depend on the indexing phase and, as a consequence, on the capability to reduce the number of trajectories to be clipped in the three-dimensional space. At a more detailed analysis, the *selectivity* of the indexes in each plane is related to the area of the corresponding MBR which, in turn, only depends on the trajectory geometry, so being (apparently) fixed. This is the reason why we decided to introduce a segmentation stage, aimed at increasing the selectivity of the indexes.

Segmentation aims at subdividing each trajectory into consecutive smaller units, which we will refer to as trajectory units. The proposed algorithm aims at exploiting the characteristics of the available bidimensional indexes by decreasing the area of the projected MBRs of each trajectory unit by recursively working. Initially (that is at iteration 0), it assumes that the trajectory T^k is composed by a single unit ${}^0U_1^k$, that is split into a set of m consecutive smaller units $\{{}^1U_1^k, \dots, {}^1U_m^k\}$; each of the ${}^1U_i^k$ is in turn inspected and, if the stop criteria are not satisfied, it is further split.

Let us analyze how a generic unit ${}^{(i-1)}U_j = \{P_1, \dots, P_m\}$ is split into $\{{}^iU_1, \dots, {}^iU_n\}$; we first choose a *split-dimension* (sd) and a *split-value* (s^*). Assume, as an example and without loss of generality, that x has been chosen as the *split-dimension* and let x^* be the *split-value*. In addition, assume that $x_1 < x^*$. According to these hypotheses, ${}^iU_1$ is the set of the consecutive points lying on the *left* side of x^*:

$${}^iU_1 = \{P_1, \dots, P_k\} \tag{5}$$

where P_k is thus the first point such that $x_k \geq x^*$. Then, the second unit will be formed by the sequence of consecutive points lying on the *right* side of x^*:

$${}^iU_2 = \{P_{k+1}, \dots, P_l\} \tag{6}$$

where P_l is the first point such that $x_k \leq x^*$. The inspection of ${}^{(i-1)}U$ ends when P_m is reached.

According to the above considerations, the criteria for the choice of sd and s^* play a crucial role. Since we aim at optimizing the indexing strategy, the proposed segmentation algorithm is based on the occupancy percentage on each 2D coordinate plane. Thus, with reference to the generic unit ${}^iU_j$ to be segmented, we calculate the three occupancy percentage values O_{xy}, O_{xt} and O_{yt} of as follows[1]:

$$O_{kz} = \frac{A(MBR_{kz}({}^iU_j))}{A(V_{kz})} \tag{7}$$

Without loss of generality, suppose that the maximum occupancy percentage value is O_{xy} and, consequently, the corresponding plane is xy; let W and H be the two dimensions of $MBR_{xy}({}^iU_j)$, respectively along the coordinates x and y; sd is defined as x if ($W > H$) and as y otherwise. Given sd, s^* is the MBR_{kz}'s average point on the coordinate sd.

The algorithm ends when all the trajectory units cannot be further subdivided, since at least one of the stop conditions has been reached for each unit; in particular, we employ two stop criteria . First, we only segment units composed by more than PS^{min} points. Furthermore, we choose not to segment trajectory units whose MBR areas are smaller than a fixed percentage of the entire scenario (PA^{min}).

3 EXPERIMENTAL RESULTS

To validate the effectiveness of the proposed method, we tested our system performing several *TIQs*. The database has been implemented by storing the trajectories' data in Postgres using PostGIS; data are indexed using the standard bidimensional R-tree over GiST (Generalized Search Trees) indexes since the specialized literature highlights that this choice guarantees higher performance in case of spatial queries,

[1]$A(\cdot)$ indicates the area while V_{kz} represents the projection of the volume of interest on the coordinate plane kz.

if compared with the PostGIS implementation of R-trees.

We represent each trajectory's unit as a tuple:

$$(\underline{ID}, \underline{UID}, U_{xyt}, MBR_{xy}, MBR_{xt}, MBR_{yt})$$

where ID is the moving objects identifier, UID identifies the trajectory's unit, and U_{xyt} is the 3D trajectory unit, represented as a sequence of segments (a PostGIS 3D multi-line). Finally, MBR_{xy}, MBR_{xt}, and MBR_{yt} are the three unit's MBRs in each coordinate plane, represented as PostGis *BOX* geometries. Once data have been indexed, PostGIS provides a very efficient function to perform intersections between boxes and MBRs in a 2D space. We conducted our experiments on a PC equipped with an Intel quad core CPU running at 2.66 GHz, using the 32 bit version of the PostgreSQL 9.1 server and the 1.5 version of PostGIS.

We tested our information retrieval system with synthetic data, which have been generated as follows. Let W_S and H_S be the width and the height of our scene and I be the time interval we are interested in. Each trajectory starting point is randomly chosen in our scene at a random time instant t_1; the trajectory length L is assumed to be fixed while the initial directions along the x axis and the y axis, respectively d_x and d_y, are randomly chosen. At each time step t, we first generate the new direction, assuming that d_x and d_y can vary with probability PI_x and PI_y respectively; subsequently, we randomly chose the velocity along x and y. The velocity is expressed in pixels/seconds and is assumed to be greater than 0 and less than two fixed maxima, V_x^{max} and V_y^{max}. Therefore the new position of the object can be easily derived; if it does not belong to our scene, new values for d_x and/or d_y are generated. We refer to the scene populated with trajectories as the *Scenario*. Table 1 reports the free parameters and the values for the creation of the 25 different scenarios used in our experiments as well as the parameters used to constrain the number of segments in each trajectory. Note that the worst case, corresponding to the maximum values of L and of the number of trajectories T, results in 10^4 trajectories with 10^4 points, for a total of 10^8 points to be stored and processed; this value is over and above the size of many real world datasets publicly available.

The time needed to process a generic *TIQ* query (QT) is a function of many parameters, since it surely depends on the number of trajectories T, on the trajectory length L, on the query cube dimension D_c (expressed as percentage of the volume $V = W_S * H_S * I$), and on the position of the query box P_c.

In particular, P_c strongly influences the time needed to extract the trajectories as, in real world scenarios, the trajectories are not uniformly distributed.

Table 1: The parameters used in our experiments.

W_S (pixels)	10^4
H_S (pixels)	10^4
I (seconds)	10^5
T	$\{1,2,3,5,10\} * 10^3$
L	$\{1,2,3,5,10\} * 10^3$
PI_x	5%
PI_y	5%
V_x^{max} (pixels/secs)	10
V_y^{max} (pixels/secs)	10
PA^{min}	1%
PS^{min}	300

To avoid the dependence on the query cube position, we decided to repeat the query a number of times inversely proportional to the query cube dimension, as shown in row N of Table 2; finally, results are averaged to obtain:

$$\overline{QT} = f(T, L, D_c). \tag{8}$$

For the description of the experimental results on our data, we propose three different set of experiments obtained fixing two of the three parameters T, L and D_c, and showing the variation of $\overline{QT}$ with respect to the third (free) parameter.

Table 2: Number N of times each query is repeated as D_c varies.

D_c	1%	5%	10%	20%	30%	50%
N	200	40	20	10	7	4

In Figure 2, $\overline{QT}$ is related to the variable number of trajectories T, both for small ($D_c = 5\%$) and large ($D_c = 30\%$) query boxes; the number of curves corresponds to the different fixed values of L. The relationship between $\overline{QT}$ and T has been analyzed by polynomially approximating $QT(T)$: $\overline{QT}$ linearly increases with T with a very small factor of approximation [2]. Diamond points in Figure 3 express $\overline{QT}$ in relation to the query box dimensions D_c and for $T = 3.000$ and $T = 10.000$ (the number of curves again corresponds to the different fixed values of L). In this case we obtain that $\overline{QT}$ quadratically depends on D_c. Finally, in Figure 4 the diamonds express $\overline{QT}$ as a function of L, for $D_c = 5\%$ and $D_c = 30\%$, while the number of curves corresponds to the different fixed values of T: $\overline{QT}$ increases quadratically with L.

[2]The semi-log scale provides a greater comprehension of the system behavior for large values of the parameters, even not permitting to display some of the lines interpolating small values.

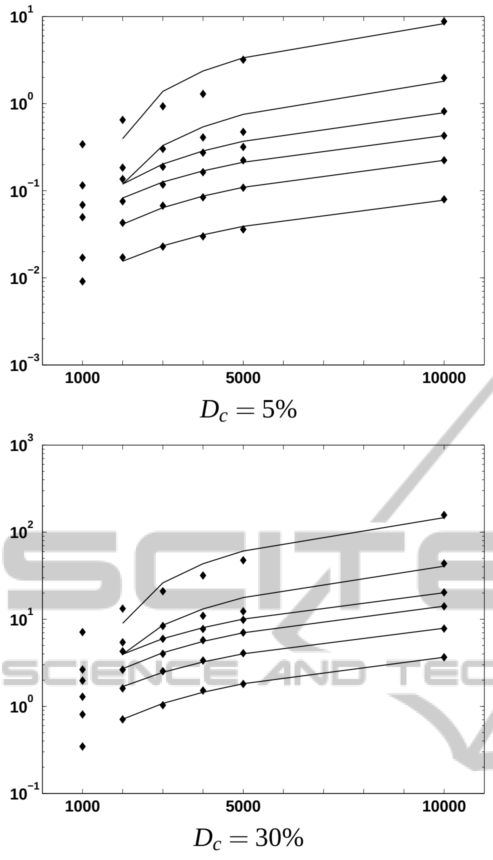


Figure 2: $\overline{QT}$ (in seconds) as T increases having L as parameter.

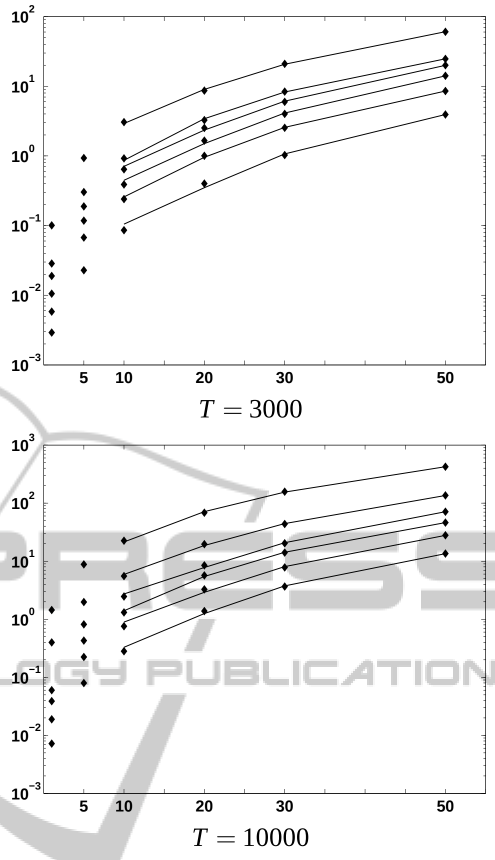


Figure 3: $\overline{QT}$ (in seconds) as D_c (in percentage of the whole volume) increases and having L as parameter.

4 CONCLUSIONS

Motivated by the fact that many of the existing solutions to index 3D data are not widely available both in commercial and freely available products, we are investigating the possibility of using widely available 2D indexes to deal with 3D data, and, in this paper, we propose a possible solution.

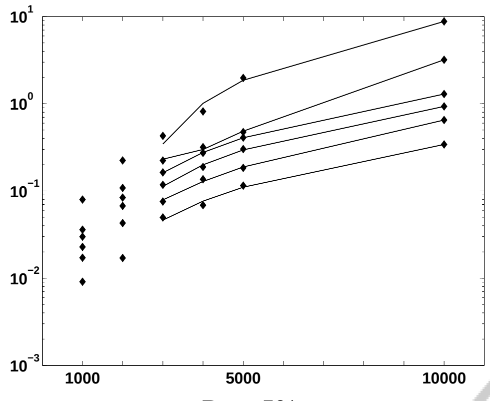
Our strategy has been implemented using PostGIS and the experimental results, obtained on *TIQs* performed on synthetic data, show that, even thanks to a segmentation algorithm we have proposed, the obtained system is able to fully exploit retrieving capabilities based on well established 2D indexes.

Our solution evolves the method presented in (d'Acierno et al., 2011), where the indexing of 3D data through bidimensional structures has been obtained using a redundant storing scheme. Spatial complexity, in fact, has been improved through the removal of almost any redundancy in the data to be stored. For what concerns the time complexity, while a careful theoretical analysis is outside the scope of this paper, it is possible to empirically define the improvement index η as follows:

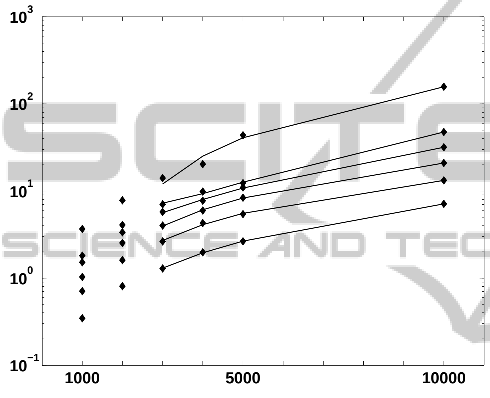
$$\eta = \frac{\overline{QT}_{\ (d'Aciernoet\ al.,2011)} - \overline{QT}}{\overline{QT}_{\ (d'Aciernoet\ al.,2011)}} \qquad (9)$$

Figure 5 shows η as T increases (with $L = 5000$) for small query cubes ($DC = 5\%$, diamonds) as well as for big query cubes ($DC = 30\%$, circles). There is a significant improvement for small query cubes and an interesting improvement for large cubes. Intuitively, this is due to the fact that $\overline{QT}$ is lower bounded by the time needed to extract trajectories.

Further improvements in the performance will be hopefully achieved first of all by applying the clipping algorithm in parallel to each candidate trajectory to take advantage of multi-core and multi-processors systems. To reduce the extraction time, strategies aiming at compressing data to be stored and retrieved are also being considered. Moreover, we are extending our system in order to answer different query typologies as well as to handle multi-dimensional data.



$D_c = 5\%$



$D_c = 30\%$

Figure 4: $\overline{QT}$ (in seconds) as L increases and having T as parameter.

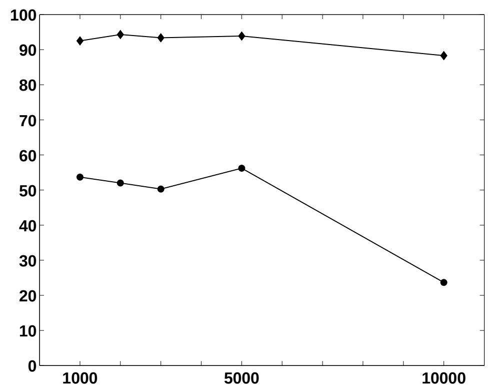


Figure 5: η as T increases for $DC = 5\%$ (diamonds) and DC=30% (circles) ($L = 5000$).

ACKNOWLEDGEMENTS

This research has been partially supported by A.I.Tech s.r.l. (a spin-off company of the University of Salerno, www.aitech-solutions.eu) and by the FLAGSHIP *InterOmics* project (PB.P05, funded and supported by the Italian MIUR and CNR organizations).

REFERENCES

Chakka, V. P., Everspaugh, A., and Patel, J. M. (2003). Indexing large trajectory data sets with seti. In *First Biennial Conference on Innovative Data Systems Research (CIDR 2003)*, Asilomar, CA, USA.

Cudre-Mauroux, P., Wu, E., and Madden, S. (2010). Trajstore: An adaptive storage system for very large trajectory data sets. In *Int. Conf. on Data Engineering*, pages 109–120, Los Alamitos, CA, USA. IEEE CS.

d'Acierno, A., Saggese, A., and Vento, M. (2011). A redundant bi-dimensional indexing scheme for three-dimensional trajectories. In *Proc. of the 1th Conf. on Advances in Information Mining and Management (IMMM11)*, pages 73–78, Barcelona, Spain.

Guttman, A. (1984). R-trees: a dynamic index structure for spatial searching. In *Proc. of ACM SIGMOD Conference*, pages 47–57, New York, NY, USA. ACM.

Newman and Sproull (1979). *Principles of interactive computer graphics*. Mc Graw-Hill, Singapore, 2nd edition.

Obe, R. and Hsu, L. (2011). *PostGIS in Action*. Manning Publications Co., Greenwich, CT, USA.

Park, Y., Seo, D., Lim, J., Lee, J., Kim, M., Bao, W., Ryu, C. T., and Yoo, J. (2010). A new spatial index structure for efficient query processing in location based services. In *Proc.of the 2010 IEEE Int. Conf. on Sensor Networks, Ubiquitous, and Trustworthy Computing*, SUTC '10, pages 434–441, Washington, DC, USA. IEEE Computer Society.

Pfoser, D., Jensen, C. S., and Theodoridis, Y. (2000). Novel approaches in query processing for moving object trajectories. In *Proc. of VLDB Conf.*, pages 395–406, San Francisco, CA, USA. Morgan Kaufmann Publ. Inc.

Priyadarshini, J., AnandhaKumar, P., Aparna, M., Geetha, J., and Shobana, N. (2011). Indexing and querying technique for dynamic location updates using r k-d trajectory trie tree. In *Int. Conf. on Recent Trends in Information Technology (ICRTIT)*, pages 1143 –1148.

Song, Z. and Roussopoulos, N. (2003). Seb-tree: An approach to index continuously moving objects. In *Proceedings of the 4th International Conference on Mobile Data Management*, MDM '03, pages 340–344, London, UK, UK. Springer-Verlag.

Zheng, Y. (2011). A fast index method for moving objects on full temporal query. In *3rd Int. Conf. on Computer Research and Development (ICCRD)*, volume 3, pages 205 –208.