# TIME SPLIT LINEAR QUADTREE FOR INDEXING IMAGE DATABASES

*Theodoros Tzouramanis*, *Michael Vassilakopoulos* & *Yannis Manolopoulos*
theo@delab.csd.auth.gr mvass@computer.org manolopo@delab.csd.auth.gr

Lab of Data Engineering, Dept. of Informatics, Aristotle University, 54006 Thessaloniki, Greece

## ABSTRACT

The Time Split B-Tree (TSBT) is modified for indexing a database of evolving binary images. This is accomplished by embedding ideas from Linear region Quadtrees that make the TSBT able to support spatio-temporal query processing. To improve query performance, additional pointers are added to the leaf-nodes of the TSBT. The resulting access method is called Time Split Linear Quadtree (TSLQ). Algorithms for processing five spatio-temporal queries have been adapted to the new structure. Such queries appear in Multimedia Systems, or Geographical Information Systems (GIS), when searched by content. The TSLQ was implemented and results of extensive experiments on query time performance are presented, indicating that the proposed algorithmic approaches outbalance respective straightforward algorithms. The region data sets used in the experiments were real images of meteorological satellite views and synthetic raster images.

## 1. INTRODUCTION

Numerous applications require efficient retrieval and querying of static spatial objects, or of spatial objects that change over time. These include Image and Multimedia databases, GIS, Urban Planning, Computer-Aided Design (CAD), etc. The traditional indexing methods are not suitable to store spatial data because of their inability to implement a total ordering of objects in space and preserve proximity, at the same time. A plethora of techniques have been developed for spatial data [1]. Recently, reseach in spatio-temporal databases has evolved and access methods for data objects that change their spatial locations and/or their shapes at different time intervals, have been proposed [2].

The objective of this report is to present an efficient spatio-temporal access method (STAM) for storing and accessing evolving binary raster images (regional data). The new structure is based on the TSBT [3], a secondary memory transaction-time method. Instead of storing independent numerical data having a different transaction-time each, for every consecutive image the new access method stores a group of codewords that share the same transaction-time (each codeword represents a spatial subregion). Moreover, additional pointers have been added to the leaf-nodes of the TSBT, aiming at increased query performance. The resulting STAM is called *Time Split Linear Quadtree* (TSLQ).

This structure has analogous functionality to other significantly different structures recently proposed by the authors, Overlapping Linear Quadtrees (OLQs) [4] and Multiversion Linear Quadtree (MVLQ) [5]. All three structures have the same origin, Linear region Quadtree [6]. Five efficient algorithms for processing spatio-temporal queries have been also adapted to an image database organized with a TSLQ. Such queries appear in Multimedia Systems, or GIS, when they are searched by content. The TSLQ was implemented and a thorough experimentation has been conducted using sequences of real and synthetic raster images. Results from these experiments that are presented, indicate that the proposed algorithmic approaches outbalance respective straightforward algorithms. A comparison with OLQs and MVLQ is a research activity in progress.

The rest of the paper is organized as follows. Section 2 provides a description of the new STAM. Section 3 discusses query processing and Section 4 presents experimental results that characterize graphically the query performance of the TSLQ. Finally, Section 5 concludes the paper introducing, also, ideas for further research.

## 2. THE NEW ACCESS METHOD

### 2.1. Linear Region Quadtrees

We assume that a two-dimensional binary raster image is represented as a $2^n \times 2^n$ array of pixels ordered by rows, where $n$ is a positive integer. The *region Quadtree* [6] is based on the successive decomposition of two-dimensional images into four quadrants of $2^{n-1} \times 2^{n-1}$ pixels. If a part of an image is not covered entirely by black or white, it is recursively subdivided into four subquadrants, until each subquadblock is entirely unicolor. An example of a binary raster image and its Quadtree appears in Fig. 1a and Fig. 1b, respectively.

The Quadtree is a main memory structure. However, the represented image may be very large and its Quadtree too

Research performed under the European Union's TMR Chorochronos project, contract number ERBFMRX-CT96-0056 (DG12-BDCN).

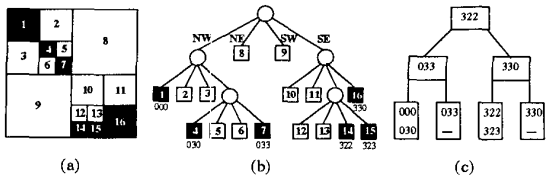


**Fig. 1**. (a) a raster image, (b) a Quadtree and (c) an LRQ

large for main memory. In such a case, information about the leaves that correspond to black quadblocks of the image array can be inserted into a B$^+$-tree producing, thus, a pointer-less version of the Quadtree. The latter method is called *Linear region Quadtree* (*LRQ* in the sequel, [6]).

Each black Quadtree leaf-node is represented by a pair of numbers $< C, L >$. The first number $C$ is termed *a locational code* and denotes the correct path to this leaf-node from the Quadtree root. Each one of the $n$ digits of $C$ can be 0,1,2 or 3, corresponding to quadrants NW, NE, SW and SE, respectively. The second number $L$ of the pair, is the Quadtree level where the node is located. This linear representation of the black Quadtree nodes is called FD (Fixed length - Depth) linear implementation [6]. Fig. 1c presents the LRQ that is obtained from the Quadtree of Fig. 1b. For simplicity, only the FD locational codes (*quadcodes* in the sequel) of the black Quadtree nodes appear in the LRQ.

### 2.2. Time Split Linear Quadtree

If a sequence of $N$ evolving images has to be stored in an LRQ, each one labeled with a unique time stamp $T_i$ (for $i$=1, 2, ..., $N$), then updates will overwrite old versions and only the last inserted image will be retained. TSLQ converts the ephemeral LRQ to a *persistent data structure*, where past states are also maintained and can be accessed. It couples time points with spatial objects in each node. Data records residing in leaves are of the form $(< C, L >, T_s)$, where $< C, L >$ is the FD code of a black Quadtree node and $T_s$ is the insertion time point of this FD code in the TSLQ. Non-leaf nodes contain entries of the form $(C', T'_s, Ptr)$, where $Ptr$ is a pointer to a descendent node, $C'$ is the smallest $C$ recorded in that descendent node and $T'_s$ is the time point at which the latter node became current (valid).

Two types of nodes are distinguished: current and historical nodes. In each TSLQ node, we added two new fields: the "$StartTime$" and the "$EndTime$" fields to register the time interval when this node is current (valid) in the database. As all new FD codes are inserted into current nodes, only current nodes are subject of split. They can be split either by key (i.e. by FD locational code), or by time. The *key split* is a split like the one in a standard B$^+$-tree. It creates one new node, whose StartTime field gets the value of the StartTime of the initially overflowing node. The *time split* is a split on transaction time. It implies the production of one historical node and the creation of one new node whose StartTime field gets the value of the time point chosen for the spit. It requires the duplication of all the records which intersect the time point chosen for the split.

The split dimension, i.e. whether a split is performed by key or time, is determined by a split policy. We chose the Isolated-Key-Split policy [3, 7], which generally achieves a good tradeoff between space consumption, i.e. the degree of redundancy of stored data, and query performance.

The algorithm of deletion in the TSLQ is, also, significantly different from the corresponding algorithm in the TSBT. The TSBT merely posts *deletion markers* (i.e. special record versions indicating the deletion) for all deleted records and does not merge sparse nodes. On the contrary, the implementation of a "real world" deletion of an FD code $< C, L >$ in the TSLQ at time point $T_j$ depends on the StartTime field of the corresponding leaf:
– If $StartTime < T_j$ then the FD code deletion is handled as a *logical* deletion, by inserting a deletion marker record of the appropriate entry.
– Otherwise, if $StartTime = T_j$ then the appropriate entry is removed from the leaf. After this *physical* deletion, if the number of entries in the leaf is above a threshold $d$ (analogous to the page consolidation threshold used in the B$^+$-tree), then the deletion is completed. Otherwise, *the node underflows*. This case is handled as in the B$^+$-tree, with the difference that if a sibling node exists then we have to check its StartTime field, before proceeding to a merge. If its $StartTime < T_j$, a current time split is made first on the sibling and the two nodes with $StartTime = T_j$ are then combined. If the new combined node has too many records, a key split is performed.

For example, consider the two consecutive images (with respect to their timestamps $T_1 = 1$ and $T_2 = 2$) in Fig. 2a. The TSLQ structure after the insertion of the first image is given in Fig. 2b. The node capacity $b$ equals 4, the threshold $d$ is the 20% of $b$ and the data records are of the form $(< C, L >, T_s)$. The second version of the structure is illustrated in Fig. 2c and it is constructed based on the first one, after the insertion of FD code $< 011, 0 >$ and the deletion of FD codes $< 102, 0 >$, $< 303, 0 >$ and $< 333, 0 >$. The deletion marker records are of the form $(< C, -1 >, T_j)$ where $T_j$ denotes a deletion time point.

## 3. SPATIO-TEMPORAL QUERY PROCESSING

The major feature of the new STAM, is that it can efficiently handle all the special types of spatio-temporal queries for quadtree-based databases, described in detail in [4]. More specifically, algorithms for processing the Strict Containment, Border Intersect, General Border Intersect, Cover and Fuzzy Cover Window spatio-temporal queries were adapted to the TSLQ. Definitions and algorithms for the processing

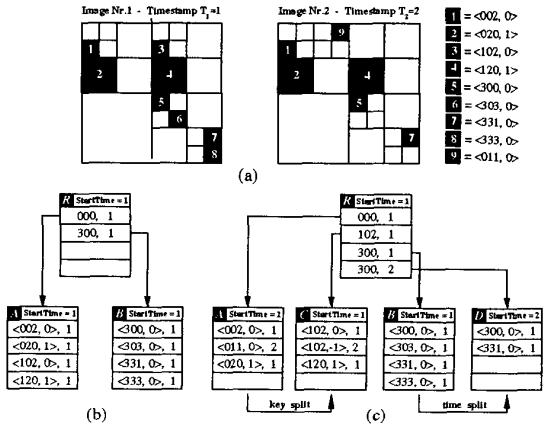


**Fig. 2.** (a) Two raster images and the TSLQ structure after the insertion of (b) the first image and (c) the second image.

of the above queries are given in [4] and can be applied to TSLQ with slight modifications.

Window queries have a primary importance since they are the basis of a number of operations that can be executed in various modern applications. For example, the Strict Containment Window Query discovers the black regions that completely fall inside the window of a query at each time point within the time interval $[T_1, T_N]$. This query is possible to appear in a Multimedia application, when a user is interested in the content of a window in the evolution of time.

In order to improve spatio-temporal query processing we added four "horizontal" pointers in every TSLQ leaf. This way there is no need to top-down traverse consecutive tree instances to search for the history of a specific FD code and excessive page accesses are avoided. The names of these pointers are: B-, BC-, F- and FC- pointer. Their roles and functions are described in detail in [4].

Alternative naive approaches for answering the above spatio-temporal queries are easy to devise. The respective algorithms would perform a suitable range search for every TSLQ version that corresponds to the given time interval, as if each one of them was separately stored in an LRQ, starting from the TSLQ root. These alternative approaches would not take into account the horizontal pointers, resulting in significantly worse I/O performance.

## 4. EXPERIMENTS

The TSLQ was implemented in C/C++. The page size was 1K, the leaf (internal) node capacity was equal to $b$=108 ($b'$=84) and the page consolidation threshold $d$ ($d'$) is the 20% of $b$ ($b'$). The evolving images were synthetic and real binary raster images of sizes 512×512 and 1024×1024 pixels, respectively. Each sophisticated algorithm for the five

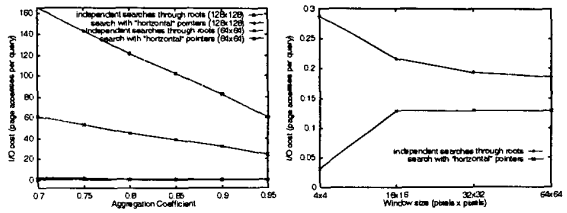


**Fig. 3.** The I/O efficiency of the Strict Containment (left) and the Cover (right) Window Queries.

spatio-temporal queries was executed several times for different window sizes and in a random window position every time. Besides, the respective naive algorithms were executed by performing independent umbrella-like searches through TSLQ root. In each run, we kept track of the average number of disk reads needed to perform the query per time point. For a more effective comparison of the two different algorithmic approaches, we excluded from the measurement the number of disk reads spent for the very first image of the sequence of the $N$ images, since we are interested only in the I/O cost profit we can succeed by the use of the horizontal pointers.

### 4.1. Experiments with Synthetic Data Sets

Every experiment was repeated 10 times using a pair of similar images. In the beginning, the first image was created with a specific black/white analogy and an aggregation coefficient $agg()$ that was increased at various amounts. The quantity $agg()$ has been defined in [8] and expresses the coherence of regions of homogeneous colors in an image. Starting from a random image and using the algorithm presented in [8], an image with exactly the same black/white analogy and higher aggregation (more realistic, including larger regions covered entirely by black or white) can be created. After the insertion of the first image, the second image was created by randomly changing the color of a given percentage (2%) of the pixels of the first image. Finally, the FD codes of that image were compared with those of the previous image and inserted in the TSLQ. Windows of sizes ranging from 4×4 to 128×128 pixels were queried 10 times each against the structure produced. Thus, every algorithm was executed 10×10 times.

The performance of the sophisticated and the naive algorithms is illustrated in Fig. 3. The left (right) part shows the I/O cost of the Strict Containment (Cover) Window Query as a function of the aggregation coefficient (of the window size) for 70% black images (50% black images and aggregation coefficient equal to 0.85). The linear decrease of the I/O cost for the naive algorithm of the Strict Containment Window Query is explained by the fact that images with larger aggregation form larger and solid black spatial re-

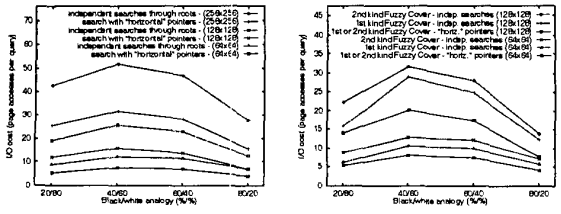

**Fig. 4**. The I/O efficiency of the Border Intersect (left) and the Fuzzy Cover (right) Window Query.

gions ("islands") and thus the corresponding LRQ holds a smaller number of FD codes.

### 4.2. Experiments with Real Data Sets

In the sequel, we provide results from some experiments based on a sequence of $N$=26 real raster images, which are meteorological views of California that were acquired from ftp://s2k-ftp.cs.berkeley.edu/pub/sequoia/benchmark/raster/. Originally, each 8-bit image pixel represented a value in a scale of 256 tones of gray. We transformed each image to a binary one, by choosing a threshold accordingly, so as to achieve a black analogy ranging between 20% and 80%. This group of meteorological satellite images corresponds to a particular 2-week interval of the same area (almost one image every 13 hours) and, thus, it is self-evident that there must appear many differences from image to image. It is obvious that the larger the image difference, the worse the query time performance of the sophisticated algorithms. However, the query performance results we obtained with these images were encouraging in such a worst case environment.

Windows of sizes ranging from $4\times4$ to $256\times256$ pixels were queried 50 times each against the structure produced. The left (right) part of Fig. 4 depicts the time performance of the Border Intersect (Fuzzy Cover) Window Query as a function of black analogy (and threshold 60%), for three (two) different window sizes.

An interesting general remark, refering to all the experiments we performed, is that the use of horizontal pointers leads to a remarkably high and stable I/O performance for all the spatio-temporal queries examined.

## 5. CONCLUSIONS

In the present paper, we proposed a new spatio-temporal structure called Time Split Linear Quadtree. This access method is able to index a database of consecutive raster images. It was demonstrated that it can be used in modern applications (multimedia, GIS) to support query processing of evolving images. Real world examples of such applications include the storage and manipulation of data of meteorological phenomena, of faunal phenomena, of urban environment, of natural catastrophes, etc. Five efficient algorithms for processing temporal window queries have also been adapted to the TSLQ. We presented experiments performed for studying the I/O efficiency of these algorithms. These experiments were based on sequences of synthetic and real evolving images. In general, our experiments showed clearly that, thanks to the "horizontal" pointers in the TSLQ leaves, our algorithms are very efficient in terms of disk activity.

In the future, we plan to investigate the use of a declustering scheme for the nodes of the TSLQ in order to exploit I/O parallelism for the above temporal window queries. In addition we plan to compare the space and time performance of TSLQ to those of OLQs and MVLQ and to develop algorithms for other new spatio-temporal queries that would take advantage of TSLQ, MVLQ, OLQs and other Quadtree-based STAMs and study their behavior. Moreover, we plan to investigate the possibility of designing analogous STAMS for grayscale and/or multicolored images.

## 6. REFERENCES


[1] V. Gaede and O. Guenther, "Multidimensional access methods," *ACM Computer Surveys*, vol. 30, no. 2, pp. 123–169, 1998.

[2] T. Abraham and J.F. Roddick, "Survey of spatio-temporal databases," *Geoinformatica*, vol. 3, no. 1, pp. 61–99, 1999.

[3] D. Lomet and B. Salzberg, "Access methods for multiversion data," in *Proc. ACM SIGMOD*, 1989, pp. 315–324.

[4] T. Tzouramanis, M. Vassilakopoulos, and Y. Manolopoulos, "Overlapping linear quadtrees and spatio-temporal query processing," *The Computer Journal*, vol. 43, no. 4, pp. 325–343, 2000.

[5] T. Tzouramanis, M. Vassilakopoulos, and Y. Manolopoulos, "Multiversion linear quadtree for spatio-temporal data," in *Proc. ADBIS-DASFAA 2000*, Prague, 2000, pp. 279–292.

[6] H. Samet, *Applications of Spatial Data Structures*, Addison-Wesley, Reading MA, 1990.

[7] D. Lomet and B. Salzberg, "The performance of a multiversion access method," in *Proc. ACM SIGMOD*, 1990, pp. 354–363.

[8] Y. Manolopoulos, E. Nardelli, G. Proietti, and M. Vassilakopoulos, "On the generation of aggregated random spatial regions," in *Proc. CIKM'95*, Washington DC, 1995, pp. 318–325.