Fully Retroactive Approximate Range and Nearest Neighbor Searching

Michael T. Goodrich and Joseph A. Simons

Department of Computer Science, University of California, Irvine, USA

Abstract. We describe fully retroactive dynamic data structures for approximate range reporting and approximate nearest neighbor reporting. We show how to maintain, for any positive constant d, a set of n points in $\mathbb{R}^d$ indexed by time such that we can perform insertions or deletions at any point in the timeline in $O(\log n)$ amortized time. We support, for any small constant $\epsilon > 0$, $(1+\epsilon)$-approximate range reporting queries at any point in the timeline in $O(\log n + k)$ time, where k is the output size. We also show how to answer $(1+\epsilon)$-approximate nearest neighbor queries for any point in the past or present in $O(\log n)$ time.

1 Introduction

Spatiotemporal data types are intended to represent objects that have geometric characteristics that change over time.

The important feature of such objects is that their critical characteristics, such as when they appear and disappear in a data set, exist in a *timeline*. The representation of such objects has a number of important applications, for instance, in video and audio processing, geographic information systems, and historical archiving. Moreover, due to data editing or cleaning needs, spatiotemporal data sets may need to be updated in a dynamic fashion, with changes that are made with respect to the timeline. Thus, in this paper we are interested in methods for dynamically maintaining geometric objects that exist in the context of a timeline. Queries and updates happen in *real time*, but are indexed in terms of the timeline.

In this paper, we are specifically interested in the dynamic maintenance of a set of d-dimensional points that appear and disappear from a data set in terms of indices in a timeline, for a given fixed constant $d \geq 1$. Points should be allowed to have their appearance and disappearance times changed, with such changes reflected forward in the timeline. We also wish to support time-indexed approximate range reporting and nearest-neighbor queries in such data sets. That is, we are interested in the dynamic maintenance of spatiotemporal point sets with respect to these types of geometric queries.

1.1 Related Work

Approximate Searching. Arya and Mount [3] introduce the approximate nearest neighbor problem for a set of points, S, such that given a query point q, a point of S will be reported whose distance from q is at most a factor of $(1+\epsilon)$ from that of the true nearest neighbor of q. Arya *et al.* [5] show that such queries can be answered in $O(\log n)$ time for a fixed constant $\epsilon > 0$. Chan [9] shows how to achieve a similar bound. Arya and Mount [4] also introduce the approximate range searching problem for a set, S, where a range R (e.g. a sphere or rectangle) of diameter w is given as input and every point in S that is inside R is reported as output and no point that is more than a distance of ϵw outside of R is reported. Let k be the number of points reported. Arya and Mount show that such queries can be answered in $O(\log n + k)$ time for fixed constant $\epsilon > 0$. Eppstein *et al.* [18] describe the skip quadtree structure, which supports $O(\log n + k)$-time approximate range searching as well as $O(\log n)$-time point insertion and deletion.

Our approach to solving approximate range searching and approximate nearest neighbor problems are based on the quadtree structure [28]. In this structure, regions are defined by squares in the plane, which are subdivided into four equal-sized squares for any regions containing more than a single point. So each internal node in the underlying tree has up to four children and regions have optimal aspect ratios. Typically, this structure is organized in a compressed fashion [7], so that paths in the tree consisting of nodes with only one non-empty child are compressed to single edges. This structure is related to the balanced box decomposition (BBD) trees of Arya *et al.* [3, 4, 5], where regions are defined by hypercubes with smaller hypercubes subtracted away, so that the height of the decomposition tree is $O(\log n)$. Similarly, Duncan *et al.* [17] define the balanced aspect-ratio (BAR) trees, where regions are associated with convex polytopes of bounded aspect ratio, so that the height of the decomposition tree is $O(\log n)$.

Computational Geometry with respect to a Timeline. Although we are not familiar with any previous work on retroactive d-dimensional approximate range searching and nearest-neighbor searching, we nevertheless would like to call attention to the fact that incorporating a time dimension to geometric constructions and data structures is well-studied in the computational geometry literature.

– Atallah [6] studies several *dynamic computational geometry* problems, including convex hull maintenance, for points moving according to fixed trajectories.
– Subsequently, a number of researchers have studied geometric motion problems in the context of *kinetic data structures* (e.g., see [22]). In this framework, a collection of geometric objects is moving according to a fixed set of known trajectories, and changes can only happen in the present.
– Driscoll *et al.* [16] introduce the concept of *persistent data structures*, which support time-related operations where updates occur in the present and queries can be performed in the past, but updates in the past fork off new timelines rather than propogate changes forward in the same timeline.

All of this previous work differs from the approach we are taking in this paper, since in these previous approaches objects are not expected to be retroactively changed "in the past."

Demaine *et al.* [13] introduce the concept of *retroactive data structures*, which is the framework we follow in this paper. In this approach, a set of data is maintained with respect to a timeline. Insertions and deletions are defined with respect to this timeline, so that each insertion has a time parameter, t, and so does each deletion. Likewise, queries are performed with respect to the time parameter as well. The difference between this framework and the dynamic computational geometry approaches mentioned above, however, is that updates can be done retroactively "in the past," with the changes necessarily being propagated forward. If queries are only allowed in the current state (i.e., with the highest current time parameter), then the data structure is said to be *partially retroactive*. If queries can be done at any point in the timeline, then the structure is said to be *fully retroactive*. Demaine *et al.* [13] describe a number of results in this framework, including a structure for fully-retroactive 1-dimensional successor queries with $O(\log^2 n)$-time performance. They also show that any data structure for a decomposable search problem can be converted into a fully retroactive structure at a cost of increasing its space and time by a logarithmic factor.

Acar *et al.* [1] introduce an alternate model of retroactivity, which they call *non-oblivious* retroactivity. In this model, one maintains the historical sequence of queries as well as insertions and deletions. When an update is made in the past, the change is not necessarily propagated all the way forward to the present. Instead, a non-oblivious data structure returns the first operation in the timeline that has become *inconsistent*, that is an operation whose return value has changed because of the retroactive update. As mentioned above, we only consider the original model of retroactivity as defined by Demaine *et al.* [13] in this paper.

Blelloch [8] and Giora and Kaplan [20] consider the problem of maintaining a fully retroactive dictionary that supports successor or predecessor queries. They both base their data structures on a structure by Mortensen [25], which answers fully retroactive one dimensional range reporting queries, although Mortensen framed the problem in terms of two dimensional orthogonal line segment intersection reporting. In this application, the x-axis is viewed as a timeline for a retroactive data structure for 1-dimensional points. The insertion of a segment $[(x_1, y), (x_2, y)]$ corresponds to the addition of an insert of y at time x_1 and a deletion of y at time x_2. Likewise, the removal of such a segment corresponds to the removal of these two operations from the timeline. For this 1-dimensional retroactive data structuring problem, Blelloch and Giora and Kaplan give data structures that support queries and updates in $O(\log n)$ time. Dickerson *et al.* [15] describe a retroactive data structure for maintaining the lower envelope of a set of parabolic arcs and give an application of this structure to the problem of cloning a Voronoi diagram from a server that answers nearest-neighbor queries.

1.2 Our Results

In this paper, we describe fully retroactive dynamic data structures for approximate range reporting and approximate nearest neighbor searching. We show how to maintain, for any positive constant, $d \geq 1$, a set of n points in $\mathbb{R}^d$ indexed by time such that we can perform insertions or deletions at any point in the timeline in $O(\log n)$ amortized time. We support, for any small constant $\epsilon > 0$, $(1+\epsilon)$-approximate range reporting queries at any point in the timeline in $O(\log n + k)$ time, where k is the output size. Note that in this paper we consider circular ranges defined by a query point q and radius r. We also show how to answer $(1+\epsilon)$-approximate nearest neighbor queries for any point in the past or present in $O(\log n)$ time. Our model of computation is the real RAM, as is common in computational geometry algorithms (e.g., see [29]).

The main technique that allows us to achieve these results is a novel, multidimensional version of fractional cascading, which may be of independent interest. Recall that in the (1-dimensional) *fractional cascading* paradigm of Chazelle and Guibas [11, 12], one searches a collection of sorted lists (of what are essentially numbers), which are called *catalogs*, that are stored along nodes in a search path of a catalog graph, G, for the same element, x. In *multidimensional fractional cascading*, one instead searches a collection of finite subsets of $\mathbb{R}^d$ for the same point, p, along nodes in a search path of a catalog graph, G. In our case, rather than have each catalog represented as a one-dimensional sorted list, we instead represent each catalog as a multidimensional "sorted list," with points ordered as they would be visited in a space-filling curve (which is strongly related to how the points would be organized in a quadtree, e.g., see [7]).

By then sampling in a fashion inspired by one-dimensional fractional cascading, we show[1] how to efficiently perform repeated searching of multidimensional catalogs stored at the nodes of a search path in a suitable catalog graph, such as a segment tree (e.g., see [29]), with each of the searches involving the same d-dimensional point or region.

Although it is well known that space-filling curves can be applied to the problem of approximate nearest neighbor searching, we are not aware of any extension of space-filling curves to approximate range reporting. Furthermore, we believe that we are the first to leverage space-filling curves in order to extend dynamic fractional cascading into a multi-dimensional problem.

2 A General Approach to Retroactivity

Recall that a query Q is *decomposable* if there is a binary operator $\square$ (computable in constant time) such that $Q(A \cup B) = \square(Q(A), Q(B))$. Demaine *et al.* [13] showed that we can make any decomposable search problem retroactive by maintaining each element ever inserted in the structure as a line segment along a time dimension between the element's insertion and deletion times.

[1] The details for our constructions are admittedly intricate, so some details of proofs are given in the full version of this paper [21].

Thus each point p in R^d is now represented by a line segment parallel to the time-axis in R^{d+1} dimensions. For example when extending the query to be fully-retroactive, a one-dimensional successor query becomes a two-dimensional vertical ray shooting query, and a one-dimensional range reporting query becomes a two-dimensional orthogonal segment intersection query.

Thus, we maintain a segment tree to allow searching over the segments in the time dimension, and augment each node of the segment tree with a secondary structure supporting our original query in d dimensions. Let S be the set of nodes in the segment tree on a root-to-leaf path searching down for t in the time dimension. To answer a fully-retroactive query, we perform the same d-dimensional query at each node in S. This transformation costs an extra $\log n$ factor in space, query time, and update time, which we would nevertheless like to avoid.

Recall that Mortensen [25] and Giora and Kaplan [20] both solve the fully-retroactive versions of decomposable search problems, and are both able to avoid the extra $\log n$ factor in query and update time. Therefore inspired by their techniques, we propose the following as a general strategy for optimally solving the fully-retroactive version of any decomposable search problem.

1. Suppose we have an optimal data structure D for the non-retroactive problem which supports queries in polylogarithmic time $T(n)$.
2. Represent each d-dimensional point as a line segment in $d + 1$ dimensions.
3. Build a weight-balanced segment tree with polylogarithmic branching factor over the segments as described by [20].
4. Augment the root of the segment tree with an optimal search structure D.
5. Augment each node of the segment tree with a colored dynamic fractional cascading (CDFC) data structure.
6. Perform a retroactive query at time t by performing the non-retroactive query on the non-retroactive data structure at the root of the segment tree, and for each node on the search path for t in the segment tree, perform the query in each of the CDFC structures (Figure 1).

The CDFC data structure must be cleverly tuned to support a *colored* (but non-retroactive) version of the d-dimensional query in $O(T(n) \cdot \log\log n/\log n)$ time. In a colored query, each element in the structure is given a color, and the query also specifies a set of colors. We only return elements whose color is contained in the query set. The colors are required because the segment tree has high degree. Each color represents a pair of children of the current node in the segment tree. Thus we encode which segments overlap the search path via their colors. Since the segment tree has a polylogarithmic branching factor, we spend $T(n)$ time searching at the root and $O(T(n) \cdot \log\log n/\log n)$ time searching in the CDFC structures at each of the $O(\log / \log n \log n)$ nodes. Therefore, the total time required by a query is an optimal $O(T(n))$. Updates follow a similar strategy, but may require us to periodically rebuild sections of the segment tree. We can still achieve the desired (amortized) update time, and the analysis closely follows [20].

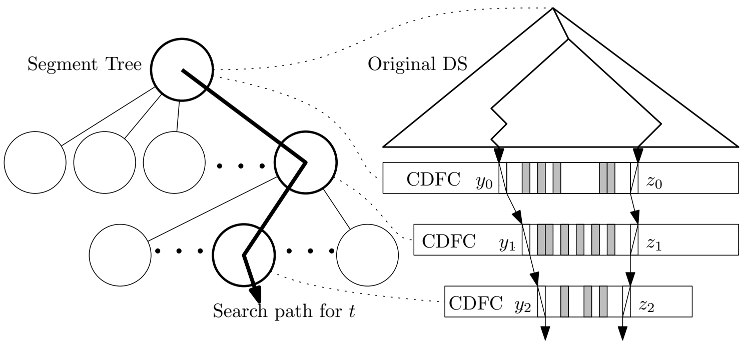

Fig. 1. An application of our strategy to a fully-retroactive one-dimensional range reporting query. Shaded elements represent elements with colors indicating they are present at time t.

One of the key difficulties in applying this strategy lies in the design of the colored dynamic fractional cascading data structure, especially in problems where the dimension $d > 1$. In fact, the authors are not aware of any previous application of dynamic fractional cascading techniques to any multidimensional search problem. However, in the following we show how techniques using space filling curves can be applied to extend the savings of fractional cascading into a multidimensional domain. First, we apply the above strategy in the simpler case when $d = 1$. Then we extend this result using space filling curves to support Fully-Retroactive range reporting queries and approximate nearest neighbor queries in $\mathbb{R}^d$. Note that in one dimension a nearest neighbor query reduces to a successor and predecessor query.

Lemma 1. *There exists a colored dynamic fractional cascading data structure which supports updates in $O(\log\log N)$ amortized time, colored successor and predecessor queries in $O(\log\log N)$ worst case time and colored range reporting queries in $O(\log\log N + k)$ worst case time, where N is the number of elements stored and k is the number of elements reported.*

Proof: We extend the generalized union-split-find structure of [20] to also support colored range queries. See [21] for details. □

Space Filling Curves. The z-order, due to Morton [26], is commonly used to map multidimensional points down to one dimension. By now space filling curves are well-studied and multiple authors have applied them specifically to approximate nearest neighbor queries [14, 10, 9, 24]. However, we extend their application to approximate range searching as well. Furthermore, we believe that we are the first to leverage space-filling curves to extend dynamic fractional cascading techniques to multidimensional problems such as these.

Lemma 2. *The set of points in any quadtree cell rooted at $[0, 1)^d$ form a contiguous interval in the z-order.*

Proof: Due to Bern *et al.* [7]. Also see [21]. □

Lemma 3. *Let P be a set of points in $\mathbb{R}^d$. Define a constant $c = \sqrt{d}(4d+4)+1$. Suppose that we have $d+1$ lists $P+v^j, j = 0, \ldots, d$, each one sorted according to its z-order. We can find a query point q's c-approximate nearest neighbor in P by examining the the $2(d+1)$ predecessors and successors of q in the lists.*

Proof: Due to Liao *et al.* [24]. Also see [21] for details. □

3 Main Results

In this section we give our primary results: data structures for fully-retroactive approximate range queries and fully-retroactive approximate nearest neighbor (ANN) queries. Recall that an approximate range query $report(q, r, \epsilon, t)$ defines an inner range Q^-, the region within a radius r of the query point q and an outer range Q^+, the area within a radius of $(1+\epsilon)r$ of q. We want to return all the points inside Q^- and exclude all points outside Q^+ for a particular point in time t. Points that fall between Q^- and Q^+ at time t may or may not be reported. Points not present at time t will never be reported.

Theorem 1 (Fully-Retroactive Approximate Range Queries). *For any positive constant $d \geq 1$, we can maintain a set of n points in $\mathbb{R}^d$ indexed by time such that we can perform insertions or deletions at any point in the timeline in $O(\log n)$ amortized time. We support for any small constant $\epsilon > 0$, $(1+\epsilon)$-approximate range reporting queries at any point in the timeline in $O(\log n + k)$ time, where k is the output size. The space required by our data structure is $O(n \log n / \log\log n)$.*

Proof: We follow the general strategy outlined in Section 2. We augment the root of the segment tree with a skip quadtree [18], an optimal structure for approximate range and nearest neighbor queries in $\mathbb{R}^d$. We also augment each node of the segment tree with a specialized CDFC structure which we now describe.

We extend the CDFC structure from Lemma 1 to maintain d-dimensional points such that given a query set of colors C_q and d-dimensional quadtree cell, it returns all points contained in that cell with colors in C_q. By Lemma 2, for all points y, q, z such that $y < q < z$ in the z-order, any quadtree cell containing y and z must also contain q. Furthermore, for a given quadtree, each cell is uniquely defined by the leftmost and rightmost leaves in the cell's subtree. Therefore, the d-dimensional cell query reduces to a one-dimensional range query in the z-order on the unique points which define the quadtree cell (Figure 2). Thus, by leveraging Lemma 1 and maintaining the points according to their z-order, we support the required query in $O(\log\log n + k)$ time.

The skip quadtree contains all the points ever inserted into our data structure, irrespective of the time that they were inserted or deleted. The inner and outer range of a query partition the set of quadtree cells into three subsets: the *inner*

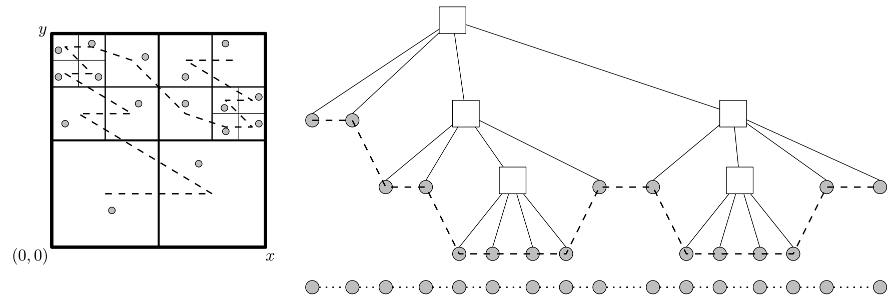

Fig. 2. The z-order curve corresponds to the order the leaves would be visited in an in-order traversal of the quadtree. We can store the points in a linked list in this order. Any element between two elements i, j in the linked list must fall in the same quadtree cell as i and j.

cells, which are completely contained in Q^+, the *outer cells*, which are completely disjoint from Q^-, and the stabbing cells, which partially overlap Q^+ and Q^-. Eppstein *et al.* [19] showed that a skip quadtree can perform an approximate range query in $O(\log n + \epsilon^{1-d})$ time expected and with high probability, or worst case time for a deterministic skip quadtree. Using their algorithm we can find the set I of $O(\epsilon^{1-d})$ disjoint inner cells in the same time bound. Based on the correctness of their algorithm we know that the set of points we need to report are covered by these cells. We report the points present at time t as follows: For each cell $i \in I$, if i is a leaf, we report only the points in i, which are present at time t in constant time. Otherwise, we find the first and last leaves y_0 and z_0 according to an in-order traversal of i's subtree in $O(\log n)$ time. Then we perform a fully-retroactive range query on cell i in the segment tree, using z_0 and y_0 to guide the query on cell i in the CDFC structures. Correctness follows since a point satisfies the retroactive range query if and only if it is in one of the cells we examine and is present at time t.

For each of the $O(\epsilon^{1-d})$ cells in I we spend $O(\log n + k_i)$ time and so the total running time is $O(\epsilon^{1-d} \log n + k) = O(\log n + k)$, since ϵ is a small constant. The skip quadtree is linear space, and thus the space usage is dominated by the segment tree. The total space required is $O(n \log n / \log \log n)$. □

Corollary 1 (Fully-Retroactive ANN Queries.). *We can maintain, for any positive constant, $d \geq 1$, a set of n points in $\mathbb{R}^d$ indexed by time such that we can perform insertions or deletions at any point in the timeline in $O(\log n)$ amortized time. We support, for any small constant $\epsilon > 0$, $(1+\epsilon)$-approximate nearest neighbor queries for any point in the past or present in $O(\log n)$ time. The space required by our data structure is $O(n \log n / \log \log n)$.*

Proof: By combining the data structure of [20] with Lemma 3 and storing the d-dimensional points in that structure according to their z-order, we already have a data structure for fully-retroactive c-approximate nearest neighbor queries.

However, c is polynomial function of d, and for $d = 2$, c is already greater than 15. In order to support $(1+\epsilon)$ nearest neighbor queries for any $\epsilon > 0$, we require the data structure of the previous theorem.

We know from Lemma 3 that we can find a c-approximate nearest neighbor by using $d + 1$ different shifts of the points. Therefore, we augment our data structure so that instead of a single CDFC structure at each segment tree node, we keep an array of $d + 1$ CDFC structures corresponding to the z-order of each of the $d + 1$ sets of shifted points. Given a query point q, we can find the predecessor and successor of q at time t in each of the $d+1$ z-orders in $O(d \log n)$ time. Out of these $2(d + 1)$ points, let p be the point that is closest to q. By Lemma 3, p is a c-approximate nearest neighbor. Let r be the distance between p and q. As observed by multiple authors [2, 27, 23], we can find a $(1 + \epsilon)$-approximate nearest neighbor via a bisecting search over the interval $[r/c, r]$. This search requires $O(\log(1/\epsilon))$ fully-retroactive spherical emptiness queries. We can support a retroactive spherical emptiness query in $O(\log n)$ time with only a slight modification to our retroactive approximate range query. Instead of returning k points in the range, we just return the first point we find, or `null` if we find none. Thus the total time required is $O(d \log n + \log(1/\epsilon) \log n) = O(\log n)$ since we assume ϵ and d are constant. The space usage only increases by a factor of d when we store the $d + 1$ shifted lists, and thus the space required is still $O(n \log n/ \log\log n)$. $\square$

References

1. Acar, U.A., Blelloch, G.E., Tangwongsan, K.: Non-oblivious retroactive data structures. Tech. Rep. CMU-CS-07-169, Carnegie Mellon University (2007)
2. Arya, S., da Fonseca, G.D., Mount, D.M.: A Unified Approach to Approximate Proximity Searching. In: de Berg, M., Meyer, U. (eds.) ESA 2010. LNCS, vol. 6346, pp. 374–385. Springer, Heidelberg (2010)
3. Arya, S., Mount, D.M.: Approximate nearest neighbor queries in fixed dimensions. In: Proc. 4th ACM-SIAM Sympos. Discrete Algorithms, pp. 271–280 (1993)
4. Arya, S., Mount, D.M.: Approximate range searching. Comput. Geom. Theory Appl. 17, 135–152 (2000)
5. Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A.: An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. J. ACM 45, 891–923 (1998)
6. Atallah, M.J.: Some dynamic computational geometry problems. Computers and Mathematics with Applications 11(12), 1171–1181 (1985)
7. Bern, M., Eppstein, D., Teng, S.-H.: Parallel Construction of Quadtrees and Quality Triangulations. In: Dehne, F., Sack, J.-R., Santoro, N. (eds.) WADS 1993. LNCS, vol. 709, pp. 188–199. Springer, Heidelberg (1993)
8. Blelloch, G.E.: Space-efficient dynamic orthogonal point location, segment intersection, and range reporting. In: 19th ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 894–903 (2008)
9. Chan, T.M.: Approximate nearest neighbor queries revisited. Discrete and Computational Geometry 20, 359–373 (1998)

10. Chan, T.M.: A minimalist's implementation of an approximate nearest neighbor algorithm in fixed dimensions (2006) (manuscript)
11. Chazelle, B., Guibas, L.J.: Fractional cascading: I. A data structuring technique. Algorithmica 1(3), 133–162 (1986)
12. Chazelle, B., Guibas, L.J.: Fractional cascading: II. Applications. Algorithmica 1, 163–191 (1986)
13. Demaine, E.D., Iacono, J., Langerman, S.: Retroactive data structures. ACM Trans. Algorithms 3 (May 2007)
14. Derryberry, J., Sheehy, D., Sleator, D.D., Woo, M.: Achieving spatial adaptivity while finding approximate nearest neighbors. In: Proceedings of the 20th Canadian Conference on Computational Geometry, CCCG 2008, pp. 163–166 (2008)
15. Dickerson, M.T., Eppstein, D., Goodrich, M.T.: Cloning Voronoi Diagrams Via Retroactive Data Structures. In: de Berg, M., Meyer, U. (eds.) ESA 2010. LNCS, vol. 6346, pp. 362–373. Springer, Heidelberg (2010)
16. Driscoll, J.R., Sarnak, N., Sleator, D.D., Tarjan, R.E.: Making data structures persistent. J. Comput. Syst. Sci. 38, 86–124 (1989)
17. Duncan, C.A., Goodrich, M.T., Kobourov, S.: Balanced aspect ratio trees: combining the advantages of k-d trees and octrees. J. Algorithms 38, 303–333 (2001)
18. Eppstein, D., Goodrich, M.T., Sun, J.Z.: The skip quadtree: A simple dynamic data structure for multidimensional data. In: 21st ACM Symp. on Computational Geometry (SCG), pp. 296–305 (2005)
19. Eppstein, D., Goodrich, M.T., Sun, J.Z.: The skip quadtree: a simple dynamic data structure for multidimensional data. In: 21st ACM Symp. on Computational Geometry, pp. 296–305 (2005)
20. Giora, Y., Kaplan, H.: Optimal dynamic vertical ray shooting in rectilinear planar subdivisions. ACM Trans. Algorithms 5, 28:1–28:51 (2009)
21. Goodrich, M.T., Simons, J.A.: Fully Retroactive Approximate Range and Nearest Neighbor Searching. ArXiv e-prints (September 2011)
22. Guibas, L.J.: Kinetic data structures — a state of the art report. In: Agarwal, P.K., Kavraki, L.E., Mason, M. (eds.) Proc. Workshop Algorithmic Found. Robot., pp. 191–209. A. K. Peters, Wellesley (1998)
23. Har-Peled, S.: A replacement for voronoi diagrams of near linear size. In: Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci., pp. 94–103 (2001)
24. Liao, S., Lopez, M., Leutenegger, S.: High dimensional similarity search with space filling curves. In: Proceedings of the 17th International Conference on Data Engineering, pp. 615–622 (2001)
25. Mortensen, C.W.: Fully-dynamic two dimensional orthogonal range and line segment intersection reporting in logarithmic time. In: 14th ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 618–627 (2003)
26. Morton, G.M.: A computer oriented geodetic data base; and a new technique in file sequencing, Tech. rep., IBM Ltd. (1966)
27. Mount, D.M., Park, E.: A dynamic data structure for approximate range searching. In: ACM Symp. on Computational Geometry, pp. 247–256 (2010)
28. Orenstein, J.A.: Multidimensional tries used for associative searching. Inform. Process. Lett. 13, 150–157 (1982)
29. Preparata, F.P., Shamos, M.I.: Computational Geometry: An Introduction, 3rd edn. Springer, Heidelberg (1990)