Kinetic Medians and *kd*-Trees

Pankaj K. Agarwal[1], Jie Gao[2], and Leonidas J. Guibas[2]

[1] Department of Computer Science, Duke University, Durham, NC 27708, USA
pankaj@cs.duke.edu

[2] Department of Computer Science, Stanford University, Stanford, CA 94305, USA
{jgao, guibas}@cs.stanford.edu

Abstract. We propose algorithms for maintaining two variants of *kd*-trees of a set of moving points in the plane. A pseudo *kd*-tree allows the number of points stored in the two children to differ by a constant factor. An overlapping *kd*-tree allows the bounding boxes of two children to overlap. We show that both of them support range search operations in $O(n^{1/2+\epsilon})$ time, where ϵ only depends on the approximation precision. As the points move, we use event-based kinetic data structures to update the tree when necessary. Both trees undergo only a quadratic number of events, which is optimal, and the update cost for each event is only polylogarithmic. To maintain the pseudo *kd*-tree, we develop algorithms for computing an approximate median level of a line arrangement, which itself is of great interest. We show that the computation of the approximate median level of a set of lines or line segments can be done in an online fashion smoothly, i.e., there are no expensive updates for any events. For practical consideration, we study the case in which there are speed-limit restrictions or smooth trajectory requirements. The maintenance of the pseudo *kd*-tree, as a consequence of the approximate median algorithm, can also adapt to those restrictions.

1 Introduction

Motion is ubiquitous in the physical world. Several areas such as digital battlefields, air-traffic control, mobile communication, navigation system, geographic information systems, call for storing moving objects into a data structure so that various queries on them can be answered efficiently; see [23, 25] and the references therein. The queries might relate either to the current configuration of objects or to a configuration in the future — in the latter case, we are asking to predict the behavior based on the current information. In the last few years there has been a flurry of activity on extending the capabilities of existing database systems to represent moving-object databases (MOD) and on indexing moving objects; see, e.g., [14, 22, 23]. The known data structures for answering range queries on moving points either do not guarantee a worst-case bound on the query time [26, 25, 19] or are too complicated [1, 15].

In this paper we develop kinetic data structures for *kd*-trees, a widely used data structure for answering various proximity queries in practice [10], which can efficiently answer range queries on moving points. The *kinetic data structure* framework, originally proposed by Basch *et al.* [5], has led to efficient algorithms for several geometric problems involving moving objects; see [13] and references therein. The main idea in the kinetic framework is that even though the objects move continuously, the relevant combinatorial structure of the data structure changes only at certain discrete times. Therefore one does not have to update the data structure continuously. The *kinetic updates* are performed on the data structure only when certain *kinetic events* occur; see [5, 13] for details.

Recall that a *kd*-tree on a set of points is a binary tree, each node v of which is associated with a subset S_v of points. The points in S_v are partitioned into two halves by a vertical or horizontal line at v and each half is associated with a child of v. The orientation of the partition line alternates as we follow a path down the tree. In order to develop a kinetic data structure for *kd*-trees, the first step is to develop a kinetic data structure for maintaining the median of a set of points moving on a line. In fact, this subroutine is needed for several other data structures.

Related work. The problem of maintaining the point of rank k in a set S of points moving on the x-axis is basically the same as computing the k-level in the arrangement of curves; the *k-level* in an arrangement of x-monotone curves is the set of edges of the arrangement that lie above exactly k curves [4]. If the points in S are moving with fixed speed, i.e., their trajectories are lines in the xt-plane, then the result by Dey [11] implies that the point of rank k changes $O(nk^{1/3})$ times; the best-known lower bound is $ne^{\Omega(\sqrt{\log k})}$ [24]. Only much weaker bounds are known if the trajectories of points are polynomial curves. Edelsbrunner and Welzl [12] developed an algorithm for computing a level in an arrangements of lines. By combining their idea with kinetic tournaments proposed by Basch *et al.* [6] for maintaining the maximum of a set of moving points on a line, one can construct an efficient kinetic data structure for maintaining the median of a set of moving points.

Since no near-linear bound is known on the complexity of a level, work has been done on computing an approximate median level. A δ-approximate median-level is defined as a x-monotone curve that lies between $(1/2-\delta)n$- and $(1/2+\delta)n$-levels. Edelsbrunner and Welzl [12] showed that a δ-approximate median level with at most $\lceil \lambda(n)/(\delta n) \rceil$ edges, where $\lambda(n)$ is the number of vertices on the median level of the arrangement, can be computed for line arrangements. Later Matoušek [16] proposed an algorithm to obtain a δ-approximate median level with constant complexity for an arrangement of n lines. However no efficient kinetic data structure is known for maintaining an approximate median of a set of moving points.

Because of their simplicity, *kd*-trees are widely used in practice and several variants of them have been proposed [3, 7, 17]. It is well known that a *kd*-tree can answer a two-dimensional orthogonal range query in $O(\sqrt{n}+k)$ time, where k is the number of points reported. Variants of *kd*-trees that support insertions and deletions are studied in [18, 9]. No kinetic data structures are known for kd-trees.

Agarwal *et al.* [1] were the first to develop kinetic data structures for answering range-searching queries on moving points. They developed a kinetic data structure that answers a two-dimensional range query in $O(\log n + k)$ time using $O(n \log n/(\log\log n))$ space, where k is the output size. The amortized cost of a kinetic event is $O(\log^2 n)$, and the total number of events is $O(n^2)$.[3] They also showed how to modify the structure in order to obtain a tradeoff between the query bound and the number of kinetic events. Kollios *et al.* [15] proposed a data structure for range searching among points moving on the x-axis, based on partition trees [3]. The structure uses $O(n)$ space and answers queries in $O(n^{1/2+\epsilon} + k)$ time, for an arbitrarily small constant $\epsilon > 0$. Agarwal *et al.* [1] extended the result to 2D. Unlike kinetic data structures, these data structures are time oblivious, in the sense that they do not evolve over time. However all these data structures are too complex to be used in practice.

In the database community, a number of practical methods have been proposed for accessing and searching moving objects (see [26, 25, 19] and the references therein). Many of these data structures index the trajectories of points either directly or by mapping to higher dimensions. These approaches are not efficient since trajectories do not cluster well. To alleviate this problem, one can parametrize a structure such as the R-tree, which partitions the points but allows the bounding boxes associated with the children of a node to overlap. To expect good query efficiency, the areas of overlap and the areas of the bounding boxes must be small. Although R-tree works correctly even when the overlap areas are too large, the query performance deteriorates in this case. Kinetic data structures based on R-tree were proposed in [25, 20] to handle range queries over moving points. Unfortunately, these structures also do not guarantee sublinear query time in the worst case.

Our results. Let $S = \{p_1, \ldots, p_n\}$ be a set of n points in $\mathbb{R}^1$, each moving independently. The position of a point p_i at time t is given by $p_i(t)$. We use $\bar{p}_i = \bigcup_t (p_i(t), t)$ to denote the *graph* of the trajectory of p_i in the xt-space. The user is allowed to change the trajectory of a point at any time. For a given parameter $\delta > 0$, we call a point x, not necessarily a point of S, a *δ-approximate median* if its rank is in the range $[(1/2 - \delta)n, (1/2 + \delta)n]$. We first describe an off-line algorithm that can maintain a δ-approximate median of S in a total time of $O((\mu/(n^2\delta^2) + 1/\delta)n \log n)$, where μ is the number of times two points swap position. We then show how a δ-median can be maintained on-line as the points of S move. Our algorithm maintains a point x^* on the x-axis, not necessarily one of the input points, whose rank is in the range $(1/2 \pm \delta)n$. As the input points move, x^* also moves, and its trajectory depends on the motion of input points. We show that the speed of this point is not larger than the fastest moving

[3] Agarwal *et al.* [1] actually describe their data structure in the standard two level I/O model, in which the goal is to minimize the memory access time. Here we have described their performance in the standard pointer-machine model.

point, and that our algorithm can be adapted so that the trajectory of x^* is C^k-continuous for any $k \geq 0$.

Next, let S be a set of n points in $\mathbb{R}^2$, each moving independently. We wish to maintain the *kd*-tree of S, so that *range searching* queries, i.e., given a query rectangle R at time t, report $|S(t) \cap R|$, can be answered efficiently, Even if the points in S are moving with fixed velocity, the *kd*-tree on S can change $\Theta(n^2)$ times, and there are point sets on which many of these events can cause a dramatic change of the tree, i.e., each of them requires $\Omega(n)$ time to update the tree. We therefore propose two variants of *kd*-trees, each of which answers a range query in time $O(n^{1/2+\varepsilon} + k)$, for any constant $\varepsilon > 0$, (k is the number of points reported), processes quadratic number of events, and spends polylogarithmic (amortized) time at each event. The first variant is called the *δ-pseudo kd-tree*, in which if a node has m points stored in the subtree, then each of its children has at most $(1/2 + \delta)m$ points in its subtree. In the second variant, called *δ-overlapping kd-tree*, the bounding boxes of the points stored in the subtrees of two children of a node can overlap. However, if the subtree at a node contains m points, then the overlapping region at that node contains at most δm points.

As the points move, both of the trees are maintained in the standard kinetic data structure framework [5]. The correctness of the tree structure is certified by a set of conditions, called *certificates*, whose failure time is precomputed and inserted into an event queue. At each certificate failure, denoted as an *event*, the KDS certification repair mechanism is invoked to repair the certificate set and possibly the tree structure as well. In the analysis of a KDS, we assume the points follow *pseudo-algebraic motions*. By this we mean that all certificates used by the KDS can switch from `TRUE` to `FALSE` at most a constant number of times. In some occasions, we may make stronger assumptions, like the usual scenario of linear, constant velocity motions. For both pseudo and overlapping *kd*-trees, we show that the set of certificates has linear size, that the total number of events is quadratic, and that each event has polylogarithmic amortized update cost. Dynamic insertion and deletion are also supported with the same update bound as for an event.

The pseudo *kd*-tree data structure uses our δ-approximate median algorithms as a subroutine. Unlike pseudo *kd*-trees, the children of a node of an overlapping *kd*-tree store equal number of points and the minimum bounding boxes of the children can overlap. However, maintaining overlapping *kd*-trees is somewhat simpler, and the analysis extends to pseudo-algebraic motion of points.

2 Maintaining the Median

2.1 Off-line maintenance

Matoušek [16] used a level shortcutting idea to compute a polygonal line, with complexity $O(1/\delta^2)$, which is a δ-approximate median. Here we extend his idea to a set S of line segments in the plane. Let μ denote the number of intersection points between the segments.

We divide the plane by vertical lines l_i's into strips such that there are at most $\delta^2 n^2/16$ intersections and at most $\delta n/4$ endpoints inside each strip. In addition, we can assume that each strip either has exactly $\delta^2 n^2/16$ intersections or exactly $\delta n/4$ endpoints, otherwise we can always enlarge the strip. The number of strips is $O(\mu/(n^2\delta^2) + 1/\delta)$. Along each l_i, we compute the median X_i of the intersections between S and l_i. We connect adjacent medians. We claim that the resulting polygonal line is a δ-approximate median level. Indeed, consider a strip bounded by l_i and l_{i+1}. We first delete the segments of S that have at least one endpoint inside the strip. Let $U \subseteq S$ be the set of line segments that intersect the segment X_iX_{i+1} and that lie above X_i at l_i. Similarly let $V \subseteq S$ be the set of line segments that intersect X_iX_{i+1} and that lie below X_i at l_i. $|U| = u$, $|V| = v$. Since X_i and X_{i+1} are on the exact median level and we have deleted at most $\delta n/4$ segments, $|u - v| \le \delta n/4$. Suppose $u \le v \le u + \delta n/4$. The intersection of a segment in U and a segment in V must be inside the strip (see Figure 1 (i)), therefore $uv \le \delta^2 n^2/16$, thereby implying that $u \le \delta n/4$ and $v \le \delta n/2$. Consequently, X_iX_{i+1} is intersected at most δn

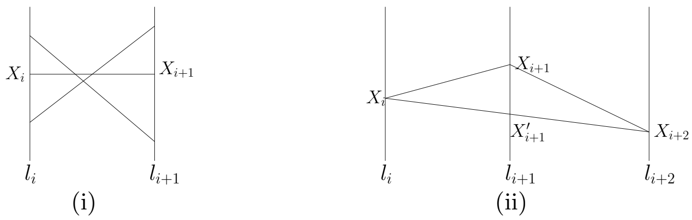

Fig. 1. (i) Approximate median level of n lines; (ii)When points change motion plan.

times, and thus it is a δ-approximation of the median level between l_i and l_{i+1}. Computing the partition line l_i involves computing the kth leftmost intersection in an arrangement. This can be done in $O(n \log n)$ time, using the slope-selection algorithm [8]. Computing a median along a cut line costs $O(n)$. So the total computational cost is $O((\mu/(n^2\delta^2) + 1/\delta)n \log n)$.

Theorem 1. *Let S be a set of n segments in the plane with a total of μ intersection points. A δ-approximate median level of S with at most $O(\mu/(n^2\delta^2) + 1/\delta)$ vertices can be computed in $O((\mu/(n\delta^2) + n/\delta) \log n)$ time.*

2.2 On-line maintenance

In previous subsection we assume that we know the motion plan of the points beforehand and compute the approximate median ahead of time. However, when the points change their motion plan, we want to maintain the approximate median online and distribute the amount of work more uniformly over time. Assume that the total number of motion plan changes over time is only linear in the number of points. We will show that the approximate median can be maintained smoothly, with linear number of events and polylogarithmic time update cost per event.

Consider the time-space plane. As in Theorem 1, the plane is divided by vertical lines l_i at time t_i into strips. We maintain the invariant that each strip either contains at most $\delta^2 n^2/36$ vertices or $\delta n/3$ flight plan changes. Also if the strip contains less than $\delta n/3$ flight plan changes, then it contains at least $\delta^2 n^2/72$ vertices. We approximate the median level by computing a polygonal line $X_1 X_2 ... X_m$ in which X_i lies on l_i. We can see that the total number of strips is bounded by $O(1/\delta^2)$ since there are only linear number of flight plan changes in total.

However, instead of computing the $O(1/\delta^2)$ cut lines all at one time, we compute them one by one. We begin with computing the median on the two leftmost cut lines l_1 and l_2 (l_1 corresponds to the starting time). There are $\delta^2 n^2/72$ vertices between l_1 and l_2. The approximate median then moves along line segment $X_1 X_2$. This preprocessing costs $O(n \log n)$. Then we compute the position of the cut line l_{i+2} and the median along l_{i+2} gradually, as we are moving along $X_i X_{i+1}$. The position of l_{i+2} is computed such that there are $\delta^2 n^2/72$ vertices between l_{i+1} and l_{i+2}. So X_{i+2} is our prediction of the median on l_{i+2}, assuming there is no flight plan change between t_{i+1} and t_{i+2}. The cost of computing l_{i+2} and X_{i+2}, which is a slope-selection problem, is amortized over the time interval $[t_i, t_{i+1}]$, using the technique proposed by Agarwal *et al.* [2]. In an online version, we need to accommodate the flight plan changes in the future. If there are $\delta n/6$ motion plans changes before we reach l_{i+1}, we start another session to compute X_{i+2} based on the current (changed) trajectories. If we reach X_{i+1} before another $\delta n/6$ motion updates, we switch to the segment $X_{i+1} X_{i+2}$ computed from the first construction. Otherwise, if we see another $\delta n/6$ motion plan changes (i.e., a total of $\delta n/3$ changes since t_i) before we reach l_{i+1}, then we just stop and move l_{i+1} to the current position and X_{i+1} to the current position of the approximate median. The next line segment that the approximate median needs to follow is $X_{i+1} X_{i+2}$, where X_{i+2} is the value returned by the second computation (the one initiated after $\delta n/6$ motion-plan changes). Thus inside one strip, the number of intersections is at most $\delta^2 n^2/36$, and the number of flight plan changes is at most $\delta n/3$. Next we prove that the polygonal line we compute is a δ-approximation of the median level.

The only problem is that if some points change their flight plans between time t_i and t_{i+1}, then X_{i+1}, which is computed before t_i, is no longer the real median X'_{i+1}. See Figure 1 (ii). However, only the points between X_{i+1} and X'_{i+1} can cross $X_{i+1} X_{i+2}$ without crossing $X'_{i+1} X_{i+2}$. Since there are only at most $\delta n/3$ flight plan changes, there are at most $\delta n/3$ points between X_{i+1} and X'_{i+1}. Since $X'_{i+1} X_{i+2}$ is crossed by at most $2\delta n/3$ lines, $X_{i+1} X_{i+2}$ is still a δ-approximation.

Theorem 2. *Let S be a set of n points, each moving with a constant velocity. Suppose the flight paths of S change a total of m times. Then a δ-approximate median of S can be maintained in an online fashion so that the median moves* *continuously with at most $O(1/\delta^2)$ flight plan changes. The total maintenance cost is $O((n+m)\log n/\delta^2)$. Each flight plan change costs polylogarithmic update time.*

Remark. Finding the next cut line in Theorem 2 is based on complex algorithms and data structures (e.g., slope selection). We can use a considerably simpler randomized algorithm, based on the random-sampling technique, for computing the next cut line. We omit the details from this version of the paper.

2.3 Approximate median with speed limit

In many applications like mobile networks and spatial databases, the maximum velocity of the points is restricted. We'll show an approximate median that moves no faster than the fastest point.

Lemma 1. *The approximate median computed above cannot move faster than the maximum speed of the n points, if the points do not change flight plans.*

Proof. Observe that the approximate median is composed of line segments connecting two median points X and Y at different time steps. So if there is a line l crossing XY and going up, there must be another line l' crossing XY and going down. So the slope of XY is bounded by the slope of l and l'. If XY is not crossed by any lines, X and Y lie on the same line as they are both medians. Hence, the approximate median moves along one of the input points between X and Y. This proves the lemma.

If the points can change their flight plan, our scheme in Theorem 2 does not guarantee that the approximate median moves within the limit restriction. But we can adapt it as follows. Let X_i and X'_i denote the precomputed and exact median, respectively. We let the approximate median to move toward the precomputed median. If we can reach there without violating the speed constraint, then everything is fine. Otherwise, we just move toward it with the maximum speed. Figure 2 (i) shows the second case. W.l.o.g., we assume X_{i+1} is above

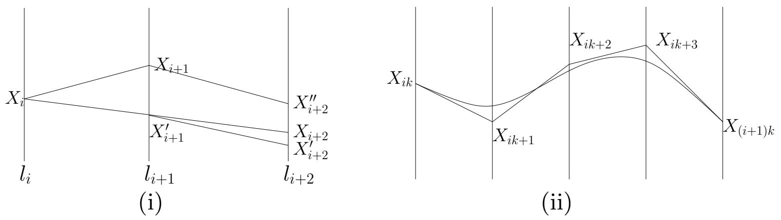

Fig. 2. (i) Speed restricted median; (ii) Smooth medians.

X'_{i+1}. Assume the approximate median gets to the point X''_{i+2} at time t_{i+2}. Since the approximate median moves with the maximum speed, the points above X_{i+1} will stay above X''_{i+2}. So at most $\delta n/3$ points, the ones that lie between X_{i+1} and X'_{i+1}, can cross $X_{i+1}X''_{i+2}$ without crossing $X'_{i+1}X_{i+2}$. Following the argument in Theorem 2, $X'_{i+1}X_{i+2}$ is crossed by at most $2\delta n/3$ lines. So $X_{i+1}X''_{i+2}$ is a δ-approximate median level. There are $n/2$ points below X'_{i+1} and X'_{i+2}, respectively, so the number of points between X''_{i+2} and X'_{i+2} can only be less than the number of points between X_{i+1} and X'_{i+1}. This imples that we can continue this process for the next time interval and we obtain the following result.

Theorem 3. *Let S be a set of n points, each moving with fixed velocity. We can maintain a δ-approximate median of S that preserves all the properties of Theorem 2 and that cannot move faster than the maximum speed of the input points.*

2.4 Smooth medians

For a set of moving points, the approximate median we computed above, follows a polygonal line composed of line segments. Although the approximate median moves continuously, it may have to make sharp turns. In practice, a smooth trajectory is preferred. A curve is said to have C^k continuity if its k-th derivative is continuous. The idea is to use Bézier interpolation on the medians along the vertical lines, see Figure 2 (ii). We will give the theorem whose proof is omitted.

Theorem 4. *Given an arrangement of n lines in the plane, we can compute a curve with C^k continuity which is a δ-approximate median level. The curve is determined by $O(k^2/\delta^2)$ control points and is computed in $O(k^2 n \log n/\delta^2)$ time.*

3 Pseudo *kd*-tree

Overmars [18] proposed the pseudo *kd*-tree as a dynamic data structure for range searching that admits efficient insertion and deletion of points.

Definition. A *δ-pseudo kd-tree* is defined to be a binary tree created by alternately partitioning the points with vertical and horizontal lines, so that for each node v with m points in the subtree, there are at most $(1/2 + \delta)m$ points in the subtrees rooted at the children of v. A δ-pseudo *kd*-tree is an almost balanced tree with depth $\log_{2/(1+2\delta)} n$. We denote by $d(u)$ the depth of a node u. The same analysis as for the standard *kd*-tree implies that a range query in a δ-pseudo *kd*-tree can be answered in $O(n^{1/2+\varepsilon} + k)$ time, k is the number of points in the answer and $\varepsilon = \log_{(1/2+\delta)^2} 2 - 1/2$.

Maintaining the pseudo kd-tree. One way to maintain the pseudo *kd*-tree is to maintain the input points sorted by their x- as well as y-coordinates and update the tree when a point crosses the partition line. Maintaining the points in two sorted lists will generate $\Theta(n^2)$ events. The subtree is rebuilt if the number of points in one child exceeds fraction $1/2 + \delta$. Each event has an amortized update cost of $O(\log n)$. Another way of maintaining the pseudo kd-tree is to use the dynamic data structure proposed by Overmars [18]. However, it maintains a forest of kd-trees instead of a single tree. Here, we will show how to maintain a single kd-tree with only polylogarithmic update cost per event without any rebuilding ,assume the points move with constant velocity.

To maintain a pseudo kd-tree, we need to maintain a hierarchical partition of the points, which always supports a δ-pseudo kd-tree. Since points are inserted into or deleted from a subtree, the trajectories of the points stored in a node are actually line segments. Maintaining the partition line of a node involves finding the approximate median of a set of line segments in an online fashion. The idea for online maintenance in Section 2.2 works here as well. The difference is, points may come in and out of the range of a node. So inside one strip the number of intersections, endpoints of the line segments and flight plan changes should be bounded all at the same time. The endpoints of the line segments can be treated in the same way as the events of motion plan changes we described before. We omit the details here and prove the following lemma and theorem in an off-line setting. We define a *δ-approximate partition line* of depth k to be a δ-approximate median level in the arrangement of the trajectories of points stored in the subtree of a node at depth k. Let V^k denote the set of nodes at depth k.

Lemma 2. *There exists a set of δ-approximate partition lines of V^k with total complexity $O(k/(\delta\alpha^k)^2)$, and it can be computed in $O((kn\log n)/(\delta\alpha^k)^2)$ time, where $\alpha = 1/2 - \delta$.*

Proof. A key observation is that the combination of the trajectories of all the points in V_k is the arrangement of n lines. So the total number of intersections of the segments in V_k is bounded by $O(n^2)$. Assume a node of depth k has n_k points associated with it. Since the child of a node with m points has at least $(1/2-\delta)m$ points, we have $n_k \geq n\alpha^k$ where $\alpha = 1/2 - \delta$. Let s_k be the total number of line segments in V^k, and let c_k be the total complexity of the δ-approximate partition lines for V^k. By Theorem 1, we have that $c_k = O((n/(\delta n_k))^2 + s_k/(\delta n_k))$. When a point crosses the partition line at a node of depth k, it ends the trajectory in the old child and begins a trajectory in the new child. Therefore we have $s_{k+1} = 2c_k \times \delta n_k = O(n^2/(\delta n_k)) + 2s_k$. By induction, we get $s_k = O(kn/(\delta\alpha^k))$, $c_k = O(k/(\delta^2\alpha^{2k}))$. The running time is bounded in the same way as in Theorem 1.

An event happens when a point crosses a partition line. Updating the pseudo tree involves moving a point from one subtree to the other, which costs $\log_{2/(1+2\delta)} n$. The number of events of depth k is bounded by $s_{k+1}/2$. So the total number of events is $O((n^2/\delta)\log_{2/(1+2\delta)} n)$. In summary, we have

Theorem 5. *For a set of n moving points on the plane, each moving with a constant velocity, we can find a moving hierarchical partition of the plane which supports a δ-pseudo kd-tree at any time. The number of events in the kinetic data structure is $O((n^2/\delta)\log_{2/(1+2\delta)} n)$ in total. Update cost for each event is $O(\log_{2/(1+2\delta)} n)$.*

4 Overlapping *kd*-tree

In the second variant of the *kd*-tree, the bounding boxes associated with the children of a node can overlap.

Definition. Assume $S(v)$ is the set of points stored in the subtree of a node v, $B(v)$ is the minimum bounding box of $S(v)$, and $d(v)$ is the depth of v. Define $\sigma(w) = B(u) \cap B(v)$ to be the *overlapping region* of two children u and v of node w. A *δ-overlapping kd-tree* is a perfectly balanced tree so that for each node w with m points in the subtree, there can be at most δm points in the overlapping region $\sigma(w)$, $0 < \delta < 1$. The overlapping *kd*-tree has linear size and depth $O(\log n)$. We call a node *x-partitioned* (*y-partitioned*) if it is partitioned by a vertical (resp. horizontal) line. An orthogonal range query can be answered in the same way as in the standard *kd*-tree.

Lemma 3. *Let ℓ be a vertical (or horizontal) line, and let v, w be two x-partitioned (y-partitioned) nodes of a δ-overlapping kd-tree such that w is a descendant of v. If ℓ intersects both $\sigma(v)$ and $\sigma(w)$, then $d(w) \geq d(v) + \log_2 \gamma$, where $\gamma = (1 - 2\delta)/(2\delta)$.*

Theorem 6. *A range query in a δ-overlapping kd-tree can be answered in $O(n^{1/2+\varepsilon} + k)$ time, where $\varepsilon = \log_\gamma 2$, $\gamma = (1 - 2\delta)/(2\delta)$, k is the number of points reported.*

Proof. As for the standard *kd*-tree, it suffices to bound the number of nodes whose bounding boxes intersect a vertical line ℓ. Let v be a x-partitioned node at depth k, and let n_v be the points stored in the subtree rooted at v. The query procedure visits both children and thus all four grandchildren of v only if ℓ intersects $\sigma(v)$. Otherwise, ℓ intersects at most two grandchildren of v. The granchildren of v are also x-partitioned. By Lemma 3, ℓ does not intersect $\sigma(w)$ for any descendent w of v at depth less than $k + \log_2 \gamma$. An easy calculation shows that ℓ intersects at most $8\sqrt{\gamma}$ nodes of depth at most $k + \log_2 \gamma$ and at most $2\sqrt{\gamma}$ nodes of depth $k + \log_2 \gamma$. On the other hand, any descendent of v at depth $k + \log_2 \gamma$ has at most n_v/γ points. Let $C(n_v)$ denote the number of nodes intersected by ℓ in the subtree rooted at a x-partitioned node that contains at most n_v points. Then we obtain the following recurrence

$$C(n_v) \leq 2\sqrt{\gamma} C(n_v/\gamma) + 8\sqrt{\gamma}$$

whose solution is $C(n_v) = O(n_v^{1/2+\varepsilon})$, where $\varepsilon = \log_\gamma 2$.

Maintaining the overlapping kd-tree. Maintaining the δ-overlapping *kd*-tree mainly involves tracking the number of points in the overlapping region. Consider the root of the *kd*-tree, assume n points are divided into $n/2$ red points and $n/2$ blue points by a vertical line at the beginning. W.l.o.g, assume red points have bigger coordinates. Consider the time-space plane, the goal is to track the depth of the lower(upper) envelope of red(blue) curves in the arrangement of blue(red) curves. An event happens when those two numbers add up to δn. So at least

$\delta n/2$ points in the overlapping region have the same color, suppose they are red. Then the highest blue point must be above at least $\delta n/2$ red points. We define a red-blue inversion to be the intersection of a red curve and a blue curve. So there must be at least $\Omega(\delta n)$ red-blue inversions since the last recoloring. The number of recoloring events in the root of the *kd*-tree is bounded by $O(n/\delta)$, if the points follow pseudo-algebraic motion.

Since the combination of the trajectories of all the points in nodes of depth k becomes the arrangement of the trajectories of n points, the total number of recoloring events in depth k is bounded by $O(n2^k/\delta)$, using the same argument as above. When an event happens at depth k, we rebuild the subtree, which costs $O(n \log n/2^k)$. So the total update cost sumed up over all events is $O(n^2 \log n/\delta)$. The amortized cost for one event is therefore $O(\log n)$. Putting everything together, we obtain the following.

Theorem 7. *Let S be a set of n points moving in the plane, and let $\delta > 0$ be a constant. We can maintain a δ-overlapping kd-tree by spending $O(\log n)$ amortized time at each event. Under pseudo-algebraic motion of S, the number of events is $O(n^2/\delta)$.*

Acknowledgements: The authors wish to thank John Hershberger and An Zhu for numerous discussions and for their contribution to the arguments in this paper. Research by all three authors is partially supported by NSF under grant CCR-00-86013. Research by P. Agarwal is also supported by NSF grants EIA-98-70724, EIA-01-31905, and CCR-97-32787, and by a grant from the U.S.–Israel Binational Science Foundation. Research by J. Gao and L. Guibas is also supported by NSF grant CCR-9910633 and grants from the Stanford Networking Research Center, the Okawa Foundation, and the Honda Corporation.

References

1. P. K. Agarwal, L. Arge, and J. Erickson. Indexing moving points. In *Proc. Annu. ACM Sympos. Principles Database Syst.*, 2000. 175–186.
2. P. K. Agarwal, L. Arge, and J. Vahrenhold. Time responsive external data structures for moving points. In *Workshop on Algorithms and Data Structures*, pages 50–61, 2001.
3. P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, volume 223 of *Contemporary Mathematics*, pages 1–56. American Mathematical Society, Providence, RI, 1999.
4. P. K. Agarwal and M. Sharir. Arrangements and their applications. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 49–119. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.
5. J. Basch, L. Guibas, and J. Hershberger. Data structures for mobile data. *J. Alg.*, 31(1):1–28, 1999.
6. J. Basch, L. J. Guibas, and G. D. Ramkumar. Reporting red-blue intersections between two sets of connected line segments. In *Proc. 4th Annu. European Sympos. Algorithms*, volume 1136 of *Lecture Notes Comput. Sci.*, pages 302–319. Springer-Verlag, 1996.

7. J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.
8. R. Cole, J. Salowe, W. Steiger, and E. Szemerédi. An optimal-time algorithm for slope selection. *SIAM J. Comput.*, 18(4):792–810, 1989.
9. W. Cunto, G. Lau, and P. Flajolet. Analysis of kdt-trees: kd-trees improved by local reorganisations. *Workshop on Algorithms and Data Structures (WADS'89)*, 382:24–38, 1989.
10. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.
11. T. K. Dey. Improved bounds on planar k-sets and k-levels. In *IEEE Symposium on Foundations of Computer Science*, pages 165–161, 1997.
12. H. Edelsbrunner and E. Welzl. Constructing belts in two-dimensional arrangements with applications. *SIAM J. Comput.*, 15:271–284, 1986.
13. L. J. Guibas. Kinetic data structures — a state of the art report. In P. K. Agarwal, L. E. Kavraki, and M. Mason, editors, *Proc. Workshop Algorithmic Found. Robot.*, pages 191–209. A. K. Peters, Wellesley, MA, 1998.
14. R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis. A foundation for representing and querying moving objects. *ACM Trans. Database Systems*, 25(1):1–42, 2000.
15. G. Kollios, D. Gunopulos, and V. J. Tsotras. On indexing mobile objects. In *Proc. Annu. ACM Sympos. Principles Database Syst.*, pages 261–272, 1999.
16. J. Matoušek. Construction of ϵ-nets. *Discrete Comput. Geom.*, 5:427–448, 1990.
17. J. Nievergelt and P. Widmayer. Spatial data structures: Concepts and design choices. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 725–764. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.
18. M. H. Overmars. *The Design of Dynamic Data Structures*, volume 156 of *Lecture Notes Comput. Sci.* Springer-Verlag, Heidelberg, West Germany, 1983.
19. D. Pfoser, C. J. Jensen, and Y. Theodoridis. Novel approaches to the indexing of moving object trajectories. In *Proc. 26th Intl. Conf. Very Large Databases*, pages 395–406, 2000.
20. C. M. Procopiuc, P. K. Agarwal, and S. Har-Peled. Star-tree: An efficent self-adjusting index for moving points. In *Proc. 4th Workshop on Algorithm Engineering and Experiments*, 2002.
21. A. Rockwood and P. Chambers. *Interactive Curves and Surfaces: A Multimedia Tutorial on CAGD.* Morgan Kaufmann Publishers, Inc., 1996.
22. A. P. Sistla and O. Wolfson. Temporal conditions and integrity constraints in active database systems. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 269–280, 1995.
23. A. P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. In *Proc. Intl Conf. Data Engineering*, pages 422–432, 1997.
24. G. Tóth. Point sets with many k-sets. *Discrete & Computational Geometry*, 26(2):187–194, 2001.
25. S. Šaltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 331–342, 2000.
26. O. Wolfson, A. P. Sistla, S. Chamberlain, and Y. Yesha. Updating and querying databases that track mobile units. *Distributed and Parallel Databases*, pages 257–287, 1999.