

Kinetic kd-Trees

Mohammad Ali Abam*

Mark de Berg*

Bettina Speckmann*

Abstract

We propose a simple variant of kd-trees, called rank-based kd-trees, for sets of points in \mathbb{R}^d . We show that a rank-based kd-tree, like an ordinary kd-tree, supports range search queries in $O(n^{1-1/d} + k)$ time, where k is the output size. The main advantage of rank-based kd-trees is that they can be efficiently kinetized: the KDS processes $O(n^2)$ events in the worst case, assuming that the points follow constant-degree algebraic trajectories, each event can be handled in $O(\log n)$ time, and each point is involved in $O(1)$ certificates.

1 Introduction

Background. Due to the increased availability of GPS systems and to other technological advances, motion data is becoming more and more available in a variety of application areas: air-traffic control, mobile communication, geographic information systems, and so on. In many of these areas, the data are moving points in 2- or higher-dimensional space, and what is needed is to store these points in such a way that *range queries* (“Report all the points lying currently inside a query range”) can be answered efficiently. Hence, there has been a lot of work on developing data structures for moving point data, both in the database community as well as in the computational-geometry community.

Within computational geometry, the standard model for designing and analyzing data structures for moving objects is the kinetic-data-structure framework introduced by Basch et al. [3]. A *kinetic data structure* (KDS) maintains a discrete attribute of a set of moving objects—the convex hull, for example, or the closest pair—where each object has a known motion trajectory. The basic idea is, that although all objects move continuously, there are only certain discrete moments in time when the combinatorial structure of the attribute—the ordered set of convex-hull vertices, or the pair that is closest—changes. A KDS contains a set of *certificates* that constitutes a proof

that the maintained structure is correct. These certificates are inserted in a priority queue based on their time of expiration. The KDS then performs an event-driven simulation of the motion of the objects, updating the structure whenever an *event* happens, that is, when a certificate fails—see the surveys by Guibas [6, 7] for more details.

Related work. There are several papers that describe KDS’s for the orthogonal range-searching problem, where the query range is an axis-parallel box. Basch et al. [4] kinetized d -dimensional range trees. Their KDS supports range queries in $O(\log^d n + k)$ time and uses $O(n \log^{d-1} n)$ storage. If the points follow constant-degree algebraic trajectories then their KDS processes $O(n^2)$ events and each event can be handled in $O(\log^{d-1} n)$ time. In the plane, Agarwal et al. [1] obtained an improved solution: their KDS supports range searching queries in $O(\log n + k)$ time, it uses $O(n \log n / \log \log n)$ storage, and the amortized cost of processing an event is $O(\log^2 n)$.

Although these results are nice from a theoretical perspective, their practical value is limited for several reasons. First of all, they use super-linear storage, which is often undesirable. Second, they can perform only orthogonal range queries; queries with other types of ranges or nearest-neighbor searches are not supported. Finally, especially the solution by Agarwal et al. [1] is rather complicated. Indeed, in the setting where the points do not move, the static counterparts of these structures are usually not used in practice. Instead, simpler structures such as quadtrees, kd-trees, or bounding-volume hierarchies (R-trees, for instance) are used. In this paper we consider one of these structures, namely the kd-tree.

Kd-trees were initially introduced by Bentley [5]. A kd-tree for a set of points in the plane is obtained recursively as follows. At each node of the tree, the current point set is split into two equal-sized subsets with a line. When the depth of the node is even the splitting line is orthogonal to the x -axis, and when it is odd the splitting line is orthogonal to the y -axis. In d -dimensional space, the orientations of the splitting planes cycle through the d axes in a similar manner. Kd-trees use $O(n)$ storage and support range searching queries in $O(n^{1-1/d} + k)$ time, where k is the number of reported points. Maintaining a standard kd-tree kinetically is not efficient. The problem is that a single event—two points swapping their order

*Department of Mathematics and Computer Science, TU Eindhoven, {mabam, mdberg, speckman}@win.tue.nl. M.A. was supported by the Netherlands’ Organisation for Scientific Research (NWO) under project no. 612.065.307. M.d.B. was supported by the Netherlands’ Organisation for Scientific Research (NWO) under project no. 639.023.301.

on x - or y -coordinate—can have a dramatic effect: a new point entering the region corresponding to a node could mean that almost the entire subtree must be restructured. Hence, a variant of the kd-tree is needed when the points are moving.

Agarwal et al. [2] proposed two such variants: the δ -pseudo kd-tree and the δ -overlapping kd-tree. In a δ -pseudo kd-tree each child of a node ν can be associated with at most $(1/2 + \delta)n_\nu$ points, where n_ν is the number of points in the subtree of ν . In a δ -overlapping kd-tree the regions corresponding to the children of ν can overlap as long as the overlapping region contains at most δn_ν points. Both kd-trees support range queries in time $O(n^{1/2+\varepsilon} + k)$, where k is the number of reported points. Here ε is a positive constant that can be made arbitrarily small by choosing δ appropriately. These KDS's process $O(n^2)$ events if the points follow constant-degree algebraic trajectories. Although it can take up to $O(n)$ time to handle a single event, the amortized cost is $O(\log n)$ time per event. Neither of these two solutions are completely satisfactory: their query time is worse by a factor $O(n^\varepsilon)$ than the query time in standard kd-trees, there is only a good amortized bound on the time to process events, and only a solution for the 2-dimensional case is given. The goal of our paper is to develop a kinetic kd-tree variant that does not have these drawbacks.

Our results. We present a new and simple variant of the standard kd-tree for a set of n points in d -dimensional space. Our *rank-based kd-tree* supports orthogonal range searching in time $O(n^{1-1/d} + k)$ and it uses $O(n)$ storage—just like the original. But additionally it can be kinetized easily and efficiently. The rank-based kd-tree processes $O(n^2)$ events in the worst case if the points follow constant-degree algebraic trajectories and each event can be handled in $O(\log n)$ worst-case time. Moreover, each point is involved only in a constant number of certificates. Thus we improve the both the query time and the event-handling time as compared to the planar kd-tree variants of Agarwal et al. [2], and in addition our results work in any fixed dimension.

2 Rank-based kd-trees

Let \mathcal{P} be a set of n points in \mathbb{R}^d and let us denote the coordinate-axes with x_1, \dots, x_d . To simplify the discussion we assume that no two points share any coordinate, that is, no two points have the same x_1 -coordinate, or the same x_2 -coordinate, etc. (Of course coordinates will temporarily be equal when two points swap their order, but the description below refers to the time intervals in between such events.) In this section we describe a variant of a kd-tree for \mathcal{P} , the *rank-based kd-tree*. A rank-based kd-tree preserves all

main properties of a kd-tree and, additionally, it can be kinetized efficiently.

Before we describe the actual rank-base kd-tree for \mathcal{P} , we first introduce another tree, namely the *skeleton* of a rank-base kd-tree, denoted by $\mathcal{S}(\mathcal{P})$. Like a standard kd-tree, $\mathcal{S}(\mathcal{P})$ uses axis-orthogonal splitting hyperplanes to divide the set of points associated with a node. As usual, the orientation of the axis-orthogonal splitting hyperplanes is alternated between the coordinate axes, that is, we first split with a hyperplane orthogonal to the x_1 -axis, then with a hyperplane orthogonal to the x_2 -axis, and so on. Let ν be node of $\mathcal{S}(\mathcal{P})$. $h(\nu)$ is the splitting hyperplane stored at ν , $\text{axis}(\nu)$ is the coordinate-axis to which $h(\nu)$ is orthogonal, and $\mathcal{P}(\nu)$ is the set of points stored in the subtree rooted at ν . A node ν is called an x_i -node if $\text{axis}(\nu) = x_i$ and a node ω is referred to as an x_i -ancestor of a node ν if ω is an ancestor of ν and $\text{axis}(\omega) = x_i$. The first x_i -ancestor of a node ν is the x_i -parent(ν) of ν .

A standard kd-tree chooses $h(\nu)$ such that $\mathcal{P}(\nu)$ is divided roughly in half. In contrast, $\mathcal{S}(\mathcal{P})$ chooses $h(\nu)$ based on a range of ranks associated with ν , which can have the effect that the sizes of the children of ν are completely unbalanced. We now explain this construction in detail. We use d arrays $\mathcal{A}_1, \dots, \mathcal{A}_d$ to store the points of \mathcal{P} in d sorted lists; the array $\mathcal{A}_i[1, n]$ stores the sorted list based on the x_i -coordinate. As mentioned above, we associate a range $[r, r']$ of ranks with each node ν , denoted by $\text{range}(\nu)$, with $1 \leq r \leq r' \leq n$. Let ν be an x_i -node. If x_i -parent(ν) does not exist, then $\text{range}(\nu)$ is equal to $[1, n]$. Otherwise, if ν is contained in the left subtree of x_i -parent(ν), then $\text{range}(\nu)$ is equal to the first half of $\text{range}(x_i\text{-parent}(\nu))$, and if ν is contained in the right subtree of x_i -parent(ν), then $\text{range}(\nu)$ is equal to the second half of $\text{range}(x_i\text{-parent}(\nu))$. If $\text{range}(\nu) = [r, r']$ then $\mathcal{P}(\nu)$ contains at most $r' - r + 1$ points. We explicitly ignore all nodes (both internal as well as leaf nodes) that do not contain any points, they are not part of $\mathcal{S}(\mathcal{P})$, independent of their range of ranks. A node ν is a leaf of $\mathcal{S}(\mathcal{P})$ if $\text{range}(\nu) = [j, j]$ for some j . Clearly a leaf contains exactly one point, but not every node that contains only one point is a leaf. (We could prune these nodes, which always have a range $[j, k]$ with $j < k$, but we chose to keep them in the skeleton for ease of description.) If ν is not a leaf and $\text{axis}(\nu) = x_i$ then $h(\nu)$ is defined by the point whose rank in \mathcal{A}_i is equal to the median of $\text{range}(\nu)$.

We construct $\mathcal{S}(\mathcal{P})$ incrementally by inserting the points of \mathcal{P} one by one. Let p be the point that we are currently inserting into the tree and let ν be the last node visited by p ; initially $\nu = \text{root}$. Depending on which side of $h(\nu)$ contains p we select the appropriate child ω of ν to be visited next. If ω does not exist, then we create it and compute $\text{range}(\omega)$ as described above. We recurse with $\nu = \omega$ un-

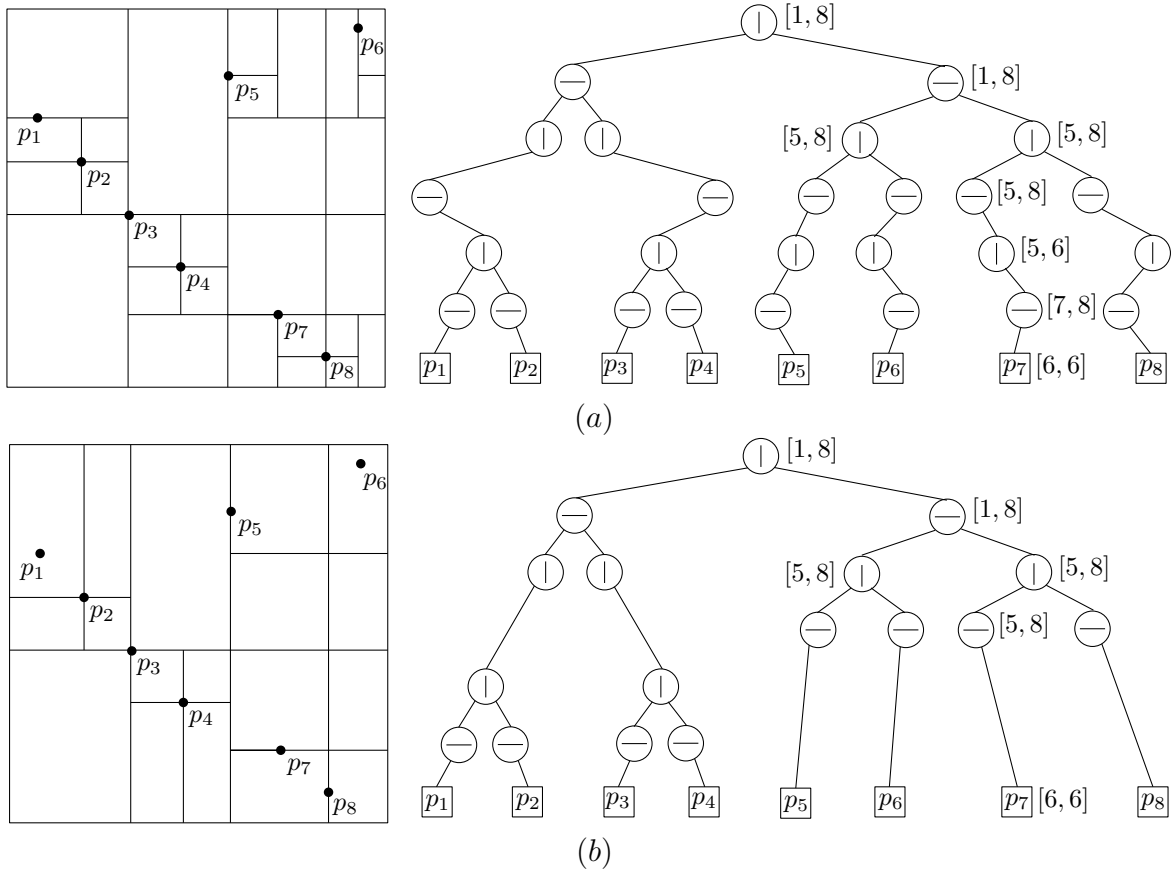


Figure 1: (a) The skeleton of a rank-based kd-tree and (b) the rank-based kd-tree itself.

til $\text{range}(\nu) = [j, j]$ for some j . We always reach such a node after $d \log n$ steps, because the length of $\text{range}(\nu)$ is a half of the length of $\text{range}(x_i\text{-parent}(\nu))$ and $\text{depth}(\nu) = \text{depth}(x_i\text{-parent}(\nu)) + d$ for an x_i -node ν . Figure 1(a) illustrates $\mathcal{S}(\mathcal{P})$ for eight points. Since the depth of each leaf is $d \log n$, the size of $\mathcal{S}(\mathcal{P})$ is $O(n \log n)$.

Lemma 1 *The depth of $\mathcal{S}(\mathcal{P})$ is $O(\log n)$ and the size of $\mathcal{S}(\mathcal{P})$ is $O(n \log n)$ for any fixed dimension d . $\mathcal{S}(\mathcal{P})$ can be constructed in $O(n \log n)$ time.*

A node $\nu \in \mathcal{S}(\mathcal{P})$ is *active* if and only if both its children exist, that is, both its children contain points. A node ν is *useful* if it is either active, or a leaf, or its first $d-1$ ancestors contain an active node. Otherwise a node is *useless*. We derive the rank-based kd-tree for \mathcal{P} from the skeleton by pruning all useless nodes from $\mathcal{S}(\mathcal{P})$. The parent of a node ν in the rank-based kd-tree is the first unpruned ancestor of ν in $\mathcal{S}(\mathcal{P})$. Roughly speaking, in the pruning phase every long path whose nodes have only one child each is shrunk to a path whose length is less than d . The rank-based kd-tree has exactly n leaves and each contains exactly one point of \mathcal{P} . Moreover, every node ν in the rank-based kd-tree is either active or it has an active ancestor among its first $d-1$ ancestors. The rank-

based kd-tree derived from Figure 1(a) is illustrated in Figure 1(b).

Lemma 2 *A rank-based kd-tree on a set of n points in \mathbb{R}^d has depth $O(\log n)$ and size $O(n)$.*

Proof. A rank-based kd-tree is at most as deep as its skeleton $\mathcal{S}(\mathcal{P})$. Since the depth of $\mathcal{S}(\mathcal{P})$ is $O(\log n)$ by Lemma 1, the depth of a rank-based kd-tree is also $O(\log n)$. To prove the second claim, we charge every node that has only one child to its first active ancestor—recall that each active node has two children. We charge at most $2(d-1)$ nodes to each active node, because after pruning there is no path in the rank-based kd-tree whose length is at least d and in which all nodes have one child. Therefore, to bound the size of the rank-based kd-tree it is sufficient to bound the number of active nodes. Let \mathcal{T} be a tree containing all active nodes and all leaves of the rank-based kd-tree. A node ν is the parent of a node ω in \mathcal{T} if and only if ν is the first active ancestor of ω in the rank-based kd-tree. Obviously, \mathcal{T} is a binary tree with n leaves where each internal node has two children. Hence, the size of \mathcal{T} is $O(n)$ and consequently the size of the rank-based kd-tree is $O(n)$. \square

Like a kd-tree, a rank-based kd-tree can be used to report all points inside a given orthogonal range search

query—the reporting algorithm is exactly the same. At first sight, the fact that the splits in our rank-based kd-tree can be very unbalanced may seem to have a big, negative impact on the query time. Fortunately this is not the case, since we can bound the number of cells intersected by an axis-parallel plane h . The following theorem summarizes our results.

Theorem 3 *A rank-based kd-tree for a set \mathcal{P} of n points in d dimensions uses $O(n)$ storage and can be built in $O(n \log n)$ time. An orthogonal range search query on a rank-based kd-tree takes $O(n^{1-1/d} + k)$ time where k is the number of reported points.*

The KDS. We now describe how to kinetize a rank-base kd-tree for a set of continuously moving points \mathcal{P} . The combinatorial structure of a rank-base kd-tree depends only on the ranks of the points in the arrays \mathcal{A}_i , that is, it does not change as long as the order of the points in the arrays \mathcal{A}_i remains the same. Hence it suffices to maintain a certificate for each pair p and q of consecutive points in every array \mathcal{A}_i , which fails when p and q change their order. Now assume that a certificate, involving two points p and q and the x_i -axis, fails at time t . To handle the event, we simply delete p and q and re-insert them in their new order. These deletions and insertions do not change anything for the other points, because their ranks are not influenced by the swap and the deletion and re-insertion of p and q . Hence the rank-based kd-tree remains unchanged except for a small part that involves p and q . A detailed description of this “small part” can be found below.

Deletion. Let ν be the first active ancestor of the leaf μ containing p —see Figure 2(a). Leaf μ and all nodes on the path from μ to ν must be deleted, since they do not contain any points anymore (they only contained p and p is now deleted). Furthermore, ν stops being active. Let ω be the first active descendent of ν . There are at most d nodes on the path from ν to ω . Since ν is not active anymore, any of the nodes on this path might become useless and hence have to be deleted.

Insertion. Let ν be the highest node in the rank-based kd-tree such that its region contains p and the region corresponding to its only child ω does not contain p —note that p cannot reach a leaf when we re-insert p , because the range of a leaf is $[j, j]$ for some j and there cannot be two points in this range. Let ν' and ω' be the nodes in $\mathcal{S}(\mathcal{P})$ corresponding to ν and ω . Let u' be the lowest node on the path from ν' to ω' whose region contains both region(ω') and p as illustrated in Figure 2(b)—note that we do not maintain $\mathcal{S}(\mathcal{P})$ explicitly but with the information maintained in ν and ω the path between ν' and ω' can

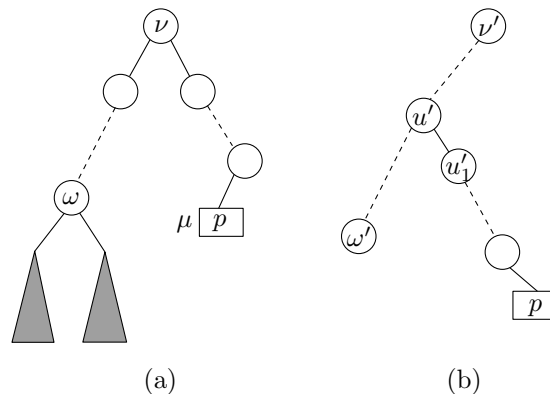


Figure 2: Inserting and deleting a point p .

be constructed temporarily. Because u' will become an active node, it must be added to the rank-based kd-tree and also every node on the path from u' to ω' must be added to the rank-based kd-tree if they are useful. From u' , the point p follows a new path u'_1, \dots, u'_k which is created during the insertion. All first $d - 1$ nodes in the list u'_1, \dots, u'_k and the leaf u'_k must be added to the rank-based kd-tree—note that $\text{range}(u'_k) = [j, j]$ for some j .

Theorem 4 *A kinetic rank-based kd-tree for a set \mathcal{P} of n moving points in d dimensions uses $O(n)$ storage and processes $O(n^2)$ events in the worst case, assuming that the points follow constant-degree algebraic trajectories. Each event can be handled in $O(\log n)$ time and each point is involved in $O(1)$ certificates.*

References

- [1] P. Agarwal, L. Arge, and J. Erickson. Indexing moving points. *Journal of Computer and System Sciences*, 66(1):207-243, 2003.
- [2] P. Agarwal, J. Gao, and L. Guibas. Kinetic medians and kd-trees. In *Proc. 10th European Symposium on Algorithms*, pages 5–16, Lecture Notes in Computer Science 2461, Springer Verlag, 2002.
- [3] J. Basch, L. Guibas, and J. Hershberger. Data structures for mobile data. *Journal of Algorithms*, 31:1–28, 1999.
- [4] J. Basch, L. Guibas, and L. Zhang. Proximity problems on moving points. In *Proc. 13th Symposium on Computational Geometry*, pages 344–351, 1997.
- [5] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.
- [6] L. Guibas. Kinetic data structures: A state of the art report. In *Proc. 3rd Workshop on Algorithmic Foundations of Robotics*, pages 191–209, 1998.
- [7] L. Guibas. Modeling motion. In J. Goodman and J. O’Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 1117–1134. CRC Press, 2nd edition, 2004.