An improvement in the build algorithm for Kd-trees using mathematical mean

Priyank Trivedi, Abhinandan Patni, Zeon Trevor Fernando and Tejaswi Agarwal

School of Computing Sciences and Engineering, VIT University, Chennai – 600048, Tamil Nadu, India

priyank.trivedi2010@vit.ac.in, abhinandan.patni2010@vit.ac.in, zeon.trevor2010@vit.ac.in, tejaswi.agarwal2010@vit.ac.in

Abstract

Querying forms a central part of dealing with data. Hence, it becomes imperative to have efficient data structures and query-search algorithms for data retrieval. Among the different types of data structures, in this paper, we have focused on 4 tree data structures. These are B, Kd, Range and Quad trees. We have made a comparative study of their build processes. Traditionally, the median of data-elements is used for the building of Kd trees. This gives a time complexity of O (k*nlogn) where n is the number of points to be sorted and k is the dimensionality of the points. We propose the building of Kd trees on the basis of mean of the data elements. The proposed algorithm has been shown to reduce the number of computations in the build process. The Kd trees produced by this methodology are balanced. Hence, this provides an improvement over the traditional building mechanism.

Keywords: Kd-tree, B tree, Range tree, Quad tree, median, mean, time complexity.

1. Introduction

Trees as a data structure has been widely been used in querying methods and extracting data from various sources. Apart from querying, trees have been used in space partitioning and clustering [1] algorithms. Kd-tree, introduced by Bentley in 1974 is a well-known space partitioning data structure for organizing points in k-dimensional space. Kd-tree, due to their versatility and wide range of application areas, are one of the most used techniques to generate efficient data structures for fast ray tracing [2] and are increasingly being utilized by researchers around the world. In Kd-trees have been dealt with varying complexity of the scene for ray tracing [3, 4].

In the process of utilizing Kd-trees for querying the performance bottleneck lies in the build algorithm. Therefore much work has gone in to enhance the Build process of the Kd tree algorithm. Bentley [5] proposed a median based build algorithm of Kd tree for which for larger inputs may produce an unbalanced Kd tree.

In this paper, we initially evaluate the build process of kd,B, Range and Quad trees and then propose a mathematical mean based Kd-tree build algorithm which produces a balanced Kd tree thereby enhancing the query time and traversal within the tree. The algorithm is experimentally proven for larger inputs; it builds a balanced Kd tree and is shown through a sample example later in the paper.
The rest of the paper is organized as follows. The next section describes the motivation and background study conducted on Kd tree and mentions about the various applications of Kd trees in a plethora of domains. Section 3 presents a detailed comparative study upon the various trees used for the purpose of querying. Section 4 explains the proposed mean based build process of the Kd tree data structure with Section 5 concluding the paper.

2. Motivation and Related Work

Kd tree has become one of the most widely used data structure in various domains. Kd tree has been used as a spatial data structure for rendering accelerators for both ray tracing [1] and rasterization [12] and forms an integral component of recent parallel real time ray tracers. Kd tree has been extensively used in computing surface area heuristic (SAH) that subdivides geometry into regions of small surface areas that contain many triangles [13,14]. Kd-trees have recently received a lot of attention and today, research is being undertaken in building them to be efficient, in traversing them quickly and optimizing their low-level implementation and memory layout.
Kd tree build process forms an integral part of the execution time complexity of Kd tree application. The traditional Kd tree [5] build process builds on the median and decides the axis split based on the median of the data set. Building a Kd tree using the median method generally leads to the creation of an unbalanced Kd tree, that is, internal nodes may have a leaf and another internal node as children, or internal nodes may only have a single child (which is an internal node).

Algorithm 1: BuildMedianKdTree(P, depth)

```
1. If(P contains only one point) then
      a. Return P
2. Else if (depth is even) then
      a. Split P into two subsets with a vertical line $l$ through the $x$- co-ordinate closest to the
         median of $x$-coordinate of the points in P. Let $P_1$ be the set of points to the left of l or on l,
         and let P2 be the set of points to the right of $l$.
3. Else
      a. Split P into two subsets with a horizontal line $l$ through the $y$-co-ordinate closest to
```

```
        the median of y-coordinate of the points in P. Let $P_1$ be the set of points below
        or on $l$, and let $P_2$ be the set of points above $l$
4.  $v_{left}$ = BuildMedianKdTree(P1,depth+1)
5.  $v_{right}$ = BuildMedianKdTree(P2,depth+1)
6.  Create a node v storing , make $v_{left}$ the left child of v, and make $v_{right}$ the right child of
    v.
7.  Return v.
```

The above algorithm for building a Kd tree has a subtle problem. Consider the points [(2, 3), (2, 4), (4,3)]. In this list there are no points with an x-value less than the median x-value (2). Similarly there is no point with a y-value less than the median value (3). Because of this, we cannot partition the list in two using either x-value or y-value, so the above algorithm would recur forever.

3. A Comparative study on B, Range and Quad trees

3. 1 B-Tree [6]: It is a tree data structure which allows insertions, deletions, sequential access and searches in logarithmic time. They are always height-balanced with all the leaf nodes at the same level. It keeps related records on the same disk page taking into consideration the locality of reference. B-tree is generally used in the databases and file systems. Insertion, Retrieval are done in $\log_m$ (I) where m is describes the page size of the device and I is the size of the index. Index refers to adjacent data items of fixed size, consisting of a key identifying a unique element in the index and a pointer to the record itself. Pages, on which the index is stored, are the nodes of the tree. Within each page the keys are sequentially arranged.

- **Cost of Insertion:** To analyze the cost of insertion we must consider the number of pages fetched from the secondary storage to the main memory to choose the appropriate leaf node and the number of pages written back to the secondary storage after modification.

Let us consider $P_{min}(P_{max})$ as the minimal(maximal) no. of pages fetched, and w_{min} and w_{max} as the minimum(maximum) no. of pages written.
For inserting a inserting a single key the least cost needed is when no page splits are needed.

$$\mathbf{P_{min} = h \; ; \; w_{min} = 1}$$

Worst case cost is when all the pages retrieved including the root node needs to split into two. Since the h pages would be retrieved, then a new root page would have to be written.

$$\mathbf{P_{max} = h \; ; \; w_{max} = 2h + 1}$$

Improvement: B-trees could apply a rotation mechanism to reduce the number of page splits. This rotation occurs when a leaf node is full, but one of its sibling nodes is not full. Instead of splitting the leaf node, we will move a few keys to its sibling to as to equally distribute the keys, adjusting the indices accordingly. Using the mechanism the storage utilization could be improved appreciably as nodes will be split only if they are full and their siblings are full.

- **The maximum fan-out of a B-tree** will be decided based on the maximum size of the node and the size of the key which is based on an index and the size of the pointer to the actual record. Generally each node of a B-tree is a disk page, and the maximum size of the node will depend on the page size which can be transferred from secondary storage to the main memory.

3.2 Range trees: The search complexity for range querying in k-d trees is $O(n^{1-1/k} + m)$ time, where n is the number of points stored in the tree, *m* is the number of the reported points, and *k* the dimension of the Kd tree.

In 1979, Jon Bentley [7] designed the range tree data-structure that would give faster query times. Using range trees, we can get query times of $O(\log^{d} n + k)$, where *n* is the number of points stored in the tree, *d* is the dimension of each point and *k* is the number of points reported by a given query.

A multi-dimensional range tree however is much more complex, in terms of building and storage. Every node denotes the 'd' dimensions of the attributes. From these nodes stem other binary trees which are of 'd-1' dimensionality. This goes on till we have a sorted array in the associated tree i.e. a one dimensional binary search tree. A range tree on a set of points in *d*-dimensions is a recursively defined multi-level binary search tree.

- **Build Process:** The following algorithm describes the building mechanism for a 2D tree. This can be further generalized to 'd' dimensions.

```
Algorithm 2: BuildRangeTree(P)
    1. Construct Associated Structure: Build a binary search tree $T_{assoc}$ on the set Py of y-
       coordinates in P.
    2. If(P contains only one point) then
        a. Create a leaf v storing this point, and make $T_{assoc}$ the associated structure of v.
    3. Else
```

```
        a. Split P into two subsets, $P_{left}$ and $P_{right}$, the subsets $\leq$ and $>$ the median x-
           coordinate $x_{mid.}$
        b. $v_{left}$ = BuildRangeTree($P_{left}$).
        c. $v_{right}$ = BuildRangeTree($P_{right}$).
    4. Create node v storing $x_{mid}$, make $v_{left}$ the left child of v, make $v_{right}$ the right child of v, and
       make $T_{assoc}$ the associated tree of v.
    5. Return v.
```

The construction algorithm takes O (n $\log^2$ n) time: **T (1) = O (1)**

T (n) = 2.T (n/2) + O (n log n) which solves to O (n $\log^2$ n) time which is more by a factor of **(log n)** when compared to O (n log n) of Kd trees.

- **Improvements:** There exist techniques such as [8] Fractional Cascading that can increase the search speed quite significantly. The first search requires logarithmic time, but the successive searches are faster. This has ensured that the search-complexity of range trees can be reduced to **O (log n + m + k)**.

3.3 Quadtrees: They are space partition trees in which each node consists of four children often used to partition a 2D region by recursive division into four arbitrary quadrants. H. Samet [9] gave an excellent introduction to quad trees and the use of spatial data structures in spatial databases.

Algorithm 3: BalancedQuadTree(v;T)

```
1.  Insert all leaves of T into a list L
2 .     while L is not empty
3.          do remove a leaf μ from L
4.              if μ has to be split
5.                  then split μ and insert four new leaves into L
6.                      check if μ had neighbors that now need to be split
                        and if so, insert them into L
```

For a quadtree of height h with n nodes, a balanced quadtree can be constructed in O (hn) time. Various enhancements in quadtrees like the compress quadtree [10], and higher dimensional octrees have been proposed for different applications.

Kd Vs Quad Tree

Kd trees are guaranteed to have a logarithmic depth, which contributes to the time complexity of a nearest neighbor query. In contrast, the depth of a quadtree is the maximum of the number of bits of precision of the input. Depth in a quadtree can be controlled theoretically, the time for which is shown to be logarithmic**.**

Advantages of quad trees include:

a. It can be converted from high resolution to lower resolution levels with each, maintaining multiple levels of detail.

b. Simultaneous mixed level of detail possible.

The main disadvantages of quad trees are:

a. The rotation of the object would lead to a completely different quadtree.

b. Recognizing and comparing similar shapes in quad trees are generally difficult.

3.4 A summary of build process time complexities

Tree	Build Process Time Complexity
B Tree	O(m * log (m*n))
Range Tree	O (n * $\log^2$ n)
Quad Tree	O (k * log n)
Kd Tree	O (n * log n)

Where m = order of the B tree

k = number of intersecting intervals (active cells) in Quad Trees

n = number of data nodes in the tree

4. Proposed Mean Based Build Algorithm

Algorithm 4: BUILDKDTREE (P, depth)

```
1. If(P contains only one point) then
       a. Return P
2. Else if (depth is even) then
3. Split P into two subsets with a vertical line l through the x- co-ordinate closest to the mean of x-coordinate of the points in P. Let $P_1$ be the set of points to the left of l or on l, and let $P_2$ be the set of points to the right of l.
4. Else
       a. Split P into two subsets with a horizontal line l through the y-co-ordinate closest to the mean of y-coordinate of the points in P. Let $P_1$ be the set of points below or on l, and let $P_2$ be the set of points above l
5. $v_{left}$ = BUILDKDTREE($P_1$,depth+1)
6. $v_{right}$ = BUILDKDTREE($P_2$,depth+1)
7. Create a node storing ,make $v_{left}$ the left child of v, and make $v_{right}$ the right child of v.
8. return v.
```

5. Experimental Results

In order to demonstrate the robustness of the proposed algorithm, a sample test is performed to build the Kd tree. The given sample test demonstrates the same.

P = (4, 2), (6, 7), (5, 3), (9, 8), (7, 3)

Figure1 shows the structure of the tree which is built after applying Algorithm 1-the original Kd-tree build process based on the median resulting in an unbalanced Kd-tree for the given data set input. Figure 2 shows the resultant tree formed using the mean based approach is balanced, reducing the querying time enhancing the overall performance of the application.

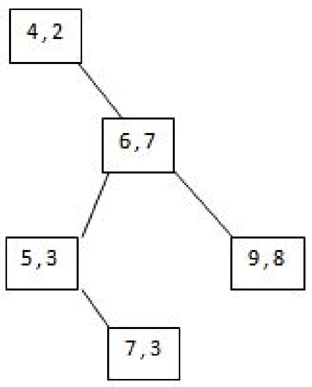

Figure 1 Median Based Kd Tree Build Process

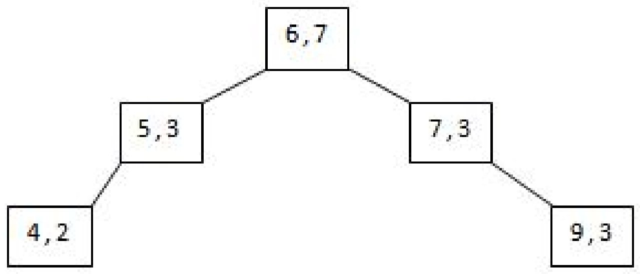

Figure 2: Resultant tree after applying the mean based build process

The construction of Kd-tree is governed by two constraints:

1. Selection of the plane which would divide the given data set into subset for spatial partitioning
2. Determination of the mean of the remaining points in the data set once the plane has been decided.

Let's analyze the construction time of a 2-dimensional Kd tree. The most expensive step that is performed at every recursive call is finding the splitting line. The median based Kd-tree build process involves the sorting of the data set before we could find the median, thus it leads to an increase in the computation cost. Whereas the mean based Kd-tree build process involves the determination of the mean for the data set which does not involve sorting of the data set thus leading to a reduced computation cost. Hence, the building time complexity T (n) satisfies the recurrence condition:

$$T(n) = \begin{cases} O(1), & n=1 \\ O(n) + 2T(\lceil n/2 \rceil), & n>1 \end{cases}$$

Which solves to *O(nlogn)* and considering k dimensions it solves to *O(k*nlogn)*.

6. Conclusion

In this paper, we evaluated the build process of B, Range and Quad trees with their asymptotic time complexities. We then proposed a mean based build process for Kd-trees which has been shown to have a better time complexity as well as a balanced tree compared to the original median based build process. This would widen the application domain for Kd-trees, especially in the areas which require a balanced tree such as spatial database indexing.

7. References

[1] Constrained K-means Clustering with Background Knowledge ; Kiri Wagsta, Claire Cardie, Seth Rogers, Stefan Schroedl, Daimler Chrysler In: Proceedings of the Eighteenth International Conference on Machine Learning, 2001, p. 577-584,

[2] Fast kd-Tree Construction for 3D-Rendering Algorithms Like Ray Tracing, Sajid Hussain and Håkan .Grahn, G. Bebis et al. (Eds.): ISVC 2007, Part II, LNCS 4842, pp. 681–690, 2007,

[3] Wald, I.: Realtime Ray Tracing and Interactive Global Illumination. PhD thesis, Computer Graphics Group, Saarland University, Saarbrucken, Germany,

[4] Zara, J.: Speeding Up Ray Tracing - SW and HW Approaches. In: Proceedings of 11th Spring Conference on Computer Graphics (SSCG 1995), Bratislava, Slovakia, pp. 1–16 (May 1995),

[5]Multidimensional binary search trees used for associative searching; Jon Louis Bentley, Communications of the ACM CACM Volume 18 Issue 9, Sept. 1975, Pages 509 – 517,

[6] Bayer, R.; McCreight, E. (1972), "Organization and Maintenance of Large Ordered Indexes", Acta Informatica 1 (3): 173–189,

[7] Bentley, J. L. (1979). "Decomposable searching problems". Information Processing Letters **8** (5): 244–201,

[8] Chazelle, Bernard; Guibas, Leonidas J. (1986), "Fractional cascading: I. A data structuring technique", *Algorithmica* **1** (1): 133–162,

[9] H. Samet. The Design and Analysis of Spatial Data Structures. Addison-Wesley, 1990,

[10] [D. Eppstein, M. Goodrich, M. T. Sun; The Skip Quadtree: A Simple Dynamic Data Structure for Multidimensional Data,SoCG 2005,

[11] Oracle Spatial 10g White Paper (2006). Oracle Spatial Quadtree Indexing, 10g Release 1 (10.1).

[12] Greene N., Kass M., Miller G.: Hierarchical Zbuffer Visibility. In Proc. SIGGRAPH (1993), pp. 231–238.

[13] Goldsmith J., Salmon J.: Automatic Creation of Object Hierarchies for Ray Tracing. IEEE Computer Graphics and Applications 7, 5 (1987), 14–20.

[14]Macdonald J. D., Booth K. S.: Heuristics for Ray Tracing using Space Subdivision. Visual Computer 6, 3 (1990),153–65.