The Unified Segment Tree and its Application to the Rectangle Intersection Problem

David P. Wagner*

Abstract

In this paper we introduce a variation on the multidimensional segment tree, formed by unifying different interpretations of the dimensionalities of the levels within the tree. Nodes in the resulting d-dimensional structure can have up to d parents and $2d$ children. In order to better visualize these relationships we introduce a diamond representation of the data structure. We show how the relative positions of the nodes within the diamond determine the possible intersections between their representative regions. The new data structure adds the capability to detect intersections between rectangles in a segment tree. We use this to solve the "Rectangle Intersection Problem" with a more straightforward algorithm than has been used previously.

1 Introduction

The Segment Tree is a classic data structure from computational geometry which was introduced by Bentley in 1977 [1]. It is used to store a set of line segments, and it can be queried at a point so as to efficiently return a list of all line segments which contain the query point.

The data structure has numerous applications. For example, in its early days it was used to list all pairs of intersecting rectangles from a list of rectangles in the plane [2], to report all rectilinear line segments in the plane which intersect a query line segment [9], and to report the perimeter of a set of rectangles [12]. More recently the segment tree has become popular for use in pattern recognition and image processing [7].

Vaishnavi described one of the first higher dimensional segment trees in 1982 [8]. Introducing his two-dimensional segment tree as a self-described "segment tree of segment trees", he attached an "inner segment tree", representing one dimension, to every node of an "outer segment tree", representing another dimension, and used this for the purpose of storing rectangles in the plane. A point query would then return a list of all rectangles containing the point. The recursive nature of this data structure meant that it could be generalized to arbitrary dimensions. This has been the standard model for high dimensional segment trees ever since.

*Department of Electronics Engineering, Hanyang University, dwagndwagn@gmail.com

In this paper, we describe a variation on the higher dimensional segment tree. In two dimensions, this new data structure is formed by merging the two segment trees which would be formed from different choices for the dimensions of the inner and outer segment trees. Our purpose in introducing this variation is not to show that it is faster for a particular application, but rather that it supports an additional operation, namely the detection of rectangle intersections, while retaining the structure and functionality of a segment tree.

In the following sections, we will define the data structure, and show a useful way to visualize it. We introduce several new definitions as they apply to this variation of data structure. We further show some relationships between the nodes, and the regions those nodes represent.

Finally, we demonstrate that the data structure can be used to solve the "Rectangle Intersection Problem". Existing methods to solve this problem have either involved using range trees, storing d-dimensional rectangles as $2d$-dimensional points [3], or sweep planes, processing a lower dimensional problem across the sweep [4]. We think that our new data structure represents a more natural way to store this data, and it allows a greatly simplified rectangle intersection algorithm.

2 Segment Tree Properties

In later sections, we will refer to several well known properties of a one-dimensional segment tree. We list them here for the conveniece of the reader.

Property 1 *A segment tree storing n segments has a height of $O(\log n)$.*

Property 2 *A segment stored in a segment tree is split into a canonical representation of $O(\log n)$ subsegments, each of which is stored at a different node of the tree.*

Property 3 *The ancestors of the nodes of the canonical representation of a segment consist of, at most, $O(\log n)$ nodes.*

Property 4 *$O(\log n)$ time is required to insert a segment or to query a point in a segment tree.*

Property 5 *If two distinct nodes each store a segment in a segment tree, and one node is a descendant of the other, then the segment of the descendant node is completely contained within segment of the ancestor node.*

Property 6 *If two distinct nodes each store a segment in a segment tree, and neither node is an ancestor of the other, then the two segments are disjoint.*

The next property is perhaps less well known, but it follows directly from the previous two properties.

Property 7 *Two line segments stored at any two nodes in a segment tree are either disjoint, or one is completely contained in the other.*

3 Two-dimensional Segment Trees

Here we introduce a number of definitions relating to the Unified Segment tree. We begin with a description of the two-dimensional segment tree given by Vaishnavi in 1982 [8]. Although we will focus primarily on two dimensions hereafter, almost every concept we will describe generalizes into arbitrary dimensions.

Construction of the Vaishnavi segment tree begins with a single one-dimensional segment tree, called the "outer segment tree", which represents divisions of the plane along one of the two axes. Attached to every node of this outer segment tree, is an "inner segment tree", representing further subdivisions along the other axis.

Note that the choice of axis for the outer segment tree could have been either the x-axis or the y-axis. Although this choice is arbitrary, it has a great effect on the organization of the data structure. Therefore, let us give different names to the different data structures resulting from this choice.

Definition 1 (xy-segment tree) *Let the xy-segment tree be the Vaishnavi two-dimensional segment tree whose outer segment tree divides the plane along the x-axis, and whose inner segment trees further divide these regions along the y-axis.*

Definition 2 (yx-segment tree) *Let the yx-segment tree be the Vaishnavi two-dimensional segment tree whose outer segment tree divides the plane along the y-axis, and whose inner segment trees further divide these regions along the x-axis.*

A rectangle inserted into a two-dimensional segment tree is first divided into subrectangles along one axis, and then further subdivided along the other axis. These subrectangles are then stored in the nodes of the tree, and if necessary any ancestors of these nodes are created. We show here that the order of these two axes does not affect the set of subrectangles.

Theorem 1 *A rectangle inserted into an xy-segment tree and into a yx-segment tree will be stored as an equivalent set of subrectangles in the each tree.*

Proof. Upon insertion into an xy-segment tree, a rectangle is first divided along the x-axis, and then along the y-axis. In the yx-segment tree the order of these axes is reversed. The canonical subdivisions are the same, regardless of the order in which the two axes are chosen. Therefore subrectangles created and stored in each of the trees are the same. □

Multiple rectangles inserted into a segment tree are stored independently of each other. So this leads to the following corollary.

Corollary 2 *A set of rectangles inserted into an xy-segment tree and into a yx-segment tree will be stored as an equivalent set of subrectangles each tree.*

Now we define the Unified Segment Tree in two dimensions based on these two structures.

Definition 3 (Unified Segment Tree (in 2 dimensions)) *Define the unified segment tree storing a set of rectangles in the plane to be the data structure created by the following procedure:*

1. *Create an xy-segment tree and a yx-segment tree, and insert the same set of rectangles into both.*
2. *Merge the root of every inner segment tree with the node of the outer segment tree to which it is attached, so that they are considered to be one node.*
3. *Merge any two nodes in the xy-segment tree and the yx-segment tree which represent the same region of the plane, so that they become one node. By Corollary 2 they would contain the same data.*
4. *Add all the possible ancestors to any node which is missing any of its possible ancestors. (Ancestors in this data structure are defined later.)*

We note several features of the new data structure which are not normally associated with segment trees. First, a node may have up to two parents, one from the xy-segment tree and one from the yx-segment tree. Second, a node may have up to four children, two nodes each from the xy-segment tree and from the yx-segment tree. Finally, the new data structure is technically no longer a tree, as it may contain cycles.

4 Parents, Children, Ancestors, Descendants

In order to accommodate the features of the new data structure, we must create new definitions of previously well-defined concepts such as parent and child.

Definition 4 (x-child) *Let an x-child of a node be either of the two nodes representing the regions created when the original node's representative region is divided in half along the x-axis.*

A node can have a left x-child, and a right x-child.

Definition 5 (x-parent) *Let the x-parent relationship be the inverse of the x-child relationship.*

Definition 6 (x-ancestor) *Let an x-ancestor be any node which can be reached by following a series of x-parent relationships.*

Definition 7 (x-descendant) *Let an x-descendant be any node which can be reached by following a series of x-child relationships.*

Analogous definitions exist for y-child, y-parent, y-ancestor, and y-descendant.

In addition to x-ancestors, y-ancestors, x-descendants and y-descendants, we define additional nodes to be simply ancestors and descendants.

Definition 8 (Ancestor) *Let an ancestor of a node be any node which can be reached by following a series of x-parent and/or y-parent relationships.*

Definition 9 (Descendant) *Let a descendant of a node be any node which can be reached by following a series of x-child and/or y-child relationships.*

Some additional properties can be seen from these definitions

Property 8 *The x-parent of the y-parent of a node is the same node as the y-parent of its x-parent.*

Property 9 *The x-children of the y-children of a node are the same nodes as the y-children of its x-children.*

5 Visualization

We find it useful to visualize the unified segment tree as a diamond, where the root of the data structure is at the top of the diamond. We divide the diamond into units, such that all nodes representing a rectangle of the same shape and size are located in the same unit.

The two x-children of any node appear in the same unit below and to the left of their x-parent. The two y-children of any node appear in the same unit below and to the right of their y-parent. Thus, each horizontal row of the diamond has double the number of nodes per unit, as the row above it. See Figure 1.

It is possible to see the xy-segment tree and the yx-segment tree embedded in the diamond representation of the unified segment tree. See Figure 2.

Using this visualization, the ancestors of a node appear in the diamond shaped region above the node. The descendants appear in the diamond shaped region below the node. See Figure 3. The number of ancestors can be bounded as follows.

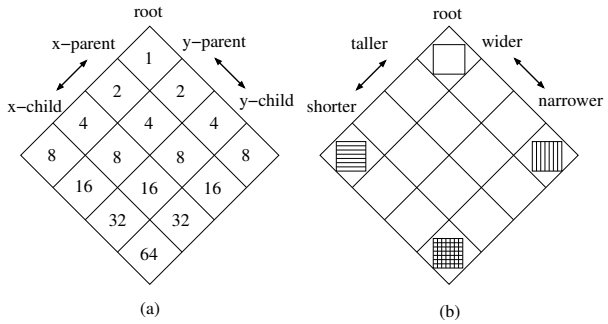

Figure 1: (a) A diamond-shaped visualization of parent child relationships, and the number of nodes in each unit of the diamond. (b) The rectangles which are represented by the nodes in each unit of the diamond.

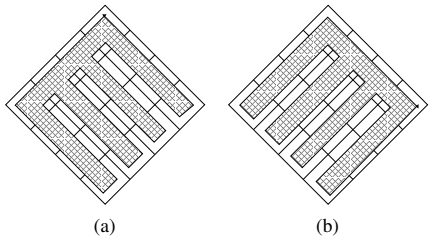

Figure 2: (a) The xy-segment tree embedded into the unified segment tree. (b) The yx-segment tree embedded into the unified segment tree.

Theorem 3 *A node has at most one ancestor per unit of the diamond.*

Proof. Assume there are two ancestors of a node within the same unit. These two nodes must represent rectangles of the same shape and size, because they are within the same unit. The representative rectangles must contain a common point, since the nodes have a common descendant. The representative rectangles must not partially overlap, by Property 7 of Segment Trees. Therefore, the rectangles, and the nodes representing them, must be the same. □

We think it is interesting that the nodes taken from the central vertical column of the diamond form a quad tree, while the central column and a neighboring column form a k-D tree. See Figure 4 for a depiction of this.

6 Relationships between Nodes

The relative positions of the nodes within a unified segment tree determine the possible intersections between the rectangular regions they represent. We examine that relationship here.

Theorem 4 *A node is a descendant of another node, if and only if it represents a rectangle that falls entirely within the rectangle represented by the ancestor node.*

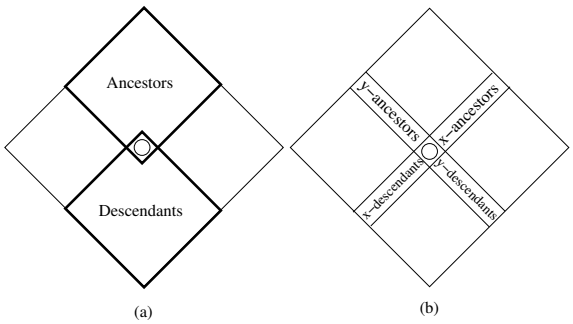

Figure 3: (a) The location of the ancestors and descendants of a node within the diamond. (b) The location of x-ancestors, y-ancestors, x-descendants, and y-descendants within the diamond.

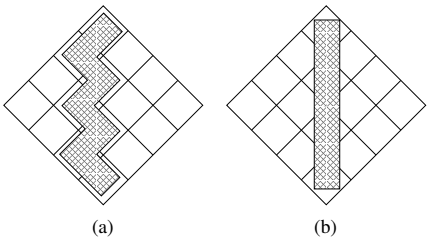

Figure 4: (a) A k-D tree embedded in the unified segment tree. (b) A quad tree embedded in the unified segment tree.

Proof. First, consider two nodes in the segment tree, one of which is a descendant of the other. The rectangle represented by the descendant was formed by successively subdividing the rectangle represented by the ancestor. Therefore, the rectangle represented by a descendant falls entirely within the rectangle represented by any of its ancestors.

Next, consider two rectangles represented by two different nodes of the tree, such that one rectangle falls entirely within the other. Follow the x-parents of the node representing the smaller rectangle, until a node is found which has the same size in the x direction as the larger rectangle. This must have the same x-coordinates as the larger rectangle, otherwise Property 7 would be violated. From there, follow y-parents until a node is found which has the same size in the y direction. This node must have the same y-coordinates as the larger rectangle, for the same reason. Therefore it must be the node which represents the larger rectangle, and the node representing the smaller rectangle must be a descendant of the node representing the larger rectangle. □

Theorem 5 *Two nodes can have a common ancestor which is an x-ancestor of one node and a y-ancestor of the other, if and only if their representative rectangles completely cross over each other, one in the x direction, and the other in the y direction (see Figure 5).*

Figure 5: Possible intersections between rectangles represented by nodes in a unified segment tree.

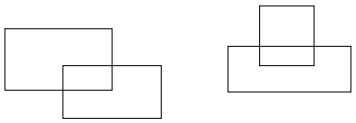

Figure 6: Impossible intersections between rectangles represented by nodes in a unified segment tree.

Proof. Consider any two nodes, having a common ancestor which is an x-ancestor of one and a y-ancestor of the other. The rectangle represented by the common ancestor of the two nodes can be formed by expanding one of the original two rectangles in the x direction, or by expanding the other in the y direction. Therefore, the rectangle of the ancestor must be completely spanned in y direction by the first rectangle, and completely spanned in the x direction by the other. Therefore the two rectangles must completely cross each other.

Next, consider two nodes which represent rectangles that completely cross over each other. The smallest rectangle enclosing both original rectangles can be found by expanding one rectangle in the x-direction or by expanding the other rectangle in the y-direction. Therefore, the enclosing rectangle is represented by an x-ancestor of one of the original nodes, and a y-ancestor of the other node. □

Theorem 6 *Two rectangles which are represented by nodes in a unified segment tree may only intersect in one of two ways. Either one rectangle can be completely inside of the other, or the two rectangles can completely cross over each other, one in the x direction, and the other in the y direction.*

Proof. By enumerating the possible rectangle intersections, we can see that all other intersections would violate Property 7. See Figures 5 and 6 for a visual depiction of the possible and impossible intersections. □

7 Analysis

Here we analyze the performance of the unified segment tree, showing bounds on its size, and on the running time of the standard segment tree operations.

Theorem 7 *After the insertion of n rectangles into a unified segment tree, the deepest x-descendant of the root and the deepest y-descendant of the root have a maximum depth of $O(\log n)$.*

Proof. Recall that the unified segment tree was created from an xy-segment tree, and a yx-segment tree. The outer segment trees of these two trees exactly comprise the x-descendants and the y-descendants of the root. By Property 1, these two trees can have a maximum height of $O(\log n)$. □

Corollary 8 *Any node in a unified segment tree can have a maximum of $O(\log n)$ x-ancestors and a maximum of $O(\log n)$ y-ancestors.*

Theorem 9 *Any node in a two-dimensional unified segment tree can have a maximum of $O(log^2 n)$ ancestors.*

Proof. A node can have a maximum of $O(\log n)$ x-ancestors, and each of these nodes can have a maximum of $O(\log n)$ y-ancestors. All ancestors can be reached along one of these routes. □

Theorem 10 *The canonical representation of a rectangle in a two-dimensional segment tree is comprised of a maximum of $O(\log^2 n)$ subrectangles.*

Proof. Any rectangle is decomposed into a maximum of $O(\log n)$ regions in the x direction by Property 2 of Segment Trees. Each of these regions is further subdivided into a maximum of $O(\log n)$ subregions in the y direction. □

Theorem 11 *All nodes representing the canonical subregions of a rectangle have a maximum of $O(\log^2 n)$ ancestors in a two-dimensional unified segment tree.*

Proof. The limit on the number of the ancestors of the canonical representation exists because there can be no more than 16 ancestors of a given shape. Consider that there are 17 ancestors of a particular shape in the tree. By Property 7, these shapes cannot overlap in their x-coordinates, or their y-coordinates, unless the coordinates are the same. Since there are 17 distinct sets of x and y-coordinates, there must be at least 5 distinct pairs of x-coordinates, or 5 distinct pairs of y-coordinates. Assume, without loss of generality that there are 5 distinct pairs of x-coordinates. Consider the node representing the middle of these 5 pairs of coordinates. The x-parent of the node representing this rectangle must be located entirely within the original rectangle. Therefore there is no reason to include the middle rectangle, or any of its descendants in the canonical representation. □

Corollary 12 *Insertion of a rectangle into a two-dimensional segment tree requires $O(\log^2 n)$ time.*

Corollary 13 *A two-dimensional unified segment tree requires $O(n \log^2 n)$ space to store n rectangles.*

Theorem 14 *A point query of a two-dimensional unified segment tree returns the list of enclosing rectangles in $O(\log^2 n+k)$ time where k is the number of rectangles reported.*

Proof. A point is represented in a unified segment tree at the deepest node. All enclosing rectangles are represented by $O(\log^2 n)$ ancestors of this node. Thus it is sufficient to report all rectangles stored in all ancestors. □

8 Higher Dimensions

Most every aspect of the unified segment tree can be generalized into arbitrary dimensions in a straightforward way. In d dimensions, each node can have up to d parents, $2d$ children, and $O(\log^d n)$ ancestors. Insertion requires $O(\log^d n)$ time. Query requires $O(\log^d +k)$ time, where k is the number of results reported. The entire data structure occupies $O(n \log^d n)$ space.

9 The Rectangle Intersection Problem

The rectangle intersection problem is a classic problem dating back to the early days of computational geometry. The problem has been studied by many authors, and numerous variations have been inspired [5, 6].

Although multiple definitions of the "rectangle intersection problem" have appeared in the literature, we use this definition. Store a set of n axis-parallel rectangles so that a rectangle query will efficiently report all rectangles from the set which intersect the query rectangle.

Edelsbrunner and Maurer gave one of the first general purpose algorithms solving this problem in 1981 [4]. Their method involves storing the rectangles in a combination of segment trees and range trees, and these are queried to detect four kinds of intersections between rectangles. In higher dimensions, they sweep across the problem, solving a lower dimensional problems during the sweep, and thereby giving an overall solution that is recursive by dimension.

Edelsbrunner demonstrated two further methods to solve the problem in 1983 [3]. The first of these involves storing the d-dimensional rectangle in a $2d$-dimensional range tree, and performing an appropriate range query. For the second, a data structure called the "rectangle tree" is developed, and this is queried similarly to the range tree.

Here we show yet another way to solve this problem using an augmented version of our unified segment tree. Our method does not claim a speedup over the previous methods. However, we feel the unified segment tree uses a more natural representation of the data than the range tree or rectangle tree. Additionally, given the machinery that we have already developed in this paper, the algorithm is quite straightforward.

9.1 Augmenting the Unified Segment Tree

For the purpose of supporting a rectangle intersection operation, we augment each node of the unified segment tree with the following additional data:

- A list of rectangles in all descendants of the node.
- A list of rectangles in all x-descendants of the node.
- A list of rectangles in all y-descendants of the node.

Thus a node of a two-dimensional augmented segment tree contains the following information.

```
struct NODE {
    struct NODE * xparent, yparent;
    struct NODE * leftxchild, rightxchild;
    struct NODE * leftychild, rightychild;
    Segment storedHere[];
    Segment storedInDescendants[];
    Segment storedInXDescendants[];
    Segment storedInYDescendants[];
}
```

When a segment is inserted, its identifier must be inserted into all canonical nodes, and all ancestors of the canonical nodes, in the appropriate lists. In two dimensions, this does not affect the asymptotic running time of the insert operation. However, in d dimensions, there can exist 2^d separate lists, so an alternate method of storage may be desirable if d is large.

9.2 Rectangle Query Algorithm

A rectangle query operation returns a list of all rectangles which intersect the given query rectangle. Our algorithm for rectangle query first divides the query rectangle into its canonical regions. It then performs a query on each rectangle individually, reporting the union of the rectangles found.

Note that the same rectangle might be found in multiple places, so care must be taken to avoid reporting duplicates, if that is undesirable. If duplicates are reported this may adversely affect the running time by a polylogarithmic factor.

Recall from Theorem 6 that rectangles can only intersect if one is completely inside the other, or if they completely cross over each other, one in each dimension. Therefore, it is sufficient to report the rectangles described in Theorems 4 and 5.

This gives us the following straightforward algorithm, which is performed on each node representing a subrectangle of canonical subdivision of the query rectangle:

1. Report all rectangles stored in ancestors of the node.
2. Report all rectangles stored in the descendant list of the node.
3. Report all rectangles stored in the x-descendant list of a y-ancestor of the node.
4. Report all rectangles stored in the y-descendant list of an x-ancestor of the node.

This information is available in the ancestors of the canonical nodes of the query rectangles. So only $O(\log^2 n)$ nodes need to be accessed. Again care must be taken to avoid reporting duplicates.

Acknowledgments.

The author is grateful to Stefan Langerman and John Iacono for their helpful discussions.

References

[1] J. L. Bentley. Solutions to Klee's rectangle problems. Technical report, Carnegie-Mellon University, 1977.

[2] J. L. Bentley and D. Wood. An optimal worst case algorithm for reporting intersections of rectangles. *IEEE Transactions on Computers*, C-29(7):571–577, July 1980.

[3] H. Edelsbrunner. A new approach to rectangle intersections - part I. *International Journal of Computer Mathematics*, 13:209–219, 1983.

[4] H. Edelsbrunner and H. A. Maurer. On the intersection of orthogonal objects. *Inform. Process. Lett.*, 13:177–181, 1981.

[5] V. Kapelios, G. Panagopoulou, G. Papamichail, S. Sirmakessis, and A. Tsakalidis. The 'cross' rectangle problem. *The Computer Journal*, 38(3):227–235, 1995.

[6] H. Kaplan, E. Molad, and R. E. Tarjan. Dynamic rectangular intersection with priorities. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, STOC '03, pages 639–648, New York, NY, USA, 2003. ACM.

[7] G. Racherla, S. Radhakrishnan, and B. J. Oommen. Enhanced layered segment trees: a pragmatic data structure for real-time processing of geometric objects. *Pattern Recognition*, 35(10):2303–2309, 2002.

[8] V. K. Vaishnavi. Computing point enclosures. *IEEE Transactions on Computers*, C-31:22–29, 1982.

[9] V. K. Vaishnavi and D. Wood. Rectilinear line segment intersection, layered segment trees, and dynamization. *Journal of Algorithms*, 3:160–176, 1982.

[10] M. van Kreveld and M. Overmars. Concatenable segment trees. In *Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science*, pages 493–504, 1989.

[11] M. van Kreveld and M. Overmars. Union-copy data structures and dynamic segment trees. *Journal of the ACM*, 40:635–652, 1993.

[12] P. M. B. Vitanyi and D. Wood. Computing the perimeter of a set of rectangles. Technical Report TR 79-CS-23, McMaster University, 1979.