An implementation of range trees with fractional cascading in C++[*]

Vissarion Fisikopoulos

March 2009

1 Introduction

This project is an implementation of range trees with fractional cascading, named layered range trees in C++ using STL and generic programming techniques. Range trees are multidimensional binary trees which are used to perform d-dimensional orthogonal range searching. Range trees were discovered independently by several people including Bentley[1], who also discovered kd-trees and Lueker, who introduces the technique of fractional cascading for range trees [7]. An introduction to orthogonal range searching, range trees and fractional cascading can be found in [6, 9]. In [2] there is a presentation of a project of efficient implementations of range trees in 2-3 dimensions including layered ones and some experimental results.

2 Complexity issues

The range trees answer a d-dimensional range query in time $O(\log^d n + k)$, where n is the whole set of points and k is the set of reported points. The construction time and the space the tree consume are $O(n \log^{d-1} n)$. Using fractional cascading we can be benefited by a $\log n$ factor in the last level of the tree and the resulting time complexity become $O(\log^{d-1} n + k)$. Intuitively, fractional cascading perform one binary search instead of two in the last level. The optimal solution to the

---

[*]This project was an assignment in the graduate course of Computational Geometry, National and Kapodistrian University of Athens 2008-2009, http://cgi.di.uoa.gr/~compgeom/2008/ . The code of this project together with a makefile and a small example can be found here: http://users.uoa.gr/~vfisikop/compgeom/layered_range_trees

orthogonal range search problem is due to Chazelle [5, 4] who propose a structure with time complexity $O(\log^c n + k)$ and $O(n(\log n / \log \log n)^{d-1})$ space consumption, where c is a constant.

3   Range trees in CGAL

Although, CGAL library [3] provides some classes for range trees there is space for optimizations in that package [10]. Firstly, there is a lack of recursive construction of d-dimensional range tree and the only way to construct a range tree of dimension d is to build a tree of dimension 1 and then make this an associative range tree of a new one which will have dimension 2. Then one must build a tree of dimension 3 with this tree as an associative tree and this technique continues until the construction of the whole d-dimensional tree.

In addition to that, the package uses virtual functions, which increases the run time and finally there is no fractional cascading.

The proposed approach uses nested templates for the representation of the d-dimensional range tree which is defined in compilation time. The dimension of the tree must be a constant and defined in the compilation time. In the last level a fractional cascading structure is constructed.

For example a 4-dimensional range tree of size n with different kind of data at each layer is given by the following nested templated definition.

```
Layered_range_tree <DataClass,
            Layered_range_tree <DataClass,
                        Last_range_tree <DataClass>
                        >
            >  tree(n);
```

Note that for each layer $i < d - 1$ the same class Layered_range_tree is used. The last two layers, in which the fractional cascading is implemented, use the Last_range_tree class. The DataClass has the defintions of each layer's own data along with the comparison operators.

4   Software implementation

Essentially, the project was implemented using the C++ language and the STL library[11]. Concisely, the design uses methods from object oriented as well as the generic programming style.

Representation. The trees are represented as stl vectors. The tree traversals are implemented using index arithmetic i.e. node's i parent is $\lfloor i/2 \rfloor$, the left, right child of i is $2i+1$ and $2i+2$ respectively. This method is optimal for a *full, static, binary* tree and in our case the third is always hold. In order to have a full binary tree we replicate the last (biggest in the fist dimension) point and in the worst case we have a tree the half of which is useless with no effect to the time complexity (the replicated nodes would not be visited). In this project we are interested in the static case of range trees but the design is sufficient for a dynamic implementation in which the tree nodes must also have some extra pointers. On the other hand, dynamization of the fractional cascading structure is not trivial [8].

Construction. For the construction of the tree we need to sort the input data with respect to the first coordinate and build recursively (top-down) the main tree in linear time. For the associative trees we don't have to sort the input data again. We build the associative trees in bottom-up manner. Every node merge the sorted lists of its children in linear time starting from the leaves which are trivially sorted. Note that this is essentially the same algorithm as merge-sort.

Memory consumption. Even the asymptotic complexity of space stated above ensures that range tree needs a lot of memory. The only constraint in the number of dimensions of data is memory. Moreover, from the asymptotic complexity follows that with fixed memory there is a trade of between the number of data and number of dimensions. [1]

References

[1] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509--517, 1975.

[2] R. Berinde. Efficient implementations of range trees. 2007.

[3] C. E. Board. *CGAL User and Reference Manual*, 3.4 edition, 2008.

[4] B. Chazelle. Lower bounds for orthogonal range searching: part i. the reporting case. *J. ACM*, 37(2):200--212, 1990.

[5] B. Chazelle. Lower bounds for orthogonal range searching: part ii. the arithmetic model. *J. ACM*, 37(3):439--463, 1990.

---

[1]See http://users.uoa.gr/~vfisikop/compgeom/layered_range_trees/10d/layered_range_trees.cpp for an example of a range tree over 10-dimensional data.

[6] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, second edition, 2000.

[7] G. S. Lueker. A data structure for orthogonal range queries. In *SFCS '78: Proceedings of the 19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*, pages 28--34, Washington, DC, USA, 1978. IEEE Computer Society.

[8] K. Mehlhorn and S. Näher. Dynamic fractional cascading. *Algorithmica*, 5:215--241, 1990.

[9] D. M. Mount. Lecture notes: Cmsc 754 computational geometry. lecture 18: Orthogonal range trees. pages 102--104, 2007.

[10] G. Neyer. Algorithms, complexity, and software engineering in computational geometry. *PhD Thesis*, 2000.

[11] B. Stroustrup. *The C++ Programming Language (Special 3rd Edition).* Addison-Wesley Professional, February 2000.