Point Region Quadtree Implementation Document

Jens Åkerblom

University of Helsinki
Faculty of Science

Course
Joint Project Group: Data Structure Project

Instructor
Henning Lübbers

Spring 2011

Contents

1 Introduction

This is the Implementation Document of my Point Region Quadtree, this document will complement the Requirements & Design Document. All of the functionality listed in the R & D Document has made it to the final application, I've also added a error handling system within the structure. This was done by creating an exception class.

The original functionality for this data structure was to simplify collision detection, but since the tree cannot consider areas my first plan got altered.

2 Analysis

2.1 Algorithms

As stated in the R & D Document, the tree has to extract points quickly from an area or position. From the pseudo-code we can determine that the time it takes to extract points is linearly dependent on the depth in that node. So for worst case scenario $O(d)$, where d is the max depth, we have $O(d) = d$. For best case scenario, the point we are looking for is in the root, then the algorithm runs in constant time.

To update points, we use a tree search algorithm. In worst case we have that all nodes are searched. The maximum number of nodes are

$$V = 1 + 4 + 4^2 + \ldots + 4^d = \frac{1 - 4^d}{1 - 4},$$

so the worst case is exponential. This illustrates why you should remove the point, move the point and then add the point back to the tree instead of updating it, especially when dealing with large trees. Finding a point is always faster than updating a point.

From an implementation point of view, this could be avoided completely by giving the tree monopoly of moving a point. However I chose not to do this because it enforces a certain structure to the user's application in whole. As I see it, the tree is not a container, it is only an area partitioner.

2.2 Storage and memory management

The points are not stored in the tree, instead I used a dynamic array of pointers to store a reference of them in the tree. Since the point storage is an array, adding more than 2^{31} points to a single subregion will not work as expected. The tree itself is stored by dynamic links and can be as large as your machine can handle.

The size of a node is dependent on its type. Assuming you're running on a 32 bit machine, an interleaved node will have the constant size of 37 bytes. A leaf node will have a size higher than or equal to 29 bytes, this size will change dynamically when adding and removing points.

3 Test report

I created a test suite to test the tree. The test is divided into two parts, the automated and the interactive.

3.1 Automated test

The automated test runs some operations on the tree and then prints the tree. The tester can then see the changes and determine if the operation has succeeded. There are four test categories in the automated test.

3.1.1 Adding and removing points

This tests adding and removing points to the tree. It has 9 parts.

- Part 1. Displays an empty tree. Should show an un-subdivided tree.
- Part 2. Adding point. The new point should put itself in the root.
- Part 3. Adding point. The tree should be subdivided and the point should be put in different node than the previous point.
- Part 4. Adding point. The new point will be put in the exact same location as one of the previous points. This should subdivide the tree and the point should be put at maximum depth.
- Part 5-7. These parts will remove points and will in effect be inverses of the previous 4 steps.
- Part 8, 9. The last two parts will check the error handling by performing non-valid operations on the tree.

3.1.2 Moving points

This tests moving points. It has 6 parts.

- Part 1. Moves point from one region to another without merging or subdividing the tree.

- Part 2. Moves point within region. Should not have any effect on the tree.
- Part 3. Moves point so that both merging and subdividing occurs.
- Part 4. Moves the points back to their original positions.
- Part 5, 6. The last two parts will check the error handling.

3.1.3 Getting at location

This tests extracting points based on their approximate location. It has 3 parts.

- Part 1. Get from empty region. Should return no points.
- Part 2. Get from region occupied with one point. Should return the point.
- Part 3. Get from region occupied with two points. Should return the points.

3.1.4 Getting in area

This tests the extracting of points inside a rectangular area. It has 6 parts.

- Part 1. Get from empty area. Should return no points.
- Part 2. Get from partial regions. The rectangle gets a point from a region it covers only partially. Should return one point.
- Part 3. Get from mixed regions. The rectangle gets points both in completely covered regions and partially covered regions.
- Part 4. Gets the whole scene.
- Part 5. Gets outside the scene, should not throw any exceptions.
- Part 6. Check error handling.

3.2 Interactive test

In the interactive test you can perform insertions, removal and updating of points. If any exception is thrown the message will be printed and you will be returned to the main menu.

4 Installing

To include the point region quadtree to your project you only need the files 'quadtree.cpp' and 'quadtree.h'. Simply include the header file in your project and link the .cpp file. If you want the tree to generate console output and activate assertions you must define the macro _DEBUG. If you have defined that macro but don't want the quadtree to generate any console output then comment the line that defines _DEBUG_QUADTREE in 'quadtree.h'.

5 Conclusion

So about this project as a whole. Since I was basically complete when I started out, this has been a more or less a boring couple of programming times. The only real learning experience was to travel the www to find how others implement Point Region Quadtrees and as I thought, it was more or less equal to my prior understanding. Also I learned some general terminology about trees and I got to use doxygen formally for the first time. Other than that I got to use LaTeXsome more, so indirectly I learned more of that. But as the Data Structure Project implementation goes, no challenges, no fun.