Skip Quadtree Pseudocode

Luo Qian

December 3, 2014

Point

```
1  float32_t x, y;
```

SkipQuadtreeNode

```
1  SkipQuadtreeNode parent;
2  SkipQuadtreeNode up, down;
3  SkipQuadtreeNode[4] children;
4  double length; // side length of square, if is_square
5  Point center; // center of the square; length/2 distance from each edge
6  boolean is_square;
```

SkipQuadtree_search(SkipQuadtreeNode node, Point p)

```
 1  if node is null or p is not in node
 2        return false
 3
 4  // call starts at the root of the highest-order tree
 5  // children[i] is quadrant i+1, relative to node
 6
 7  if node has no appropriate child
 8        if node is not a square and its center is p
 9              return true
10        return recurse on node.down if possible
11  return recurse on appropriate subchild
```

```
SkipQuadtree_add(SkipQuadtreeNode node, Point p)
1  if p is not in node
2       return false
3  while random() < 0.5 // still winning the coin toss
4       if node.up is null
5            create node.up and link properly // adding more levels
6       node = node.up
7  SkipQuadtree_add_helper(node, p)
8  return true
```

```
SkipQuadtree_add_helper(SkipQuadtreeNode node, Point p)
 1  // call starts at the root of the highest-order tree
 2  parent = parent of where p should be inserted
 3
 4  if parent has a down node
 5      downNode = recursive call to SkipQuadtree_add_helper on the down node
 6  else
 7      downNode = null
 8
 9  // children[i] is quadrant i+1, relative to node
10  quadrant = getQuadrant(node, p);
11  newNode = new SkipQuadtreeNode(x = p.x, y = p.y, is_square = false)
12  update up/down/parent pointers for newNode and downNode
13  if parent.children[quadrant] is null
14      // search for corresponding parent node
15      while downParent is an ancestor of downNode and does not share parent's center
16          downParent = downParent.parent
17      update parent's up/down pointers, and child pointer to newNode
18      return newNode
19  else // node.children[quadrant] is preexisting point
20      // square is the smallest square containing both node.children[quadrant] and newNode
21      square = new SkipQuadtreeNode(x, y = center of square)
22      square's children are now node.children[quadrant] and newNode
23      while the existing child and p are in the same quadrant of square
24          square = smaller square representing the coinciding quadrant
25
26      // find corresponding square/node in down tree
27      square.down = down of where square is to be inserted
28      while square.down has a different center than square
29          square.down = square.down.parent
30
31      square.down.up = square
32
33      replace node.children[quadrant] with square
34      update parent pointers of square, newNode, and sibling
35  return newNode
```

SkipQuadtree_remove(SkipQuadtreeNode node, Point p)

```
1  //check base case of node being null or p being out of range
2
3  // call starts at the root of the highest-order tree
4  // children[i] is quadrant i+1, relative to node
5  quadrant = getQuadrant(node, p)
6  if node.is_square
7        removedFlag = SkipQuadtree_remove(node.children[quadrant],p)
8        if removedFlag
9              update square's pointers, removing tree level if needed
10       else
11             removedFlag = recurse on node.down
12       return removedFlag
13
14 // here, we're at a leaf node now
15 if node does not match p
16       return false
17 // note that this happens only at the highest-level root node each time
18 remove the node's pointers
19 update parent square's pointers, removing tree level if needed
20 free matching node
21 recurse on the down-instance of node
22 return true
```

Next steps: serial k-d tree, concurrent k-d tree using pthread.mutex