Data Structures used for the Geolocalization of robotic bees and avoiding crashes between them

María Camila Morales Ríos
EAFIT University
Colombia
mcmoralesr@eafit.edu.co

Mauricio Toro
EAFIT University
Colombia
mtorobe@eafit.edu.co

ABSTRACT:

The target of this paper is to show a solution for a futuristic related problem that is a concerning one for humankind: the disappearance of the Anthophila species. With that on mind, the main aspect that we describe here is the possibility of creating robotic bees that will held the work of doing pollination.

With the growth of mass extinction of bees and this solution being one alternative for rebuilding what they are doing, it will be a challenge to localize each individual bee and avoid crashes between all of them, for example, if we talk of a large amount of these Robots working at the same time in a particular area with Geodetic coordinates being received, for that reason, the principal obstacle will be solutioned creating a data structure to detect possible collisions between bees that are less than 328.084 feet away from each other, some examples of algorithms and data structures that solve this problem are mainly used in the videogame industry and I will describe them in the development of this research. The solution I proposed to solve this problem with a Data structure is using a quadtree to report if a bee is near another enough to provide a report and advice the user that those bees are going to crash.

CCS Concepts:

- **Computer systems organization** → Embedded and cyber-physical systems → Robotics → Robotic control→ Evolutionary robotics.
- **Information systems** → Data management systems → Data structures → Data access methods → Proximity search.
- **Theory of computation** → Design and analysis of algorithms → Data structures design and analysis

Keywords:

“Data Structures”; “Computer science”; “Algorithm”; “Geographical coordinates”; “bees”; “Collision Detection”; “Spatial searching”; “Geolocalization”; “Geolocation”; “Robotics”

1. INTRODUCTION:

Based on the dramatic drop of bee population that started in 1997 and got worse during the year 2005, the constant use of pesticide and some parasites were slowly eradicating the specie; an issue had begun for agricultural workers who depended on bees to pollinate crops and, during that era and the following years, importing bees was a provisional solution for them. [1]

Recently, some Universities had been interested on using Robotic Pollinators to stop the problem of bees being exterminated and not being capable of find another specie capable of doing the work that they did, for that, a general question is ¿how the robotic pollinators will be located and avoid collisions between them? For that, the point of this whole investigation is to find a way to solve the inquiry.

2. PROBLEM:

The difficulty of this problem is to find a precise data structure for the robotic bees that will not allow wrecks among them, we are solving this problem because is not a distant problem that computer science will investigate in the future, is a daily matter of interest to put effort into, and is a problem often used in other areas, is an adequate example of issues that will appear on other fields that need constant alternatives that are both efficient and effective.

3. RELATED JOBS

3.1 Quadtree

A quadtree is a tree data structure in which each internal node has exactly four children. Quadtrees are the two-dimensional analog of octrees and are most often used to partition a two-dimensional space by recursively subdividing it into four quadrants or regions. The data associated with a leaf cell varies by application, but the leaf cell represents a "unit of interesting spatial information". [2]

- A quadtree starts as a single node. Objects added to the quadtree are added to the single node.

- When more objects are added to the quadtree, it will eventually split into four subnodes. Each object will then be put into one of these subnodes according to where it lies in the 2D space. Any object that cannot fully fit inside a node’s boundary will be placed in the parent node.

- Each subnode can continue subdividing as more objects are added. [3]

3.2 Spatial Hashing

A spatial hash is a 2 or 3-dimensional extension of the hash table. The basic idea of a hash table is that you take a piece of data (the 'key'), run it through some function (the 'hash function') to produce a new value (the 'hash'), and then use the hash as an index into a set of slots ('buckets').

To store an object in a hash table, you run the key through the hash function, and store the object in the bucket referenced by the hash. To find an object, you run the key through the hash function, and look in the bucket referenced by the hash.

Typically, the keys to a hash table would be strings, but in a spatial hash we use 2 or 3 dimensional points as the keys. In addition, here is where the twist comes in: for a normal hash table, a good hash function distributes keys as evenly as possible across the available buckets, in an effort to keep lookup time short. The result of this is that keys which are very close (lexicographically speaking) to each other, are likely to end up in distant buckets. However, in a spatial hash we are dealing with locations in space. [4]

3.3 AABB Trees

An AABB tree is nothing but simply a binary tree, where all the AABBs are stored at the leaves. The main advantage for this kind of broad-phase is that this is a border-less data structure, and it doesn't require you to explicitly specify an area which other kinds of data structures such as grids or Quad Trees require.

A Dynamic AABB Tree is a binary search tree for spatial partitioning. [5]

It is very efficient in all aspects in terms of game physics, including ray casting, point picking, region query, and, most importantly, generating a list of collider pairs that are potentially colliding (which is the main purpose of having a broad phase) Each collider has its own AABB, and they are stored inside a binary tree as leaf nodes. Each internal node, a.k.a. branch node, holds its own AABB data that represents the union of the AABB of both of its children. [6]

3.4 R-Tree

R-trees are tree data structures used for spatial access methods, i.e., for indexing multi-dimensional information such as geographical coordinates, rectangles or polygons.

The key idea of the data structure is to group nearby objects and represent them with their minimum bounding rectangle in the next higher level of the tree; the "R" in R-tree is for rectangle. Since all objects lie within this bounding rectangle, a query that does not intersect the bounding rectangle also cannot intersect any of the contained objects. At the leaf level, each rectangle describes a single object; at higher levels the aggregation of an increasing number of objects. This can also be seen as an increasingly coarse approximation of the data set.

Similar to the B-tree, the R-tree is also a balanced search tree (so all leaf nodes are at the same height), organizes the data in pages, and is designed for storage on disk (as used in databases). Each page can contain a maximum number of entries, often denoted as M. It also guarantees a minimum fill (except for the root node), however best performance has been experienced with a minimum fill of 30%–40% of the maximum number of entries (B-trees guarantee 50%-page fill, and B*-trees even 66%). The reason for this is the more complex balancing required for spatial data as opposed to linear data stored in B-trees. [7]

-Every leaf node contains between m and M index records unless it is the root

- For each index record in a leaf node, I 1s the smallest rectangle that spatially contains the n-dimensional data object represented by the indicated tuple

- Every non-leaf node has between m and M children unless it is the root

- For each entry in a non-leaf node, I 1s the smallest rectangle that spatially contains the rectangles in the child node

- The root node has at least two children unless it is a leaf

- All leaves appear on the same level [8]

4. Quadtree

The main idea with the implementation this data structure, is based on the idea that it represents the archetypical tree-based spatial partitioning, the preminent reason why I choose the Quadtree to develop this project is because of its space partition based mostly on the recursive subdivision of the region in four equal quadrants, subquadrants, and so on, depending on the number of leaves.

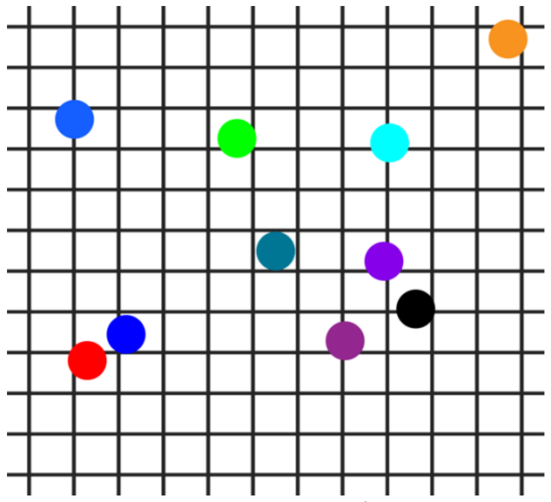

Figure 1: standard 2D vizualisation of the location between bees on a squared grid

What this project needs to do is to compare the position of each bee against the others, discriminate the ones that do not need analysis because they are not near and, in this case for a more efficient Quadtree, there is a need to limit the depth of the tree because the complexity is unbounded in the worst case, under a reasonable input, the Space has a O(n) complexity and the Query time has a complexity of O(n) in the worst case.

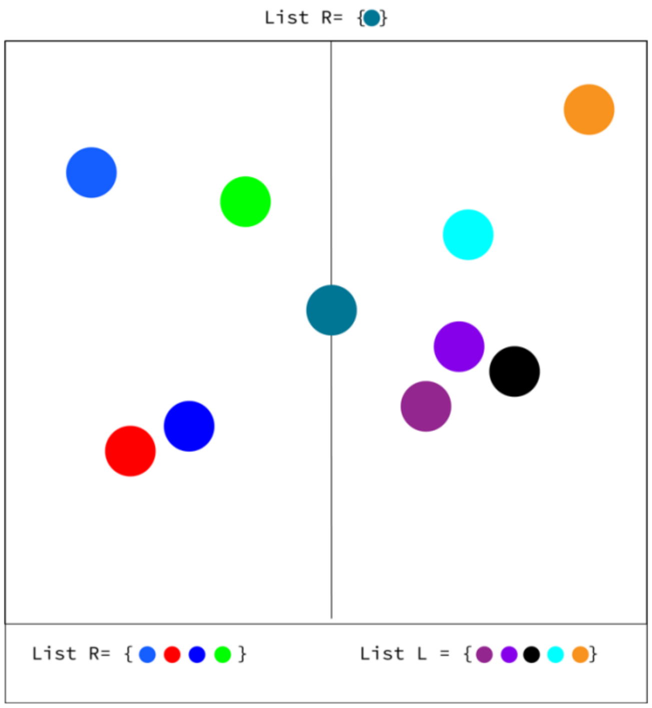

Figure 2: Lists of bees dividing the space in two, one of the list is the root, the others are a lists of bees depending on their location

For a better use of the Quadtree, the implementation of a list that will be the root, other the right division of a stablished space, and the other being the left can help to watch intersections between each list from the inside, a robotic bee on the left list will probably crash between another from the same subgroup. [9]

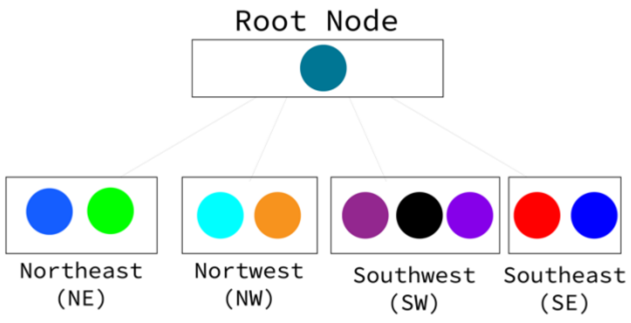

Figure 3: Quadtree data structure

Operations of the data structure

Figure 4: Defining a node Root

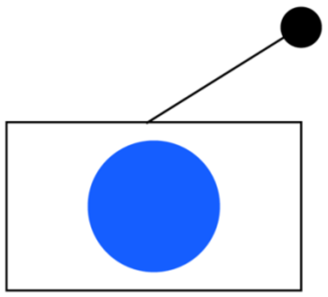

Figure 4: inserting a node in the quadtree

F**f**

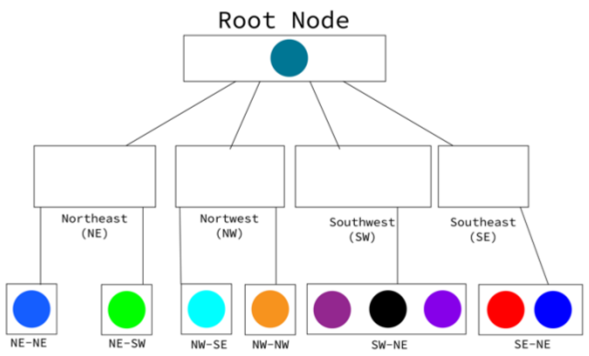

Figure 5: Defining s a region quadtree of a shape in the plane and its tree data structure in which each node has four children. Every node in the three corresponds to a quadrant or squared box in the region of interest hence, the quadrants NE (northeast), NW (northwest), SW (southwest) and SE (southeast).

4.2 Design criteria of the data structure

I choose the quadtree because the idea of dividing a square area into smaller square and quadrants was one of the most interesting spatial Partitioning I found, these subcubes form the child nodes of the root node. The child nodes are, in turn, recursively subdivided in the same fashion, the Quadtree stops being a recursive creation when the tree reaches a maximum depth, or the child nodes keep getting smaller than some preset minimum size, with a reasonable input the complexity of a quadtree in the worst case is o(n) for the query time, and with space is o(n) too. The query time is based on consulting the time that takes to: Given a point q, find the deepest node inwhich q lies, by performing binary search on the depth, each time checking whether the node in depth i containing q exists in the tree, in the complexity analysis, n are the number of data that are being inserted in the tree, q is a node and the idea is to watch out how much it takes to reach the maximal depth in the tree if its necessary. At comparing it with the other data structures that were investigated I discovered that implementing a the spatial hash algorithm was not the best based on the fact that spatial attributes are of higher dimensions, and thus more complicated, make spatial hashing a difficult problem, the AABB tree is very efficient for queries,, but a quadtree is more flexible over time, as the modification are kept more locally.

4.3 Complexity analysis.

Method	Complexity
Remove	O(1)
Insert	O(log s)
GetCollisions	O(s + t)*O (m)

4.4 Execution time

	10 bees (Ms)	100 bees (Ms)	10000 bees (Ms)	1000000 bees (Ms)
Remove	12,3	12,3	12,3	12,3
Insert	15	27	7034	1346000
GetCollisions	34	127	60340	42331000

4.5 Memory used

	10 bees	100 bees	10000 bees	1000000
Memory consumed	0,012 MB	0,036 MB	2,034 MB	45,632 MB

4.6 Result analysis

The results are not optimized, the memory consumption for each data set are dissatisfactory, the complexity for the worst case are not bad based on the data management it takes but it could be improved.

5. Quadtree

I will follow to use the quadtree as the data structure for the development of this project, in my opinion, as I analyzed the results, they were as expected, and I could correct some minor problems I had for the memory consumption based on the number of bees, I had problems with the stack in the first code I did but I made some solutions to this one.

5.1 Operations of the data structure

The operations stayed the same, I followed the same tree implementations I had before, but I tried to optimize some methods and put a better scheme into the performance of the code in action.

5.2 Design criteria of the data structure

The reason why I choose this data structure are the same as I mentioned before, plus at researching more about the quadtree I found out that that the different levels of the tree provide different resolutions of detail and this can be useful in many different ways and that data search for collisions can be much faster changing the developed data structure a little bit.

5.3 Complexity analysis

Method	Complexity
Remove	**O(1)**
Insert	**O(log s)**
GetCollisions	**O(s)*O(m)**

5.4 Execution time

	10 bees (Ms)	100 bees (Ms)	10000 bees (Ms)	1000000 bees (Ms)
Remove	12,3	12,3	12,3	12,3
Insert	15	27	7034	1346000
GetCollisions	17	118	57894	34591008

5.5 Memory used

	10 bees	100 bees	10000 bees	1000000
Memory consumed	0,010 MB	0,036 MB	2,013 MB	43,218 MB

5.6 Result analysis

At comparing the data structure results with the first implementation, we see that the upgrade in the code was changed some aspects in the comparisons of bees being near to collide, the complexity of GetCollisions was reorganized, and in the results of the execution time it shows how it was enhanced but did not had a significant change of values because we are working with the same data structure.

6. Conclusions

The disappearance of an important specie like the bees had the beginning point for this paper and the development of robotic bees as a way to replace their labor, in my opinion we have the technology for implementing a data structure that constantly watches over the collisions between them, even in Korea with 5g telecommunications we can achieve to watch a large number of bees in real time.

What I achieved with the data structured that was enhanced shows that if we want to watch at the collisions in real time and we do not want a large memory consumption we should start watching and analyzing a number of data less than 10.000 bees at the same time in a certain area for a better investigation and inquiry into their target that is to make a replacement of pollination.

At comparing both of the solutions, there is a notable improvement in the last one at the method that compares if two bees collided, the complexity now takes less execution time and the memory consumed is less than the beginning.

For the future of this project I would really like to rebuild this analysis when the technology can be more suitable for large scale data scrutiny, the goal I will have in mind is to reduce the data memory consumption because the ideal of an algorithm is that is effective and does not take much time at comparing, searching and inserting a value, I will try to design another data structure and not work with an existent one or try to mix up a some characteristics of other data structures, I had a lot of trouble with the stack of data, it generally stopped worked for 10.000 bees, I tried to improve it and It works for a large number of coordinates because I tried searching for a solution that was based on Hash sets.

6.1 Future work

I would like to improve the time it takes at comparing two or more bees and the time it takes to search into the deepest part of the tree and try to blend and incorporate another data structure into the quadtree, for example the aabb tree which was very similar to the quadtree or creating a new data structure with space partitioning that could compare at an efficient rate, I would like to improve the number of insertions as well, but in mi opinion this data structure solves the problem in the best way I could think of.

ACKNOWLEDGEMENTS
I would like to thank Jennifer Modelevsky for helping me with this project and correct the parts that were unsteady.

REFERENCES

1. Bees in peril: a timeline, Margaret Badore, Retrieved February 24, 2018 from https://www.treehugger.com/sustainable-agriculture/bees-peril-timeline.html

2. Quadtree, Wikipedia, Retrieved February 24, 2018 from https://en.wikipedia.org/wiki/Quadtree

3. Quadtrees to Detect Likely Collisions in 2D, Retrieved February 24, 2018 from Space. https://www.gamedev.net/articles/programming/general-and-gameplay-programming/spatial-hashing-r2697/

4. Spatial Hashing, Retrieved February 24, 2018 from https://www.gamedev.net/articles/programming/general-and-gameplay-programming/spatial-hashing-r2697/

5. AABB Trees for Collision Detection, Retrieved February 24, 2018 from https://goharsha.com/blog/aabb-trees-forcollision-detection/

6. Game Physics: Broadphase – Dynamic AABB Tree, Retrieved February 24, 2018 from http://allenchou.net/2014/02/game-physics-broadphase-dynamic-aabb-tree/

7. R-tree, Wikipedia, Retrieved February 24, 2018 from https://en.wikipedia.org/wiki/R-tree

8. R-trees: a dynamic index structure for spatial searching Retrieved February 24, 2018 from, https://dl.acm.org/citation.cfm?id=602266

9. Introduction to Octrees Retrieved April 14, 2018 from https://www.gamedev.net/articles/programming/general-and-gameplay-programming/introduction-to-octrees-r3529/