NON-COLLISION DATA STRUCTURES: A QUALITATIVE STUDY OF DATA STRUCTURES FOR COLLISION PREVENTION WITH ROBOTIC BEES

Juan Felipe Londoño
EAFIT University
Colombia
jflondonog@eafit.edu.co

Juan Pablo Giraldo Restrepo
EAFIT University
Colombia
jpgiraldor@eafit.edu.co

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

Mateo Sánchez Toro
EAFIT University
Colombia
msanchezt@eafit.edu.co

ABSTRACT

The future of pollination lies in the creation of mini drones that replace a bee's labor. In order for these drones to work they have to have certain characteristics and abilities, one of them being the skill to not crash into one another,when being closer than 100 meters to each-other. Many problems rely on collision based algorithms as their solution. Such will be seen later on.

Authors keywords: Data structures, collision algorithms, collision detections, Spatial hash.

ACM Keywords:

CCS → Software and its engineering → Software notations and tools → General programming languages → Language features → Data types and structures

1. INTRODUCTION

We need bees. 70 of the top 100 human food crops, which produce 90% of the food we ingest, are pollinated by bees . Since 1947 to 2008 there has been a significant 60% reduction in bee hives. In the future we might not see bees anymore. Thanks to all of these factors and our dependence on bees, investigators are trying to develop artificial bees to pollinate our plants. Scientist such as Eijiro Miyako have been trying to develop insect sized drones to help or replace bees in their labor. All of this is not made easily. The drones must be able to work by themselves for them to have some weight in the pollination process. A very important factor for the robot bees to be autonomous is for them to be able to not crash between each-other which takes us to our problem. How can we make bees not crash into each-other when in distances that are under 100 meters? In this paper the reader will be able to see different solutions to the same problem using data structures.

2. PROBLEM

As read in the preceding paragraph in the future bees may be extinct. For man kind to subsist alternatives such as autonomous pollinating bees must be made. In order for these mini robots to be independent they must be capable of not crashing with one another. For this to happen algorithms to control all of the bees and prevent crashes must be used.

3. RELATED WORK

3.1 AABB tree

AABB trees also known as Axis Aligned Bounding Box is a data structure based on trees which contain a coordinate (X,Y). This data structure can be used to to detect collisions in a rapid manner. This can be done so time effective because what this does is divide the tree in many subtrees and this way you can detect collisions easily. This can all be done by creating a bounding box around the object. This data structure is widely used in the videogame industry as a first line of detection.

3.2 Quadtree

This is another data structure that is also used to store two dimensional coordinates, at most a branch will have four children. This detects collisions by dividing the area recursively into 4 other subregions. The subregions may be squares or rectangles but the shapes must be arbitrary. The quad tree represents a bitmap as a tree

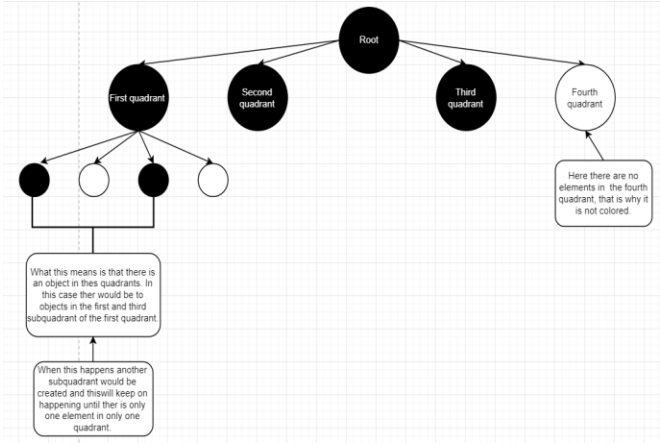
3.3 R-trees

This is another data structure based on trees. They were created by Antonin Guttman in 1984. This data structures has many uses but mainly it is used for spatial access methods, such as geographical coordinates, rectangles or polygons. In real life this tree may be used to store locations on a map like streets or landmarks. It is also very effective for finding the nearest neighbors.

3.4 Spatial hash

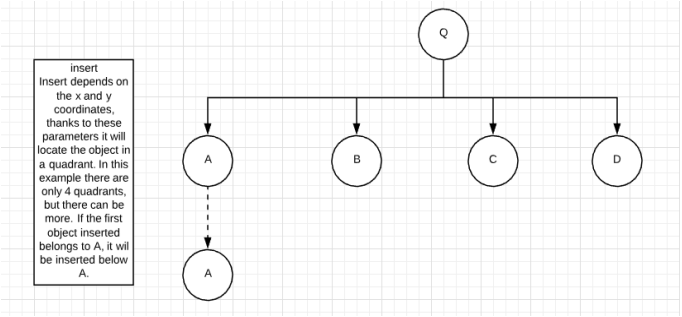
A spatial hash is based on the Hash table data structure. The benefit of doing a spatial hash would be it's low complexity if designed well. If the hash function has a good function for assigning the keys, it could easily have a complexity of O(1). What's stored in each key are 2D coordinates of the objects.

4. Quadtree

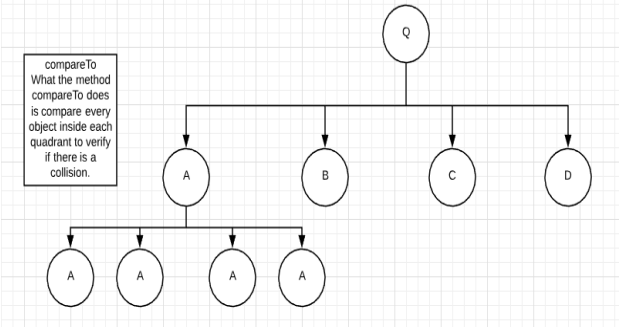


4.1 Operations of the data structure

Insert



CompareTo



4.2 Design criteria of the data structure

At first, this data structure was taken into consideration because of its efficiency in order to compare different nodes which in this case it is very necessary because of the amount of points that have to be checked. The problem with the quad tree is that there may be some collisions that are not detected because to objects may be colliding but in different quadrants.

4.3 Complexity analysis

Methods	Complexity
Insert	O(nLog n)
Compare	O(nLog n)
CreateBee	O(1)
CalculateDistance	O(nLog n)

CalculateDistance is nLog n because needs to go and look for the bee and recover from it his position in X and in Y and to compare it with the next bee, which he must also look for.

4.4 Times

It's the time it takes to run all the code to know how many collisions it finds.

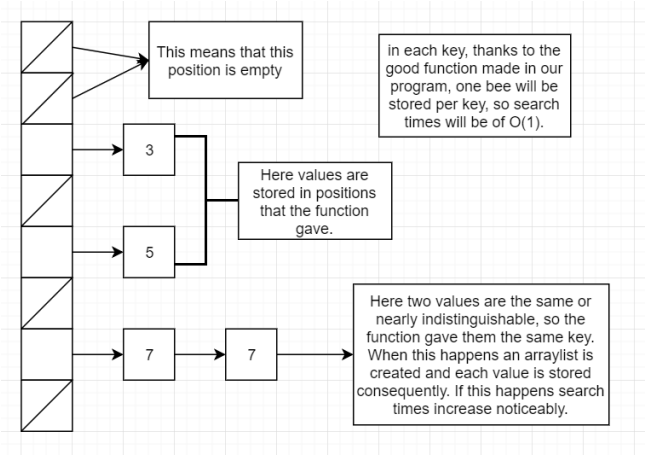
QUADTREE	
N	TIME
4	15
10	15
100	19
1000	56
10000	1173
100000	94285
1000000	N/A

4.5 Analysis of results

-For time:

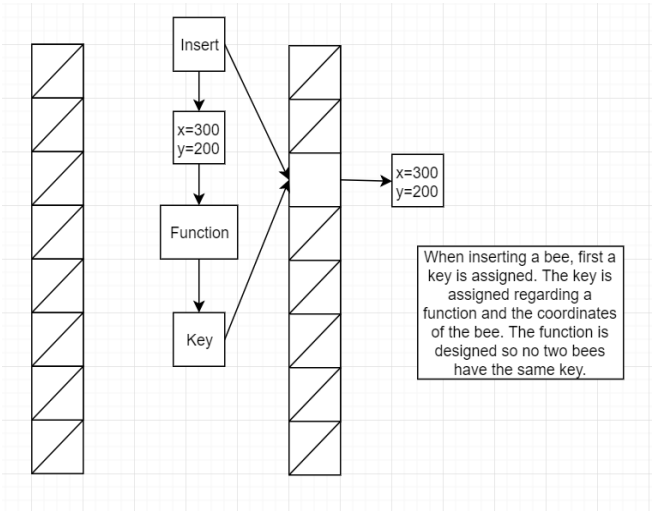
It is seen that as there are more bees, it starts to take much longer, in fact we see that for a million it was impossible to calculate the time. So it was considered not viable

5. Spatial hash

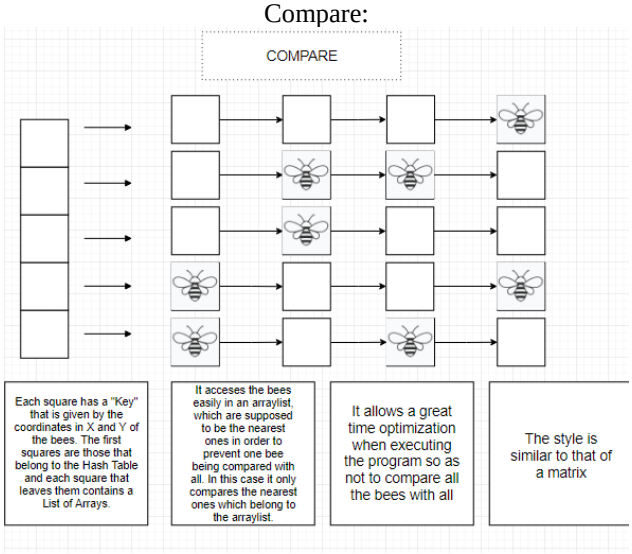


5.1 Operations of the data structure

Insert:

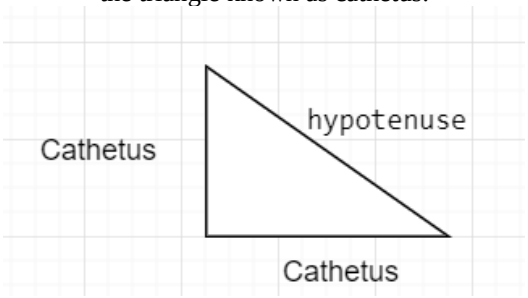


Compare:



CreateBee: Gets the parameters of the text file reading

CalculateDistance: Applies the Pythagorean Theorem, first it obtains the positions in X and Y for the two bees. To compare, subtract the b-a to obtain the value of the side of the triangle known as cathetus.



5.2 Design criteria of the data structure

This data structure was chosen as our final one because it's very effective, even more than the tree. The benefit of the hash table(Hash map in Java) is that with a good function, it has a complexity of O(1) for search. Thanks to this it can be done very fast but also it solves the problem of colliding bees in two different quadrants.

5.3 Complexity analysis

Methods	Complexity
Insert	O(1)
Compare	O(nLog n)

CreateBee	O(1)
CalculateDistance	O(nLog n)

5.4 Times

SPATIAL HASH	
N	TIME
4	2
10	2
100	3
1000	14
10000	126
100000	3276
1000000	17225

5.5 Analysis of results

For time:

Spatial hash has lower execution times when compared to quadtree, for this reason it's less complex. Also since the spatial hash is based on hash tables, many processes can be done with a complexity of O(1) which compared to a tree who has a complexity of O(n log n)in the best case scenario. All this means that execution times in almost every aspect, are lower for hash tables.

6. Conclusions

To conclude bees are extremely important in our lives, but due to many different reasons they are dying. Inorder to subside without bees we have to draw upon other solutions. Here the robotic bees come in. In order for them not to collide we have come up with a great solution after a comparison of many data structures. At the end we took out the best two options which were the spatial hash and the quadtree.

Spatial Hash is more optimal in terms of time than Quadtree, as shown in the comparison of the time tables. (For this problem)

The insert function has less complexity in Spatial Hash than in QuadTree, because in Spatial Hash it is O(1) and QuadTree is O(nLog n)

The comparison function has the same complexity for both data structures

For CalculateDistance and CreateBee the complexity is the same for both data structures.

To conclude, after comparing multiple data structures we found that the best was spatial hash for the properties mentioned before. Also, this may be very possible in a near future due to the lack of bees and death rate. T

6.1 Future works

For future work, we would like to implement a graphical interface, optimize the functions used to detect collisions, implement other data structures to make a more complete comparison between them. We would like to achieve a very optimal program that does not waste a lot of memory and that is done in less time. Also we would like to check for collisions in 3D

6.2 Acknowledgements

We would like to thank EAFIT university for helping us become excellent professionals. Our classmates and everyone who helped this project become possible.

REFERENCES

James. 2017. Introductory Guide to AABB Tree Collision Detection. (January 2017). Retrieved April 14, 2018 from https://www.azurefromthetrenches.com/introductory-guide-to-aabb-tree-collision-detection/

Anon. 2018. Quadtree. (April 2018). Retrieved April 14, 2018 from https://en.wikipedia.org/wiki/Quadtree

Aditya Kamath. 2018. Quad Tree. (February 2018). Retrieved May 14, 2018 from https://www.geeksforgeeks.org/quad-tree/

Steven Lambert. 2012. Quick Tip: Use Quadtrees to Detect Likely Collisions in 2D Space. (September 2012). Retrieved May 14, 2018 from https://gamedevelopment.tutsplus.com/tutorials/quick-tip-use-quadtrees-to-detect-likely-collisions-in-2d-space--gamedev-374

Anon. 2018. R-tree. (May 2018). Retrieved May 14, 2018 from https://en.wikipedia.org/wiki/R-tree

Christer Byström. 2014. Quadtree vs Spatial Hashing - a Visualization. (January 2014). Retrieved April 25, 2018 from http://zufallsgenerator.github.io/2014/01/26/visually-comparing-algorithms/

Omar Shehata. 2016. Redesign Your Display List With Spatial Hashes. (December 2016). Retrieved April 24, 2018 from https://gamedevelopment.tutsplus.com/tutorials/redesign-your-display-list-with-spatial-hashes--cms-27586

Tristam MacDonald. 2009. Spatial Hashing. (October 2009). Retrieved April 24, 2018 from https://www.gamedev.net/articles/programming/general-and-gameplay-programming/spatial-hashing-r2697/