# COLLISION DETECTION BETWEEN BEES

## THIS PROJECT'S PURPOSE IS TO IDENTIFY THE NEAREST ROBOTIC BEES TO AVOID COLLISIONS BETWEEN THEM.

Mariajose Franco Orozco
Universidad Eafit
Colombia
mfrancoo@eafit.edu.co

Susana Álvarez Zuluaga
Universidad Eafit
Colombia
salvarezz1@eafit.edu.co

Mauricio Toro
Universidad Eafit
Colombia
mtorobe@eafit.edu.co

## ABSTRACT

Bees are insects with an important role in the reproduction process of plants and crops. Pesticides and deforestation have caused serious problems for bees because they have reduced the population of these, affecting not only bees but also crops and farmers.

**Keywords**
Data structure, Complexity, Collision prevention, Execution time, Algorithms.

**ACM CLASSIFICATION Keywords**
Theory of computation → Design and analysis of algorithms→ Data structures design and analysis → Sorting and searching

## INTRODUCTION

The goal of every living organism is to create offspring. Plants generate pollen with the aim of being transmitted to other plants in order to generate offspring. There are some insects, such as bees, that help to transmit pollen among plants. The process of pollination is necessary in all crops.

Unfortunately, the world has undergone very drastic climate changes, which have caused the deforestation of some land. Many crops have suffered because of this deforestation and also because of the pesticides that are applied to them to prevent any pest. The problem with these pesticides is that they affect the insects that are responsible of completing the process of pollination between plants. This has caused the extinction of a huge quantity of bees and at the same time of many crops.

## PROBLEM

It is important to be able to find a solution to this problem since it is expected to keep the crops alive but with no plague. For this reason robotic bees were invented. These robotic bees consist of some drones that are in charge of completing the process of pollination. However, there is a problem with these drones because there may be collisions between them. With this project we intend to ensure that this collisions between robotic bees do not happen. In a few words, the objective of this project is to identify the bees that are at 100 meters of distance and send a notification to the user saying that those bees are in risk of collision.

## RELATED WORKS

These are some similar data structure problems:

### 1. Dynamic AABB tree

This data structure consists of an arrangement of nodes, which are represented in two forms: branches and leaves. The idea of this tree is to allow the data to be stored in the nodes of the leaves. The nodes of the branches contain a single leaf and from it there are two others, etc. AABB trees are based on 3 main functions: insert, remove and update. The first function, that is insert, consists in creating a thick AABB to join data with a new leaf node, then, one must go through the tree to find with which brother can pair and create another branch. The remove function can only be done with the leaf nodes since each branch must have 2 valid children and only the leaves contain the user's data. Lastly, the update function is responsible for verifying that the current and adjusted AABB is still contained in the tree.

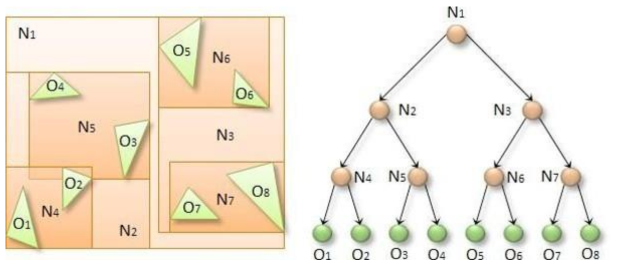


**Figure 1:** Representation of an AABB tree.

### 2. Spatial Hashing

This data structure consists of an extension of the hash tables in which two-dimensional or three-dimensional work can be done. Hash tables are data structures that associate keys with values. The first operation that is performed is searching the data with which one wants to work. This data is saved as a key and then through the hash functions a transformation of this key is made into a position and through this process the desired value is located. Generally, hash tables work with one-dimensional keys (strings), but in a spatial hash you can work with 2-dimensional and 3-dimensional keys. Hash tables are also known as cells where every object has been located in one of these. To know if there may or may not be a collision, the displacement of the objects in the cells can be observed.

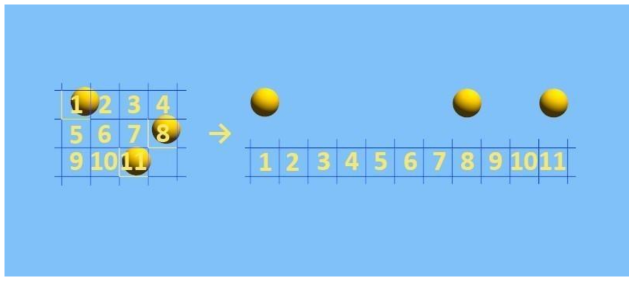


**Figure 2:** Representation of a Spatial Hashing.

### 3. Octree

An octree is a tree data structure in which each internal node has exactly eight children. Octrees are most often used to partition a three-dimensional space by recursively subdividing it into eight octants. Octrees are the three-dimensional analog of quadtrees. Each node corresponds to a single cube and has exactly eight sub-nodes. Notice that all of the sub-node cubes are contained within the parent cube. As in the case of the quadtree, the octree can be used to find a three-dimensional point very quickly. As before you can flatten out the octree and draw it as a standard tree structure. The octree branches very rapidly and it doesn't take many levels to generate lots of nodes. Implementing this data structure isn't difficult. What tends to be difficult is managing the geometry needed to divide the cubes and storing the details in each node.

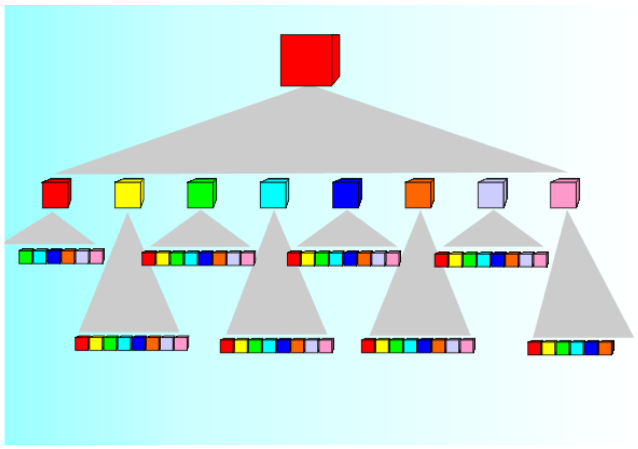


**Figure 3:** Representation of an octree.

### 4. R-tree

It is a tree data structure used to deal with spatial information. This structure was created by Antonin Guttman in 1984. The R-tree is normally used to search for data using a location. The key idea of the data structure is to group objects close to the location and represent them with their minimum bound rectangle in the next level of the tree. Since all objects are within this bounding rectangle, this allows the query not to intercept with objects that are not contained in this rectangle. Each leaf represents a single object. A common use of this data structure in everyday life is to look for open restaurants within a radius of 1 km from the current location of a person, look for the market closest to the current location, among others.

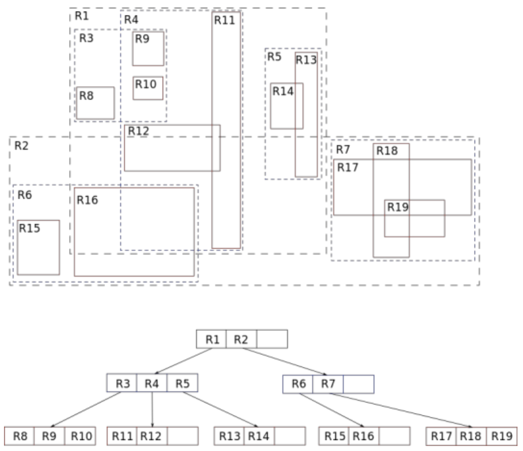
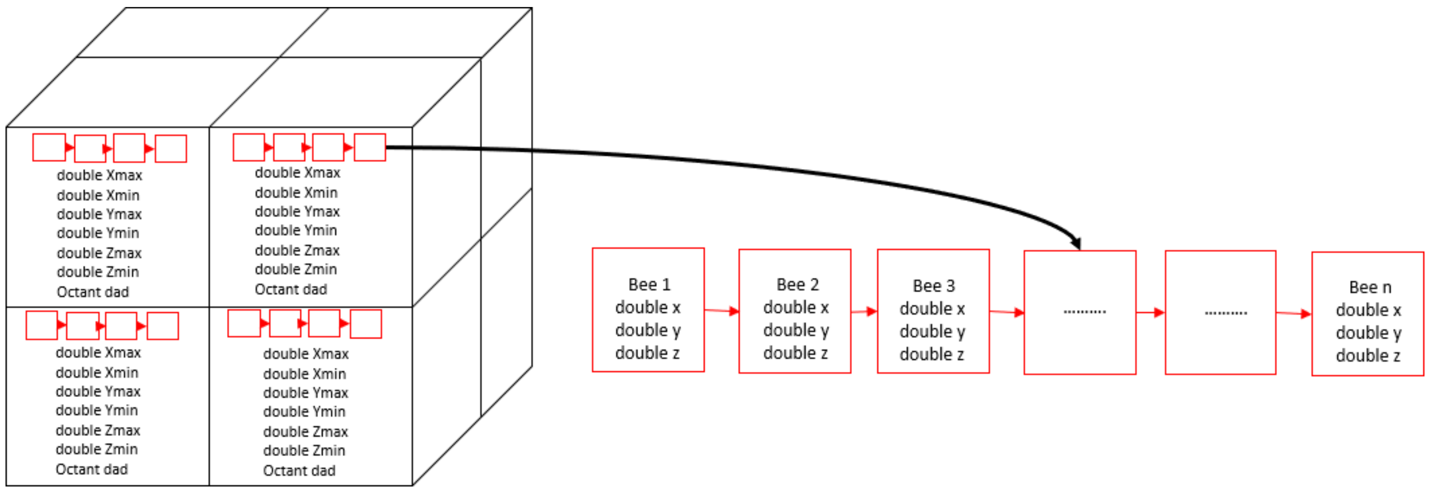


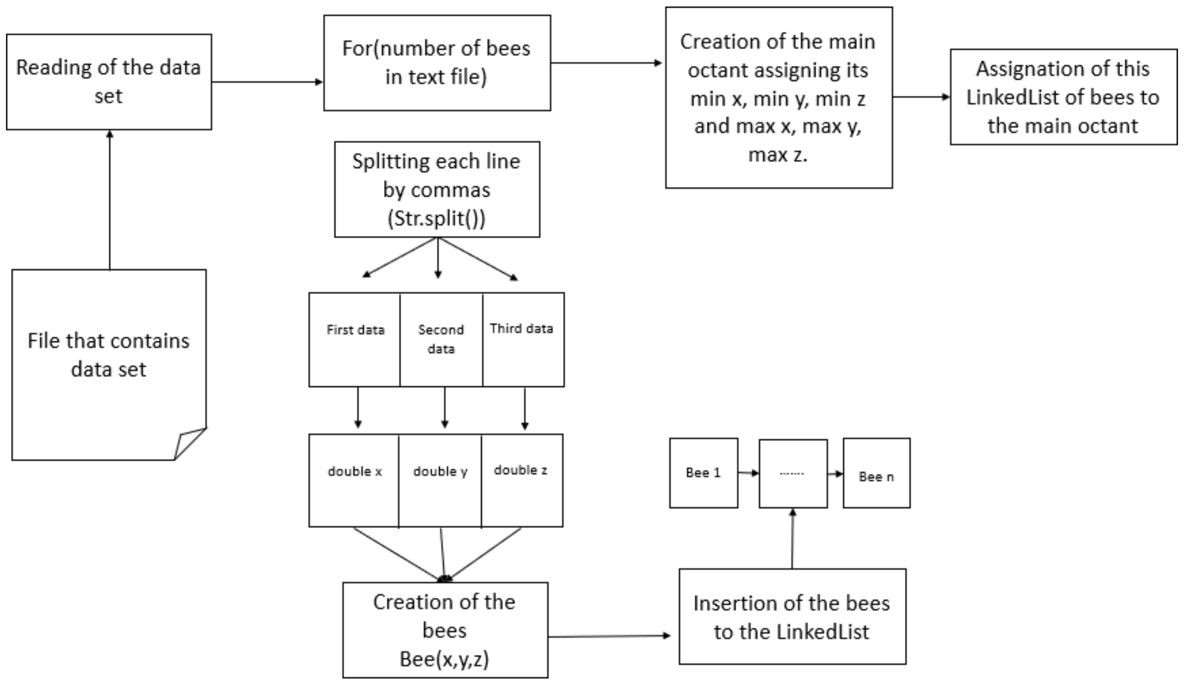
**Figure 4:** Representation of a R-tree.

**Design of the data structure: Octree**



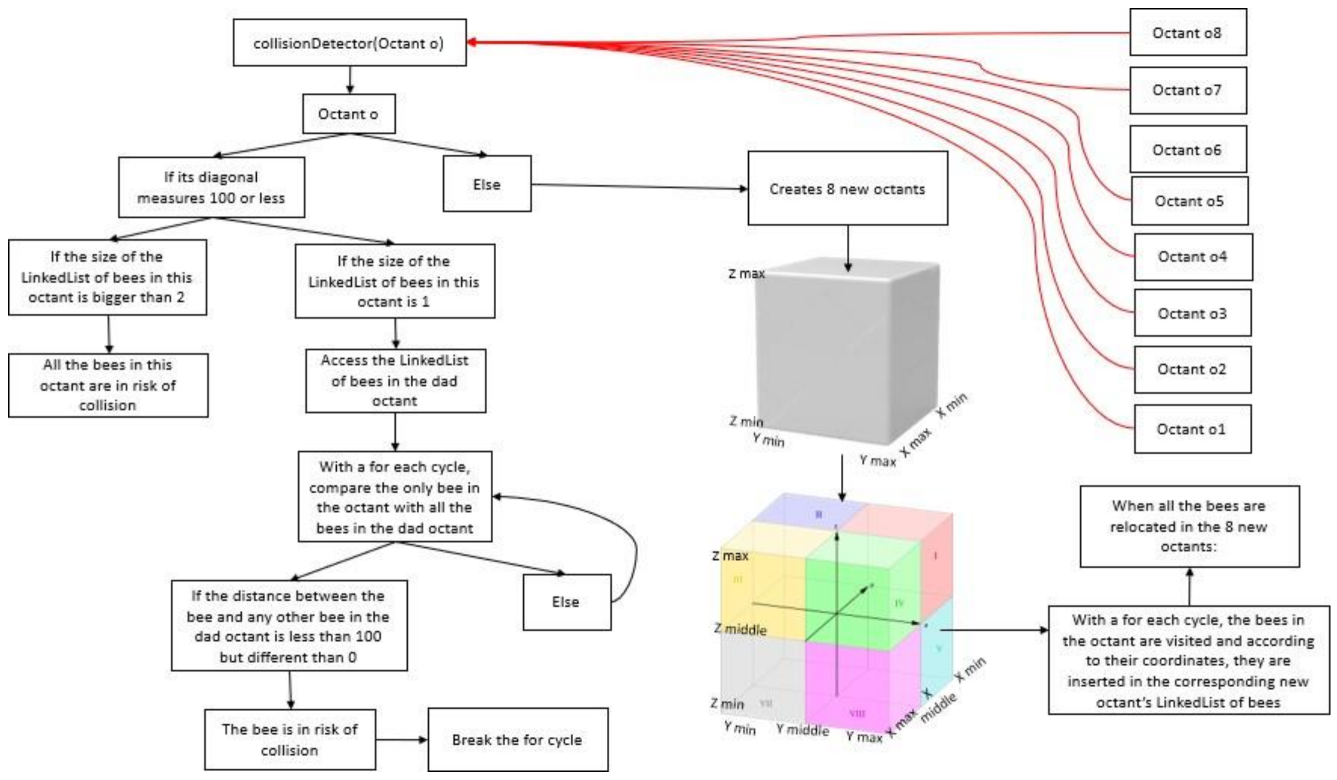
**Graph 1:** Octree: Every octant contains a LinkedList of bees, an octant dad, and the maximum and minimum coordinates of the bees. The LinkedList of bees contains in each position a bee which has coordinates x, y, z.

**Operations of the data structure**
**1. Read and save data**


Reading of the data set
For(number of bees in text file)
Creation of the main octant assigning its min x, min y, min z and max x, max y, max z.
Assignation of this LinkedList of bees to the main octant
Splitting each line by commas (Str.split())
First data
Second data
Third data
File that contains data set
double x
double y
double z
Bee 1
.......
Bee n
Creation of the bees Bee(x,y,z)
Insertion of the bees to the LinkedList

**Graph 2:** Image of the process of reading and saving data

## 2. Collision Detection


collisionDetector(Octant o)
Octant o
If its diagonal measures 100 or less
Else
Creates 8 new octants
If the size of the LinkedList of bees in this octant is bigger than 2
If the size of the LinkedList of bees in this octant is 1
All the bees in this octant are in risk of collision
Access the LinkedList of bees in the dad octant
With a for each cycle, compare the only bee in the octant with all the bees in the dad octant
If the distance between the bee and any other bee in the dad octant is less than 100 but different than 0
Else
The bee is in risk of collision
Break the for cycle
Octant o8
Octant o7
Octant o6
Octant o5
Octant o4
Octant o3
Octant o2
Octant o1
Z max
Z min
Y min
Y max
X max
X min
Z middle
Y middle
X middle
When all the bees are relocated in the 8 new octants:
With a for each cycle, the bees in the octant are visited and according to their coordinates, they are inserted in the corresponding new octant's LinkedList of bees

**Graph 3:** Image of the process of detecting collisions

### Design criteria of the data structure

We decided to implement this data structure since it is not difficult to interpret and understand. We think this is a very useful data structure for what we want to achieve because it works well and it has a complexity of O(n*x). This complexity is low and makes the program run really fast. We can prove this with the results displayed in the tables of execution time in this document. We can see that the average execution time of the different data sets does not exceed 7 seconds. This means that the data structure is working well and at high speed, which is our main objective.

### Complexity analysis

| Method | Complexity |
|---|---|
| readAndSaveData() | O(n) |
| collisionDetector() | O(n*x) |

**Table 1:** Complexity of the implemented methods

### Execution time

Time consumption of the method collisionDetector():

| Number of bees in data set | Average time (ms) | Maximum time (ms) | Minimum time (ms) |
|---|---|---|---|
| 15 | 0 | 1 | 0 |
| 150 | 1 | 5 | 0 |
| 1,500 | 25 | 114 | 2 |
| 15,000 | 333 | 1180 | 19 |
| 150,000 | 1326 | 3817 | 286 |
| 1,500,000 | 6477 | 50423 | 1668 |

**Table 2:** Time consumption in detecting collisions between bees in Colombia

| Number of bees in data set | Average time (ms) | Maximum time (ms) | Minimum time (ms) |
|---|---|---|---|
| 10 | 0 | 1 | 0 |
| 100 | 1 | 7 | 0 |
| 1,000 | 16 | 53 | 1 |
| 10,000 | 204 | 411 | 13 |
| 100,000 | 654 | 2068 | 109 |
| 1,000,000 | 5521 | 19101 | 599 |

**Table 3:** Time consumption in detecting collisions between bees in Bello

**Memory used**

Memory consumption of the method collisionDetector():

| Number of bees in data set | Average memory consumption (bytes) | Maximum memory consumption (bytes) | Minimum memory consumption (bytes) |
|---|---|---|---|
| 15 | 1581889 | 2268224 | 1501800 |
| 150 | 2319488 | 3328800 | 1444976 |
| 1,500 | 13587445 | 23506208 | 3787488 |
| 15,000 | 125064289 | 226554160 | 24661424 |
| 150,000 | 290214425 | 501598064 | 34990000 |
| 1,500,000 | 864622235 | 1079491184 | 577702712 |

**Table 4:** Memory consumption in detecting collisions between bees in Colombia

| Number of bees in data set | Average memory consumption (bytes) | Maximum memory consumption (bytes) | Minimum memory consumption (bytes) |
|---|---|---|---|
| 10 | 1506222 | 2176096 | 1433072 |
| 100 | 2073384 | 2866976 | 1558168 |
| 1,000 | 10130280 | 16733168 | 2909576 |
| 10,000 | 83864497 | 153037704 | 18197352 |
| 100,000 | 257127238 | 391560296 | 12917168 |
| 1,000,000 | 676561120 | 998138408 | 188690616 |

**Table 5:** Memory consumption in detecting collisions between bees in Bello

**Result analysis**

The time complexity of detecting collisions not only depends on the number of bees, but also on how big the space were the bees are located is. Let n be the number of bees and x be the larger number between the longitude, latitude and height of the space were the bees are located in. In the worst-case scenario, the method visits all the bees in the LinkedList to relocate them in the new octants that have a reduced space. The time complexity of this is O(n). Additionally, for being a recursive method in which for every octant the method calls itself other eight time but with the space reduced in eight, the complexity is T(n*x)=8n*T(x/8)+c. However, this equation in big O notation equals O(n*x).
As observed in the execution of the data structure implemented, we could notice that the average time it takes for the method collisionDetector() to perform their operations is not greater than 7 seconds, we consider that it isn't a bad time taking into account that it is the average time of the data sets of 1,500,000 and 1,000,000 of bees. Analyzing the maximum times, we can observe that as long as the data set is greater, the time for executing the method will be longer, but the maximum time for all the data sets, is 50 seconds, that is the time consumption for performing the method in the 1,500,000 data set. This time is less than a minute. We also evidenced that the size of the space were the bees are located in also affects the execution time. If you compare table 2 with table 3, the execution times displayed in table 2 are greater than those in table 3. This happens because table 2 displays the results of time consumption in detecting collisions between bees in Colombia and table 3 displays the results of bees in Bello. Colombia is a much bigger space than Bello so the program has to create more octants for Colombia than for Bello and this is why it takes more time to detect the collisions in Colombia. As for the memory, we could observe that the consumption of this is not as low as it would be ideal. The memory consumption was directly proportional to the number of bees and the size of the space the bees are located in. As the number of bees increases and as the size of the space increases, more memory is required.

## CONCLUSIONS

This project consists in a data structure designed for preventing collisions between robotic bees. The 3D data structure we implemented is an octree. To understand it we can consider the octants as cubes. Each cube is divided in other 8 smaller cubes. When the diagonal of the smaller cubes measure 100 meters, the program concludes that all the bees situated in this cube are in risk of collision. When a cube only has one bee, the program compares this bee's coordinates with the coordinates of all the bees in the dad octant and if the distance between them is less than 100, meters they will collide.

When designing the last data structure, we were able to lower its complexity in comparison to the data structure in the first solution. In the first solution we used an arrayList to store the bees in each octant, however; the complexity of adding elements to an arrayList is O(n), so this really increased the execution time of our program. When doing the last version of the data structure we used a linked list instead of an arrayList to store the bees in each octant. The insertion in a linked list is O(1), but accessing to a specific element in a linked list is O(n). So this was a problem we had because we didn't know if we preferred having a data structure that took O(n) to add objects and O(1) to access to an specific object or a data structure were the insertion was O(1) but accessing a specific object was O(n). After analyzing the situation we realized that we never really needed to access to a specific object in the LinkedList. What we really needed was to go through all the object in the LinkedList so by using a for-each loop to go through the LinkedList we were able to avoid using the "get" which had a complexity of O(n). This helped us reduce the complexity of going through all the objects in a LinkedList from $O(n_2)$ to O(n). Now, we had created a data structure were insertion took O(1) and going through all the elements in a LinkedList took O(n). This data structure was much better than the one used in the first version of the project were insertion took O(n) and going through all the elements in a took O(n). When comparing the execution time of the previous data structure with the last data structure, the reduction in time was significant because we were able to decrease the time complexity.

**Future work**

Even though we consider our actual project is really good, in the future, we would like to research more about data structures and know if there's another one that can make it easier and better. We would like to research more about how our data structure can be faster than it is and have less memory consumption.

We would like to improve our data structure and apply it to other kinds of problems in the world like the prevention of car accidents.

## ACKNOWLEDGEMENTS

This research was supported/partially supported by [Name of Foundation, Grant maker, Donor].

We thank for assistance with finding time consumption and memory consumption to Lorenzo Cannafarina, who is monitor in the class of Data Structure and Algorithms 1 in EAFIT.

## REFERENCES

Myopic Rhino. 2009. Spatial Hashing. (October 2009). Retrieved February 16, 2019 from https://www.gamedev.net/articles/programming/general-and-gameplay-programming/spatial-hashing-r2697

Anon. 2019. Tabla hash. (January 2019). Retrieved February 15, 2019 from https://es.wikipedia.org/wiki/Tabla_hash

Randy Gaul. 2013. Dynamic AABB Tree. (October 2013). Retrieved February 17, 2019 from https://www.randygaul.net/2013/08/06/dynamic-aabb-tree/

Anon. 2018. Quadtree. (October 2018). Retrieved February 16, 2019 from https://es.wikipedia.org/wiki/Quadtree

Anon. 2019. R-tree. (February 2019). Retrieved February 15, 2019 from https://en.wikipedia.org/wiki/R-tree