Bees in action: Avoiding the collision of robotic bees as an alternative for pollination

Camila Barona Cabrera
Universidad EAFIT
Medellín, Colombia
cbaronac@eafit.edu.co

Mariana Gómez Piedrahita
Universidad EAFIT
Medellín, Colombia
mgomezp10@eafit.edu.co

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

ABSTRACT

During the last years, the world has come across the problem of the disappearance and extermination of bees that achieve pollination and give multiple advantages to crops and therefore the human being, for which reason they have taken remedial measures by means of robotics and programming, however, the problem of the collision of these robotic animals has been worrying in this field because their possible impacts would cause significant damage. To solve everything else before said, an algorithm was created which would indicate that bees are at risk of collision, for this adjacency matrices are used with lists, thus, the input information will be analyzed quickly obtaining results instantly. Taking into account that the main objective is to minimize the complexity of the method to detect collisions, from the results that will see below it can be said that a good final optimization of the problem was achieved.

KEY WORDS

-Theory computation

Analysis and design of algorithms

-Real-time operating systems

-Feature interaction

-Sorting and searching

1.INTRODUCTION

Almost two thirds of the cultivated plants that are predestined for human consumption depend on the pollination of bees. For 15 years these beings have diminished considerably their number, due to the use of pesticides and attack of parasites; for this reason, different universities around the world have put in the work of creating small mechanical bees that make the work of these living beings, combining innovative technologies such as miniature robotics and processing algorithms.

PROBLEM

Treating the problematic spoken in motivation we concentrate mainly on the processing algorithm; here we will focus on identifying which robotic bees are 100 or less meters away from any other bee, taking as a priority the algorithm's runtime efficiency.

RELATED WORK

For the current report, an investigation was conducted of different data structures that promoted to give an effective solution to the problem posed.

3.1 Octree

An octree or octal tree is a tree structure of data in which each internal node has exactly 8 children. The octree structures are based on the recursive division of the Envelope box of the figure to be represented in eight octants that completely cover the volume. If an octant is completely occupied by the solid is labeled as black, if it is not as white and if on the other hand it is partially occupied as gray, in which case it is necessary again divide it into 8 children that correspond to the subdivision. The complexity of this algorithm is O (log (n)) and for obtain a complete and correct solution must be had in account:

- Ensure that the entire geometry contained in the node is used when calculating its envelope
- Ensure that the envelope at level n this completely contained at level n-1
- Avoid gaps between adjacent nodes.

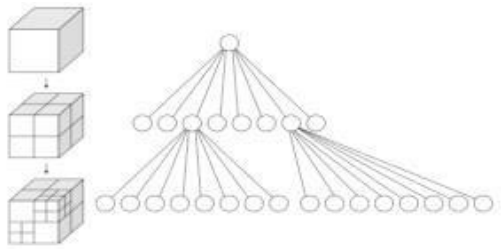


Figure 1.

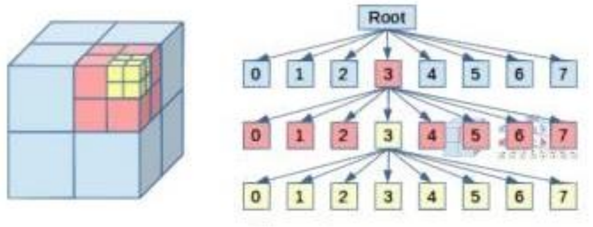


Figure 2.

BSP

Binary space partitioning or Binary Space Partition (BSP) is a method or algorithm used to subdivide recursively a space in convex elements using hyperplanes. This subdivision gives rise to a representation of the scene by means of a structure of data from the tree known as the BSP tree. These two Semi-spaces are usually referred to as positive and negative, the positive half-space being in the part of the plane towards which your normal points.

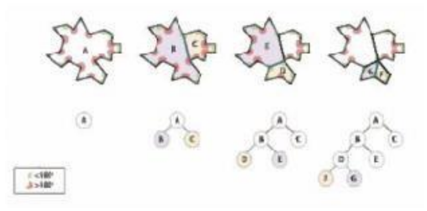


Figure 3.

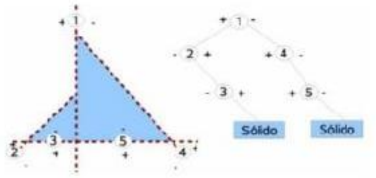


Figure 4.

Quadtree

The term Quadtree, or quaternary tree, is a algortimo which is used to describe classes of data structures hierarchical whose common property is that they are based on the principle of recursive decomposition of space. In a QuadTree of points, the center of a subdivision is always at one point. When inserting a new element, the space is divided into four quadrants. By repeating the process, the quadrant is divided again into four quadrants, and so on.

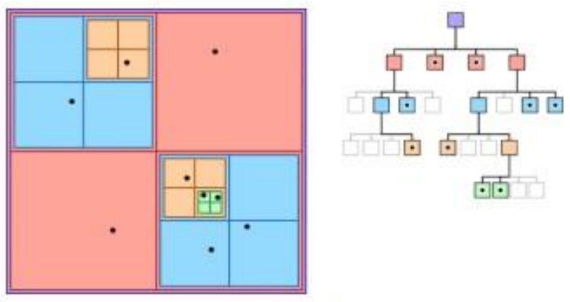


Figure 5.

This type of algorithms are used for different problems related to the space of objects and in this case, avoid collisions of figures, since their different levels provide different resolutions of details being useful in many ways. For example, if has a two-dimensional data point, x, and then they will be immersed in exactly one node at each level of the tree, so it can be used to perform a search fast of a data set.

K-Dimensional tree

U Tree kd (abbreviation of k-dimensional tree) is a Partitioning data structure of the space that organizes the points in a Euclidean space of k dimensions. Since there are many possible ways to choose blueprints aligned to the axes, there are many ways to generate trees kd. The usual system is: As it descends on the tree, cycles are used through the axes to select the planes. (For example, the root can have a plane aligned with the x axis, its descendants would have planes aligned with him and the grandchildren of the root aligned with the z, and so on) In each step, the selected point to create the plane of cut will be the median of the points placed on the kd tree, what respects their coordinates in the axis that is being used.

- Construct a static kd tree from n points is from O (nlogn).

- Insert a new point in a balanced kd tree it's O (logn).

- Remove a point from a balanced kd tree is from O (logn).

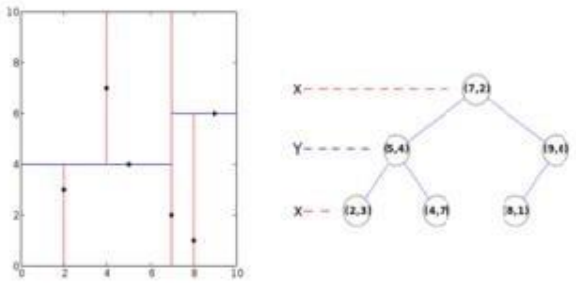


Figure 6.

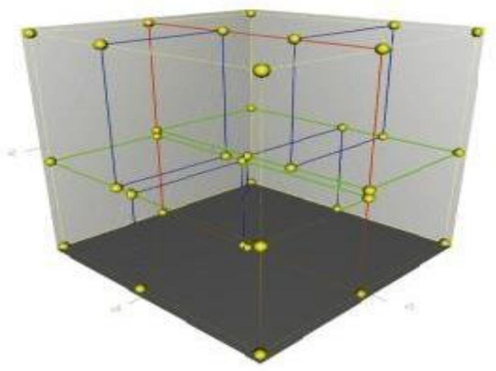


Figure 7.

Arraylist With Bees

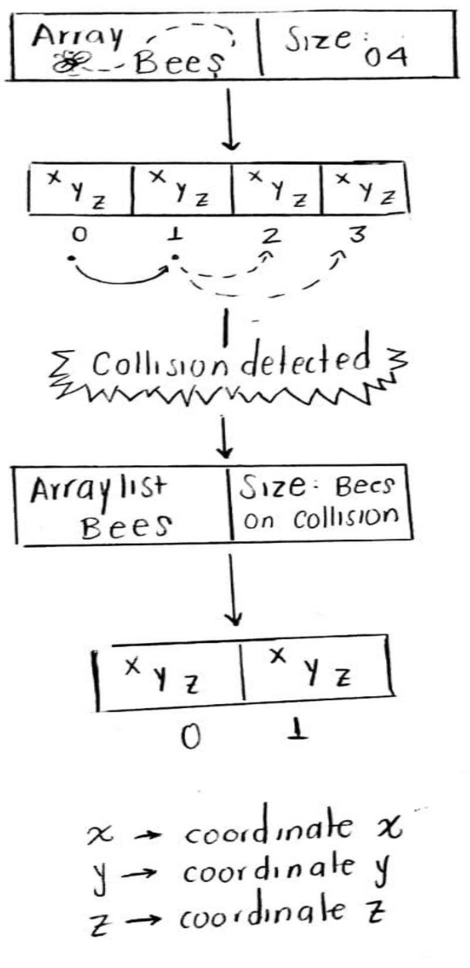


Figure 8

Grafic 1: Array of a number of bees extracted of an archive. Each bee has 3 coordinates: Altitude, Latitude and Height. When the collision is detected, it bees will pass to an ArrayList, for been recognized.

4.1 Operations of data structure

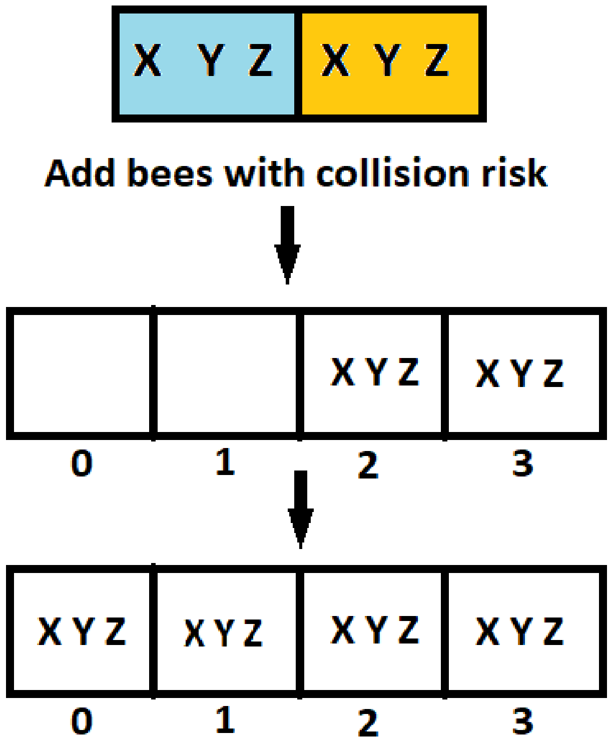


Figure 9

Design criteria of data structure

In the beginning, the array is implemented, since the number of elements are defined. First of all, the set of bees that are at risk of collision are started, they are entered in an Array List, since the size of an array cannot be modified.

It is also clear that the number of bees that are going to collide is not defined, so there is no specific number to create the Array, in this way, they avoid wasting spaces inside the Array, since the total number of bees (those entered at the beginning) is not the same as that of collided bees; thus the use of the memory space is reduced and the travel time of the Array List is much smaller with the number of data required.

4.3 Complexity Analysis

Method	*Complexity*
Read an archive	O(n)
Detecting risk of bee's collition	O(n^3)
Save an archive	O(n)

4.4 Execution Time

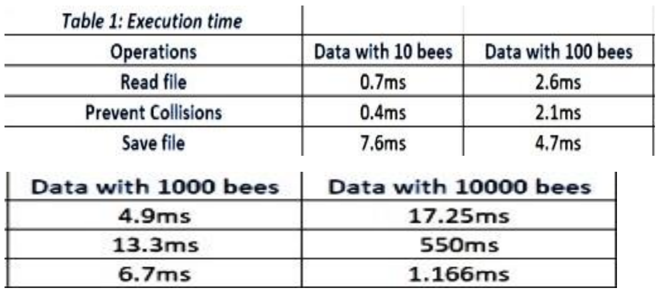


Table 1: Execution time

Operations	Data with 10 bees	Data with 100 bees	Data with 1000 bees	Data with 10000 bees
Read file	0.7ms	2.6ms	4.9ms	17.25ms
Prevent Collisions	0.4ms	2.1ms	13.3ms	550ms
Save file	7.6ms	4.7ms	6.7ms	1.166ms

4.5 Memory

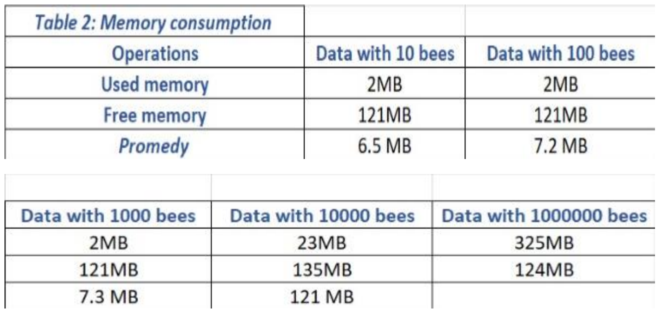


Table 2: Memory consumption

Operations	Data with 10 bees	Data with 100 bees	Data with 1000 bees	Data with 10000 bees	Data with 1000000 bees
Used memory	2MB	2MB	2MB	23MB	325MB
Free memory	121MB	121MB	121MB	135MB	124MB
Promedy	6.5 MB	7.2 MB	7.3 MB	121 MB	

4.6 Analysis of results

In the algorithm to calculate collisions of bees less than 100 meters away, we can see in the table the execution times, separated by operations, that is, methods, that the one that takes more time is that of keeping a file, in which we have the answer; which are the coordinates of the bees that could collide. In the worst case we have a time of 1193 milliseconds and in the average case, it takes 1116 milliseconds, the second operation that demands the most time is to prevent collisions, we have as the worst case 695 milliseconds and in the average case 550 ms.

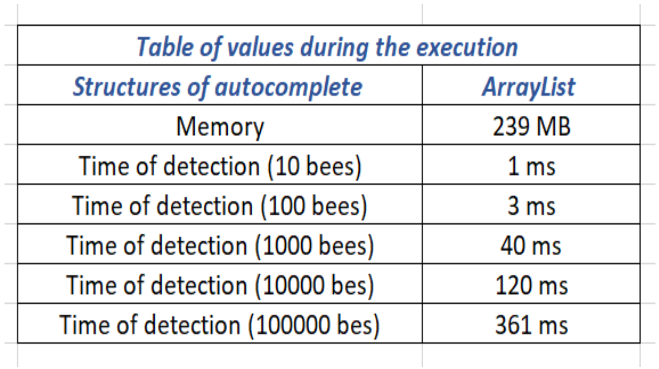


Table of values during the execution	
Structures of autocomplete	***ArrayList***
Memory	239 MB
Time of detection (10 bees)	1 ms
Time of detection (100 bees)	3 ms
Time of detection (1000 bees)	40 ms
Time of detection (10000 bes)	120 ms
Time of detection (100000 bes)	361 ms

5. Avoiding airborne crashes with matrices

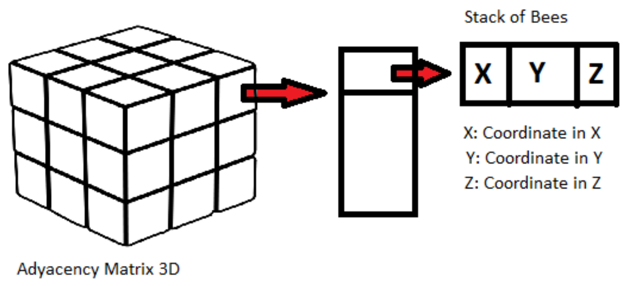


Figure 10: Adjacency matrix with lists, each position of the matrix has a list representing each bee with its respective data.

5.1 Operations of the data structures

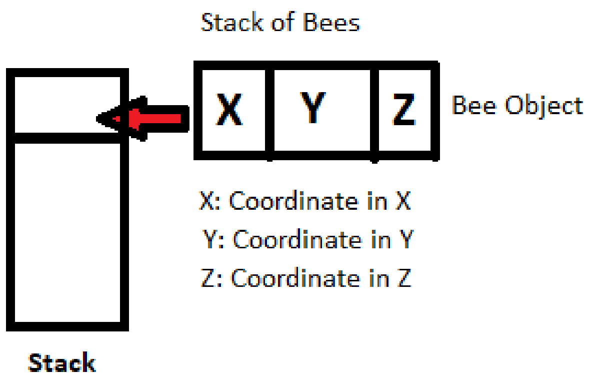


Figure 11: Insert a Bee object in a Bee's Stack (Inserting to the beginning of the Stack).

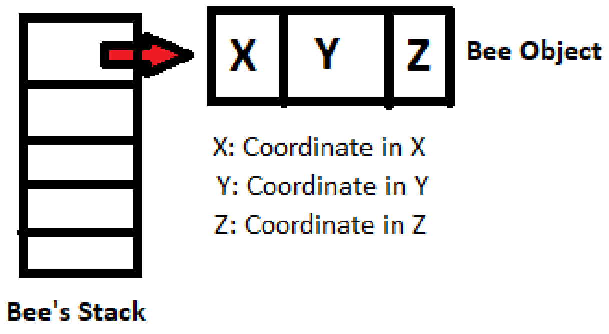
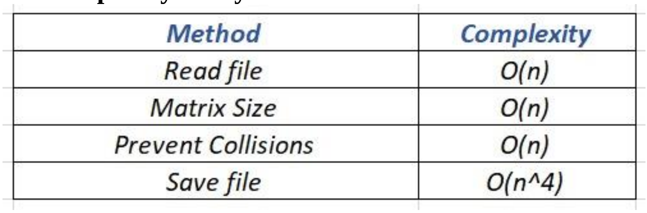


Figure 12: Remove a bee object from the Stack (the last one that entered).

5.2

As the main objective is to minimize the complexity of the main method, resorting to the use of adjacency matrices is one of the best options, since insert and read is O (n), counting them as the basic operations; likewise with the lists its complexity is very little O(1), ideal to maintain the low complexity that is needed. In addition, using matrices and lists is very familiar.

5.3 Complexity Analysis



Method	*Complexity*
Read file	*O(n)*
Matrix Size	*O(n)*
Prevent Collisions	*O(n)*
Save file	*O(n^4)*

Figure 13: Table of Analysis complexity of all methods.

5.4 Execution Time

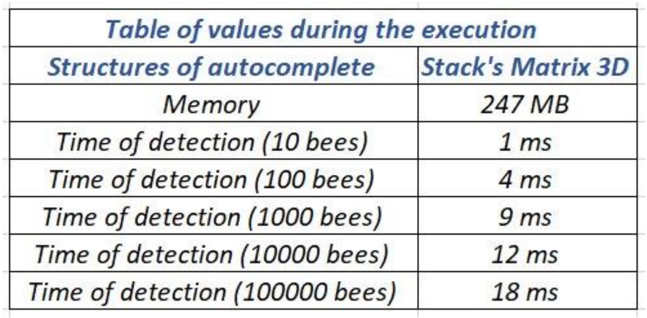


Table of values during the execution	
Structures of autocomplete	*Stack's Matrix 3D*
Memory	*247 MB*
Time of detection (10 bees)	*1 ms*
Time of detection (100 bees)	*4 ms*
Time of detection (1000 bees)	*9 ms*
Time of detection (10000 bees)	*12 ms*
Time of detection (100000 bees)	*18 ms*

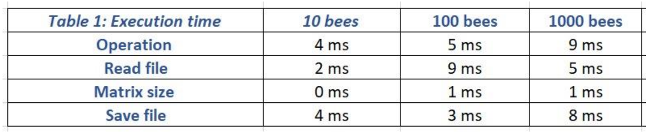
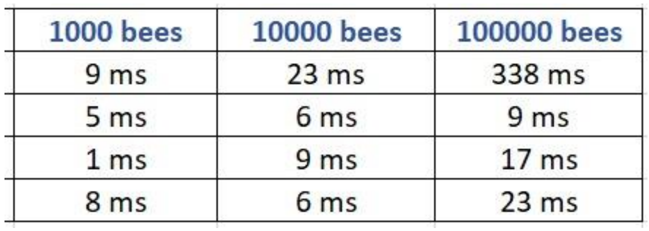


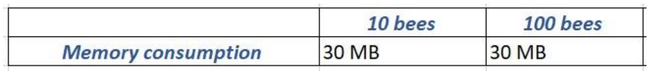
Table 1: Execution time	*10 bees*	100 bees	1000 bees
Operation	4 ms	5 ms	9 ms
Read file	2 ms	9 ms	5 ms
Matrix size	0 ms	1 ms	1 ms
Save file	4 ms	3 ms	8 ms



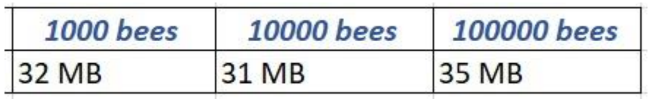
1000 bees	10000 bees	100000 bees
9 ms	23 ms	338 ms
5 ms	6 ms	9 ms
1 ms	9 ms	17 ms
8 ms	6 ms	23 ms

Figure 14: Table of Execution time of set of bees

5.5 Memory



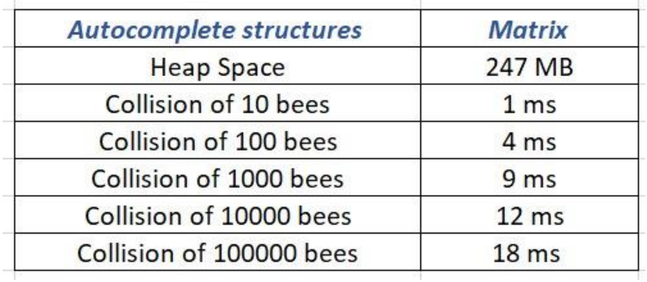
	10 bees	*100 bees*
Memory consumption	30 MB	30 MB



1000 bees	*10000 bees*	*100000 bees*
32 MB	31 MB	35 MB

Figure 15: Memory consumption

5.6 Analysis of the results



Autocomplete structures	*Matrix*
Heap Space	247 MB
Collision of 10 bees	1 ms
Collision of 100 bees	4 ms
Collision of 1000 bees	9 ms
Collision of 10000 bees	12 ms
Collision of 100000 bees	18 ms

The results obtained from the final data structure developed in comparison with the previous structure are of total difference, taking into account that execution times and memory consumption are much lower; and in the focal point that is the complexity of the methods we can also observe a great difference, finally obtaining the results that were wanted.

6. CONCLUSIONS

Among the most important things that were discussed throughout the report were in general 3:
The data structure to be used, since here we had to take into account many variables, such as: efficiency in time and space, if we knew how to develop it, etc.
Another is the operations of the data structure, from which we plotted its structure and functioning; finally the analysis of the results is of vital importance, since that is where the feedback of the whole implementation of the solution for the problem is made.

The most important result of this practice is the complexity of each method, with which we meet the needs for a good execution of the project.

6.1 Future Works

As a possible improvement of the work we would like to take it to the graphic part, we believe that it would be a good plus for the project; Achieving even lower complexity would also be a good improvement for the job.

It is clear that applying other data structures the project would be seen in a different way, therefore it would be good to experiment with data structures that could help us simplify many operations in relation to the current solution.

Finally, it would be good to see the algorithm working as in a type of simulation, in order to get a better view of what the problem would be like on a real implementation scale.

Gratefulnes

We thank Daniel Mesa Arango, monitor of the course, for his help in different areas of the project, who helped us constantly to solve all our doubts about the project.

REFERENCES

1. Octree, 2017. Consultado el 25 de Agosto del 2018, Wikipedia, Free encyclopedia. Retrieved of: http://www.adobe.com/products/acrobat/.
2. BSP,2018. Consultado el 25 de agosto del 2018, Wikipedia, Free encyclopedia. Retrieved of:
https://es.wikipedia.org/wiki/Partici%C3%B3n_bi naria_del_espacio

3. Quadtree, 2017. Consulted on August 25 of 2018, Wikipedia, Free encyclopedia Retrieved of: https://es.wikipedia.org/wiki/Quadtree

4. Árbol kd, 2016. Consulted on August 25 of 2018, Wikipedia, Free encyclopedia. Retrieved of: https://es.wikipedia.org/wiki/%C3%81rbol_kd

5. Mike James. August 09, 2018. Quadtrees and Octrees. Retrieved of: https://www.iprogrammer.info/programming/theory/1679- quadtrees-and-octrees.html

6. Adolfo Guzmán Arenas. Use of kd tres as supervised classifiers and to replace expert systems. Retrieved of: http://www.cic.ipn.mx/aguzman/papers/89%20Arboles%20kd.htm

7. Francisco Javier Melero Rus. Bp-Octree: An hierarchical structure of enveloping volumes. Taking of Granada University: https://hera.ugr.es/tesisugr/17693895.pdf

8. Program used for vectorize imágenes: Inkscape. https://inkscape.org/es Ilustraciones:

9. Figure 1: https://es.wikipedia.org/wiki/Octree

10. Figure2: https://geidav.wordpress.com/2014/07/18/advancedoctrees-1-preliminaries-insertion-strategies-andmax-treedepth/

11. Figure3: https://es.wikipedia.org/wiki/Partici%C3%B3n_binaria_del_espacio

12. Figure4: https://hera.ugr.es/tesisugr/17693895.pdf

13. Figure5: https://developer.apple.com/documentation/gameplaykit/gkquadtree

14. Figure 6: https://www.cienciaexplicada.com/2015/10/atajos-paramovereficientemente-un-quadrotor-a-traves-delgrupoespecial-euclideo-se3.html

15. Figure 7:https://es.wikipedia.org/wiki/%C3%81rbol_kd

16. Key words:

https://www.acm.org/publications/class-2012