Robotic bees: Data structures and algorithms for collision detection and prevention.
or
Robotic bees and how to stop them from "*bumper cart*-ing" into each other.

Vincent Alejandro Arcila
Universidad EAFIT
Colombia
vaarcilal@eafit.edu.co

Isabel Piedrahita
Universidad EAFIT
Colombia
ipiedrahiv@eafit.edu.co

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

Key words:

Author selected keywords: Spatial data structures, complexity, nearest neighbor, collision detection, efficient.

ACM CLASSIFICATION keywords: CCS → Information systems → Data management systems → Data structures → Data access methods → Proximity search

ABSTRACT

With the population of bees rapidly dropping in recent years, we are left with the task of finding ways in which to carry out tasks bees usually handle for us. One of such solutions might be the implementation of robotic bees, but with it arises the need to stop them from crashing into one another, resulting in a problem of collision detection in rigid yet constantly moving objects. **(To be completed in following deliverables)**

1. INTRODUCTION

In recent years the population of bees has been constantly decreasing . More than a billion of bees have dyed in Colombia alone in the span of the last four years[1], worldwide, bees are not doing any better. With the number of bees dwindling dramatically in the past 10 years it is time to consider the importance of bees in the ecosystem. Out of the 369,000 species of flowering plants, 90% depend solely on insect pollination[2], and taking into account that bees can visit anywhere between 50 and 1000 plants a day[3], it is undeniable that their role on this task is fundamental. Without bees not only would countless plants die, but the food chain would also be considerably damaged, since a sizable amount of primary producers would disappear. This urges us to find ways in which to cover up for the bees, which leads to solution such as the creation and deployment of robotic bees.

2. PROBLEM

A problem that will rise is how to get the robotic bees not to crash when they are pollinating. A small step to get with a solution is to find an efficient way to identify which bees are in risk to collide. In other words, alert bees when they are near to others.

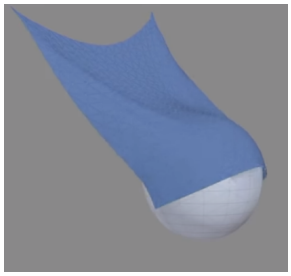
3. RELATED WORKS

3.1 Better collisions and faster cloth for Pixar's Coco[4]

While making Pixar's Coco animators ran into a complication, it was nearly impossible to get the cloth of clothes to sit properly on the huge cast of esqueletal

characters, whose bones pinched the fabric causing what producers referred to as “Skeleton Wedgies”. Because of this, their in-house cloth system, Fritz, had to implement a more efficient and easily systematized version of continuous collision detection.

In order to fix this, the Global Intersection Analysis, an algorithm that uses constant collision detection to identify when cloth has crossed over to the wrong side by analysing all the intersections between the digital mesh that composes the costumes. This was used to create a fast way to stop the cloth from going through and getting stuck between the small geometries that make up Coco’s skeletons.



3.2: Search and tracking algorithms for swarms of robots: A survey[5]
This paper serves as an introduction to any scientist interested on understanding ways to keep an eye on groups of robots and to make them work more efficiently. It covers various algorithmic related problems in order to give the reader an idea of what to have in mind when designing systems that involves groups of robots.

One of the problems that needs to be solved in software is the problem of target search and tracking, understanding a target as an object with which the robots need to interact directly in order to succeed on their task. Understanding how to handle with software all the different environments that the robots might have to face is not as simple as one might think. The number of targets may be unknown, or may vary with time. Also, how to manage robots so that they do not act on the same target simultaneously, known as the task allocation problem is a thing to have in mind. Targets might not be static, which has to be considered.

Cooperation and coordination among the robots is an essential aspect that has to be taken into account when designing swarm robots systems. An ideal system would be the one on which all robots are exchanging information, combining measurements from multiple positions to accurately determine the targets position, and get the best synchronization possible.

3.3: Cooperative Path Planning of Robot Swarm Based on ACO[6]
Ant Colony Algorithm is the abstraction of the way that ants find food. It is based on the fact that they secrete pheromones when walking, so the more pheromone is on a path the more ants have used it, meaning that there is likely to be food. The problem arises when a group of ants finds a deadlock area, meaning they have to go back to try a different path.

Can this algorithm be improved? Clearly, if no pheromone were on the path that leads to a deadlock ants would find food faster. The solution given in this paper states a barely obvious but hard to implement way to fix this issue. If the individual that found a deadlock removed the pheromone left on that path other individuals would not waste time on that path.

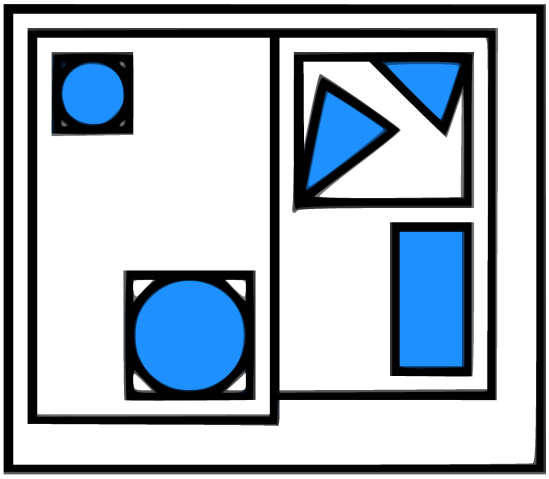
So, being robots instead of ants, we get a great increase on performance when a robot communicates to its peers that there is no point going through certain areas, due to the lack of targets on

certain points of an open field.

3.4: Real-Time Robot Motion Planning in Dynamic Environments[7] Navigation is one of the main fields collision detection is used on. As soon as a robot can move on its own, ithas to decide where to, and this is where collision detection comes in very handy. Although there has already been extensive study of collision detection, further study is very important, because current algorithms understand the world around them via a group of queries, instead of using geometrical analysis.

Because of this, the process of collision detection is usually divided in two phases.

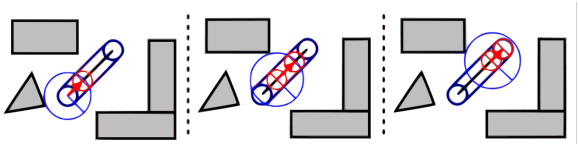
“Broad-Phase: Determines which geometric objects need to be checked for a certain query, ruling out distant objects based on bounding boxes or other partitionings of the space. Two data structures stand out throughout the broad-phase, namely, spatial hashing and spatial partitioning trees.



AABB-tree

Narrow-Phase: For the objects that overlap in the broad-phase, a check is performed between the query and a single primitive object that represents an obstacle. Often objects correspond to rigid bodies.”

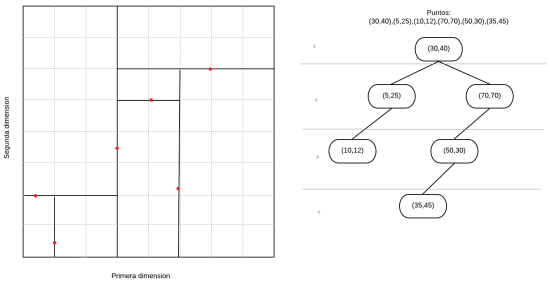
In the narrow-phase some very useful algorithms stand out when we take into account that most of the structure is described by primitive objects such as spheres. These algorithms include checking a point q and radius r for collision, calculating the distance from q to the nearest point on an obstacle and Calculating the nearest point p on the obstacle to a query point q.



Example swept circle obstacle check using only distance queries.

4. TITLE OF THE FIRST DATA STRUCTURE DESIGNED

The data structure chosen is the KD-tree. KD-trees are binary trees generated by repeatedly splitting each of any k dimensions into two chunks. By doing this values greater than each split are placed on the right, and values smaller than the value of the split are placed on the left branch.

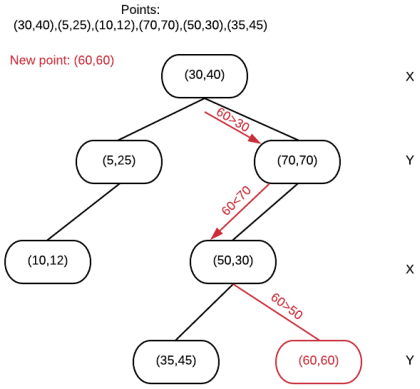


4.1: Operations of the data structure

The main operations carried out on our data structure are insert, delete, search nearest neighbor, search and space.

Insert: This operation will receive a one dimensional array containing all of the dimensions of a point. It will transverse

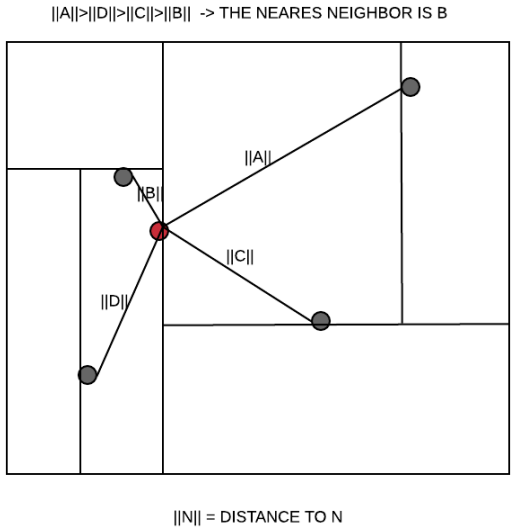
the tree comparing each of the dimensions stored in the array with the cutting dimension values of every level of the three.



Delete: This operation will take a point represented by an array, and will recursively transverse the tree searching for it. If the current node contains the the querie's datum, then if the node is a leaf, it will simply delete the reference that ties that value to the tree. If the node has a right sub-tree different from NULL then it will look for the minimum of the current node's cutting dimension in said subtree, will replace the subtree to be deleted with the found minimum, and will recursively delete the found minimum. Else if the node has a left sub-tree it will have the same behaviour as the above described scenario of the right child, but it will additionally make the new left sub-tree the right child of the node.

Nearest neighbor: The NN algorithm will transverse the tree, visiting the most promising sub-trees first, while it is doing this, it will store partial results, and prune the tree, this means it will not iterate over fragments of the tree it has deemed unimportant for the particular query. Although the word iteration was used, this algorithm is in fact recursive.

Search: This recursive algorithm will transverse the KD-Tree. It will do this by receiving an array with length k and comparing for each level of the tree the cutting dimension of the current node with the equivalent dimension of the node that's being looked for. It will discard subtrees by comparing the sizes of the two values of the nodes in the given dimension, if the value of the current node is larger that the value of the node we`re looking for, them the datum will be in the left child, and if this is not the case, then it has to be on the right tree or not be at all. When the algorithm reaches a leaf, it know that it in fact did not find the required value and returns a boolean false, if at any point of the transversal it finds the requires node, then it will return a boolean true.



4.2: Design criteria for the data structure

Our data structure is required to represent the space intuitively, in order to improve the complexity of the *search collisions* method by decreasing the amount of operations performed to detect these events for a single point in space. With this on mind and with lots of literature analyzed, we chose KD-tree as

our base data structure. This data structure gives the level of abstraction that we need, and the efficiency expected.

The data structure needed to be simple to understand, and we required it to be present in different tools that would make it both more easy to use and more effective.

4.3: Complexity analysis

The kd-tree complexity analysis is shown in the next table:

Método	**Complejidad**
Search	O(n)
Insert	O(n)
Delete	O(n)
Search nearest neighbor	O(n)

4.4: Execution time and Memory used

4.4.1: Search query:

	Time (sec)
100000	**0.00000558**
10000	**0.000004234**
1000	**0.0000035867**
100	**0.000003088**
10	**0.0000004847**

4.4.2: Tree generation

	Time (sec)
100000	0.4108500541
10000	0.0320463161999
	9994
1000	0.0030059084999 99954
100	0.0005630852999 999991
10	0.0003572091000 000555

4.4.3: Memory used

	MiB
100000	**91.1**
10000	**63.4**
1000	**60.1**
100	**60.1**
10	**59.8**

4.4.4: Search for collisions

	Time (sec)
100000	**521.12312**
10000	**24.2908690127**
1000	**0.6712638075**
100	**0.001641400799 9999481**
10	**0.000496442600 0000355**

4.5: Result analysis

With the results given in the las section is proven that this data structure still needs optimization in order to perform well on large datasets. Also, it is shown the least

expensive methods on which KD-trees outperforms. Unfortunately, collision detection is not one of them.

REFERENCES

[1] RCN radio, Carlos Brand (2018) Más de mil millones de abejas han muerto en Colombia en los últimos tres años. (16, February 2018). Retrieved February 15, 2019 from https://www.rcnradio.com/recomendado-del-editor/mas-de-mil-millones-de-abejas-han-muerto-en-colombia-en-los-ultimos-tres

[2] BBC News, Rebecca Morelle (2016) Kew report makes new tally for number of world's plants. (10,May 2016). Retrieved February 15, 2019 from https://www.bbc.com/news/science-environment-36230858

[3] Nicola Bradbear BEES AND THEIR ROLE IN FOREST LIVELIHOODS: A guide to the services provided by bees and the sustainable harvesting, processing and marketing of their products. FAO, Rome 2009, 13-16.

[4] David Eberle. 2018. Better collisions and faster cloth for Pixar's Coco. In ACM SIGGRAPH 2018 Talks (SIGGRAPH '18). ACM, New York, NY, USA, Article 8, 2 pages. DOI: https://doi-org.ezproxy.eafit.edu.co/10.1145/3214745.3214801

[5] Madhubhashi Senanayake, Ilankaikone Senthooran, Jan Carlo Barca, Hoam Chung, Joarder Kamruzzaman, Manzur Murshed. 2015. Search and tracking algorithms for swarms of robots: A survey. *Robotics and Autonomous Systems 75, B (2016) 422-434.* DOI:https://doi.org/10.1016/j.robot.2015.08.010.

[6] Li Yong, Li Yu, Guo Yipei,,Cai Kejie. 2017. *Cooperative Path Planning of Robot Swarm Based on ACO*. Key Laboratory of Industrial Internet of Things and Intelligent Instrument, Ministry of Education. Industrial IoT Collaborative Innovation Center Chongqing Education Commission. Chongqing University of Posts and Telecommunications, Nanshan Scenic, .

[7] James Robert Bruce. 2006. Real-Time Motion Planning and Safe Navigation in Dynamic Multi-Robot Environments. Ph.D. thesis Dissertation. Carnegie Mellon University Pittsburgh, PA 15213.