Algorithm to detect collisions with the spatial hash algorithm

Omar Alexis Becerra Sierra
Universidad EAFIT
Colombia
oabecerras@eafit.edu.co

Juan José Tamayo
Universidad EAFIT
Colombia
jjtamayoa@eafit.edu.co

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

SUMMARY

The main objective of the construction of this algorithm is the resolution of the problem of collisions between bees and give an optimal solution to the problem. A very similar problem is the problem of collision handling of video games where you have to detect the collision of one object with another in as long as possible so that the game does not lose fluidity and give an excellent user experience. We used spatial hash to reduce the complexity to O (log (n)). The results we obtained were favorable since we managed to reduce the time necessary to find the collision between bees. We concluded that there are more efficient ways to solve any problem.

KEY WORDS

Spatial Hash , HasMap, Quadrant, Complexity

Keywords of the ACM classification

Theory of computation, algoritm, Sorting , Searching, Colitions

1. INTRODUCTION

Bees are one of the species that is in danger of extinction in the world, due to the daily human actions. The use of chemicals and pesticides in agricultural production, pollution and global warming are some of the factors that have put the lives of these insects at risk. Because of this Scientists from the Polytechnic University of Warsaw have created the first robotic bee designed to artificially pollinate flowers capable of finding a flower, pick up its pollen, and transfer it carefully from the male to the female flower to fertilize. The problem is that If this were to be done in a massive way, there would be collisions between the same robotic bees. In this document, possible solutions will be given to the problem based on a computer algorithm where the positions of the bees were handled through geodetic coordinates.

2. PROBLEM

The importance of this problem lies in the fact that bees are responsible for the pollination process, one of the most rudimentary food production tools in the world. Three quarters of the food crops in the world depend on pollination

3. Related Algorithms

3.1 V-COLLIDE Algorithm

It is a collision detection library for large environments. It is designed to operate in large quantities of polygonal objects. It does not make assumptions about the input structure and it works on arbitrary models, also known as polygon soups

3.2 DEFORMCD Algorithm

It is a rapid collision detection library designed to accelerate the calculation to deform objects whose vertices are vibrating, an AABB retrofit solution is used for collision detection

3.3 I-COLLIDE Algorithm

It works for models that are convex polyhedra. But it exploits the special characteristics of the convex polyps to determine very quickly the state of the contact. It also takes advantage of temporal consistency, so collision query times are extremely fast when models move only a relatively small amount between frames.

3.4 Gilbert-Johnson-Keerthi distance Algorithm

GJK algorithms are often used incrementally in simulation and videogame systems. In this mode, the final simplex of a previous solution is used as an initial conjecture in the next iteration, or "frame". If the positions in the new table are similar to those in the previous table, the algorithm will converge into one or two iterations. This produces collision detection systems that operate in almost constant time.

3.5 Quadtree

It is used to represent points in several ways. The chosen representation depends on the specific task that one wants to execute with a group of points. The most common tasks that are performed with a group of points are to find a group of points that are related given some criteria.

4. Spatial Hash

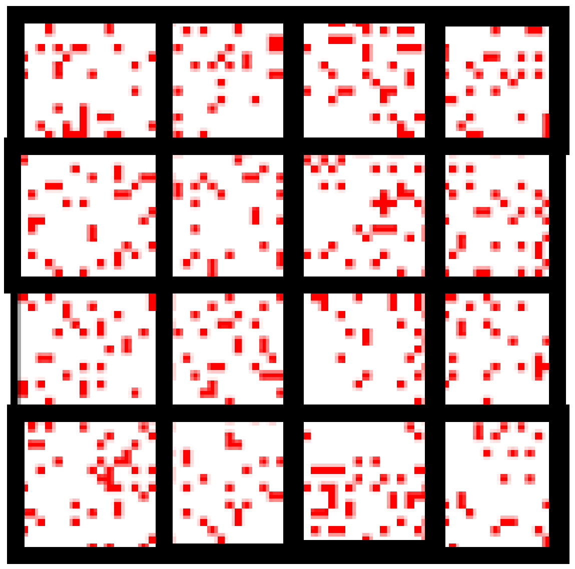

Figura 1: Representation of a spatial hash where each red dot is a bee and each square is a quadrant

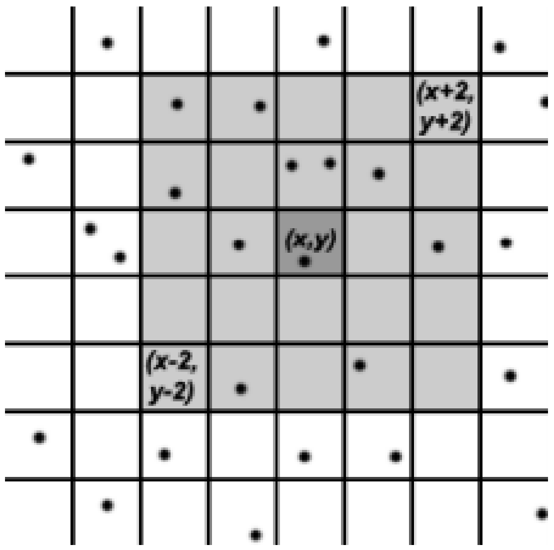

Figura 2: Representation of a spatial hash where each black dot is a bee and each frame is a quadrant

4.1 Data structure operations

Main(class)

Run the program creating a visible window to enter dice

Ventana(class)

Create a panel where you enter the data to analyze the number of data with a button to start when they are already entered.

Posicionpintar(class):

class position(subclass)

Receive the position in "x" and "y" of the bee.

Posicionpintar (builder):

Create a square where you can enter bees in each of the positions of the bee.

getCell(method)

Returns the square where the bee is

Returns the indexes of the cells occupied by the bee

getNearObjects();

Obtain nearby objects in all the cells it occupies

Draw:

Draw a circle that represents the bee graphically.

cuadradocolisiones(class):

cuadradocolisiones(builder):

Receive where the size of the map and the size of the divisions of the cell start and end.

addObject:
Add a bee to the map.

getObjectsInCell:

Returns the bees in each cell.

getCell:

Given some coordinates, it obtains the cell to which the bee belongs.

Paint:

Paint all the divisions of the cells and then paint the bees on the drawn.

run():

It detects which bees are colliding in each cell but collide the blue paint, and if they collide the pint of red.

Info(class)

info (builder):

Receive the information of the initial map the maximum height, the number of objects to be analyzed, the radius of the bee when drawn. Then he begins to read the data from a text file and begins to determine which cell each bee belongs to and adds it to the cell.

4. Spatial hash design criteria

This structure was designed in this way thanks to the efficiency in the identification of collisions and its low algorithmic complexity, the efficiency in the time of execution, this idea was granted thanks to a suggestion from our teacher.

4.3 Complexity Analysis

Draw	O(n)
cuadradocolisiones(constructor)	O(1)
addObject	O(n)
getObjectsInCell	O(n)
Paint	O(n)
run	O(n)
info (constructor)	O(n)

Tabla 1: Complexity algorithm methods

Execution times

N	4	10	100	1000	10000	100000
Tiempo de ejecucion promedio	0.00004 s	0.0009	1.189s	15.678 s	120.345 s	200.634 s

Tabla 2: Execution times of each data sets

4.5 Memory

N	4	10	100	1000	10000	100000
Memoria promedio que gasta	0.37 mg	1.29mg	30.78 mg	87.657 mg	123.354 mg	200.654 mg

Tabla 3: Spent memory for each data sets

4.6 Analysis of the results

N	4	10	100	1000	10000	100000
Memoria promedio que gasta hash espacial	0.37 mg	1.29 mg	30.78 mg	87.65mg	123.35 mg	200.65 mg
Memoria que gasta con n^2	0.56 mg	1.32 mg	15.32 mg	53.23 mg	90.54 mg	127.30mg
Velocidad de decteccion de colisiones hash espacial	0.00004 s	0.0009 s	1.189s	15.678 s	120.345 s	200.634 s
Velocidad de detecciongasta con n^2	0.00032 s	0.0043 s	4.234 s	44.234 s	300.678 s	1000.456s

Tabla 4: Analysis of the results obtained with the implementation of a special hash and two cycles with complexity n^2

5. CONCLUSIONS

In this Project we learned that there are multiple ways of approaching a problem depending on the needs of the user. The applications that this can have in the field of video games besides the multiple branches can also be demonstrated if an algorithm like this can be used.
We could show that an n2 algorithm spends more time dectectar collisions but hash space spends more memory one of the main problems we found was that we found very little information of these algorithms as a quadtree there is very little information to be documented more about the algorithm .
In the future we would like to explore other algorithms that help us solve this problem more efficiently and continue to learn more about this great topic and someday build a game or application that uses these algorithms

6.1 Future projects

We would like for a future project to make the graphical user interface function appropriately and graph the double points appropriately and not only to integer points and explore new solutions for this problem. It is considered that this topic is very important besides that the applications of this subject are many.

Thanks

We thank the EAFIT University for their help which contributed our knowledge to our colleagues, parents and friends for their comments that helped improve this research and provided moral support during the development of the algorithm.

REFERENCES

1. Bystr&ouml;m, C. (2018). Quadtree vs Spatial Hashing - a Visualization. Retrieved from http://zufallsgenerator.github.io/2014/01/26/visually-comparing-algorithms

2. Spatial hashing implementation for fast 2D collisions. (2018). Retrieved from https://conkerjo.wordpress.com/2009/06/13/spatial-hashing-implementation-for-fast-2d-collisions/

3. hashing?, W. (2018). When is a quadtree preferable over spatial hashing?. Retrieved from https://gamedev.stackexchange.com/questions/69776/when-is-a-quadtree-preferable-over-spatial-hashing

4.(2018).Retrieved from http://www.cs.ucf.edu/~jmesit/publications/scsc%202005.pdf

5.rubencm/spatial-hashing. (2018). Retrieved from https://github.com/rubencm/spatial-hashing/blob/swing/src/net/rubencm/spatialhashing/FormWindow.java