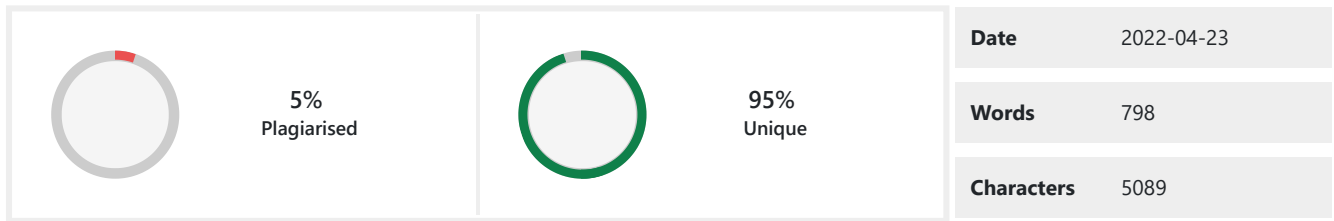


PLAGIARISM SCAN REPORT



Content Checked For Plagiarism

We propose to develop a program that can show a QuadTree view and data model architecture. Nowadays, many digital map applications have the need to present large quantities of precise point data on the map. Such data can be weather information or the population in towns. With the development of the Internet of Things (IoT), we expect such data will grow at a rapid pace. However, visualizing and searching in such a magnitude of data becomes a problem as it takes a huge amount of time. QuadTrees are data structures that are used to efficiently store point data in a two-dimensional environment. Each node in this tree has a maximum of four children. QuadTrees allow us to visualize the data easily and rapidly compared to other data structures. This project aims to build an application for interactively visualizing such data, using a combination of grid-based clustering and hierarchical clustering, along with QuadTree spatial indexing. This application illustrates the simulation of the working of the QuadTree data structure.

The QuadTree is a spatial data structure with a hierarchical structure. It's a tree with each level corresponding to a further refinement of the space in question. Though QuadTrees come in a variety of shapes and sizes, they can be used in a variety of ways. The concept can be applied to any dimension, and it is always a recursive subdivision of space that aids in the storage of information and provides the most vital or interesting details regarding space. In QuadTrees, we begin by adding pointers to its root node, which defines all potential space. The node divides into four child nodes when the number of points in the node reaches a predetermined maximum capacity. When any of those nodes has reached its full point capacity, it splits into four child nodes, and so on. QuadTrees have a range of applications; from internet services handling millions of requests every second, image compression, handling geolocation services, searching for nodes in 2-D areas, collision detection and more. Collision detection is a crucial feature in the majority of video games. Detecting when two entities collide is critical in both 2D and 3D games, since bad collision detection may lead to some very intriguing effects. Numerous games need the use of collision detection techniques to identify whether two objects have interacted, however, these algorithms are frequently costly procedures that may significantly slow down a system. In this paper, we will be addressing QuadTrees, and how we can use them to speed up collision detection by skipping pairs of objects that are too far apart to collide. We'll be writing a general-purpose, scalable and re-usable QuadTree library in Typescript and importing it into a visualization tool to depict its internal workings.

This project aims to provide a web application for visualizing the QuadTree structure. QuadTree. The users should be able to understand the working of the QuadTree and experience the simulation provided on the web application. This Visualizer provides an interactive environment where users can change configurations of the QuadTree and environment conditions at runtime.

A QuadTree [1][2][3] is a tree data structure with zero or four offspring at each node. Its key distinguishing feature is its method of recursively partitioning a flat 2-D [2] space into four regions. The data associated with a leaf cell differs depending on the application, but the leaf cell is a "unit of relevant spatial information." The subdivided areas or regions can be square or rectangular or any other form. This data structure was named a QuadTree by Raphael Finkel and J.L. Bentley in 1974.

The QuadTree is a data structure for organizing objects based on their locations in a two-dimensional space. By definition, a QuadTree [2] is a tree in which each node has at most four children. QuadTree implementations ensure that as points are added to the tree, nodes are rearranged such that none of them has more than four children. Figure 1 below illustrates the

general concept of QuadTree data structure.

The QuadTree partitioning strategy divides space [1][2] into four quadrants at each level.

When a quadrant contains more than one object, the tree subdivides that region into four smaller quadrants, adding a level to the tree.

A similar partitioning is also known as a Q-tree. QuadTrees are a way of partitioning space so that it's easy to traverse and search.

It is used extensively in computer graphics, image compression and is also used to represent spatial relations. Visualizing data points with a QuadTree [3] and checking and detecting collisions. The computer issue of identifying the collision of two or more bodies is known as collision detection. Collision detection [5] is a basic problem in computational geometry that has applications in a wide range of computer domains. Figure 2 shows the use case of Quadtree Visualizer.

Matched Source

Similarity 3%

Title: [GKQuadtree - Documentation - Apple Developer](https://developer.apple.com/documentation/gameplaykit/gkquadtree?changes=__7_10)

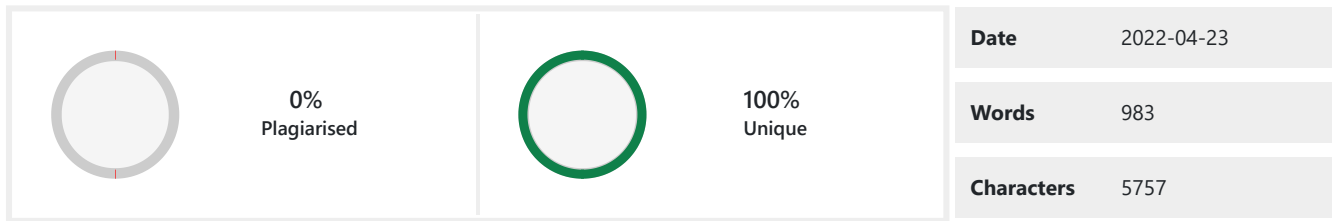
https://developer.apple.com/documentation/gameplaykit/gkquadtree?changes=__7_10

Similarity 3%

Title: [Quadtree - Wikipedia](https://en.wikipedia.org/wiki/Quadtree)

<https://en.wikipedia.org/wiki/Quadtree>

PLAGIARISM SCAN REPORT



Content Checked For Plagiarism

QuadTrees are also implemented for spatial indexing [3] while searching a particular point or location in a map. QuadTrees are very efficient as they can sparse through the maps very easily and quickly compared to other methods. Figure 3 shows the use case of QuadTree Spatial Indexing.

QuadTrees, for example, can handle a sparse Mario level a billion tiles across, where one of the tiles contains the finishing spot. A QuadTree will split the arrival spot into different cells and still use gigantic cells for the empty spaces. Figure 4 shows the use case of QuadTree in Gaming.

Hit detection:

For example, as seen in the maps above, there are a lot of points in space. If we wish to discover an arbitrary point P , we can do it inside that set of points. This quickly turns into a frantic process. We could check each and every single point to P , but when there are 1000 points yet none of them are P , we will have to do 1000 comparisons to figure out which point is P . Alternatively, we may get a very rapid lookup by retaining a matrix (a 2D array) of Booleans for each and every conceivable point in this space. However, suppose the area occupied by these points is 1 million \times 1 million so we need to keep 1,000,000,000,000 variables.

A QuadTree would seem to be a better choice in this scenario. To find P , the QuadTree [1] will determine which quadrant it is in. Then it will determine which quadrant within that quadrant it is in. Even if there are 1000 points in the space, it will only have to execute this seven times for a 100x100 space (provided points can have numerical value only). Once it's found that rectangle node, it just needs to test whether any of the four-leaf equals P .

Sparse Data using QuadTree:

QuadTrees are ideal for sparse data to search for a particular point. By only performing computations between items in comparable nodes/quads, QuadTrees aid in obtaining knowledge about which collisions in an environment are worth investigating.

QuadTree nodes split into four evenly-sized leaf nodes when the number of objects inside them reaches a certain capacity. Objects are inserted into a fresh QuadTree every iteration, which places each object in its deepest possible node. The QuadTree algorithm improves upon the naive $T(n) = \theta(n^2)$ algorithm and achieves $T(n) = O(n^2)$, $T(n) = \Omega(n \log(n))$.

Inadvertently, QuadTrees depending on pixels are a sort of trie.

B. Limitations of QuadTree

The fundamental drawback of QuadTrees is that comparing two pictures [4] that vary only in rotations or translation is nearly difficult. This is due to the fact that the QuadTree depiction of such pictures will be so distinct. The picture rotation methods offered are limited to revolutions of 90 degrees. There is no alternative rotation available, thus there is no translation facility. Figure 5.1 shows the original image and Figure 5.2 shows the rotated images. As we can see, it is not possible to compare two images that are different in terms of rotation.

C. Types of QuadTree

There are three types of QuadTrees:

1. Point QuadTree
2. Edge QuadTree
3. Polygonal Map QuadTree.

Some characteristics are shared by all QuadTrees:

- They break space down into flexible cells.
- There is a maximum capacity for each cell (or bucket). When the bucket reaches its full capacity, it separates.
- The QuadTree's spatial decomposition is followed by the tree directory.

D. Working of QuadTree

The Figure 6 below depicts how a QuadTree [7] alters as a result of insertion of a point E:

1. Make four boxes out of the current two-dimensional space.
2. If a box includes one or more points, make a child object that stores the box's two-dimensional space.
3. Do not generate a child for a box that does not contain any points.
4. Repeat with each of the children.

E. Algorithm

Three types of nodes are used in quadtree:

1. Point node: It is used to represent a point. It is always a leaf node.
2. Empty node: It is used as a leaf node to represent that no point exists in the region it represents.
3. Region node: This is always an internal node. It is used to represent a region. A region node always has 4 children nodes that can either be a point node or an empty node.

a. Insertion in QuadTree

Insertion: A recursive function for storing a point in a QuadTree.

1. As the current node, begin with the root node.
2. If the specified point is not within the boundary indicated by the current node, the insertion should be terminated with an error.
3. Determine the best child node to store the point.
4. If the child node is empty, it should be replaced with a point node that represents the point. Insertion should be stopped.
5. Replace the child node with a region code if it is a point node. For the point that was just replaced, use insert. Set the current node to be the region's freshly generated node.
6. Set the specified child node as the current node if it is a region node. Proceed to step 2.

b. Search in QuadTree

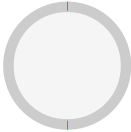

Search: This is a boolean function that determines whether or not a point exists in 2D space.

1. As the current node, begin with the root node.
2. If the specified point is not within the boundary indicated by the current node, the search should be terminated with an error.
3. Determine the best child node to hold the point in.
4. Return FALSE if the child node is empty.
5. Return TRUE if the child node is a point node and matches the specified point, else return FALSE.
6. Set the current node as the child region node if the child node is a region node. Proceed to step 2.

Matched Source

No plagiarism found

PLAGIARISM SCAN REPORT

 <div>0% Plagiarised</div>	 <div>100% Unique</div>	Date 2022-04-23
		Words 803
		Characters 5135

Content Checked For Plagiarism

c. Complexity

1. Time complexity:

- Find: $O(\log 2N)$
- Insert: $O(\log 2N)$
- Search: $O(\log 2N)$

2. Space complexity:

- $O(k \log 2N)$, where k is the count of points in the space and Space is of dimension $N \times M$, $N \geq M$

F. Collision in QuadTree

Because the data points in the visualizer are always shifting, collisions are unavoidable. Collision [5] is the meeting of two bodies, in this case data points in the shape of circles. The QuadTree visualization is built on top of a 2D Collision System with a restitution coefficient that can be adjusted to differentiate between elastic and inelastic impacts. Collision detection is a costly activity. One method for speeding up collision detection is to use QuadTrees

G. Coefficient of Restitution

The coefficient of restitution [6] is the ratio of the final velocity to the starting velocity of two objects after they collide. The restitution coefficient, written as 'e,' is a unitless quantity with values ranging from 0 to 1.

The coefficient of restitution is a quantity that represents the nature of the colliding materials. The coefficient of restitution informs about the elasticity of the collision. A fully elastic collision is one in which there is no loss of total kinetic energy.

The greatest coefficient of restitution for this sort of collision is $e = 1$. A fully inelastic collision is one in which all of the kinetic energy is wasted. They have a restitution coefficient of $e = 0$. The majority of real-life crashes occur in the middle.

The Coefficient of Restitution mathematical formula is as follows:

You can see from the following equation that you always divide the smaller number by a larger number. As a result, the restitution coefficient is always positive.

An architectural model is a simplified representation of a system. It is an estimate that captures the various system characteristics. It is a generalized form that has all of the system's critical elements. The process of modelling architecture entails determining the system's features and expressing them as models so that the system may be understood.

Architecture models make it possible to see information about the system represented by the model. Figure 8 depicts the web application's model architecture.

- Since we are using Next.js in our project, we first need to have Node.js.
- The web application works on `http://localhost:3000`.
- To run the application locally, we need to install the packages required using the npm command: `npm install package.json`
- Figure 9 shows the command prompt with the packages installed using the npm install commands.
- After installing all the dependencies, we then run the command: `npm run dev`.

- After we run the command: npm run dev. It will run the developer server.
- Figure 10 depicts the compilation and running of the server. The server is working on <http://localhost:3000>.

Since QuadTrees are a type of tree data structure in which each internal node has exactly four children, they are most often used to partition a two-dimensional space by recursively subdividing it into four quadrants or areas. The areas can be rectangular, square, or any other form. The QuadTree is used as a utility as part of the Maps SDK for iOS Utility Library. They've also been heavily used in image compression algorithms and higher-level design of 8-bit games like Mario. Eventually, we believe that QuadTrees can be used for memory management in a big and hierarchical database. It is one of the most crucial places we can use the QuadTree and it can be used to access varied data points and make searching efficient and fast.

Further work in this project can be to let the user visualize their own QuadTree using their own dataset. Users will have to give a dataset for the input and the visualizer will create a QuadTree based on the given dataset. Additional features could be added here such as the different color and shapes for different data points. Moreover, this project can be implemented as part of bigger projects such as Geolocation, Collision Detection Systems.

We explored a type of tree data structure named QuadTree, that can be used to represent 2-D spaces. In this process, we learnt how/why they are used in a range of applications from scaling up internet services to handle millions of requests per minute to their ever-present use in geolocation-based services like Maps and how we can build applications/libraries to implement the same in our apps/services. It can be concluded QuadTrees are extremely powerful data structures that are still heavily under-utilized in both the industry and community applications. By the time of completion of this project we've learned to develop scalable and reusable codebases for large projects, understood the fundamentals of API build and interaction and understood function in a time-bound manner and collaborate at scale across various tasks and disciplines.

Matched Source

No plagiarism found

Check By:  Dupli Checker