

Quad Tree Visualization for Effective Learning

Akash Sutar, Bhavesh Sutar

Department of Computer Engineering from Mumbai University, India

sutaraakash123@gmail.com , bhaveshsutar622@gmail.com

Mr. Sharique Ahmad

Designation - Assistant Professor, Universal College of Engineering, Mumbai University, India

sharique.ahmad@universal.edu.in

Abstract:

we intend to develop a program that provides a visualization of the quadtree structure and its data model architecture in modern times numerous digital mapping applications require the capability to display vast amounts of precise point data on a map such data can include meteorological information or demographic statistics for various towns with the expansion of the internet of things iot the volume of such data is expected to increase significantly however handling and searching through such an immense dataset poses a challenge as it demands considerable computational time quadtrees are an efficient data structure used for storing point data in a two-dimensional space each node in this hierarchical tree can have up to four children compared to other data structures quadtrees offer a more effective method for visualizing and processing large datasets rapidly the goal of this project is to create an application that enables interactive visualization of extensive point data this will be achieved through a combination of grid-based clustering and hierarchical clustering techniques alongside quadtree spatial indexing the application will serve as a simulation to demonstrate how the quadtree data structure functions in managing and displaying large-scale data efficiently

Keywords — QuadTree Visualizer, Q-Tree, Data Structure, Spatial Indexing, Coefficient of Restitution, Collision Detection, QuadTree Algorithm.

1. Introduction

The QuadTree is a hierarchical spatial data structure used for efficient space partitioning. Each level of the tree represents a progressively refined subdivision of the space it encompasses. Although QuadTrees can take on different forms, they serve multiple purposes across various domains. The fundamental principle can be extended to any dimension,

as it recursively divides space to store information while highlighting crucial or relevant details.

In a QuadTree, the structure begins with pointers to the root node, which represents the entire possible space. When the number of points within a node surpasses a predefined threshold, the node splits into four child nodes. This process continues recursively as child nodes reach their maximum capacity, further dividing into smaller sections.

QuadTrees find applications in several areas, including internet services managing vast numbers of requests, image compression, geolocation services, efficient node searching in two-dimensional areas, and collision detection. Collision detection plays a significant role in numerous video games, both in 2D and 3D environments. Properly detecting when two entities collide is essential, as poor collision detection can result in unintended behaviors. Many games rely on collision detection algorithms to determine whether objects interact, but these methods can be computationally expensive and slow down performance.

This paper focuses on QuadTrees and their ability to optimize collision detection by excluding object pairs that are too far apart to interact. We aim to develop a general-purpose, scalable, and reusable QuadTree library in TypeScript, which will be integrated into a visualization tool to illustrate its internal mechanisms.

The goal of this project is to build a web application that visually represents the QuadTree structure. Users will be able to explore the functionality of the QuadTree and interact with the simulation through the web interface. The visualizer will provide an interactive environment where users can modify

QuadTree configurations and environmental settings in real-time to observe their effects.

2. Literature Survey

A Modern Approach to Electoral Delimitation using the Quadtree Data Structure - The paper explores electoral delimitation using the quadtree data structure to address inefficiencies, gerrymandering, and uneven seat-to-population ratios in current methods. The proposed approach automates districting by recursively subdividing geographical space into four quadrants based on population. This model efficiently allocates constituencies while maintaining location-specific information. The algorithm is evaluated against existing techniques and proves to be effective in terms of computational complexity and boundary visualization, offering an optimized solution for political districting.[1]

Quadtree Generating Networks: Efficient Hierarchical Scene Parsing with Sparse Convolutions - Quadtree Generating Networks (QGNs) optimize semantic segmentation by using quadtrees instead of dense pixel grids, significantly reducing memory usage. This approach applies computationally intensive layers only to image regions with class boundaries, improving efficiency. It also enables flexible inference for constrained environments like embedded devices. Experiments on Cityscapes, SUN-RGBD, and ADE20k datasets show a 3% mIoU gain over similar-memory networks and a 4× reduction in memory consumption with minimal accuracy loss.[2]

Using Quadtree Representations in Building Stock Visualization and Analysis - This article explores the use of a quadtree representation for managing building data in Germany, addressing data protection concerns when aggregating geospatial information. Unlike traditional administrative zones, the quadtree approach enables flexible grid sizes for statistical and cartographic analysis. Using Hamburg and its surrounding areas as a case study, the method demonstrates improved data visualization and analysis, making it applicable to other sensitive datasets and regions. The approach supports European cooperation

standards (INSPIRE) and facilitates future cell-based analyses. [3]

“An Effective Way to Represent Quadtrees” - A quadtree can be efficiently represented without pointers by encoding each black node as a quaternary integer, forming a “linear quadtree.” This approach reduces storage requirements by at least 66% compared to traditional quadtrees. Several algorithms utilizing linear quadtrees are introduced, including encoding pixels, finding adjacent nodes, determining node colors, and superposing images. The first three algorithms run in logarithmic time, while image superposition operates in linear time relative to the number of black nodes. Additionally, the dynamic behavior of quadtrees can be effectively simulated using this method. [4]

3. Proposed system

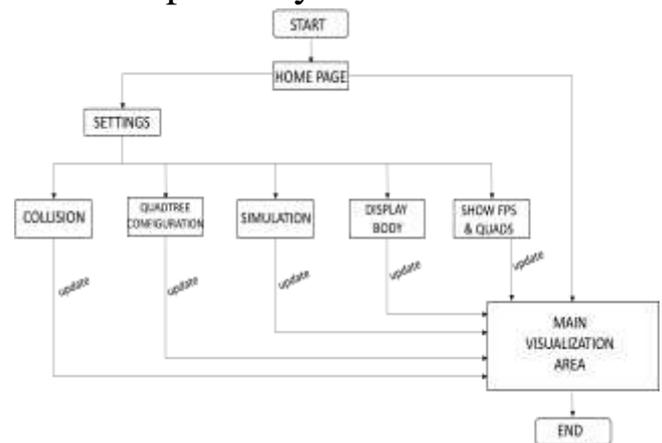


Fig 3.1

An architectural model is a simplified abstraction of a system, capturing its essential characteristics. It serves as a structured representation that includes all the key components of the system. The process of architectural modeling involves identifying the system's features and representing them as models to enhance understanding. These models provide a visual framework that helps in analyzing and interpreting the system's structure and behavior. Figure 1.0 illustrates the model architecture of the web application. The Quadtree Visualization System consists of a structured interface that allows users to observe and interact with a quadtree-based spatial partitioning system. The system is designed with a home page that presents the initial visualization and a settings

sidebar that provides extensive configuration options for fine-tuning the simulation.

A. Home Page

When a user accesses the application, the server processes the request and delivers the home page, which displays the initial quadtree visualization. By default, the system presents a 4x4 quadtree, showcasing the concept of spatial partitioning, where objects are divided into hierarchical regions. This setup enables users to understand how data is organized within the quadtree structure. Additionally, a settings button is available, allowing users to access further configuration options to modify various aspects of the simulation.

B. Settings Sidebar

Upon clicking the settings button, a sidebar panel appears on the left side of the screen, providing users with multiple customization options. The sidebar consists of five configuration sections, each dedicated to specific aspects of the simulation. These options empower users to manipulate the quadtree properties, collision settings, simulation behavior, and visualization preferences effectively.

1. Collision Settings

This section allows users to adjust the collision speed of objects, directly influencing the dynamics of interactions within the system. Higher collision speeds cause objects to interact more rapidly, impacting their movement and behavior. This setting is essential for controlling the realism and responsiveness of the simulation.

2. Quadtree Configuration

The quadtree configuration settings enable users to define critical properties that govern the structure and efficiency of the quadtree. Users can modify the Node Capacity, which determines the maximum number of objects a node can accommodate before it subdivides into smaller sections. Additionally, the Maximum Tree Depth setting establishes a limit on the depth of the quadtree, preventing excessive subdivisions that could degrade performance. These parameters significantly influence data organization,

computational efficiency, and object grouping within the system.

3. Simulation Settings

This section provides controls for managing the properties of objects within the simulation. Users can adjust the Radius, which defines the size of objects and impacts their visibility and interaction range. Additionally, the Spawn Count determines how many objects are generated during each spawning event, influencing the density of elements within the quadtree environment. These settings play a crucial role in balancing performance and visual complexity.

4. Display Body Options

Users have the ability to enable or disable various body-related features that affect how objects are generated and managed. The Spawn Bodies option allows users to dynamically generate new objects within the quadtree, enhancing the simulation's variability. The Clear Bodies function removes all existing objects, providing a reset mechanism to start a fresh simulation state. Additionally, the Random Bodies option spawns objects with randomized attributes, including size, position, and velocity, making each simulation instance unique and unpredictable.

5. Show FPS and Quads

This section offers visual debugging tools that assist in performance monitoring and spatial analysis. Enabling the Show FPS (Frames Per Second) option displays the current rendering speed, allowing users to assess the efficiency and responsiveness of the system. The Show Quads option toggles the visibility of quadtree boundaries, making spatial divisions within the environment clearly distinguishable. This feature is particularly useful for understanding how the quadtree adapts to object distribution and subdivision thresholds.

Quadtree Algorithm Explanation

A quadtree is a data structure that organizes points in a 2D space by recursively subdividing the space into four regions. There are three types of nodes in a quadtree:

1. **Point Node** – Represents a single point in the quadtree and is always a leaf node (a node without children).
2. **Empty Node** – Acts as a leaf node when there are no points in a specific region.
3. **Region Node** – An internal node that represents a larger area. A region node always has four child nodes, which can be either point nodes or empty nodes.

Insertion in a Quadtree

To insert a point into a quadtree, a recursive function is used:

1. Start at the root node of the quadtree.
2. If the given point is outside the boundary of the current node, stop the insertion process (since the point cannot be stored in the quadtree).
3. Identify the correct child node where the point should be placed.
4. If the child node is empty, replace it with a point node that stores the new point, and stop the insertion process.
5. If the child node already contains a point node, convert it into a region node (since multiple points need to be stored in this area). Then, insert the existing point into the new region node and proceed with inserting the new point.
6. If the child node is already a region node, move to that node and repeat the process from Step 2.

Searching in a Quadtree

The search operation in a quadtree is a boolean function that checks whether a specific point exists in the 2D space. The process follows these steps:

1. Start at the root node of the quadtree.
2. If the target point is outside the boundary of the current node, stop the search (as the point cannot be stored in the quadtree).
3. Identify the child node where the point would be located.
4. If the child node is empty, return FALSE (indicating that the point is not in the quadtree).

5. If the child node is a point node and matches the target point, return TRUE. Otherwise, return FALSE.
6. If the child node is a region node, move to that node and repeat the process from Step 2.

Collision Handling in Quadtree

In a Quadtree visualization, data points (represented as moving circles) are constantly shifting, making collisions inevitable. A collision occurs when two objects come into contact with each other. In this case, it refers to two moving points colliding within the simulation.

The Quadtree visualization system is built on a 2D collision detection system that includes a restitution coefficient. This coefficient determines whether a collision is elastic (where objects bounce off each other with minimal energy loss) or inelastic (where energy is lost upon impact).

Collision detection is a computationally expensive task, especially when dealing with multiple moving objects. One efficient way to speed up collision detection is by using Quadtrees. By organizing objects into a hierarchical structure, Quadtrees minimize the number of collision checks required, improving performance significantly.

Coefficient of Restitution

The **coefficient of restitution** (denoted as e) is a measure of how much energy is retained after two objects collide. It is calculated as the **ratio of the relative speed after collision to the relative speed before collision**.

This coefficient is **unitless** and has a value between **0 and 1**, depending on how much kinetic energy is conserved:

- **Perfectly Elastic Collision ($e = 1$)** – The objects bounce off each other without losing any kinetic energy.
- **Perfectly Inelastic Collision ($e = 0$)** – The objects stick together after colliding, losing all kinetic energy in the process.
- **Real-World Collisions ($0 < e < 1$)** – Most real-world impacts fall between these two extremes, meaning some energy is lost, but objects do not completely stick together.

Users have the option to **clear the Quadtree**, removing all existing data points while keeping the structure intact. When the Quadtree is empty, only the **grid structure** remains visible, as shown in **Figure 5.2**.

C. Spawning Bodies

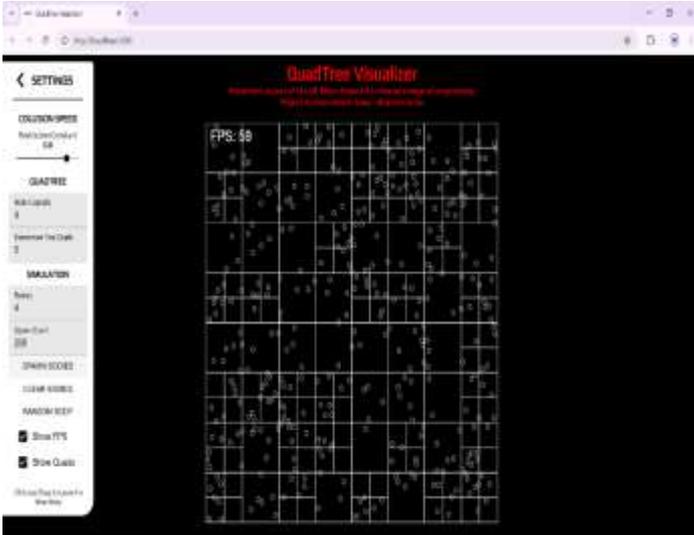


Fig 5.3 Spawning Bodies

The spawn bodies feature allows users to generate new objects (circles) inside the Quadtree. As objects are added, the Quadtree dynamically adjusts, subdividing regions as necessary. Figure 5.3 demonstrates how the Quadtree adapts to the placement of spawned objects.

D. Generating Random Bodies

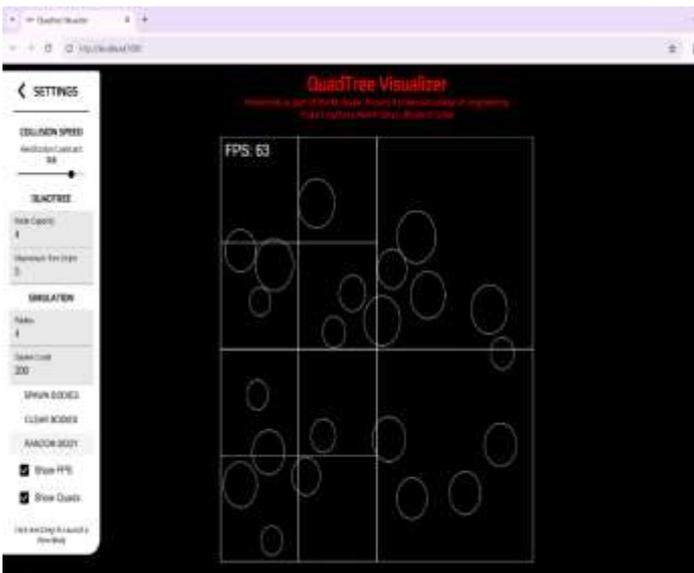


Fig 5.4 Generating Random Bodies

The application also supports **randomly generating bodies** within the Quadtree. This feature creates **circles at random positions** in the environment, leading to varied and unpredictable spatial divisions. **Figures 5.4** illustrate how objects are randomly distributed within the Quadtree.

E. Combining Random and Spawn Bodies



Fig 5.5 Combining Random and Spawn Bodies

Users can **combine both random and manually spawned bodies** to observe how the Quadtree handles different types of object placements. This creates a more complex environment with varied **spatial distribution and subdivisions**. **Figures 5.5** visualize this combined scenario.

F. Control Panel

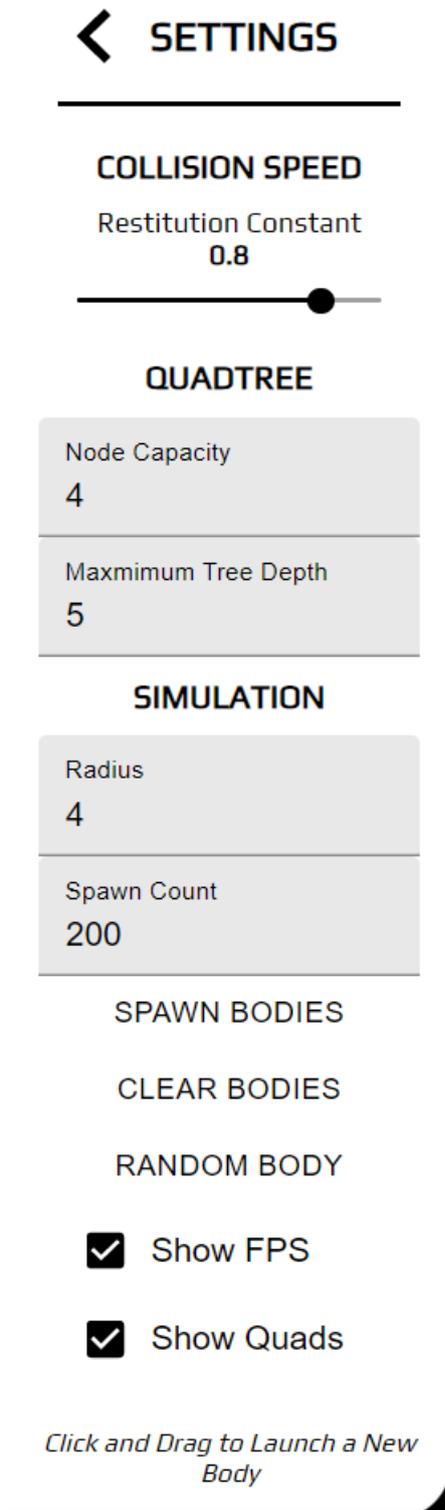


Fig 5.6 Control Panel

-The **control panel** allows users to fine-tune the simulation settings. It enables adjustments such as:

- Selecting **different object types**
- Modifying the **coefficient of restitution** (which affects collision behavior)

- Monitoring **frames per second (FPS)** for performance analysis

-**Figure 5.6** illustrates how the control panel provides flexibility in configuring different simulation parameters within the Quadtree.

6. Future Scope

Quadtrees are a specialized tree data structure where each internal node is divided into exactly four child nodes. They are commonly used to partition two-dimensional spaces, breaking them down into four smaller regions. These regions can take various shapes, including squares, rectangles, or other forms.

Quadtrees have been widely applied in spatial indexing, such as in the Maps SDK for iOS Utility Library, where they help organize geographical data efficiently. They also play a significant role in image compression algorithms and have been used in classic 8-bit games, such as Super Mario, for efficient rendering and collision detection.

Looking ahead, Quadtrees could be highly beneficial for memory management in large hierarchical databases. They provide an efficient way to store, organize, and search data, making querying large datasets much faster.

One potential enhancement to this project is to allow users to generate and visualize their own Quadtree structures using custom datasets. By providing their own dataset as input, users could observe how the Quadtree adapts to different data distributions.

7. Conclusion

In this project, we explored Quadtrees, a tree-based data structure used to efficiently represent and manage two-dimensional spaces. Through our research and implementation, we gained insights into why and how Quadtrees are applied across various domains, from handling large-scale internet services that process millions of requests per minute to their essential role in geolocation-based applications like Maps.

Our work demonstrated that Quadtrees are a highly efficient yet underutilized data structure with significant potential for broader adoption in both industry and community-driven projects.

Throughout the development process, we gained hands-on experience in building scalable and reusable codebases, deepened our understanding of API design and interaction, and enhanced our ability to work collaboratively on complex, time-sensitive tasks. This project not only strengthened our technical skills but also provided valuable experience in structuring and optimizing spatial data processing systems.

References

- [1] Sahil Kale, Gautam Khaire, Jay Patankar, and Pujashree Vidap (February 2024). "A Modern Approach to Electoral Delimitation using the Quadtree Data Structure" by arXiv for Quadtree model for automated electoral districting and boundary visualization.
- [2] Kashyap Chitta, Jose M. Alvarez, and Martial Hebert (September 2019). "Quadtree Generating Networks: Efficient Hierarchical Scene Parsing with Sparse Convolutions" by arXiv Quadtree networks for memory-efficient scene parsing.
- [3] Martin Behnisch, Gotthard Meinel, Sebastian Tramsen, and Markus Diesselmann (June 2013). "Using Quadtree Representations in Building Stock Visualization and Analysis" by Erdkunde Journal of Human and Physical Geography for Quadtree use in building data visualization and analysis.
- [4] Hanan Samet (August 1989). "An Effective Way to Represent Quadtrees" by The ACM Digital Library for Efficient quadtree representation for image compression and spatial indexing.