IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 10, 2021, accepted September 21, 2021, date of publication September 24, 2021,
date of current version November 2, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3115510

Research on Hybrid Index Based on 3D Multi-Level Adaptive Grid and R+ Tree

YONGSHAN LIU[1], TIANBAO HAO[1], XIANG GONG[1], DEHAN KONG[2], AND JIANJUN WANG[1]

[1]Department of Information Science and Engineering, Yanshan University, Qinhuangdao 066004, China
[2]Department of Information Engineering, Hebei University of Environmental Engineering, Qinhuangdao 066102, China

Corresponding author: Tianbao Hao (283454988@qq.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 61972334, in part by the 2019 Hebei University Higher Education Science and Technology Research Youth Fund Project under Grant QN2019044, and in part by Qinhuangdao Science and Technology Bureau (Research and Development of Science and Technology in Qinhuangdao) under Grant 201902A028.

ABSTRACT With the widespread application of Geographic Information System (GIS), three-dimensional spatial data, as the reflection of the real world entity, has an increasing amount of data, and the phenomenon of uneven data distribution appears. If a single spatial index structure is used to store and manage these data, there will be a waste of storage space and low query efficiency. A hybrid index structure based on 3D multi-level adaptive grid and R+ tree was proposed to solve these problems. The index structure was mainly composed of two structures, multi-level grid and R+ tree. Firstly, the data set was processed by the multi-level automatic grid algorithm based on normal distribution, and the length, width and height of the grid were obtained. Secondly, a multi-level adaptive grid structure was used to partition the data space quickly and effectively, and the advantage of zero overlap of the intermediate nodes of the R+ tree was used for efficient indexing. Finally, the maintenance and query algorithms of the index structure were given in detail, which solved the problem of low index establishment and retrieval efficiency under the condition of uneven distribution of massive data sets. In this paper, a data set subject to Gauss distribution was used to simulate the distribution of three-dimensional data. Through a large number of experimental comparison tests, it was proved that the hybrid index structure based on 3D multi-level adaptive grid-R+ tree proposed in this paper had good performance in both index structure construction and query in the case of massive data sets or uneven data distribution.

INDEX TERMS Adaptive algorithm, hybrid index, multilevel grid, R+ tree, spatial database.

I. INTRODUCTION

With the widespread application of Geographic Information System (GIS), spatial indexing technology affects the efficiency of spatial data query and GIS retrieval to a certain extent. The storage efficiency and query performance of the spatial database can be improved effectively by a reasonable spatial index structure. For example, the catalogue of a book can be regarded as an index. With this catalogue, the structure of the whole book can be understood more clearly, and the pages that readers want to read can be located quickly. Especially when dealing with a large amount of spatial data, the search will take a long time if there is no reasonable and efficient index structure. Therefore, the establishment of an efficient index structure is essential for the retrieval of spatial data.

In real life, there are many situations similar to the following. The data may be denser in some areas, while in other areas may be sparser, that is, the amount of data is large and the data distribution is uneven. For example, the distribution of shared bicycles. In urban centers, the number of shared bicycles is large and densely distributed. In remote mountainous areas, the number is small and unevenly distributed. There are many similar examples, such as the distribution of high-rise buildings in multi-center cities, the distribution of atmospheric pollutants, the vegetation coverage, population density, precipitation distribution, and so on. Grid index structure and R tree index structure are used mostly in the current spatial databases for data query processing. However, in view of the large amount of spatial data and uneven data distribution, if a single spatial index structure is used to store

The associate editor coordinating the review of this manuscript and approving it for publication was Vlad Diaconita.

and manage these data, there will be a waste of storage space and low query efficiency. For example, if a single grid index structure is used to partition the data, there will be a large number of sub-grids without data nodes, and it will bring unnecessary pressure to queries. If a single R tree is used to build the index structure, the depth of the tree will increase and a large number of intermediate nodes will appear, and it will increase the space complexity and time complexity of the system.

In three-dimensional space, the most commonly used data models are Axle Aligned Bounding Box (AABB) model, Oriented Bounding Box (OBB) model, 8-DOP Model and convex hull model [1]. However, there is spatial redundancy between the actual spatial object and the representation model. Generally, the spatial redundancy of the AABB model is greater than that of the OBB model. In order to make the results of the query more accurate, the impact of spatial redundancy should be minimized when the query is performed. In the existing research on index structure, the most widely used model for representing three-dimensional spatial objects is the Minimum Bounding Box (MBB) model [2]. The MBB model is easier to represent the spatial objects and the query speed is faster. However, compared with the OBB model, 8-DOP model and convex hull model, the MBB model is more prone to spatial overlap. Using it to build an R tree index structure will lead to serious multi-path problems, thereby reducing the speed and accuracy of the query. Therefore, the combined use of data model representation and index structure establishment plays a vital role in queries in three-dimensional space.

At present, the index structure for three-dimensional space mainly includes R tree, octree, grid, or hybrid index and so on [3]–[8]. The single index structure has many problems when dealing with three-dimensional objects. The octree is the expansion of the quadtree in three-dimensional space, and it has the problem of depth imbalance. The grid index divides the spatial area into sub-grids of equal or unequal size. Each sub-grid contains a certain number of spatial objects. It has a good I/O performance and fast search speed in general. But this index structure wastes main memory buffer and the secondary storage space as its catalog is loose. In addition, when local matching query or range query is performed, adjacent items may point to the same grid unit, so there may be multiple scans of the same item. Hybrid index structure mainly includes several modes such as octree and R tree or grid tree and R tree [9], [10]. Hybrid index can take advantage of the unique advantages of different index structures. It is more flexible and more efficient than a single index structure. Therefore, in recent years, the study on the structure of the hybrid index has received widespread attention.

In order to solve the problem of large amount of data and uneven data distribution in three-dimensional space, an in-depth study had been conducted. The traditional MBB model is used to represent the data in three-dimensional space, and a hybrid index structure of grid and R+ tree is adopted. The grid can divide multiple sub-grids automatically by an adaptive algorithm according to the spatial data distributed unevenly, and then the sub-grids are further divided according to the threshold set by the user. The spatial data within the sub-grid adopts a zero-crossing R+ tree index structure to avoid multi-path problems during query. The structure of the paper is as follows. In section 1, an overall introduction of the paper was described. In section 2, a brief summary of related work was given. In section 3, a hybrid index structure was proposed based on three-dimensional multi-level adaptive grid and R+ tree. The maintenance algorithm of the hybrid index structure was described in section 4. Section 5 described the query algorithm based on the index structure, including precise point query and k-nearest neighbor query. In section 6, the experimental environment was set up and experimental comparison was made to prove the effectiveness and robustness of the index structure. In section 7, we gave the conclusion of this paper.

II. RELATED WORK

At present, the studies of spatial index structure are mainly based on tree structure and grid structure. There have been many researches based on tree structure, such as R tree [11]–[14], quadtree [15], octree [16], and some other variant trees of R tree and so on [17]–[20]. As early as 1984, the concept of R tree was proposed by Guttman *et al.* [21]. R tree is an extension of B+ tree, which could better solve the problem of data storage and query. Sellis *et al.* [22] had proposed R+ tree which is a variant of the R tree. This index structure could avoid the problem of a large number of overlaps effectively in the middle node of the R tree, and improve the query of spatial data. R* tree was designed by Guttman in 1990 [23]. It forced the node to be inserted again using the principle of node optimization. R* tree could improve the space utilization rate and reduce the number of node splits. Yang and Huang [24] proposed a hybrid index based on extended quad-tree and three-dimensional R tree organization. This index structure was applied to large-area, high-density ground point cloud data management and visualization applications. Grid index structure was based on the idea of hashing [25]. It built the index structure after reducing the dimension by dividing the data set into "$m \times n$" small blocks, and then encoded and stored it according to a certain method. The structure of grid was simple, and it could divide two-dimensional or three-dimensional space quickly for efficient query. Huang *et al.* [26] described the grid structure systematically and proposed the concept of multi-level grid to solve the problem of uneven data distribution in two-dimensional space. In 2016, Tang *et al.* [27] proposed an algorithm to divide the space into grids, and realized the rapid division of spatial data in the form of z-sorting curves.

Each index structure had its own advantages, but also has its own limitations. Scholars at home and abroad had conducted in-depth research to improve the index structure and make it have better query efficiency. Xu *et al.* [28] proposed a nearest neighbor query algorithm based on space-filling curve grid division. This algorithm sorted the points linearly

in the grid, using the space-fulfilling curve's dimensionality reduction characteristics and data clustering characteristics. The nearest neighbor could be obtained by visiting the points in the grid where the query point was located and the points in the nearby grid around it. A ciphertext sorting search encryption scheme based on B+ tree index structure was proposed by Niu *et al.* [29]. The index structure of B+ tree was used to improve the retrieval speed of ciphertext transactions on the blocked chain. Zhang [30] analysed the electric power big data and proposed a hybrid index structure based on the B+ tree to improve the query efficiency. Gong [31] proposed an expansion method of three-dimensional R tree index that took into account detail of multiple levels. Based on global optimization and three-dimensional clustering analysis, the balanced structure of index was established. And they designed the node selection algorithm. In the algorithm, the searching was from bottom to top firstly, and then from top to bottom. The node splitting algorithm based on k-medoids clustering algorithm was used. So the node size was uniform, the shape was regular, and the overlap was reduced. At the same time, they proposed an adaptive algorithm [32] based on the extended structure of the 3D R tree index in 3D city model. This algorithm could modify the multi-level of detail definition parameters in real time according to the current performance and adjust the complexity scene quantitatively. A database management algorithm based on combined 2D and 3D indexing of very large point-cloud data was proposed by Wang and Guo [33], for extracting the point cloud in need and improving the query efficiency. The algorithm used 3D-R tree and QMBB tree index to access self-organized spatial data and point cloud data. Sharifzadeh and Shahabi [34] proposed a VoR-Rtree hybrid index built by R tree combined with Voronoi diagram. It merged the Voronoi diagram into the R tree. Using the index, the V diagram searched the neighborhood quickly and reduced R tree overlapping space for various nearest neighbor queries. Gong *et al.* [35] proposed an efficient management method for point cloud data based on octree and 3D R tree, which could solve the problem of large-scale data management, but it would result in wasting huge storage space when the octree was too deep. Song *et al.* [36] proposed a hybrid tree spatial index structure in 3D GIS. Octree and R* tree were used to solve the problem that a single spatial index structure restricted retrieval performance as the amount of data increased. The experiment used distributed random data and the results were satisfactory, but the R* tree still had intermediate nodes overlapping, and the multi-path query still restricted the improvement of index performance. Gong *et al.* [37] proposed a hybrid index method based on 3d grid-R Tree. Although the problem of uneven data distribution was solved to a certain extent, the single-level grid used an artificial parameter to divide the grid. It did not make sure the rationality of the division, and resulted in data redundancy. The R tree had multi-path queries that still restricted the improvement of index performance.

Among them, R+ tree was a variant tree of R tree. It could realize the path clearly and uniquely when the data set was searching. The R+ tree divided the objects that spanned the subspace into two or more, and then stored each part separately in the node of the tree. It could ensure that the intermediate nodes would not overlap. However, the R+ tree needed to be split up down to some child nodes, which would increase the number of layers of the tree, and reduced the efficiency and space utilization rate during the query. Grid index was used for the 3D data set. If the sub-grids spanned multiple grids at the same time, there would be redundancy in the index. In this situation the space complexity would increase. This paper proposed a hybrid index structure based on 3D multi-level adaptive grid and R+ tree, which used a multi-level grid automatic division algorithm based on normal distribution to divide the grid and obtained $m \times n \times t$ sub-grids. If the number of the data set in the sub-grid exceeded the threshold set by the user, the sub-grid continued to be divided. The R+ tree was used to build the index structure in the sub-grids. Since the intermediate nodes of the R+ tree had zero overlap characteristics, it had high query performance.

III. 3D MULTI-LEVEL ADAPTIVE GRID AND R+ TREE HYBRID INDEX STRUCTURE

A. MULTI-LEVEL ADAPTIVE GRID STRUCTURE DESIGN

Multi-level grid [38]–[40], which is a kind of grid structure, is based on the idea of hashing. The idea is to divide the entire data space horizontally and vertically into several equal small blocks, each of which is a bucket. The identification number of the entity object in the small block is put into the corresponding bucket of the small block. In order to improve performance, the small blocks are continued to be divided until the cut-off condition is met. Dividing the multi-level grid is the first and most critical step to determine the performance of the entire index structure.

The data structure of the index is generally composed of two parts, an array and a single list. Each bucket of the grid has a pointer to the previous physical node. There is no entity in the bucket if the pointer is empty. In addition to the actual label, the entity node also has a pointer to the next entity and some other information indicating the subordinate spatial index of the entity.

In the 3D space, the space is divided into $m \times n \times t$ blocks, an $m \times n \times t$ grid have $m \times n \times t$ buckets, and the i-th bucket can be expressed by $Buck[index](0 \leqslant index < m \times n \times t)$, the relationship between *Buck* and *Block* is as follows.

$$\begin{aligned} Buck[index] &\leftrightarrow Block[i, j, k] \\ i &= index/(n \times t) \\ j &= (index - i \times n \times t)/t \\ k &= (index - i \times n \times t)\%t \end{aligned} \tag{1}$$

Among them, when $1 \leqslant index \leqslant m \times n \times t$, "$\leftrightarrow$" indicates a one-to-one correspondence. The number of layers of the i-th bucket in the three-dimensional grid is represented by i, $i = index/(n \times t)$, the number of rows of the i-th bucket in the three-dimensional grid is represented by j, $j = (index - i \times n \times t)/t$, and the number of columns of the i-th bucket in the

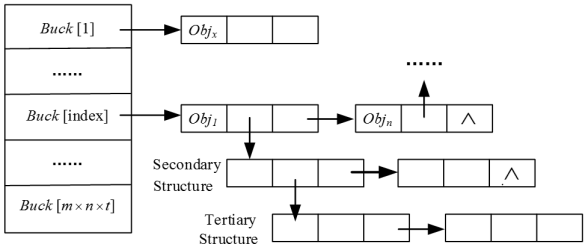


FIGURE 1. 3D multi-level grid index structure.

three-dimensional grid is represented by k, $k = (index - i \times n \times t)\%t$. Therefore, a one-to-one correspondence is formed. If we know the bucket number, we can locate the grid position quickly.

Taking into account that the grid is divided into too many levels, it will lead to excessive storage space inevitably and reduced query efficiency, so three levels of division is supported at most in the grid index of our paper. If the second or third level uses buck structure, it will take up a lot of storage space inevitably. Therefore, in order to save storage space and improve query efficiency, the second and third level spatial indexes are represented by a dynamic linked list.

The first-level division of the multi-level grid structure is to divide the whole spatial space, and $m \times n \times t$ sub-grids $Block[i, j, k](0 \leqslant i < m, 0 \leqslant j < n, 0 \leqslant k < t)$ are obtained. The second-level grid divides the $Block[i, j, k](0 \leqslant i < m, 0 \leqslant j < n, 0 \leqslant k < t)$ that exceeds the threshold range again into $m_2 \times n_2 \times t_2$ sub-grids $Block_2[i_2, j_2, k_2](0 \leqslant i_2 < m_2, 0 \leqslant j_2 < n_2, 0 \leqslant k_2 < t_2)$, The third level grid divides the $Block_2[i_2, j_2, k_2](0 \leqslant i_2 < m_2, 0 \leqslant j_2 < n_2, 0 \leqslant k_2 < t_2)$ that exceeds the threshold value into $m_3 \times n_3 \times t_3$ small blocks, then the division is over. The subscript 2 in the above formula represents a 2-level division. The three-dimensional multi-level grid index structure is shown in Figure 1. $Buck[index]$ stores the first divided grid $Block[i, j, k]$ in the form of array, and the subsequent grid divisions are stored in the form of linked list Obj_i. Each Obj_i has a pointer to the next node. The pointer of the last node points to null.

In order to split the grid reasonably, a definition of the density of distribution ($disd$) is proposed. $disd$ is the density of the element distribution in a unit volume. The number of objects in the sub-grid is represented by $count$, and the volume of the sub-grid is represented by $volume$, the calculation formula is as follows.

$$disd = \frac{count}{volume} \tag{2}$$

The density distribution size threshold is ε, and it is the density threshold entered by the user. When $disd \geqslant \varepsilon$, we consider the grid is a dense unit and need to be divided again. When $disd < \varepsilon$, we consider the grid is a sparse unit, and do not continue to divide it down.

The data structure designed for the multi-level grid is expressed in C # language. According to the need, we give the definitions in 3D space as follows.

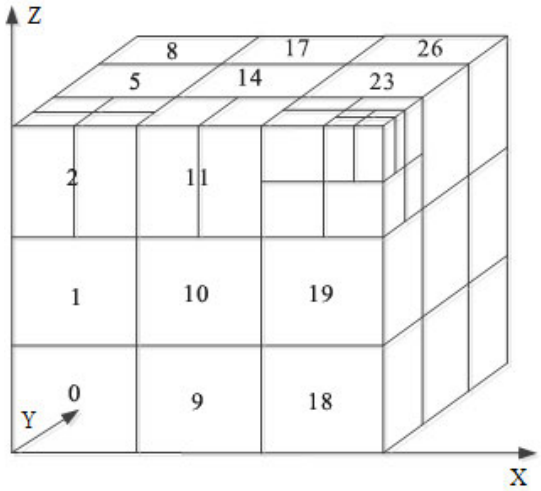


FIGURE 2. Multi-level grid.

Definition 1 (Point): Point(ID, x, y, z) represents the point in 3D space. *ID* is the unique identification of the point in 3D space. x, y, z is the coordinates of the point on the axis.

Definition 2 (GridNode): GridNode(*ID*, *gridCount*, *leftPoint*, *rightPoint*, *flag*, *List<Point> Data*) represents the node in multi-level grid. *ID* is the index number of the node, which represents the unique identification of the sub-grid formed after the grid is divided. The *leftPoint* and *rightPoint* represent the space index coordinates of the lower left front and upper right rear of the sub-grid respectively after the grid is divided, which is used to represent the actual position of the node cube in the space. The *flag* is a boolean type, used to determine whether the grid needs to be divided into the next level. If it is true, it means the number of objects in the current subspace is still too large, and this subspace needs to be re-divided. If it is false,the subspace no longer divide down. The division of multi-level grids is shown in Figure 2. The space is first evenly divided into 27 small blocks. The 3rd block, the 12th block, and the 21st block exceed the threshold and need to be further divided. The steps for dividing grid are as follows.

Step 1. The boundary coordinates X_{max}, X_{min}, Y_{max}, Y_{min}, Z_{max}, Z_{min} can be obtained by the given data set, then we can get the root node on the X axis, Y axis, and Z axis by the following formula.

$$\begin{aligned} L_X &= X_{max} - X_{min} \\ L_Y &= Y_{max} - Y_{min} \\ L_Z &= Z_{max} - Z_{min} \end{aligned} \tag{3}$$

Step 2. The grid is divided for the first time. We use the multi-level grid automatic division algorithm based on normal distribution, and divide it into $m \times n \times t$ basic grids $G_{000}, G_{001}, \ldots G_{ijk}, \ldots (0 \leqslant i < m, 0 \leqslant j < n, 0 \leqslant k < t)$. Then in accordance with the order of the $Z \rightarrow Y \rightarrow X$ axis, we put the sub-grids formed after the first grid division are sequentially coded. The next division of the sub-grid also prioritizes the division of the sub-grid in the order of the

$Z \rightarrow Y \rightarrow X$ axis, we can remove the sub-grid that does not exist in the grid to free up storage space.

Step 3. Continue to divide the sub-grid. We can count the number of objects contained in the sub-grid space, and calculate the distribution density in this sub-grid. According to the degree of density, we can decide whether to divide the sub-grid again. If the conditions for continued division are met, divide it and all the elements into the corresponding subset according to the coding order of the grid division in step 2. For example, G_{ijk} divides into eight sub-grids, as G_{ijk1}, G_{ijk2}, G_{ijk3}, G_{ijk4}, G_{ijk5}, G_{ijk6}, G_{ijk7}, G_{ijk8}.

Step 4. During the division process, two situations may be encountered. An object falls completely in the grid, and we store it in the linked list corresponding to the grid directly. The object spans multiple grids, and we do not save it in the divided sub-grid directly, but store it in the linked list corresponding to the upper-level grid for better query.

Step 5: Step 3 is called recursively in a loop, until the distribution density of all subsets is less than the threshold we set, or the grid has been divided for three times, then all the multi-level grid construction is completed.

The determination of grid parameters is the key factor for meshing. Although the three-dimensional multi-level grid and R+ tree hybrid index allows users to assign different values to improve the adjustability of the spatial index, it adds great difficulty to the user. Therefore, if the system can adjust the spatial index parameters automatically in combination with the given data set during the operation of the spatial index, and obtain more reasonable and effective three-dimensional grid spatial index parameters, it will be the best solution.

Based on the idea, a method that can adjust the spatial index parameters $m \times n \times t$ dynamically, is proposed to realize the establishment of an efficient index according to the data set. As the method is studied on the basis of normal distribution, it is called a multi-level grid automatic division algorithm based on normal distribution.

The multi-level grid automatic division algorithm based on normal distribution is mainly used for the data that obey Gauss distribution, and it still adapt to data that does not meet normal distribution. The algorithm is mainly used to deal with the length, width and height of all objects in the three-dimensional space. Firstly, we judge whether the data set of object obeys the normal distribution. If the data set does not obey, a scale factor $K(0 < K \leqslant 1)$ is designed. This scale factor less than 1 and can be used to control the side length of the grid. If the data set satisfies the Gauss distribution, the scale factor K will not play any role in the realization of the whole algorithm. Therefore, when the data meets the Gauss distribution, the "3σ criterion" and the calculated MBB length, width, and height values are used to divide the grid. If the data set does not meet the Gauss distribution, the scale factor K is used, because the value obtained according to $\mu + \sigma$ no longer meet the user's requirements well, and this scale factor needs to be adjusted. Experiments have proved that when K is in the interval of 0.90-0.95, the query efficiency is relatively high, so we no longer set the value of K, and default it to 0.9. The calculation formulas for the mean and standard deviation are shown below.

$$\mu = \frac{1}{n}\sum_{j=1}^{n} x_j$$
$$\sigma = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(x_j - \overline{x})^2} \tag{4}$$

The base idea of the algorithm is as follow. Firstly, we take the length of all spatial entities of MBB in the three-dimensional space as the study object, and sort the sample quickly to obtain a set of arrays sorted from small to large. Secondly, we use the normal distribution test algorithm to calculate the mean value μ and standard deviation σ of the sample, and use $\mu + \sigma$ as the initial value of the grid division, and increase by 0.1σ each time by m times. Finally, $\mu + \sigma + m \times 0.1\sigma$ is used as the length of the grid division. Thus we get the number of rows m of the grid division. The m is determined by the proportional coefficient K. In the same way, if the width of all spatial entities of MBB is used as a sample, the number of grid divisions n can be obtained. If the height of all spatial entities of MBB is taken as a sample, the number of grid divisions t can be obtained. Taking the length of the grid as a sample, the specific algorithm is shown in Algorithm 1.

Firstly, the quick sort algorithm is used to sort the Dataset P of the samples $(x_1, x_2, \ldots, x_n)$ to get the set of array sorted in ascending order, $array[0], array[1], \ldots, array[n-1]$, the total number of samples is n. Secondly, we use the formula 4 to get the mean value μ of the sample and the standard deviation σ. Finally, we judge $\mu + \sigma + proportion \times \mu \leqslant array[i]$. If $(i+1) \setminus n < K$, $proportion = proportion + 0.1$, $i++$, the process continue to traverse.If $(i+1) \setminus n \geqslant K$, $position = i + 1$, the algorithm is end and return the array[position-1].

B. R+ TREE INDEX STRUCTURE DESIGN

R+ tree index structure is similar to the R tree. It is a variant of the R tree. It mainly solves the overlap problem of the directory rectangles in the R tree and realizes the zero overlap of the nodes in the middle of the tree, which improves the query performance greatly.

The data structure designed for the multi-level adaptive grid and R+ tree structure is described in C # language. According to the need, we give the definitions of structure as follows.

Definition 3 (Root): *Root*(*maxNodeEntries*, Dictionary <*int*, *Node*> *nodeMap*, *treeHeight*, *rootNodeId*, Dictionary <*int*, *T*> *IdsToItems*) represents the root of the multi-level adaptive grid and R+ tree. *maxNodeEntries* is the maximum number of leaf nodes. The node information of the structure is stored in *nodeMap*, *treeHeight* represents the height of R+ tree. Root node id is the *rootNodeId*, and its leaf node items are recorded by *IdsToItems*.

Algorithm 1: Adaptive Algorithm

Input: Dataset P
Output: $Array[position]$

$array[n] \leftarrow Quicksort(x)$;
$\mu \leftarrow \frac{1}{n}\sum_{j=1}^{n} x_j$;
$\sigma \leftarrow \sqrt{\frac{1}{n}\sum_{j=1}^{n}(x_j - \bar{x})^2}$;
int *position*, *proportion* ;
for *each i in* $n+1$ **do**
  if $\mu + \sigma + proportion \times \mu \leqslant array[i]$ **then**
    if $(i+1) \setminus n < K$ **then**
      $proportion = proportion + 0.1$
    end
    else
      $position = i + 1$;
      break ;
    end
  end
  else
    $i++$;
  end
end
return $Array[position - 1]$;

Definition 4 (Node): *Node*(*nodeId*, *Rectangle* MBB, *Rectangle*[] *entries*, *ids*, *level*, *entryCount*) represents internal nodes of the multi-level grid and R+ tree. *nodeId* is the id of the node. MBB is the bounding box of the node. All the bounding boxes contained in the MBB is represented by *entries* and their ids are *ids*. The number of the layer where the node is located is *level* and the number of MBB in node is *entryCount*.

Definition 5 (MBB): MBB(*DIMENSIONS* = 3, *max*, *min*) represents the cuboid information of each node. *DIMENSIONS* = 3 means the dimension is 3. *max* represents the MBB's upper right rear coordinates and *min* represents the MBB's left front and bottom coordinates.

It can be seen from Figure 3 that the dashed boxes A, B, C, and P are the directory rectangles, and they do not overlap each other. When a spatial data object is in two directory rectangles at the same time, it will be split into two spatial objects and stored in two directory rectangles respectively. For example, object G crosses A and P directory rectangles, and it will be split into two objects and stored in the nodes A and P. Thus, creating an unique search path by avoiding the overlap of the directory rectangles.

C. THE OVERALL STRUCTURE DESIGN

The multi-level adaptive grid structure and the R+ tree structure have been introduced above. And we propose a combination of these two structures and a new hybrid index structure based on the three-dimensional multi-level adaptive grid and R+ tree.

In the process of dividing the multi-level adaptive grid and R+ tree hybrid index structure, the three-dimensional space

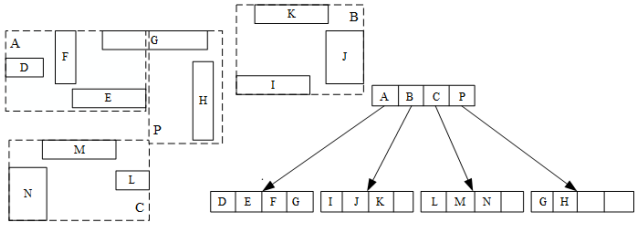


FIGURE 3. R+ tree structure.

is divided into $m \times n \times t$ equal sub-grids through the first grid division. However, if the m, n and t are divided, the index structure will occupy a huge storage space, the redundancy and query time will also become larger and longer. The overall performance will decrease. Therefore, this paper proposes an automatic multi-level meshing algorithm based on normal distribution to solve the problem.

Firstly, we use the adaptive algorithm to divide the grid reasonably. Then each sub-grid as the entire spatial range of the R+ tree index structure is the root node of the R+ tree. On this basis, the R+ tree index structure starts to be built. Secondly, the information of the spatial object is stored in the leaf node of the R+ tree. Therefore, the overall idea of the three-dimensional multi-level adaptive grid and R+ tree is to use the advantages of the multi-level adaptive grid to divide the space area quickly at first, and divide the entire data space reasonably. Finally, we use the R+ tree directory rectangle that do not intersect and overlap each other to build an R+ tree for the divided sub-grids, which can reduce the space overlap, and improve the overall performance.

For the convenience of description and display the grid structure, we take a two-dimensional multi-level grid as an example, as shown in Figure 4. Supposing 3 objects are stored in the grid at most, it will be graded down if the number exceed it. The four point objects a, b, c, and d are all in the same grid, corresponding to disk P, if we put them into page P, it exceeds the maximum value that the grid can hold. At this time, the grid is divided until the set conditions are met, then the R+ tree index structure is built.

Figure 4 shows the physical storage mode and index structure of the three-dimensional spatial data in the system. After the first partition of the grid, the data is stored in the form of a matrix. The query efficiency is the highest, and the time complexity is O(1). However, there are a large number of sub-grids in the matrix without data, and a lot of storage space is wasted. We need to release the sub-grids that have no spatial data. Then there are fewer sub-grids in which the number of data exceeds the threshold. Therefore, the linked list is adopted for further grid partition, which reduces the storage space and facilitates insertion and deletion operations. Figure 5 is a logical structure of the entire hybrid index structure. From the figure, we can easily see that the data is displayed hierarchically. Firstly, the entire three-dimensional spatial data is partitioned by an adaptive algorithm to generate many sub-grids, and the sub-grids continue to be partitioned when the number of data exceeds the set threshold.

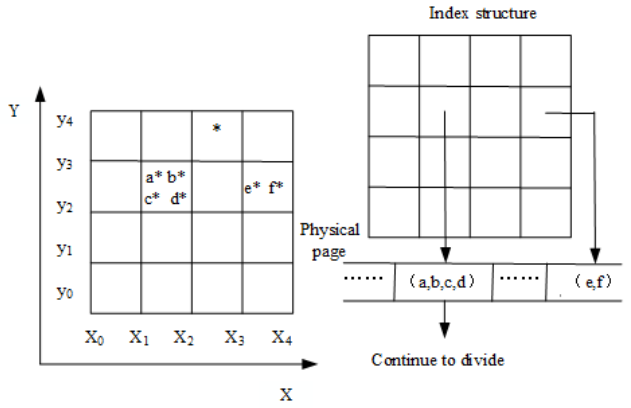


FIGURE 4. Grid space index structure.

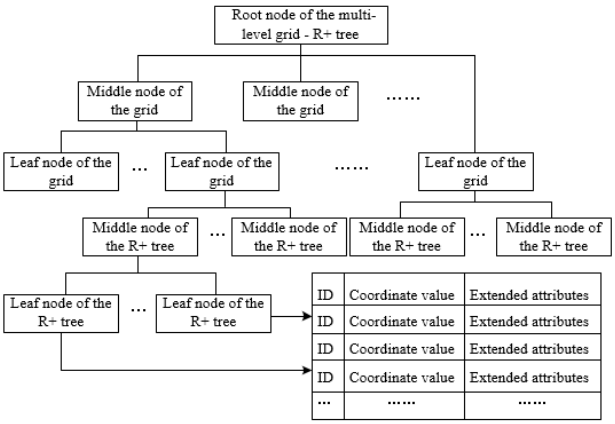


FIGURE 5. Index structure of multi-level grid and R+ tree.

Secondly, when the grid partition is completed, the number of data in the sub-grids is controlled within the threshold set by the user. R+ tree index structure is used for the data in the sub-grids. The depth of the R+ tree is controllable. Finally, the leaf nodes of the R+ tree store the three-dimensional spatial data, including the ID of the data, the spatial coordinate value and other extended attributes of the data.

This hybrid index structure mainly uses a multi-level grid automatic division algorithm to preprocess the data set. It can ensure that the multi-level adaptive grid is reasonable and the three-dimensional space is divided quickly when the data is distributed unevenly and the amount of data is large. In order to reduce the overhead of storage space, the grid is divided into three levels at most. If the grid can't be divided, the R+ tree will be built on each sub-grid, and the data information of the object will be stored in the leaf nodes of the R+ tree. At this point, the construction of the three dimensional multi-level adaptive grid and R+ tree is completed. Next, the maintenance of the hybrid index structure and the query based on the hybrid index structure are introduced, and the excellent performance of the index structure is verified through experiments.

IV. MAINTENANCE OF THREE-DIMENSIONAL MULTI-LEVEL ADAPTIVE GRID AND R+ TREE HUBRID INDEX STRUCTURE

After the construction of the three-dimensional multi-level adaptive grid and R+ tree hybrid index has been completed, certain maintenance (for data insertion, deletion, and query algorithms) of the index are required in the realization of the index function in a real sense. In this section, we introduce the insertion and deletion of hybrid indexes only. We will focus on the introduction of query retrieval in the next section. There are three types of spatial objects in two-dimensional space, point, line, and area. So the insertion algorithm and deletion algorithm are also relative to them. In three-dimensional space, in addition to points, lines, and area, spatial objects also have body types, so insertion and deletion algorithms are the description of these four types. For the sake of convenience and clarity to describe, all the insertion and deletion operations involved here are for point objects. As others are composed of points.

A. INSERT OPERATION

The insert operation algorithm of the three-dimensional multi-level adaptive grid and R+ tree hybrid index structure is shown in Algorithm 2.

The specific description of the algorithm is as follows. According to the automatic division algorithm based on normal distribution, the grid can be divided. The grid where the new inserted 3D point element object can be located by formula 1, and the grid index ID can be obtained. Then, the insert operation of the three-dimensional multi-level adaptive grid and R+ tree will insert the sub-R+ trees of the independent and non-overlapping multi-level grids. The detailed process is as follows.

The root node is the incoming parameter node N, we start the R+ tree insert operation from the root node. If N is a non-leaf node, we insert the point object P directly according to the principle of "minimum enclosing MBB". We find the node with the smallest expansion after the MBB adds a new object. If the expansion is the same, we check smallest index of the original MBB and continue to traverse each node. If the node is a leaf node, then return it, otherwise loop recursively until the leaf node meets the condition. If N is a leaf node, we judge whether the leaf node is full and whether there is overflow after insertion. If leaf node is not full, we insert the index information of the object P directly, and adjust the R+ tree from the leaf node to the parent node. If leaf node is full, object P is added to the leaf node and split operation is performed.

The essence of node splitting is the insertion operation. After the new node is added, the parent node does not have its index information yet, so an index item needs to be added to the parent node, it point to the newly added leaf node. However, if the parent node overflows after adding the index item, the parent node also needs to perform the same split operation until the conditions are met and no overflow occurs.

Algorithm 2: Insert Algorithm

Input: Point P,GridsNode N
Output: Node CP

```
indexID ← GetMultGrid(P);
Root ← N.getNodeID(indexID);
if Root.level = 0 then
    if N is overfill then
        CP ← SplitNode(N);
    end
    else
        Node(N) ← P;
        CP ← N;
    end
end
else
    CP ← getMinSize(Root.MBR);
    insert(N.CP, P);
end
return CP;
```

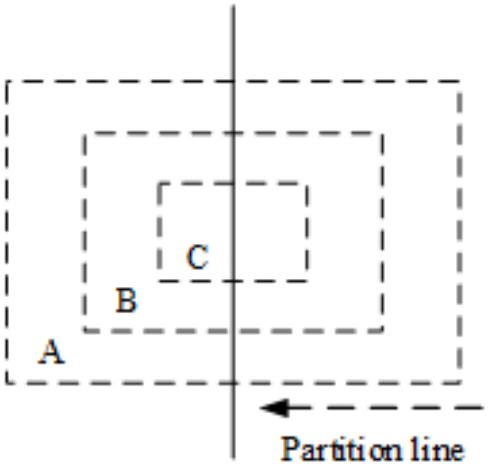


FIGURE 6. Node split.

As shown in Figure 6, the split of the R+ tree is different from the split of the R tree slightly. The split of the R tree will propagate upwards only, but the R+ tree may propagate downwards. The R+ tree reduces overlap by splitting intermediate nodes. When the added non-leaf node covers or overlaps multiple areas, we re-divide it to ensure that its child nodes only belong to a certain intermediate node, and cannot span multiple partitions region, which ensures the uniqueness of the query path. If A is the parent node of B, and B is the parent node of C, using a straight line to divide A, then B and C also need to be split. Of course, the leaf nodes cannot be split, because they are not affected by the effect of recursive splitting, and will not cause the splitting of the underlying nodes.

B. DELETE OPERATION

The delete algorithm of the three-dimensional multi-level adaptive grid and R+ tree hybrid index structure is shown in Algorithm 3.

The specific description of the algorithm is as follows. According to the automatic division algorithm based on normal distribution, the grid can be divided quickly. The grid where the deleted 3D point element object can be located by formula 1, and the grid index ID can be obtained.

The delete operation of the three-dimensional multi-level adaptive grid and R+ tree removes the sub-R+ trees of the independent and non-overlapping multi-level grids. The detailed process is as follows. The delete operation in the R+ tree is traversed from the root node. We start to find the specific location of the leaf node where the object P is stored. Only when the location is found the corresponding data item can be deleted. We need to set node N as the root node, judge whether N is a leaf node, if N is a leaf node,

Algorithm 3: Delete Algorithm

Input: Point P,GridsNode N
Output: Boolean $Result$

```
indexID ← GetMultGrid(P);
Root ← N.getNodeID(indexID);
if Root.level = 0 then
    if N contains P then
        delete(P);
        N.Count = N.Count − 1;
        Result ← true;
    end
    else
        Result ← false;
    end
end
else
    for each Node N in Root[indexID] do
        if N.MBB intersect with P then
            Delete(P, N) ;
        end
    end
end
Adjust(N.MBB);
return Result;
```

we check whether N contains object P, if it does, delete the information of object P directly. Then we adjust the smallest outer rectangle of the index item corresponding to its parent node up to the root node. If the object is deleted, there are no other data items in the leaf node, we need to release the storage space where the leaf node was located. Then, the index corresponding to the leaf node in its parent node just delete the item. If the parent node has no other data items during the process of deleting, the same operation need to be performed.

V. QUERY BASED ON HYBRID INDEX STRUCTURE

A. ACCURATE POINT QUERY

The algorithm of accurate point query based on the 3D multi-level adaptive grid and R+ tree hybrid index structure is shown in Algorithm 4.

Algorithm 4: Accurate Query Algorithm

```
Input: Point P,GridsNode N
Output: Point PosNode
indexID ← GetMultGrid(P);
Root ← N.getNodeID(indexID);
for each Node n in Root[indexID] do
    if n is leaf p is in n then
        PosNode.index = index;
        PosNode.path = each node id in Root;
    end
end
return PosNode;
```

The algorithm is described specifically as follows. According to the automatic division algorithm based on normal distribution, the grid can be divided, we could locate 3D point element object in the grid by formula 1, and the grid index number can be obtained. It traverses from the root node to the leaf node, if the leaf node contains the data information of the query point, the result will be returned. The query result set represents the path taken to query the precise point. 3D multi-level adaptive grid and R+ tree root node → 3D multi-level adaptive grid and R+ tree intermediate node → 3D multi-level adaptive grid and R+ tree leaf node, the end sign is the final query of the leaf node.

B. K-NEAREST NEIGHBOR QUERY

K-nearest neighbor query [41]–[44] based on the 3D multi-level adaptive grid and R+ tree can make use of the advantages of the index structure, combined with the idea of space division, and query the result set according to certain rules. Operations of the intermediate nodes of the lattice-R+ tree is pruned to obtain a preliminary candidate set, and then we perform a refinement operation to obtain the final result set. Query performance can be improved by this way.

In the primary structure of the hybrid index, the specific location of the grid center point in the spatial area is represented by the multi-level adaptive grid center node, so that it can locate and determine the distance from the query point. The calculation method for each central node of the 3D multi-level adaptive grid and R+ tree primary structure is shown in formula 5.

$$p_{center} = \frac{p_{li} + p_{ri}}{2}, \quad i = x, y, z \tag{5}$$

$p_{li}(x, y, z)$ represents the value of the coordinate point of the left front boundary of each intermediate node of the multi-level adaptive grid and R+ tree, and $p_{ri}(x, y, z)$ represents the value of the coordinate point of the upper right boundary.

The k-nearest neighbor query algorithm based on multi-level adaptive grid and R+ tree is described in detail.

Algorithm 5: K-Nearest Neighbor Query Algorithm

```
Input: Integer k, GridsNode ListNode
Output: List<Node> KList
KList ← KList(k);
BuckArray ← BuildMulGrid(ListNode);
for each barray in BuckArray do
    if q is in barray then
        i = index/(n × t);
        j = (index−i × n × t)/t ;
        k = (index−i × n × t)%t ;
        index ← Buck(T);
        if Buck(T).count > k then
            KList = Knearest(q, Buck(T));
        end
        else
            i − 1 ← m1 → i + 1;
            j − 1 ← n1 → j + 1;
            k − 1 ← t1 → k + 1;
            for each g in 27 do
                T1 = m1 × (m × n) + n1 × m + t1;
                if Buck(T).count + Buck(T1).count > k
                 then
                    KList = Knearest(q, Buck(T1));
                end
            end
        end
    end
end
return KList;
```

The algorithm is described as follows in Algorithm 5. Firstly, a linked list KList is created to store the returned result set of K objects. According to the automatic division algorithm based on normal distribution, the grid structure can be divided for pruning query, and the bucket structure can be obtained. We use formula 1 to locate the grid where the query object can be found, and the number of the bucket obtained. Then we judge whether the number of objects in the bucket is larger than the K value. If it is larger than the K value, the k-nearest algorithm is called to perform k nearest neighbor query and return the result set. If it is less than K, pruning query is required, that is, the sub-grid extends one unit up and down on the X, Y, and Z axes to expand the range of the candidate set. According to the traversal order of the $(Z \rightarrow Y \rightarrow X)$ axis, the coding order of the sub-grid is increased, and the coordinates of the grid where the object is located are increased sequentially, and then we calculate the number of the bucket and the number of objects in each bucket. If the number of objects in the nearest intermediate nodes is larger than the value of K, we add them to the

candidate set. Then the *Knearest* algorithm is called to obtain the result set of the k-nearest neighbor query. Otherwise, the process continues until the bucket structure is traversed, then the *Klist* result set is returned, and the algorithm ends.

VI. EXPERIMENTAL RESULTS AND ANALYSIS

The operating system of this experiment is based on the 64-bit Microsoft Windows10 system. The software used in the experimental development environment is Visual Studio 2015, and the development language uses the C# programming language. The hardware environment is set to: Intel(R) Core(TM) i5-3230M CPU @2.60 GHz, quad-core, 8GRAM.

In order to verify the good query performance of the three-dimensional multi-level adaptive grid and R + tree hybrid index structure in different scenarios, this experiment uses the software *SpatialDataGenerator* to generate three-dimensional spatial objects randomly. The data set coordinates are random numbers between 0 and 400 that follow a normal distribution. The data set uses 50,000, 100,000, and 200,000 data sets in size. The k values of k-nearest neighbor query and Anti-k nearest neighbor query are 100, 200, 300, and 400, and the data sets are tested for different scenarios. The specific data set format is shown in Table 1.

TABLE 1. Data set.

format of data set
1:240 245 115 243 248 117
2:236 186 216 239 189 218
3:123 128 170 126 131 172
.....

Each piece of the data represents the position and size of a three-dimensional object. We use the form of ID and coordinate set to represent the MBB model of the object, such as (ID: Xmin Ymin Zmin Xmax Ymax Zmax). Take the first piece of data in the data set for an example. ''1'' is the ID of the first piece of the data. Each piece of data has its ID followed by '':'' and then 6 numbers. The first three represent the lower front left coordinate value of the MBB object and the last three numbers represent the upper right rear coordinate value of the MBB object. Each number is separated by a space.

It often takes a lot of time to construct the index structure. The larger the data set, the longer it takes. This experiment compares the construction time with other index structures based on different data sets. As shown in Figure 7, the traditional R tree index structure, the grid-R tree index structure and the three-dimensional multi-level adaptive grid and R+ tree index structure are constructed based on 50000, 100000 and 200000 data sets. Under the same data set, the indexing construct time of the three-dimensional multi-level adaptive grid and R+ tree is shorter than that the traditional R tree or the grid R tree. When the data set is small, such as the size of the 50,000 data set, the time for the three-dimensional multi-level adaptive grid and R+ tree and

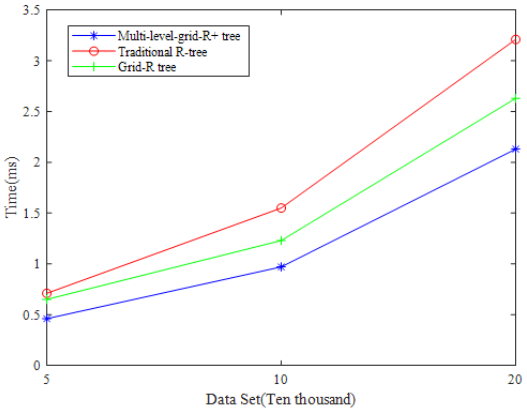


FIGURE 7. Index structure construction comparison chart.

the traditional R tree, the grid R tree to build the index structure is very close, and the time is very short. However, with the increase of the data set, such as the size of 200,000 data set, the time required to build the index structure of the three-dimensional multi-level adaptive grid and R+ tree is much shorter than the traditional R tree. As the amount of data increases, it can be seen that the indexing of the three-dimensional multi-level adaptive grid and R+ tree is more advantageous than the others.

The construction of the index is to insert each spatial data object into the index structure. When the amount of data is very large and distributed unevenly, the three-dimensional multi-level adaptive grid and R+ tree index structure are more advantageous. This is because through the division of the three-dimensional multi-level adaptive grid and R+ tree one-level grid structure, the original space has been divided, which makes the data objects that appear distributed densely in the original space range to be placed in independent Non-overlapping accordingly and interdependent three-dimensional multi-level adaptive grid and R+ tree secondary structure. On this basis, a three-dimensional R+ tree index structure is established for each small spatial data area. Compared with the traditional R tree index structure, although some storage space will be wasted, because of the characteristics of the grid structure, it can locate the node location quickly where the search object is located, reducing the coverage rate of the middle node MBB of the R+ tree, without traversing every intermediate node, reducing I/O overhead and improving indexing efficiency. The number of nodes in the sub-grid is controlled within a certain threshold set by user, and the R+ tree index structure is constructed concurrently through the sub-threads, which improves the efficiency and reduces the construction time.

For the k-nearest neighbor query under the three-dimensional multi-level adaptive grid and R+ tree index structure, the three-dimensional multi-level adaptive grid and R+ tree, the grid R tree and the traditional R tree index structure are compared under the same data set and different k values. Table 2 shows a comparison of the query time of three index structures k-nearest neighbors under the default parameter of 50,000 normal distribution data sets.

TABLE 2. k-nearest neighbor query time (different k values).

Index structure	k=100	k=200	k=300	k=400
Traditional R tree	86	90	102	113
Grid-R tree	37	69	89	106
Multi-level grid and R+ tree	33	60	83	95

When the amount of data is very large and distributed unevenly, the three-dimensional multi-level adaptive grid and R+ tree index structure have more advantageous. This is because through the division of the three-dimensional multi-level adaptive grid and R+ tree one-level grid structure, the original space has been divided quickly, which makes the data objects that appear distributed densely in the original space range to be placed in independent non-overlapping accordingly and interdependent three-dimensional multi-level adaptive grid and R+ tree secondary structure. On this basis, a three-dimensional R+ tree index structure is established for each small spatial data area. Compared with the traditional R tree index structure, although some storage space will be wasted, because of the characteristics of the grid structure, it can locate the node location quickly where the search object is located, reducing the coverage rate of the middle node MBB of the R+ tree, without traversing every intermediate node, reducing I/O overhead and improving indexing efficiency. The number of nodes in the sub-grid is controlled within a certain threshold, and the R+ tree index structure is constructed concurrently through the sub-threads, which improves the efficiency and reduces the construction time.

For the k-nearest neighbor query under the three-dimensional multi-level adaptive grid and R+ tree index structure, firstly, the three-dimensional multi-level adaptive grid and R+ tree, the grid R tree and the traditional R tree index structure are compared respectively under the same data set and different k values.

As shown in Figure 8, as the value of k continues to increase, and the time required for k-nearest neighbor query under the three index structures increases. This is because the larger the value of k, the more the number of objects to be searched, and the more the distance between the query point and the object has to be calculated, so the time will increase. But it is not difficult to see that under the same k value, the k-nearest neighbor query under the three-dimensional multi-level adaptive grid and R+ tree index structure is shorter than the others, so the k-nearest neighbor query under the hybrid index in this article still has good query performance.

For the same k value and different data sets, we will compare and analyze the k-nearest neighbor query under the three-dimensional multi-level adaptive grid and R+ tree, the grid R tree and the traditional R tree index structure. As shown in Table 3, the default value of k is 100. As shown in Figure 9, when the value of k is the same, under each different test data set, the time required for the three-dimensional multi-level adaptive grid and R+ tree index in the k-nearest

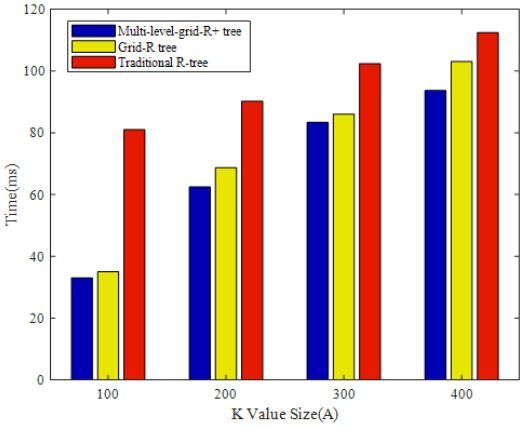


FIGURE 8. Under the size of 50,000 data set.

TABLE 3. k-nearest neighbor query time (k = 100).

Index structure	5	10	20
Traditional R tree	86	135	271
Grid-R tree	37	50	126
Multi-level grid and R+ tree	33	46	106

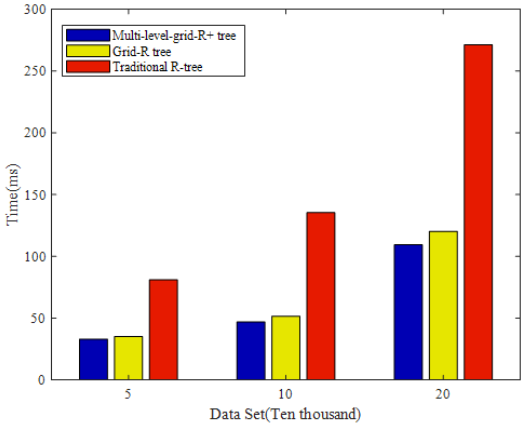


FIGURE 9. Under k = 100 different data set sizes.

neighbor query is shorter than the others. With the increase of the data set, the time required for the k-nearest neighbor query under the three-dimensional multi-level adaptive grid and R+ tree have less change. Especially when the data set is large, compared with the k-nearest neighbor query under the traditional R tree index, the time required of three-dimensional multi-level adaptive grid and R+ tree is less smaller. Therefore, when faced with a large amount of unevenly distributed data, the query performance based on the three-dimensional multi-level adaptive grid and R+ tree index structure is excellent. The size of the data set has a great impact on the query time of the R tree index structure. The larger the data set, the longer the query time. The hybrid index based on 3D multi-level adaptive grid and R+ tree has less impact, because after the threshold is set by user, the time required for the R+ tree query is ascertained. However, the grid may have a secondary structure and a tertiary

structure, which use linked list and the query will take some time. Therefore, as the data set increases, the query time of this structure will increase accordingly.

VII. CONCLUSION

Aiming at the uneven distribution of massive data, this paper proposes a hybrid index structure based on a three-dimensional multi-level adaptive grid and R+ tree. This kind of index structure has better performance in terms of fast building index and data query. Firstly, a multi-level grid automatic division algorithm based on normal distribution is used to process the data set to construct a multi-level adaptive grid. The multi-level adaptive grid structure can partition the data space quickly and effectively, thus solving the problem of irregular division caused by the single-level grid. Secondly, our method constructs the R+ tree index structure according to the threshold conditions, and uses the advantage of zero overlap of the R+ tree's intermediate node directory rectangles to perform efficient indexing. In this way, the construction of the overall mixed index is completed. Finally, this paper gives maintenance algorithm of inserts and deletes the hybrid index to ensure the safety and reliability of the index structure. The researches on precise point query and k-nearest neighbor query algorithm based on three-dimensional multi-level adaptive grid and R+ tree hybrid index structure are given respectively. The experimental results prove that the three-dimensional multi-level adaptive grid and R+ tree hybrid index structure has good query performance when the amount of data is large and the distribution is uneven.

The hybrid index proposed in this paper is just suitable for static spatial databases, but for moving objects or multi-source objects, this will make the database index update very frequent, resulting in lower database operating efficiency. Therefore, it is of great significance to develop an index structure based on dynamic environment or multi-source situation. In addition, the reverse K-nearest neighbor query could be performed based on the hybrid index structure. And we could add obstacles to the space and define the obstacle distance of the model to study the visual query of the three-dimensional obstacle space.

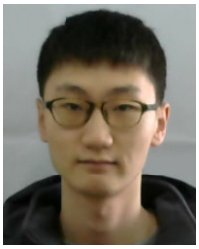
REFERENCES

[1] Y. Shen, J. Xu, and D. Liu, "Research on spatial data index of 3D digital map in embedded system," *Comput. Appl. Softw.*, vol. 29, no. 7, 2012.
[2] W. Wang *et al.*, "Parallel algorithm for arc intersection based on grid index and R tree," *Sci. Surv. Mapping*, vol. 39, no. 3, 2014.
[3] Z. Li, X. Li, S. Yang, and T. Liu, "Spatial index built in cloud computing environment," *Geomatic Spatial Inf. Technol.*, vol. 38, no. 10, pp. 13–17, 2015.
[4] M. Zhang, F. Lu, P. Shen, and C. Cheng, "The evolvement and progress of R-tree family," *Chin. J. Comput.*, vol. 28, no. 3, 2005.
[5] Y. Wang and M. Guo, "A combined 2D and 3D spatial indexing of very large point-cloud data," *Acta Geodaetica et Cartograph. Sinica*, vol. 41, no. 4, pp. 605–612, 2012.
[6] S. Xu, M. Wang, and W. Wang, "A new index structure for moving object spatial database based on R tree and Quan tree," *Comput. Digit. Eng.*, vol. 34, no. 3, 2006.
[7] Z. Jiang, "A survey on spatial prediction methods," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 9, pp. 1645–1664, Sep. 2019.
[8] H. Liao, J. Han, and J. Fang, "All-nearest-neighbor queries processing in spatial databases," *J. Comput. Res. Develop.*, vol. 48, no. 1, p. 86, 2011.
[9] N. Li, X. Wu, J. Ma, and Z. Wang, "A line clip algorithm of based on cell and R-tree spatial indexes against arbitrary polygon window," *Comput. Eng. Sci.*, vol. 34, no. 11, 2012.
[10] H. Liang and M. Wu, "Performance analysis and evaluation for two typical methods of spatial index in GIS," *J. Anyang Inst. Technol.*, vol. 20, no. 2, 2006.
[11] M. Wu, Y. Guo, and T. Chen, "An effective hybrid spatial indexing mechanism," *Comput. Eng. Appl.*, vol. 42, no. 29, 2006.
[12] N. Chen, X. Zhong, and L. Li, "Research on optimized R-tree high-dimensional indexing method based on video features," in *Proc. Int. Conf. Cloud Comput. Big Data Anal. (ICCCBDA)*, Apr. 2017, pp. 123–146.
[13] S. Pramanik, A. Watve, C. R. Meiners, and A. Liu, "Transforming range queries to equivalent box queries to optimize page access," *Proc. VLDB Endowment*, vol. 3, nos. 1–2, pp. 409–416, Sep. 2010.
[14] D. Han, "Research on nearest neighbor query based on R-tree," Harbin Univ. Sci. Technol., Harbin, China, Tech. Rep., 2011.
[15] T. Wu *et al.*, "Research on routing technology based on quadtree," Univ. Sci. Technol. China, Hefei, China, Tech. Rep., 2015.
[16] A.-V. Vo, L. Truong-Hong, D. F. Laefer, and M. Bertolotto, "Octree-based region growing for point cloud segmentation," *ISPRS J. Photogramm. Remote Sens.*, vol. 104, pp. 88–100, Jun. 2015.
[17] K. Zheng *et al.*, "Research on the spatial index structure of LOD_OR tree in 3D DIS," *Bull. Surv. Mapping*, vol. 39, no. 5, 2005.
[18] D. Z. Chen and H. Wang, "Weak visibility queries of line segments in simple polygons," *Comput. Geometry*, vol. 48, no. 6, pp. 443–452, Aug. 2015.
[19] N. Sultana, T. Hashem, and L. Kulik, "Group nearest neighbor queries in the presence of obstacles," in *Proc. 22nd ACM SIGSPATIAL Int. Conf. Adv. Geograph. Inf. Syst.*, Nov. 2014, pp. 481–484.
[20] A. Arman, M. E. Ali, F. M. Choudhury, and K. Abdullah, "VizQ: A system for scalable processing of visibility queries in 3D spatial databases," in *Proc. ACM Conf. Inf. Knowl. Manage.*, Nov. 2017, pp. 2447–2450.
[21] A. Guttman, "R-Trees: A dynamic index structure for spatial searching," in *Proc. ACM SIGMOD*, 1984, pp. 47–57.
[22] T. Sellis, N. Roussopoulos, and C. Faloutsos, "The R+-tree: A dynamic index for multi-dimensional objects," *Comput. Sci. Dept.*, vol. 9, pp. 507–518, 1987.
[23] Guttman, "Research history and latest progress analysis of R-tree spatial indexing algorithm," Mod. Comput., Tech. Rep., 1990.
[24] J. Yang and X. Huang, "A hybrid spatial index for massive point cloud data management and visualization," *Trans. GIS*, vol. 18, pp. 97–108, Nov. 2014.
[25] J. Nievergelt, H. Hinterberger, and K. C. Sevcik, "The grid file: An adaptable, symmetric multikey file structure," *ACM Trans. Database Syst.*, vol. 9, no. 1, pp. 38–71, Mar. 1984.
[26] Z. Huang *et al.*, "Research on hybrid index based on multi-level grid and STR tree," Zhejiang Univ., Hangzhou, China, Tech. Rep., 2013.
[27] X. Tang, B. Han, and H. Chen, "A hybrid index for multi-dimensional query in HBase," in *Proc. Int. Conf. Cloud Comput. Intell. Syst.*, Aug. 2016, pp. 332–336.
[28] H. Xu and Z. Hao, "Nearest neighbor query algorithm based on space-filling curve meshing," *Comput. Sci.*, vol. 37, no. 1, 2010.
[29] S. Niu *et al.*, "A ciphertext sorting search scheme based on the B+ tree index structure on the blockchain," *J. Electron. Inf.*, vol. 41, no. 10, 2019.
[30] X. Zhang, "Design and analysis of electric power big data hybrid index based on B+ tree," *Electron. Des. Eng.*, vol. 28, no. 22, 2020.
[31] J. Gong, "A sub-three-dimensional R-tree index expansion method that takes into account multiple levels of detail," *J. Surv. Mapping*, vol. 40, no. 2, 2011.
[32] J. Gong, "Adaptive method of 3D city model based on the extended structure of 3D R-tree index," *J. Surv. Mapping*, vol. 40, no. 4, 2011.
[33] Y. Wang and M. Guo, "A two-dimensional and three-dimensional hybrid indexing method for large-scale point cloud data," *J. Surv. Mapping*, vol. 41, no. 4, 2012.
[34] M. Sharifzadeh and C. Shahabi, "VoR-Tree: R-trees with Voronoi diagrams for efficient processing of spatial nearest neighbor queries," *Proc. VLDB Endowment*, 2013.
[35] J. Gong *et al.*, "A laser point cloud data management method integrating octree and three-dimensional R-tree," *J. Surv. Mapping*, vol. 41, no. 4, 2012.
[36] X. Song *et al.*, "Research on the spatial index structure of hybrid tree in 3D GIS," *J. Shenyang Jianzhu Univ.*, vol. 22, no. 3, 2006.

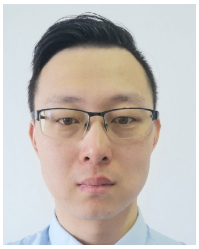
[37] X. Gong *et al.*, "Research on hybrid index method based on three-dimensional grid-R tree," *J. Yanshan Univ.*, vol. 44, no. 2, 2020.
[38] G. Wu *et al.*, "A multi-level grid division method for urban distribution network planning," Patent CN 108 876 204 A, 2018.
[39] L. Cai and H. Jiang, "Geographical national conditions information representation based on multi-level grid," *Sci. Surv. Mapping*, vol. 41, no. 3, 2016.
[40] J. Hu *et al.*, "Optimal partitioning of spatial database grid indexing mechanism," *Chin. J. Comput.*, vol. 25, no. 11, 2002.
[41] W. Wang, W. Wang, and J. Wang, "Algorithm for finding the smallest circle containing all points of a point set," *J. Softw.*, vol. 28, no. 9, 2000.
[42] L. Liu, L. Zhang, and J. Yu, "Line segment reverse k nearest neighbor query based on Voronoi diagram in spatial database," *J. Chin. Comput. Syst.*, vol. 38, no. 4, 2017.
[43] L. Zhu, W. Sun, and Y. Jing, "Voronoi graph-based k-nearest nearest neighbor query method for road network," *Chin. J. Comput. Res. Develop.*, vol. 48, no. 3, 2011.
[44] P. Pei, D. Zhang, and F. Guo, "A density-based clustering algorithm using adaptive parameter K-reverse nearest neighbor," in *Proc. IEEE Int. Conf. Power, Intell. Comput. Syst. (ICPICS)*, Jul. 2019, pp. 455–458.

YONGSHAN LIU was born in Zhangjiakou, Hebei, China, in 1963. He received the M.S. degree in computer science and technology from Yanshan University, in 1989, and the Ph.D. degree in computer and applications from Harbin University of Science and Technology, in 2006. Since 1994, he has been a Professor with the Department of Information Science and Engineering, Yanshan University. He currently serves as a tutor for Ph.D. students.

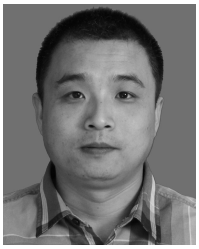
TIANBAO HAO was born in Handan, Hebei, China, in 1985. He received the M.S. degree in computer software and theory from Yanshan University, in 2011, where he is currently pursuing the Ph.D. degree in computer science and technology. His research interest includes spatial database.



XIANG GONG was born in Taiyuan, Shanxi, China, in 1991. He received the M.S. degree in software engineering from Yanshan University, in 2017, where he is currently pursuing the Ph.D. degree in computer science and technology. His research interests include spatial database, information visualization, and visual analysis of obstacle environment.



DEHAN KONG was born in Dalian, Liaoning, China, in 1986. He received the Ph.D. degree in computer science and technology from Yanshan University, in 2017. He is currently teaching with the Department of Information Engineering, Hebei University of Environmental Engineering. His research interests include spatial database and point cloud reconstruction and registration.



JIANJUN WANG is currently pursuing the Ph.D. degree with Yanshan University, Qinhuangdao, China. Prior to that, he worked with IT industry for more than ten years as an enterprise architect and the program manager. His research interests include machine learning, deep learning, and complex networks.