*remote sensing*

MDPI

*Article*

# Large Quantities of Acoustic Multibeam Bathymetric Point Clouds: Organizing Method for Efficient Storage and Retrieval

**Xianhai Bu [1,2], Shuaibing Dou [1], Jianxing Zhang [3], Tianyu Yun [1,*], Yabing Zhu [4], Yi Huang [5] and Xiaodong Cui [1,2]**

[1] College of Geodesy and Geomatics, Shandong University of Science and Technology, Qingdao 266590, China; buxianhai0826@sdust.edu.cn (X.B.); 202283020113@sdust.edu.cn (S.D.); cuixiaodong@sdust.edu.cn (X.C.)
[2] Key Laboratory of Ocean Geomatics, Ministry of Natural Resources, Qingdao 266590, China
[3] Key Laboratory of Marine Geology & Environment, Institute of Oceanology, Chinese Academy of Sciences, Qingdao 266071, China; zhangjx@qdio.ac.cn
[4] Guangdong Provincial Department of Land, Resources Surveying and Mapping Institute, Guangzhou 510700, China; 202383020033@sdust.edu.cn
[5] Unit 91001, Beijing 100143, China; 202383020102@sdust.edu.cn
* Correspondence: cloucky@sdust.edu.cn

**Abstract:** To efficiently organize large quantities of acoustic multibeam bathymetric point clouds, this paper proposes an improved oriented quadtree-based method for establishing a data indexing structure stored on a hard disk. First, the spatial characteristics of the multibeam swath data are integrated into the traditional quadtree structure, resulting in an oriented quadtree for data organization. Then, the primary orientation of the root node's bounding box, which reflects the main orientation of the swath, is consistently applied to all child nodes, eliminating the need to calculate the orientation for each individual child node by the conventional oriented quadtree. Finally, index files containing the point cloud offset, oriented bounding box, and child node information for root, child, and leaf nodes are designed and stored in external storage. Experimental results indicate that, in terms of tree construction time, although the traditional quadtree reduces time consumption by approximately 50% compared to the improved oriented quadtree, the improved oriented quadtree still achieves a 70% reduction in time consumption compared to the conventional oriented quadtree. Regarding point cloud retrieval, within the same retrieval range, the improved oriented quadtree achieves similar average retrieval times as the conventional oriented quadtree but reduces the maximum time consumption by approximately 20.83% compared to the traditional quadtree. Furthermore, by storing the constructed index in binary format on external storage, the space occupancy was reduced by 50%. The approach effectively organizes acoustic multibeam bathymetric point clouds, providing valuable insights for enhancing point cloud retrieval efficiency and reducing data memory usage.

**Keywords:** multibeam sonar; large quantities of point clouds; oriented quadtree; index files; data retrieval



Academic Editors: Jorge Vazquez and Antonino Maltese

Received: 18 December 2024
Revised: 23 March 2025
Accepted: 29 May 2025
Published: 13 June 2025

**Citation:** Bu, X.; Dou, S.; Zhang, J.; Yun, T.; Zhu, Y.; Huang, Y.; Cui, X. Large Quantities of Acoustic Multibeam Bathymetric Point Clouds: Organizing Method for Efficient Storage and Retrieval. *Remote Sens.* **2025**, *17*, 2039. https://doi.org/10.3390/rs17122039



## 1. Introduction

Serving as one of the most commonly used acoustic remote sensing techniques, the multibeam echo-sounder systems (MBESs) are widely used for underwater topography and geomorphology surveys and sediment classification, with the characteristics of high efficiency and wide coverage [1–3]. In the shallow water environment, the maximum ping rate of the MBES can reach 70 Hz (e.g., Teledyne Reson T50 multibeam sonar), with each ping generating between 256 and 1024 sounding points. Consequently, the MBES can produce a high-coverage swath consisting of large quantities of soundings along

the trackline [4,5], which presents a challenge for post-data processing [6]. Due to the limitations of computer memory, relying solely on in-memory for data storage is impractical. Furthermore, it is essential to construct a suitable data structure to organize the data for efficient retrieval.

To achieve the efficient processing of point clouds, a commonly used method is to pre-build indexes on the point cloud data based on computer memory for better organization. This approach offers the advantages of fast speed, low latency, convenient data operations, and reducing disk I/O bottlenecks [7]. However, when handling massive point clouds, even if the data can be stored in memory, excessive memory usage can easily lead to computer or software freezes, necessitating external hard-disk storage [8,9]. Relying solely on external storage for data organization and I/O operations may result in slower performance and increased time latency, which hinders efficient processing. Therefore, it is essential to develop an effective out-of-core mechanism that optimizes the use of limited memory resources in conjunction with external storage to facilitate the rapid operation of large-scale datasets [10]. This hybrid data organization strategy leverages the speed of memory while utilizing the capacity of external storage, enabling the efficient handling and processing of massive point clouds [11]. To maximize effectiveness, the data scheduling system between memory and external storage must be meticulously designed and optimized [12,13].

The effective organization and scheduling of large-scale point cloud data rely on optimized strategies, including spatial partitioning, hierarchical storage management, multi-resolution representation, and distributed computing frameworks. These methodologies substantially improve processing efficiency and enable practical applications in massive multibeam point cloud analysis [14,15]. Current mainstream point cloud management systems adopt distinct technical approaches: Open Scene Graph (OSG) demonstrates superior performance in local real-time rendering through its advanced level-of-detail (LOD) management and view frustum culling algorithms [16], while Potree specializes in the web-based visualization of billion-scale point clouds using octree spatial indexing and adaptive chunk-loading mechanisms [17]. Notably, the Point Cloud Library (PCL) provides comprehensive data organization capabilities through integrated KD-tree and octree structures, establishing itself as an essential toolkit for large-scale point cloud processing.

The key to the efficient management of spatial data lies in the creation of a well-structured data index [18]. Spatial indexes utilizing data structures such as the KD tree [19], R-tree [20], quadtree [21], and octree [22] are commonly employed to organize spatial point clouds. While the K-D tree enhances retrieval efficiency, it demands substantial memory for extensive index lookups [23]. R-trees provide flexibility and adjustability; however, overlaps in intermediate nodes may reduce lookup efficiency [24]. Quadtrees are particularly suited for the spatial indexing of two-dimensional data [25], while octrees are more appropriate for three-dimensional data [26]. To facilitate operations such as loading, displaying, and rendering large point cloud datasets, contemporary methods predominantly utilize the octree structure for data organization, combined with out-of-core and level-of-detail (LOD) techniques, allowing for efficient point cloud operations with minimal memory usage [27,28]. In contrast to land terrain derived from 3D LiDAR, seabed terrain typically consists of extensive flat areas with relatively subtle elevation changes and few prominent features. Utilizing an octree structure for organizing multibeam sonar point clouds can result in storage redundancy in the Z-direction [29]. Consequently, employing a quadtree structure to index and organize multibeam soundings is more appropriate. Furthermore, based on the principles of multibeam field operation, ensuring the comprehensive coverage of the seabed terrain necessitates that survey vessels follow designed survey lines, resulting in the acquisition of long-swath data. During the actual data acquisition process utilizing multibeam systems, the direction of the swathes is generally parallel to the isobaths. In most

cases, this orientation is not aligned with the axial direction [30,31]. When constructing a conventional quadtree index, the calculation of its bounding box does not consider the principal direction of the point cloud. Therefore, particularly in the case of multibeam swath (the long coverage area formed by quantities of pings when the ship sails along the survey line [32]) point clouds, the use of a conventional quadtree index may lead to an increase in empty nodes and excessive index depth [33].

Thus, to address these issues, this paper presents an improved oriented quadtree-based method for organizing large quantities of multibeam sonar bathymetric soundings. By incorporating the characteristics of multibeam swath-shaped point clouds, we introduce an oriented bounding box into the conventional quadtree structure. Additionally, by leveraging the principle of out-of-core technique, we manage these point clouds with both computer internal and external storage, thereby achieving efficient organization and an index of extensive multibeam point clouds.

## 2. Methods

To manage and query large multibeam bathymetric datasets efficiently, this study integrates quadtree structures with an oriented bounding box, forming an oriented quadtree. The improved oriented quadtree simplifies the tree-building process by utilizing the main direction of the root node. The corresponding indexing mechanism further enables rapid data retrieval within search ranges, i.e., the region-of-interest area. The following sections detail the design and integration of these components for efficient bathymetric data management.

### *2.1. Quadtree Structure*

In 1998, Tayeb proposed quadtree indexing, a tree-like data indexing structure comprising three sorts of elements: root nodes, child nodes, and leaf nodes, with up to four children per node [34]. The root node represents the entire spatial domain of interest. It serves as the entry point of the quadtree hierarchy and stores all data points before any subdivision occurs, as depicted by the topmost node in Figure 1. Child nodes are the direct subdivisions of the root node. Each child represents a quadrant of the parent node's spatial region, as depicted by the second-level node in Figure 1. Leaf nodes in the quadtree are terminal nodes that are not subdivided further, and they do not have child nodes, making them the "endpoints" of tree traversal and representing the lowest level in the hierarchy, as depicted by the lowest-level node in Figure 1.

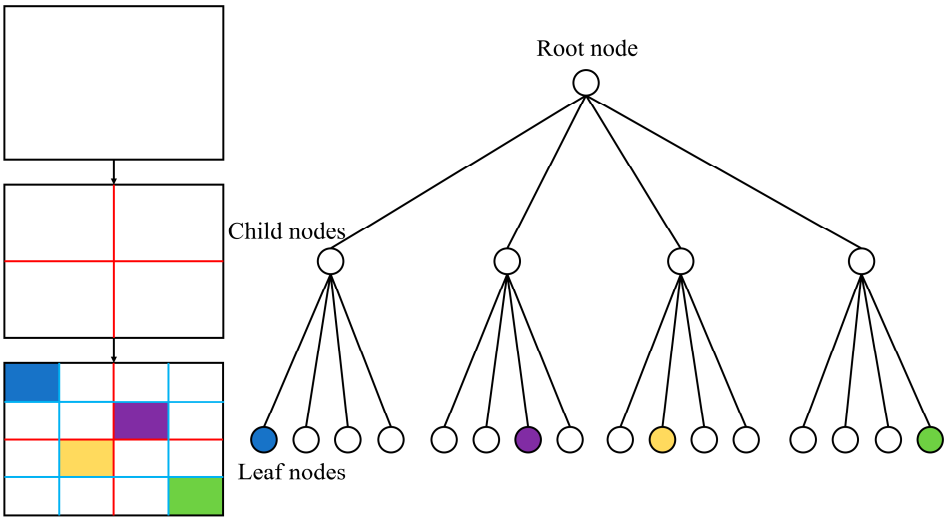



**Figure 1.** Schematic diagram of quadtree structure.

It represents the smallest spatial unit in the hierarchy, containing the final set of data points. The idea behind quadtree indexing is to divide two-dimensional objects in each space into a cyclic recursive framework. The two-dimensional object at the root node is split into four equal child nodes through a recursive procedure in this process. To avoid infinite recursion, a threshold must be set to restrict the number of recursive divisions. The hierarchical division's termination criterion is this threshold. Establishing a maximum division depth or establishing a minimum value for each node in the set are two popular approaches. The quadtree indexing procedure is finished once the threshold is reached [35]. The two-level complete quadtree division and its schematic structure are shown in Figure 1.

### *2.2. Oriented Bounding Box and Oriented Quadtree*

An oriented bounding box (OBB) is the smallest bounding box generated in the direction of the principal components of an object that surrounds it. An axis-aligned bounding box (AABB) is the smallest rectangle (or rectangular prism in three-dimensional space) that is aligned with the coordinate axes and fully encloses all data points, and the edges of the bounding box are always parallel to the X, Y (and Z in 3D) axes. In comparison to the AABB, the OBB can approximate the object with the greatest possible degree of precision in accordance with the shape characteristics of the object. This results in a more compact representation and the elimination of redundant space [36,37]. Figure 2 illustrates a comparison between AABB and OBB using a tiger model. In the first image, where the tiger is aligned with the coordinate axes, the AABB and OBB overlap. However, in the second and third images, where the tiger is not aligned with the axes, the AABB leaves noticeable gaps, while the OBB, oriented along the object's principal direction, closely fits the tiger's shape. This demonstrates that AABBs are better suited for data with weak directionality, whereas OBBs are ideal for data with strong directional characteristics [38].

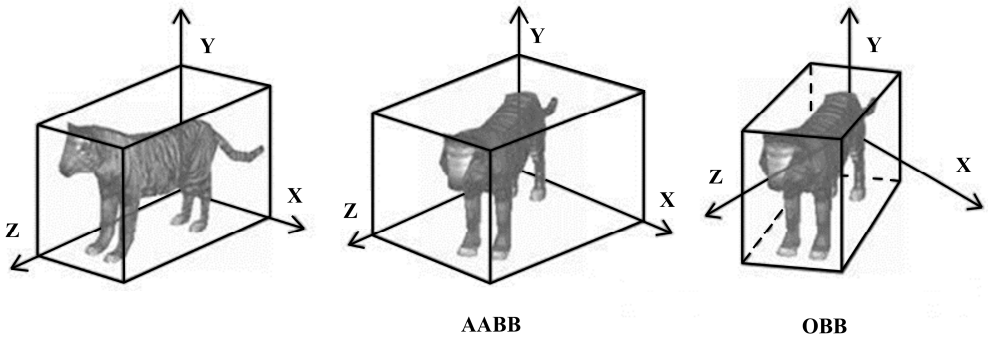



**Figure 2.** Comparison between AABB and OBB [37].

Since the multibeam bathymetric point clouds are uniformly distributed along the trajectory line, with most of them not perpendicular to the axial direction, the oriented bounding box is the best option for the root node of the multibeam bathymetric point clouds for quadtree partitioning. A comparison of the data quadtree and oriented quadtree is presented in Figure 3. The dataset is composed of multibeam-measured data, comprising approximately half a million points. The threshold for quadtree division is set at a maximum of 50,000 for each node, and finally, the comparison of quadtree and oriented quadtree is obtained, and it can be seen from Figure 3a that the node distribution of the quadtree is not uniform, and it is easy to produce empty nodes, and it can be seen from Figure 3b that the oriented quadtree is more able to closely surround the point cloud data, the node distribution is balanced, and the probability of generating empty nodes is greatly reduced. The comparison of Figure 3a,b shows that the oriented quadtree creates a quadtree with a shallower depth and fewer nodes than the standard quadtree. By performing this,

the number of redundant and empty nodes is successfully decreased, improving the query's efficiency.

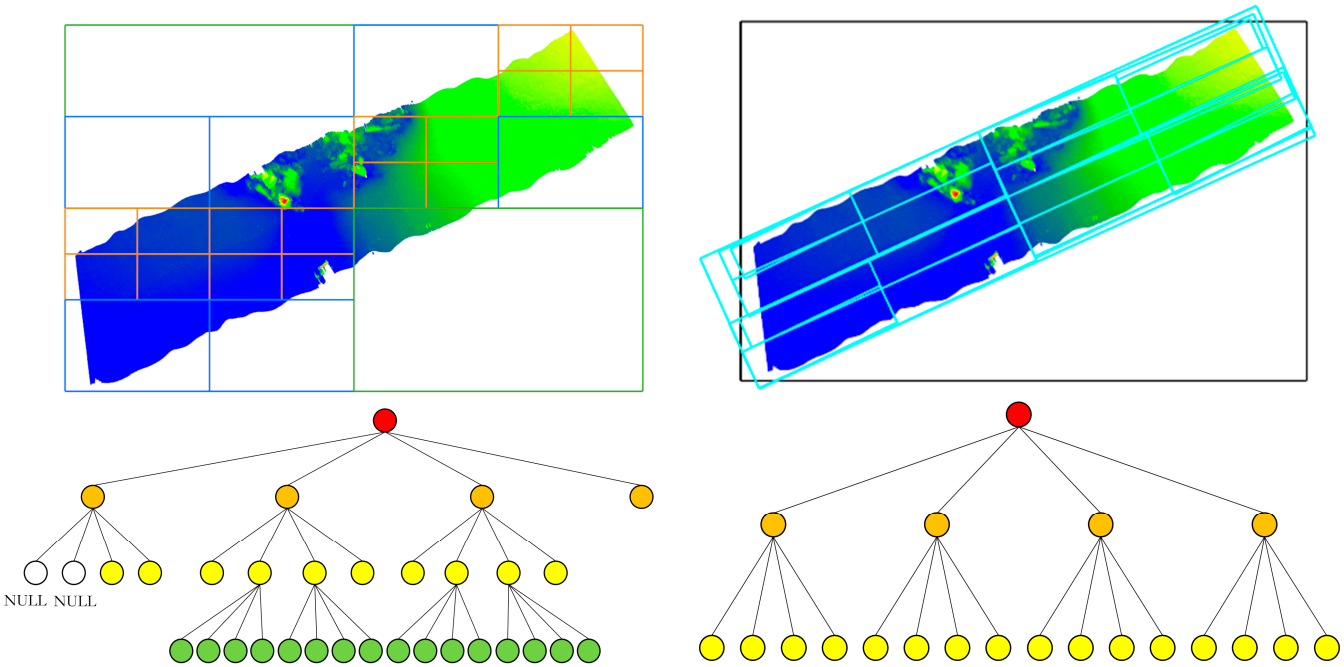



**Figure 3.** Schematic comparison of quadtree and oriented quadtree: (**a**) quadtree index structure; (**b**) Oriented quadtree index structure.

The calculation of the oriented bounding box is mainly to determine the optimal orientation using the first and second order statistical properties of the vertex coordinates and to find the minimum size of the bounding box in this orientation [39]. In this paper, the principal component analysis (PCA) algorithm is applied to the original point cloud to determine the final bounding box based on the analysis results. The effect of the $z$-axis orientation can be ignored since only the corresponding oriented bounding box of the point cloud plane needs to be calculated and divided by the quadtree index. The specific steps are as follows:

1. The principal axis orientation of the point cloud is determined by performing PCA on the raw point cloud data. The main steps of PCA are as follows:
   1.1 Calculate the center of mass $\overline{p}$ of the point cloud as shown in Equation (1).

$$\overline{p} = (\overline{x}, \overline{y}); \overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i; \overline{y} = \frac{1}{y}\sum_{i=1}^{n} y_i \quad (1)$$

   where $\overline{x}$ and $\overline{y}$ refer to the average x-coordinate and y-coordinate of all points, respectively; $n$ is the number of points.
   1.2 Translate the point cloud data $p_i'$ to the center of mass as shown in Equation (2) so that the center of mass is located at the origin of the coordinates.

$$p_i' = p_i - \overline{p} \quad (2)$$

   where $p_i'$ is the centralized point and $p_i$ is the original point.

1.3 Construct the covariance matrix $C$ of the centered point cloud data and calculate the covariance matrix as shown in Equation (3).

$$C = \frac{1}{n-1}\sum_{i=1}^{n} \boldsymbol{p}_i'(\boldsymbol{p}_i')^T \tag{3}$$

where the superscript $T$ represents the matrix transpose.

1.4 The eigenvalue decomposition of the covariance matrix is performed to solve for the eigenvectors as shown in Equation (4).

$$Cv_j = \lambda_j v_i \tag{4}$$

where $\lambda_j$ is the eigenvalue, and $v_j$ is the corresponding eigenvector.

1.5 Arrange the eigenvalues in descending order and obtain the corresponding eigenvectors and the principal axis direction of the point cloud data.

2. Rotate the centralized point cloud $\boldsymbol{p}_i'$ along the coordinate origin by the major axis square degree so that the major direction of the rotated point cloud aligns with the coordinate axis to obtain the rotated point cloud $\boldsymbol{p}_o'$.
3. At this point, the main direction of the rotated point cloud $\boldsymbol{p}_o'$ has been aligned with the coordinate axes, so the AABB (axis-aligned bounding box) is calculated for the rotated point cloud.
4. Rotate the AABB back to the corresponding position of the original point cloud and perform decentralization processing. The resulting oriented bounding box is the OBB (oriented bounding box) of the original point cloud.

### *2.3. Construction and Retrieval of Defined Index Files*

The oriented quadtree is a tree structure model that describes two-dimensional space by integrating the oriented bounding box with the traditional quadtree. It uses the oriented bounding box of the original point cloud as the root node to subdivide the quadtree. Each non-leaf node represents an oriented bounding box corresponding to the current spatial data, aligned with the principal component direction of the original point cloud.

To address the issue of large-scale data memory usage, this paper employs the out-of-core strategy, constructing the oriented quadtree index file in external storage. To minimize file storage space usage and enhance the read/write efficiency of external storage files, a binary method is used to organize the quadtree index file. The index construction, external storage organization, and retrieval process of the oriented quadtree are illustrated in Figure 4. Additionally, this paper proposes an improvement to the oriented quadtree based on the spatial distribution characteristics of the multibeam swath point cloud. Specifically, when subdividing the quadtree of a single swath point cloud, the main direction of the root node is used instead of the direction of each child node. This approach eliminates the need to compute the direction for the point cloud within each child node, thereby significantly improving the efficiency of constructing the oriented quadtree compared to conventional methods.

The following are the precise steps for creating an oriented quadtree external memory index, with the pseudo-code displayed in Algorithm 1:

1. The multibeam bathymetric point cloud data are accessed using memory mapping technology, with the coordinates of the first point serving as the reference offset. Subsequently, the coordinates of all following points are adjusted by subtracting this offset to obtain the relative offset point cloud data. This step aims to reduce the numerical range in subsequent calculations, improving computational efficiency and

accuracy. Next, the segmentation threshold is initialized and set at 2.5% of the total number of points in the original point cloud data for this experiment.

2. Calculate the oriented bounding box of the point cloud and record its principal axis direction. Use this oriented bounding box as the root node of the quadtree, then divide the space into four equal parts based on the oriented bounding box, assigning the corresponding point set in each area to the four child nodes of the current node. Recalculate the bounding box of the child nodes according to the principal axis direction of the original point cloud. If the number of points in a child node exceeds the threshold, continue subdividing the quadtree until all leaf nodes contain fewer points than the threshold.
3. In constructing the oriented quadtree index, the offset and oriented bounding box information of the root node and its child nodes are recorded in the root node file. The child node files contain the oriented bounding box information and its corresponding child nodes. For leaf nodes, the files include the oriented bounding box data, neighboring node information, and point-set details. As shown in Table 1, this oriented quadtree index information is iteratively serialized to the computer's hard disk, creating the external oriented quadtree index file. This process completes the establishment of the oriented quadtree spatial index. This process not only completes the construction of the oriented quadtree spatial index but also ensures efficient storage and fast access to the index data.

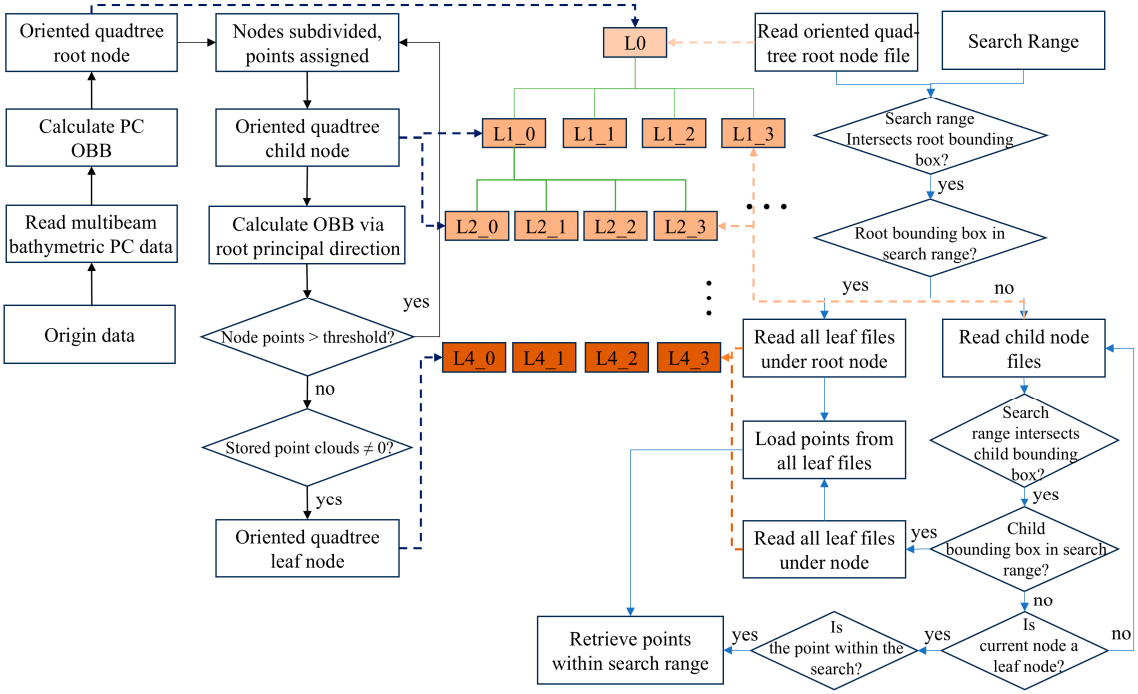



**Figure 4.** Flowchart of index construction and retrieval.

**Table 1.** Oriented quadtree external storage index file information.

| Root Node | Child Node | Leaf Node |
|---|---|---|
| Point cloud offset | Oriented bounding box information | Oriented bounding box information |
| Oriented bounding box information | Subnode file information | Neighbor node file information |
| Subnode file information | | Current node point set information |

**Algorithm 1.** External memory oriented quadtree index construction

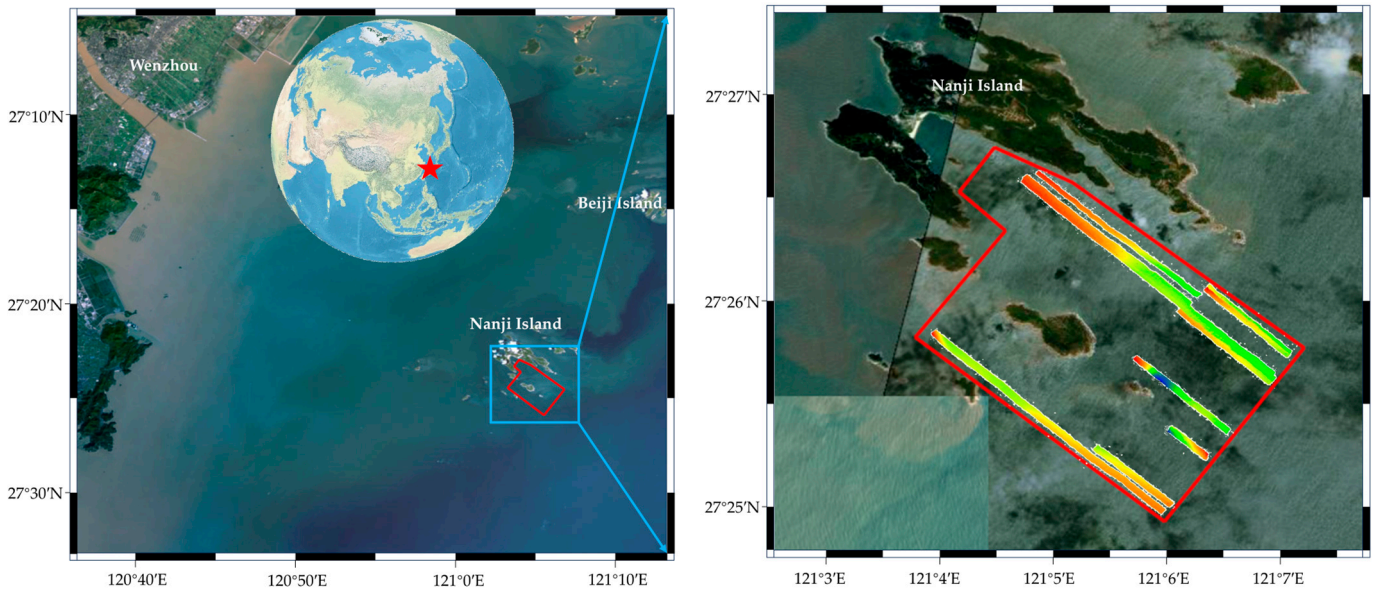
**Input**: Points, oriented bounding box (OBB), maxPointsPerNode and mainDirection
**Function** QuadTreeSegmentation(Points, OBB, maxPointsPerNode, mainDirection):
**if** size(Points) < maxPointsPerNode
Create a leaf node and store Points in it
return leaf node
**end if**
**Initialize:** Sub-obbs list SubOBBs = ComputeSubOBBs(OBB)
**Initialize:** Sub-points list SubPoints = [[], [], [], []]
**for** each point p in Points:
Determine sub-obb p belongs to and add p to the corresponding Sub Points[i]
**end for**
Recalculate SubOBBs using mainDirection, SubPoints
**for** each sub-region i (from 1 to 4):
**if** SubPoints[i] is not empty:
SubNode = QuadTreeSegmentation(SubPoints[i], SubOBBs[i], maxPointsPer Node, mainDirection)
Add SubNode to the children of Node
**end if**
**end for**
return Node
**Function** ComputeSubOBBs(OBB):
Center = Calculate the center point of the OBB
Side-center-point list SideCenters = Calculate the side center points of the OBB
OBBs = Calculate the first sub-boundary using OBB, Center and SideCenters
return OBBs
**End Function**
**# Initialize the quadtree segmentation process**
RootNode = QuadTreeSegmentation(Points, OBB, maxPointsPerNode, mainDirection)

## 3. Results

Experimental data acquired by a Kongsberg EM2040 multibeam echo-sounder (the maximum range is 600 m; range resolution is 1.25 cm; the angular sector used in this survey is approximately ±60°; the number of beams is 512) are utilized in this study. The data were collected near Nanji Island, Wenzhou City, China, in 2018. The study area encompasses three islands, with geographic coordinates approximately ranging from 121.022°E to 121.122°E and 27.405°N to 27.471°N. Ten multibeam swathes were selected for experimental analysis, each containing over one million points. The minimum number of points was 1,026,000, and the maximum number was 10,776,400 (as shown in Table 2). The selected multibeam swathes were 1~5 km in length and approximately 12.3~26.5 m in depth, with an average resolution of approximately 0.2 m. The specific research area and the seabed topography map of the selected multibeam survey line are illustrated in Figure 5. To validate the effectiveness of the proposed method, the index building time and retrieval time for the traditional quadtree, conventional oriented quadtree, and improved method were measured and compared. The algorithms were implemented using the VTK (Visualization Toolkit, version 8.2.0) library and compiled with Visual Studio 2022 and QT 6.5.2. The computer configuration used for this study was an Intel® Core™ i5-12490F CPU with six cores and twelve threads running at 3.0 GHz with 32 GB of RAM.

**Table 2.** Comparison of index construction time by different methods.

| Swath Number | Number of Point Clouds | Traditional Quadtree (s) | Conventional Oriented Quadtree (s) | Improved Oriented Quadtree (s) | Constructing Quadtree (s) |
|---|---|---|---|---|---|
| 1 | 1,026,000 | 0.581 | 3.063 | 1.107 | 0.515 |
| 2 | 1,760,000 | 0.864 | 4.751 | 1.698 | 0.694 |
| 3 | 2,417,200 | 1.152 | 6.554 | 2.306 | 0.905 |
| 4 | 3,137,600 | 1.325 | 8.212 | 2.995 | 1.182 |
| 5 | 3,538,800 | 1.489 | 9.218 | 3.352 | 1.331 |
| 6 | 4,016,000 | 1.629 | 10.470 | 3.687 | 1.427 |
| 7 | 5,388,800 | 2.010 | 14.593 | 5.100 | 1.897 |
| 8 | 6,235,600 | 2.136 | 18.007 | 5.545 | 1.947 |
| 9 | 9,267,600 | 3.687 | 25.824 | 8.221 | 2.829 |
| 10 | 10,776,400 | 4.102 | 31.039 | 9.741 | 3.515 |




(**a**) (**b**)

**Figure 5.** Study area: (**a**) geographical location image of the study area; (**b**) the seabed topography corresponding to the selected multibeam swathes. The red star represents the geographical location of the experimental data on the global map, the red line represents the range of the measurement area, and the blue line represents the specific location of the measurement area in Figure 5b.

### *3.1. Analysis of Index Construction and Retrieval Time*

Ten selected multibeam swathes are organized on the hard disk using traditional quadtree, conventional oriented quadtree, and improved oriented quadtree methods to construct index files, with a segmentation threshold set at 2.5% of the total number of points in each swath. The time consumption and external storage requirements for the index construction process using these different methods are calculated and presented in Table 2, and the number of layers and effective child nodes in the quadtree, generated using different index construction methods, is shown in Table 3.

The conventional oriented quadtree requires the longest construction time, significantly exceeding that of both the traditional quadtree and the improved oriented quadtree. This time difference becomes more pronounced as the number of point clouds increases. The incorporation of oriented information results in longer construction times for both the conventional and improved oriented quadtrees compared to the traditional quadtree, primarily due to the time-consuming PCA needed to identify the main direction of the

point cloud. However, the improved oriented quadtree necessitates only one process of PCA, leading to a substantially reduced construction time compared to the conventional oriented quadtree.

**Table 3.** Number of quadtree layers and child nodes under different methods (the bolded part in the table below represents the number of child nodes).

| Swath Number | Number of Point Clouds | Traditional Quadtree (s) | Conventional Oriented Quadtree (s) | Improved Oriented Quadtree (s) |
|---|---|---|---|---|
| 1 | 1,026,000 | 4/**106** | 3/**85** | 3/**85** |
| 2 | 1,760,000 | 5/**120** | 3/**85** | 3/**85** |
| 3 | 2,417,200 | 5/**132** | 4/**101** | 4/**101** |
| 4 | 3,137,600 | 5/**125** | 4/**89** | 4/**89** |
| 5 | 3,538,800 | 5/**125** | 4/**81** | 4/**81** |
| 6 | 4,016,000 | 5/**132** | 3/**85** | 3/**85** |
| 7 | 5,388,800 | 5/**125** | 4/**88** | 4/**88** |
| 8 | 6,235,600 | 5/**123** | 4/**137** | 4/**137** |
| 9 | 9,267,600 | 6/**185** | 4/**125** | 4/**125** |
| 10 | 10,776,400 | 6/**176** | 5/**124** | 5/**124** |

When considering only the time required to construct the quadtree, i.e., subtracting the time taken for PCA, the improved oriented quadtree is faster than the traditional quadtree. A comparative analysis of quadtree structures (Table 3) reveals significant performance advantages of oriented quadtrees over conventional implementations. For all multibeam survey lines, the oriented approach demonstrates a 1–2 layer reduction in tree depth and 19.8–35.6% fewer effective child nodes, indicating substantial redundancy reduction in bounding box generation. This structural optimization directly translates to accelerated tree construction, with time savings proportional to the child node reduction. In certain special cases, the distribution of point clouds and the chosen segmentation threshold may cause the effective number of nodes in the oriented quadtree to exceed that of the traditional quadtree, as seen in Swath 8. However, since the oriented quadtree has fewer layers than the traditional quadtree, its construction time remains shorter. Notably, while the improved oriented quadtree maintains equivalent structural complexity to conventional oriented implementations in terms of layer depth and node count (last two columns, Table 3), its computational efficiency gains primarily stem from optimized directional computation algorithms (the fourth and fifth columns in Table 2). Therefore, for multibeam survey lines with particularly straight trajectories, the trajectory direction itself can be used as an alternative to the PCA main direction, significantly reducing the overall tree construction time for the oriented quadtree.

To verify the performance of the proposed method in point cloud retrieval, the 10th group of multibeam swathes containing over ten million points in Table 2 was selected to calculate the time consumption under different retrieval ranges, and compared with the traditional quadtree, conventional oriented quadtree, and traversal methods. The 11 retrieval ranges are illustrated in Figure 6. Except for region 1, all other regions are rectangular areas originating from points A and B. Region 1 does not intersect with the survey line, whereas the range from region 2 to 11 gradually increases uniformly until it includes the entire survey line. The comparison results obtained through querying these 11 regions are shown in Table 4.

As illustrated in Table 4, when the retrieval range does not intersect with the multibeam swath (region 1 area in Figure 6), the retrieval time based on the quadtree is approximately 0.10 ms, significantly lower than that required for traversal retrieval. As the number of point clouds increases to half of the total swath, the quadtree demonstrates a considerable

retrieval advantage over the traversal method. However, as the number of point clouds within the retrieval range continues to rise, the time required for quadtree indexing gradually approaches that of traversal retrieval. When retrieving all point clouds (region 11 area in Figure 6), the traditional quadtree retrieval takes even longer than the traversal method due to its deeper tree layers and empty child nodes. This issue does not occur with oriented quadtrees and improved oriented quadtrees, as both methods exhibit a similar time consumption, which is lower than that of the traditional quadtree, particularly when dealing with a large number of points.

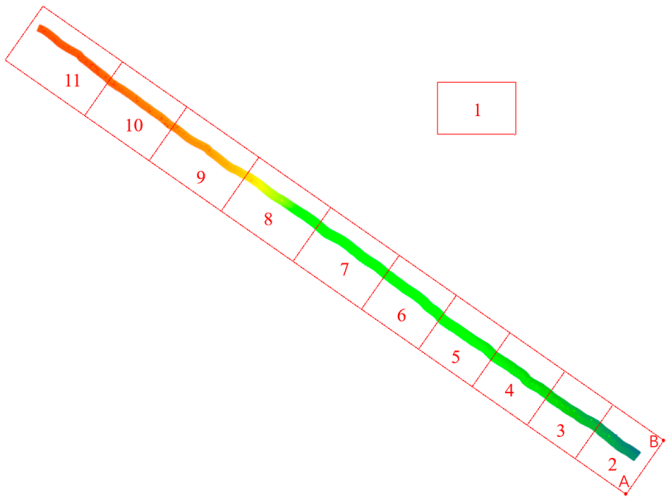


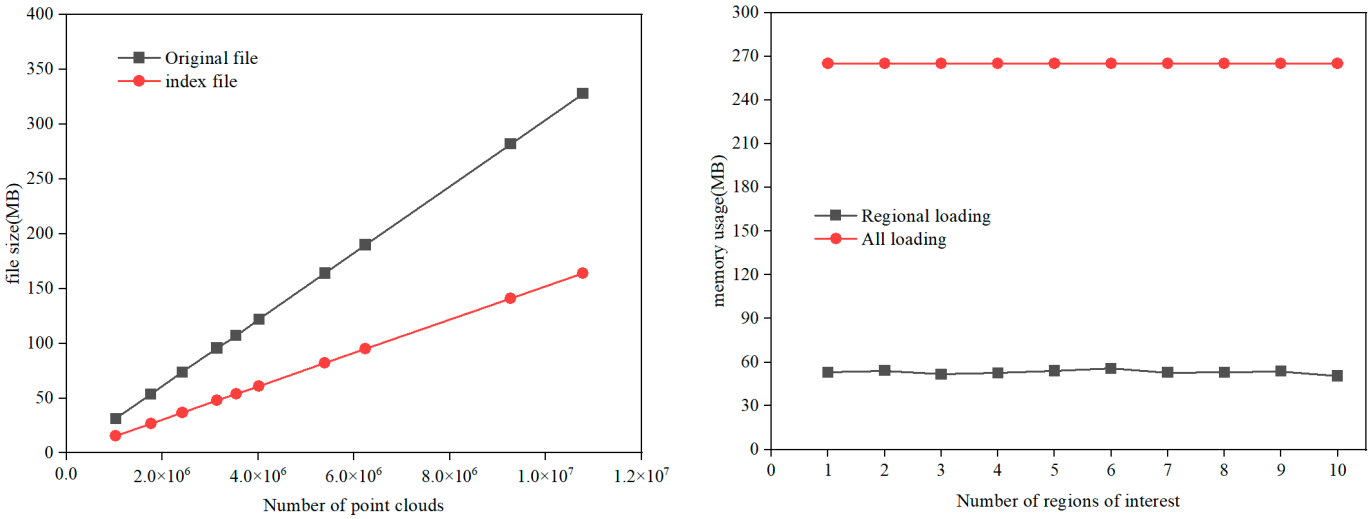

**Figure 6.** Schematic diagram of different retrieval ranges.

**Table 4.** Comparison of retrieval time by different methods under different search ranges.

| Retrieval Region | Point Number | Traditional Quadtree (ms) | Conventional Oriented Quadtree (ms) | Improved Oriented Quadtree (ms) | TRAVERSAL (ms) |
|---|---|---|---|---|---|
| 1 | 0 | 0.11 | 0.10 | 0.10 | 212 |
| 2 | 1,075,204 | 53 | 51 | 52 | 241 |
| 3 | 2,176,067 | 94 | 79 | 82 | 259 |
| 4 | 3,393,547 | 145 | 118 | 118 | 282 |
| 5 | 4,219,178 | 192 | 159 | 162 | 315 |
| 6 | 5,003,093 | 214 | 173 | 180 | 331 |
| 7 | 6,220,027 | 277 | 238 | 233 | 363 |
| 8 | 7,047,486 | 311 | 248 | 251 | 378 |
| 9 | 8,021,641 | 391 | 321 | 320 | 401 |
| 10 | 9,028,699 | 429 | 333 | 335 | 426 |
| 11 | 10,776,400 | 456 | 360 | 361 | 453 |

### *3.2. Analysis of Internal and External Storage Occupancy*

In Section 3.1, the proposed method was applied to organize 10 multibeam swath point clouds stored on a hard disk. Figure 7a illustrates the size of the original file and the corresponding index files for each multibeam swath. The organized quadtree index files in binary format significantly optimize external storage usage, reducing it by approximately 50%. To examine the memory usage of the proposed method during actual data retrieval, the 10th multibeam swath in Table 2 was tested, with the designated retrieval range depicted in Figure 6. A comparison of memory usage between the proposed method and the traditional full-load approach across 10 typical region-of-interest (ROI) scenarios is depicted in Figure 7b. The number of points and the memory usage within the specified search range are presented.




(**a**) (**b**)

**Figure 7.** Point cloud file size and memory usage: (**a**) file size of the original data and corresponding index files for each multibeam swath; (**b**) memory usage of the computer within the different retrieval ranges for the 10th swath in Table 2.

By storing the point cloud data in external storage and constructing a quadtree index, the conventional full-load mode is discarded in favor of a dynamic region-loading mechanism. Initially, spatial range retrieval is performed based on a user-defined ROI, with only the relevant quadtree node data being loaded through internal mapping [40]. After processing, a node evaluation mechanism is triggered, which updates the region's position and actively releases node data outside the current ROI range, while incrementally loading node data that newly falls within the retrieval range. This approach effectively enables the efficient processing of large-scale point cloud data in memory-constrained environments. By only loading a subset of the point clouds within the retrieval range into memory, the index memory usage remains relatively low, enabling the efficient handling of large quantities of multibeam point cloud data and facilitating the rapid retrieval of points or regions of interest.

The primary objective of this method is to accelerate large-scale point retrieval by storing all point clouds solely in leaf nodes while keeping only the relevant child node information for the root and child nodes. In future research, to address the demands of point cloud data processing and visualization, existing index file formats will be expanded to accommodate certain attributes and the topological information of the point clouds.

## 4. Discussion

### *4.1. Effects of the Different Segmentation Thresholds*

The process of building a quadtree index is closely related to the number of point clouds and segmentation thresholds, which ultimately affects the time required for building the tree and point cloud retrieval. The setting of the segmentation threshold will be influenced by the resolution and density of the point clouds. Therefore, different segmentation thresholds were set to construct indexes for the 10 swathes mentioned above. The time consumed and the depth of the quadtree layers were then calculated, as shown in Figure 8 and Table 5, respectively. The time required to build an index increases with the number of points, gradually decreases with the increase in the segmentation threshold, and tends to flatten out. At the initial stages of increasing the segmentation threshold, the tree depth decreases significantly, accompanied by substantial changes in the number of child nodes. This results in notable variations in quadtree construction time, as shown in

the time consumption for segmentation thresholds between 1% and 2% in Figure 8. As the segmentation threshold continues to increase and reaches a certain point, the tree depth stabilizes. At this stage, the small differences in the number of generated child nodes lead to relatively minor variations in construction time, as illustrated by the time consumption for thresholds between 3% and 5% in Figure 8.

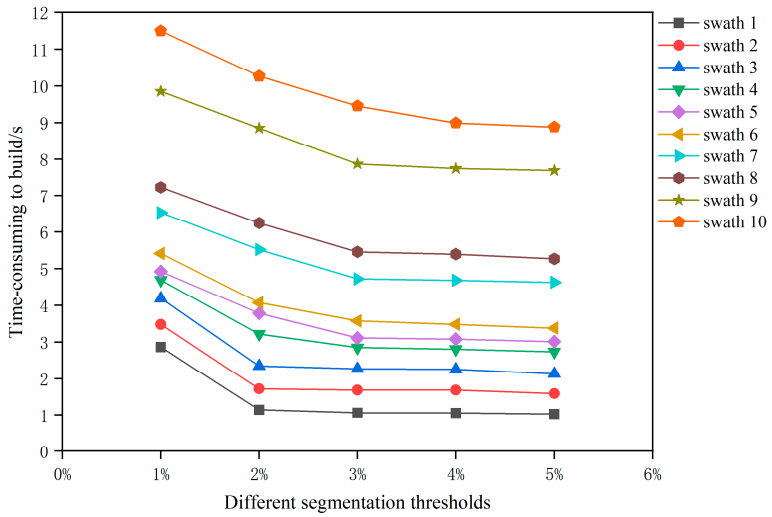



**Figure 8.** Time consumption for index construction with different thresholds for the 10 swathes.

**Table 5.** Number of quadtree layers and child nodes under different segmentation thresholds (the bolded part in the table below represents the number of child nodes).

| Swath Number | Number of Point Clouds | 1% of Total Point Clouds | 2% of Total Point Clouds | 3% of Total Point Clouds | 4% of Total Point Clouds | 5% of Total Point Clouds |
|---|---|---|---|---|---|---|
| 1 | 1,026,000 | 4/**304** | 3/**100** | 3/**84** | 3/**80** | 3/**72** |
| 2 | 1,760,000 | 4/**292** | 3/**92** | 3/**84** | 3/**80** | 3/**68** |
| 3 | 2,417,200 | 4/**300** | 3/**104** | 3/**84** | 3/**80** | 3/**64** |
| 4 | 3,137,600 | 4/**288** | 4/**140** | 3/**84** | 3/**80** | 3/**60** |
| 5 | 3,538,800 | 4/**280** | 4/**172** | 3/**80** | 3/**68** | 3/**64** |
| 6 | 4,016,000 | 4/**280** | 4/**156** | 3/**84** | 3/**76** | 3/**56** |
| 7 | 5,388,800 | 5/**266** | 4/**179** | 3/**71** | 3/**68** | 3/**64** |
| 8 | 6,235,600 | 5/**255** | 4/**171** | 3/**88** | 3/**64** | 3/**60** |
| 9 | 9,267,600 | 5/**251** | 4/**183** | 4/**80** | 3/**64** | 3/**64** |
| 10 | 10,776,400 | 5/**270** | 4/**186** | 4/**111** | 3/**67** | 3/**55** |

The 10th multibeam swath in Table 2 was also selected to examine the retrieval times under different segmentation thresholds (Figure 9). When the number of retrieved point clouds is small (less than half of the total points), the retrieval time increases significantly between the 1% and 3% threshold, after which it levels off. As the number of retrieved point clouds increases (exceeding half of the total number of points), the overall retrieval time curve shows a jumping growth, and rapidly increases between the 3% to 4% threshold. However, when the retrieval range fully encompasses the root node's oriented bounding box, a smaller segmentation threshold results in more nodes and files, leading to an increased retrieval time (as shown by the brown line in Figure 9).

The analysis of the quadtree construction times under different thresholds and the retrieval times for different thresholds also reveals a significant phenomenon: as the threshold decreases, the quadtree construction time increases, while retrieval times for different ranges decrease. Specifically, the smaller, more finely partitioned nodes allow for the quicker and more precise location of the point cloud data within a given region of interest,

leading to decreased retrieval times. Conversely, as the threshold increases, the changes in construction time diminish due to smaller variations in the number of tree layers and child nodes, and retrieval times for different ranges tend to stabilize. Therefore, selecting an appropriate segmentation threshold is crucial when dealing with large quantities of multibeam bathymetric point clouds. A carefully chosen threshold can greatly reduce the time needed to construct the index and simultaneously decrease retrieval times, thereby optimizing the overall processing efficiency. This balance between construction and retrieval performance is essential for practical applications where both speed and accuracy are paramount.

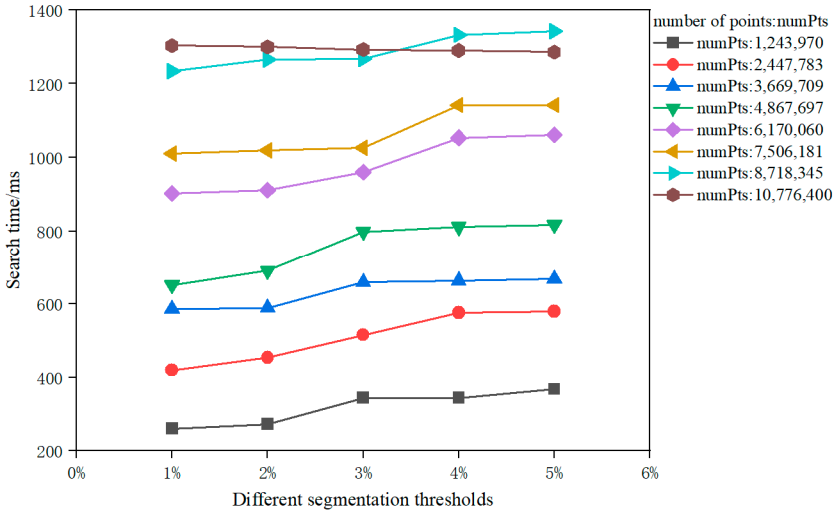



**Figure 9.** Time consumption for point cloud retrieval with different thresholds.

### *4.2. Combining Multi-Threading Technology for Multiple Swathes*

In practical retrieval operations, it is common to perform localized point cloud searches spanning multiple adjacent swathes (i.e., survey lines). When the retrieval region covers multiple overlapping or non-overlapping swathes, the system must independently query point cloud data from each individual swath within the spatial range. This sequential processing approach results in a linear accumulation of retrieval times, where the total execution duration equals the sum of processing times for all involved swathes. Such a methodology becomes particularly inefficient when handling large-scale marine survey datasets containing dozens or hundreds of swathes, significantly hindering real-time processing capabilities and scalability.

To overcome this performance bottleneck, we implement a parallel computing strategy leveraging multi-threading technology. Specifically, our solution employs Qt's advanced thread pool framework to orchestrate concurrent retrieval tasks across multiple swathes [41]. More than 50 multibeam survey lines (over 200 million points) in the Nanji Island area were used to further verify the effectiveness of our method for large-scale data indexing and retrieval processing. The specific multibeam seabed point clouds are shown in Figure 10. Firstly, the time required for constructing the point cloud index was calculated. Without multi-threaded acceleration, constructing the index for more than 50 multibeam swath point clouds took 135 s. With multi-threaded processing, the construction time was reduced to 51 s, representing a time reduction of approximately 60%.

Then, for large-scale cross-band multibeam point cloud data retrieval, five regions were selected (Figure 10). Regions 1 to 5 gradually expanded in area from points A and B, with the number of survey lines covered increasing sequentially. The number of survey lines in the retrieval region, the number of point clouds within the region, and the time consumption for multi-threaded and single-threaded retrieval are shown in Table 6.

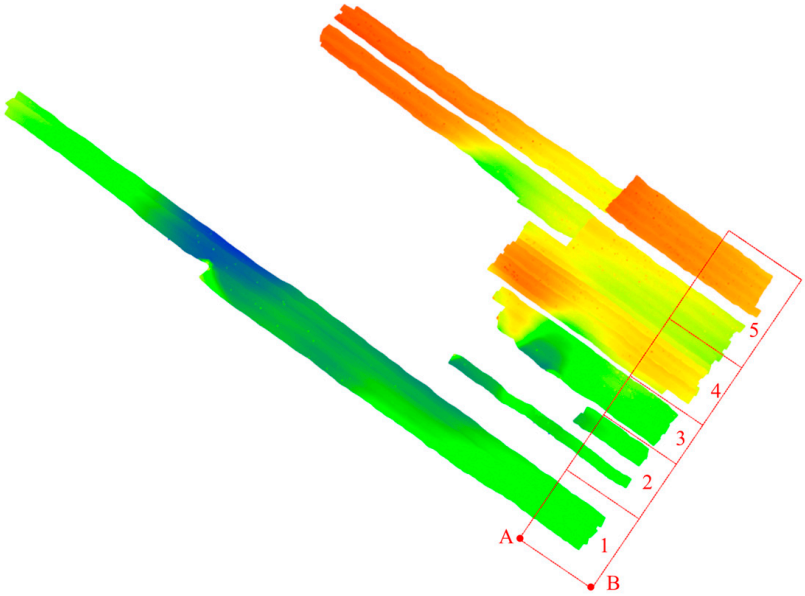



**Figure 10.** Retrieve regions with different numbers of swathes.

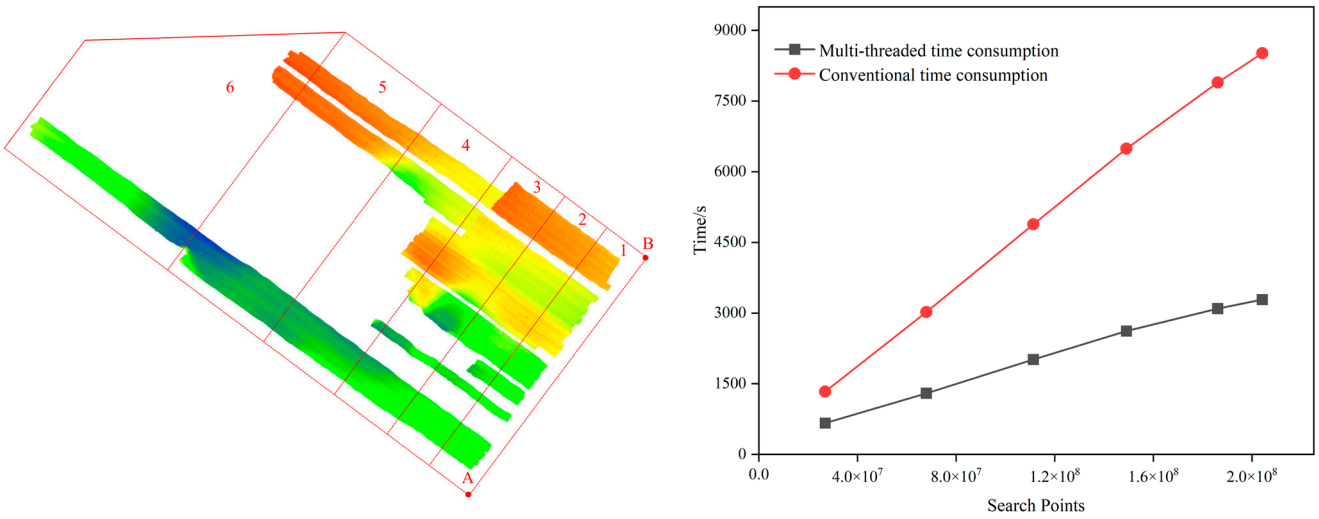
**Table 6.** Comparison of retrieval time under different search ranges.

| Retrieval Region | Number of Coverage Swathes | Point Number | Multi-Threaded Time Consumption (ms) | Conventional Time Consumption (ms) | Improve Efficiency |
|---|---|---|---|---|---|
| 1 | 5 | 3,509,194 | 163 | 200 | 18% |
| 2 | 8 | 5,582,571 | 167 | 271 | 38% |
| 3 | 13 | 9,001,712 | 207 | 431 | 52% |
| 4 | 25 | 18,365,086 | 359 | 890 | 60% |
| 5 | 39 | 32,956,272 | 641 | 1575 | 60% |

As Table 6 indicates, when the number of survey lines covered by the retrieval region does not exceed 12 processor threads (in region 1 and region 2), the multi-thread time consumption increases slightly with the number of survey lines, while the single-thread time consumption increases significantly. Exceeding the number of threads on the processor (in region 3–5), multi-threaded processing reveals its advantage through efficient resource utilization, and though thread scheduling and queuing delays occur when exceeding hardware concurrency limits, the optimized parallelization achieves approximately 60% reduction in processing time compared to single-threaded execution.

Then, six regions (see Figure 11a) were selected to verify the effectiveness of the proposed method for overall region retrieval, with regions 1–6 gradually increasing in area from points A and B. The number of point clouds in each region increased sequentially, with the total number of points reaching approximately 200 million. The time consumption for multi-threaded and single-threaded retrieval is shown in Figure 11b. The results indicate that as the number of retrieved points increases, the time-saving advantage of multi-threaded retrieval becomes more evident, achieving a maximum time reduction of approximately 61% compared to single-threaded retrieval.

The experimental results confirm the effectiveness of multi-threading in practical data processing, particularly for large-scale point cloud indexing and retrieval. This optimization proves especially impactful for large-area seabed mapping projects where datasets frequently encompass hundreds of overlapping swathes. The threading architecture not only enhances operational efficiency but also establishes a foundation for real-time processing applications such as underwater obstacle detection and bathymetric change monitoring.




(**a**) (**b**)

**Figure 11.** Points retrieval for multiple swathes: (**a**) retrieval regions; (**b**) retrieval time.

## 5. Conclusions

This article presents a new method to organize large quantities of multibeam point clouds based on an improved oriented quadtree structure, which considers the swath-shaped characteristics of multibeam bathymetric point clouds. The method proposed in this article is applicable to different densities of multibeam swath point clouds and irregular depth measurement surfaces. Inspired by out-of-core technology, the proposed approach organizes and stores the original multibeam swath points with an improved oriented quadtree structure and a defined binary index file on a hard disk, respectively. To further improve the efficiency of our method, memory mapping and multi-thread techniques are also employed to read the point clouds from the index file. Experimental data demonstrate that the proposed method effectively resolves the challenges of storing large quantities of multibeam bathymetric points in limited memory and mitigates lag caused by storage issues. Moreover, our method achieves higher retrieval efficiency for points within the interest region when compared with the traditional methods.

However, the current method primarily focuses on enhancing point retrieval efficiency. The approach proposed in this paper is particularly well suited for multibeam swath point clouds with inherent directional characteristics. When the principal direction of the multibeam swath is aligned parallel to the axis, traditional quadtree structures are more appropriate for organization and management. In future work, determining the main orientation of the multibeam swath will be crucial for enhancing the method's general applicability. Additionally, for multibeam swathes exhibiting significant redundancy in the oriented bounding box (the trajectory line is an arc), it will be necessary to preprocess the data using alternative data structures before constructing an oriented quadtree. Finally, to enable the efficient editing, computation, and visualization of large-scale point clouds, enhancements to the index file, including the integration of point attributes and topology information, are also required.

**Author Contributions:** Conceptualization, X.B., T.Y. and X.C.; methodology, X.B. and S.D.; software, S.D.; validation, X.B. and T.Y.; formal analysis, J.Z.; investigation, X.B. and Y.Z.; resources, Y.H.; data curation, S.D.; writing—original draft preparation, S.D.; writing—review and editing, X.B. and T.Y.; visualization, S.D.; supervision, X.B. and X.C.; project administration, X.B.; funding acquisition, X.B. All authors have read and agreed to the published version of the manuscript.

**Funding:** The work was supported by the National Natural Science Foundation of China [No. 42204049], the National Natural Science Foundation of Shandong Province [No. ZR2022QD008], the Open Fund of the Key Laboratory of Ocean Geomatics of the Ministry of Natural Resources [No. 2024B12], the National Natural Science Foundation of Hainan Province [No. 624QY578], Special Fund for Basic Scientific Research Business Expenses of Central Public Welfare Scientific Research Institutes [No. TKS20240303].

**Data Availability Statement:** The data presented in this study are available on request from the corresponding author due to privacy.

**Acknowledgments:** We greatly appreciate the constructive comments and suggestions provided by the editor and reviewers on this paper.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Šiljeg, A.; Marić, I.; Domazetović, F.; Cukrov, N.; Lovrić, M.; Panđa, L. Bathymetric Survey of the St. Anthony Channel (Croatia) Using Multibeam Echosounders (MBES)—A New Methodological Semi-Automatic Approach of Point Cloud Post-Processing. *J. Mar. Sci. Eng.* **2022**, *10*, 101. [CrossRef]
2. Rowley, T.; Ursic, M.; Konsoer, K.; Langendoen, E.; Mutschler, M.; Sampey, J.; Pocwiardowski, P. Comparison of terrestrial lidar, SfM, and MBES resolution and accuracy for geomorphic analyses in physical systems that experience subaerial and subaqueous conditions. *Geomorphology* **2020**, *355*, 107056. [CrossRef]
3. Wang, M.; Wu, Z.; Yang, F.; Ma, Y.; Wang, X.H.; Zhao, D. Multifeature Extraction and Seafloor Classification Combining LiDAR and MBES Data around Yuanzhi Island in the South China Sea. *Sensors* **2018**, *18*, 3828. [CrossRef] [PubMed]
4. Hughes Clarke, J.E. First wide-angle view of channelized turbidity currents links migrating cyclic steps to flow characteristics. *Nat. Commun.* **2016**, *7*, 11896. [CrossRef] [PubMed]
5. Zhang, K.; Li, Q.; Zhu, H.; Yang, F.; Wu, Z. Acoustic Deep-Sea Seafloor Characterization Accounting for Heterogeneity Effect. *IEEE Trans. Geosci. Remote Sens.* **2020**, *58*, 3034–3042. [CrossRef]
6. Stateczny, A.; Błaszczak-Bąk, W.; Sobieraj-Żłobińska, A.; Motyl, W.; Wisniewska, M. Methodology for Processing of 3D Multibeam Sonar Big Data for Comparative Navigation. *Remote Sens.* **2019**, *11*, 2245. [CrossRef]
7. Wang, Z.; Zhang, L.; Fang, T.; Mathiopoulos, P.T.; Tong, X.; Qu, H.; Xiao, Z.; Li, F.; Chen, D. A Multiscale and Hierarchical Feature Extraction Method for Terrestrial Laser Scanning Point Cloud Classification. *IEEE Trans. Geosci. Remote Sens.* **2015**, *53*, 2409–2425. [CrossRef]
8. Scheiblauer, C.; Wimmer, M. Out-of-core selection and editing of huge point clouds. *Comput. Graph.* **2011**, *35*, 342–351. [CrossRef]
9. Baert, J.; Lagae, A.; Dutré, P. Out-of-core construction of sparse voxel octrees. In Proceedings of the 5th High-Performance Graphics Conference, Anaheim, CA, USA, 19–21 July 2013; pp. 27–32.
10. Richter, R.; Kyprianidis, J.E.; Döllner, J. Out-of-Core GPU-based Change Detection in Massive 3D Point Clouds. *Trans. GIS* **2012**, *17*, 724–741. [CrossRef]
11. Derigs, U.; Metz, A. An in-core/out-of-core method for solving large scale assignment problems. *Z. Oper. Res.* **1986**, *30*, A181–A195. [CrossRef]
12. Zhang, Y.; Lv, X.Q. In-core and Out-of-core Exchange Rendering Technique of Large-scale Point Cloud. *Comput. Eng.* **2014**, *40*, 49–54.
13. Zhang, Y.; Taylor, M.; Sarkar, T.K.; Moon, H.; Yuan, M. Solving large complex problems using a higher-order basis: Parallel in-core and out-of-core integral-equation solvers. *IEEE Antennas Propag. Mag.* **2008**, *50*, 13–30. [CrossRef]
14. van Oosterom, P.; Martinez-Rubi, O.; Ivanova, M.; Horhammer, M.; Geringer, D.; Ravada, S.; Tijssen, T.; Kodde, M.; Gonçalves, R. Massive point cloud data management: Design, implementation and execution of a point cloud benchmark. *Comput. Graph.* **2015**, *49*, 92–125. [CrossRef]
15. Huang, H.C. Construction of Multi-resolution Spatial Data Organization for Ultralarge-scale 3D Laser Point Cloud. *Sens. Mater.* **2023**, *35*, 87–102. [CrossRef]
16. Shi, L.; Zhang, X.P.; Ma, X.Y.; Ye, K.; Liu, Y.W. Data Analysis System of Oblique Photography Based on OpenSceneGraph. In Proceedings of the 2017 2nd International Conference on Electrical, Control and Automation Engineering (ECAE 2017), Xiamen, China, 24–25 December 2017; pp. 282–285.
17. Guimaraes, N.; Pádua, L.; Adao, T.; Hruska, J.; Peres, E.; Sousa, J.J. VisWebDrone: A Web Application for UAV Photogrammetry Based on Open-Source Software. *ISPRS Int. J. GEO-Inf.* **2020**, *9*, 679. [CrossRef]

18. Yixuan, W.; Xudong, L.; Fenglin, Z.; Zhehui, J.; Yong, T.; Huijie, Z. Design of point cloud data structures for efficient processing of large-scale point clouds. In Proceedings of the International Conference on Optical and Photonic Engineering (icOPEN 2023), Singapore, 27 November–1 December 2023; SPIE: Bellingham, WA, USA; p. 130690.
19. Goswami, P.; Erol, F.; Mukhi, R.; Pajarola, R.; Gobbetti, E. An efficient multi-resolution framework for high quality interactive rendering of massive point clouds using multi-way kd-trees. *Vis. Comput.* **2012**, *29*, 69–83. [CrossRef]
20. Anbin, Y.; Wensheng, M. Index model based on top-down greedy splitting R-tree and three-dimensional quadtree for massive point cloud management. *J. Appl. Remote Sens.* **2019**, *13*, 028501. [CrossRef]
21. Ding, L.; Qiao, B.; Wang, G.; Chen, C. *An Efficient Quad-Tree Based Index Structure for Cloud Data Management*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 238–250.
22. Elseberg, J.; Borrmann, D.; Nüchter, A. One billion points in the cloud—An octree for efficient processing of 3D laser scans. *ISPRS J. Photogramm. Remote Sens.* **2013**, *76*, 76–88. [CrossRef]
23. Bentley, J.L. K-d trees for semidynamic point sets. In Proceedings of the Sixth Annual Symposium on Computational Geometry, Berkley, CA, USA, 6–8 June 1990; pp. 187–197.
24. Wang, W.; Zhang, Y.; Ge, G.; Jiang, Q.; Wang, Y.; Hu, L. A Hybrid Spatial Indexing Structure of Massive Point Cloud Based on Octree and 3D R*-Tree. *Appl. Sci.* **2021**, *11*, 9581. [CrossRef]
25. Zhang, B.; Liu, Y.; Dong, Z.; Li, J.; Chen, Y.; Tang, Q.; Huang, G.; Tao, J. An Optimal Denoising Method for Spaceborne Photon-Counting LiDAR Based on a Multiscale Quadtree. *Remote Sens.* **2024**, *16*, 2475. [CrossRef]
26. Han, S. Towards Efficient Implementation of an Octree for a Large 3D Point Cloud. *Sensors* **2018**, *18*, 4398. [CrossRef] [PubMed]
27. Amorim, P.H.J.; de Moraes, T.F.; da Silva, J.V.L.; Pedrini, H. Out-of-Core Rendering of Large Volumetric Data Sets at Multiple Levels of Detail. In *Multi-Modality Imaging: Applications and Computational Techniques*; Abreu de Souza, M., Remigio Gamba, H., Pedrini, H., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 191–215.
28. Schütz, M.; Ohrhallinger, S.; Wimmer, M. Fast Out-of-Core Octree Generation for Massive Point Clouds. *Comput. Graph. Forum* **2020**, *39*, 155–167. [CrossRef]
29. Ghazizadeh, M.A.; Mohammadian, A.; Kurganov, A. An adaptive well-balanced positivity preserving central-upwind scheme on quadtree grids for shallow water equations. *Comput. Fluids* **2020**, *208*, 104633. [CrossRef]
30. Wang, J.; Tang, Y.; Jin, S.; Bian, G.; Zhao, X.; Peng, C. A Method for Multi-Beam Bathymetric Surveys in Unfamiliar Waters Based on the AUV Constant-Depth Mode. *J. Mar. Sci. Eng.* **2023**, *11*, 1466. [CrossRef]
31. Hamilton, T.; Beaudoin, J.; Clarke, J.H. A more precise algorithm to account for non-concentric multibeam array geometry. In Proceedings of the Canadian Hydrographic Conference, St. John's, NL, Canada, 14–17 April 2014.
32. Bu, X.; Yang, F.; Ma, Y.; Wu, D.; Zhang, K.; Xu, F. Simplified calibration method for multibeam footprint displacements due to non-concentric arrays. *Ocean. Eng.* **2020**, *197*, 106862. [CrossRef]
33. Yang, L.J.; Cui, J.L.; Yang, Z.Q.; Zhai, G.C.; Wang, C. Multi-level index structure based on spatial distribution characteristics of point cloud. *Laser Infrared* **2023**, *53*, 137–145.
34. Beltrami, M.; da Silva, A.C.L. A grid-quadtree model selection method for support vector machines. *Expert Syst. Appl.* **2020**, *146*, 113172. [CrossRef]
35. Samet, H. The Quadtree and Related Hierarchical Data Structures. *ACM Comput. Surv.* **1984**, *16*, 187–260. [CrossRef]
36. Gottschalk, S.; Lin, M.C.; Manocha, D. OBBTree: A hierarchical structure for rapid interference detection. In Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, New Orleans, LA, USA, 4–9 August 1996; pp. 171–180.
37. Chang, J.-W.; Wang, W.; Kim, M.-S. Efficient collision detection using a dual OBB-sphere bounding volume hierarchy. *Comput. Aided Des.* **2010**, *42*, 50–57. [CrossRef]
38. Wang, W.; Ma, J.; Liu, W. Research and Application of Collision Detection Based on Oriented Bounding Box. *Comput. Simul.* **2009**, *26*, 180–183+312.
39. Siwei, H.; Baolong, L. Review of Bounding Box Algorithm Based on 3D Point Cloud. *Int. J. Adv. Netw. Monit. Control.* **2021**, *6*, 18–23. [CrossRef]
40. Song, N.Y.; Son, Y.; Han, H.; Yeom, H.Y. Efficient Memory-Mapped I/O on Fast Storage Device. *ACM Trans. Storage* **2016**, *12*, 1–27. [CrossRef]
41. Evans, N. Verifying QThreads: Is Model Checking viable for User-Level Tasking runtimes? In Proceedings of the 2nd IEEE/ACM International Workshop on Software Correctness for High-Performance Computing (HPC) Applications, Dallas, TX, USA, 12 November 2018; pp. 25–32.