Efficient Visualization of Large-Scale Oblique Photogrammetry Models in Unreal Engine

Yuhao Huo 1, Anran Yang 1, Qingren Jia 1,*, Yebin Chen 2,3, Biao He 2 and Jun Li 1

1 College of Electronic Science, National University of Defense Technology, Changsha 410073, China; huoyuhao19@nudt.edu.cn (Y.H.); yanganran@nudt.edu.cn (A.Y.); junli@nudt.edu.cn (J.L.)
2 Research Institute of Smart City, School of Architecture and Urban Planning, Shenzhen University, Shenzhen 518060, China; chenyebin@whu.edu.cn (Y.C.); hebiao@szu.edu.cn (B.H.)
3 School of Resource and Environmental Sciences, Wuhan University, Wuhan 430072, China
* Correspondence: jiaqingrengis@nudt.edu.cn; Tel.: +86-138-0822-8391

Abstract: Oblique photogrammetry models are indispensable for implementing digital twins of cities. Geographic information system researchers have proposed plenty of methods to load and visualize these city-scaled scenes. However, when the area viewed changes quickly in real-time rendering, current methods still require excessive GPU calculation and memory occupation. In this study, we propose a data organization method in which we merged all quadtrees and used a binary encoding method to encode nodes in a merged tree so that the parent–child relationship between the tree nodes could be calculated using rapid binary operations. After that, we developed a strategy to cancel the loading of redundant nodes based on the parent–child relationship, which helped to reduce the hard disk loading time and the amount of memory occupied in visualization. Moreover, we introduced a parameter to measure the area of the triangle mesh per pixel to achieve unified data scheduling under different production standards. We implemented our method based on Unreal Engine (UE), and three experiments were designed to illustrate the advantages of our methods in index acceleration, frame time, and memory reduction. The results show that our methods can significantly improve visualization fluency and reduce memory usage.

Keywords: oblique photogrammetry model; digital twin; 3D GIS; game engine; dynamic scheduling; data organization

1. Introduction

Michael W. Grieves presented digital twins [1] as a virtual asset of physical products to achieve a replica of the real world, which provides visualized, high-fidelity three-dimensional (3D), city-oriented virtual scenes to support simulations and decision making in smart cities [2,3]. For example, “Virtual Singapore” [4] is the first digital twin of an existing city-state and is a dynamic 3D digital platform that can provide Singaporeans with an effective way to engage in the digital economy and urbanization [5].

With the quickly growing sector of oblique airborne cameras, the potential of aerial photogrammetry for detailed reconstruction and footprint extraction has been demonstrated [6,7]. Given the necessity to cover city-scaled scenes in a digital twinning city, this method is absolutely noninvasive and can obtain high-precision models under various terrain conditions. It is advantageous because it provides information on 3D geometry and ground texture [8] with low time and labor costs. Due to these advantages, oblique photogrammetry models play an important role in the digital twin city scene and can provide a unified spatial reference [9,10]. Many data formats have been designed for this model, such as GLTF [11], 3DTiles [12], and OpenSceneGraph (OSGB) [13]. The volume of data in these models and the complexity of the scenes are constantly increasing [14] with the development of sensor technologies and modeling methods. Force rendering cannot be applied to visualize these massive volumes of data, such as a city-wised high-precision 3D scene...
dataset that contains hundreds of millions of triangles. To address this problem, numerous efforts were made by researchers mainly from two aspects, i.e., model simplification and visualization strategy optimization. For simplification of the oblique photogrammetry model, Li et al. proposed a half-edge collapse method of the quadratic error measure (QEM), which reduces the pressure from rendering [15]. Papageorgiou et al. presented a simplification algorithm for triangular meshes driven by a quadric error metric [16]. Their method improves the speed of simplification, and the resultant models are of comparable quality. Moreover, some researchers proposed visualization strategies including tile pyramid modeling, scene dispatch algorithm, and memory scheduling methods to exploit the transmission performance of the internet and the computational performance of the hardware. Such approaches were successfully applied in Google Earth [17], Cesium [18], and other web-GIS platform projects to visualize city scenes. However, improving rendering performance by reducing hard disk read times, and memory overhead is still a demanding task when employing these methods.

In addition, to simulate actual scenes in the digital world, materials, lighting, rendering, and other computer graphics (CG) technologies are also of great importance [19]. Game engines including Unity3D, Unreal Engine (UE), etc. employ cutting-edge CG technologies that have attracted cross-studies that reconstruct actual scenes in these game engines.

For example, in Unity3D, the visual terrain editor allows users to import the digital elevation model (DEM) and helps developers design terrain scenes effectively. Other game engines, such as TGE and CryEngine, also allow for terrain generation after some conversion [20]. Mohd Hafiz et al. compared Unity3D’s visualization when superimposing contour data at different distance intervals using unmanned aerial vehicle (UAV) images and proposed a method of 3D terrain visualization in the game engine [21]. Buyuksalih et al. visualized artificial 3D city models based on Unity3D and carried out a few projects that estimated potential solar energy on buildings and 3D underground utility mapping for Istanbul City [22]. They showed the potential of 3D Unity visualization and game engine for 3D GIS visualization. However, compared with the oblique photogrammetry model, artificial models are relatively small regarding the number of triangles, which is beneficial in improving the rendering efficiency, but acquiring these data requires a significant amount of time and has high economic costs. Moreover, manual modeling may introduce larger position errors in these real-world scene models. Wang et al. developed an application for visualizing regional oblique photogrammetry data in Unity3D [23] and proposed a double-detail hierarchical loading method that loads the entire low-level-of-detail (LOD) model as a panoramic view of the scene and loads a few dynamic high-LOD model blocks around the viewpoint while roaming; a nine-palace mode was adopted for the high-LOD model block-selection strategy. However, the visualization of large-scale, high-precision oblique photogrammetric models still needs improvement, especially when users quickly change the area viewed.

This paper aims to optimize the visualization strategy and asynchronous loading process of oblique photogrammetry models. In terms of visualization strategy, we propose a data organization method that can improve the indexing efficiency and dispatch algorithm to achieve unified data scheduling under different production standards. In terms of asynchronous loading, we propose an encoding method for quadtree that can reduce real-time loading tasks to improve the efficiency of asynchronous loading-unloading and to reduce memory usage.

The second part of this paper summarizes the visualization strategies of large-scale oblique photogrammetry models in the literature, including commonly used data formats, data organization methods, and data scheduling algorithms. In the third part, we present a data organization method and a data scheduling method and then optimize the data loading process using the encoding method for quadtree. In the fourth part, a prototype system for oblique photogrammetry model visualization is implemented in UE. We perform
experiments with this prototype system and prove that the proposed methods improve the visualization efficiency and reduce the amount of memory occupied.

2. The Visualization Strategy of Oblique Photogrammetry Models

In order to visualize oblique photogrammetry models, first, the entire area containing data is partitioned into blocks; each block is divided based on a quadtree, sometimes not strictly “quad” (a node does not always have four child nodes), and a copy of the data with different precisions is created on each node of the quadtree from the bottom to the top to build a tile pyramid model. Then, the culling method eliminates redundant data outside the viewing frustum by performing intersectional tests. For the remainder of the area containing data, the dispatch algorithm defined by the screen space error (SSE) [24] is employed to choose an appropriate copy of the data for a sub-region, ensuring that the tile can show this region with sufficient accuracy under the current screen resolution of the hardware used and has an appropriate file size that can be loaded into memory. In practice, due to the frequent switching of geographical ranges, data loading and unloading operations are performed frequently, and strategies can be adopted to maintain and optimize existing queues containing tiles that are loaded or to be loaded. Moreover, asynchronous loading is necessary to ensure that the user interface (UI) thread is not blocked in this process.

In the whole process, both the data organization and selection approaches are crucial in achieving efficient rendering of the oblique photogrammetry model. In this section, we demonstrate related methodologies and analyze the bottlenecks of these methods.

2.1. Tile Pyramid Structure for Organizing the Large-Scale Oblique Photogrammetry Model

The tile pyramid model [25] is designed based on the Hierarchy Level of Detail (HLOD) algorithm. This model first divides a large-scale data set into small segments based on spatial partition and organization strategies. A quadtree is established from coarse to fine with a copy of the data from each segment attached to a tree node. Since this approach can help eliminate redundant data according to the accuracy requirements during the visualization process, it plays a vital role in the design of GIS software platforms.

To avoid an increase in data requests caused by too many oblique photogrammetry tile files, the oblique photogrammetry model in ContextCapture, the most commonly used data production tool, is divided into blocks based on regions and a LOD model with a tile pyramid structure is created within each region [26]. Compared with the original tile pyramid structure, it performs model geometry simplification and texture compression on the root node in the upper part of the tile pyramid (as shown in Figure 1). The lower part of each tile pyramid is divided according to the quadtree structure [27]. However, due to the different amounts of original data in each region, the depths of the tile trees are not equivalent in this oblique photogrammetry model.

Figure 1. Hierarchy of the oblique photogrammetry model.
2.2. Data Selection Methods in Visualization

2.2.1. Scene Culling Method

The viewing frustum is used to simulate human vision. In the case of perspective viewing, it is a truncated pyramid [28]. Due to the large volume of data in oblique photogrammetry models, they cannot be entirely loaded into memory. The viewing frustum can be used to operate intersectional tests, with the 3D bounding boxes of all objects attached to the tree nodes in the scene. When users browse the scene, only the tiles within the visual frustum are sent to the dispatch algorithm and the corresponding level of detail of the tiles related to the visible objects is chosen.

For example, in Figure 2, we created a bounding box for each object in the scene. If the bounding box does not intersect the frustum, we do not need to add the object to the loading queue. Thus, frustum culling can significantly reduce the processing pressure of the client in the later scene rendering.

2.2.2. Dispatch Algorithm

For each visible tree node passing the viewing frustum intersection test, a dispatch algorithm is employed to select the appropriate tile level along the tile pyramid tree. For example, in cesium, two parameters are used to determine which level of nodes to be loaded, including the screen space error (SSE), which is calculated according to the metadata of the tile and the relative positional relationship between the viewpoint and the tiles (Equations (1) and (2)), and the preset maximum screen space error threshold (mSSE), which represents the precision required for screen display.

\[
SSE = \frac{GE}{D} \cdot k
\]  
\[
k = \frac{H}{2 \cdot \tan \left(\frac{FOV}{2}\right)}
\]

In Equation (1), \( GE \) is the geometric error of the current tile, which is the diagonal (DB) in Figure 3a. \( D \) represents the closest distance from the viewpoint to the tile center, and \( k \) is a constant determined by \( H \) and \( FOV \), in which \( H \) represents viewpoint height and the field of view (FOV) is the included angle formed by the upper and lower planes of the near plane and the far plane of the frustum (Figure 3b).

![Viewing frustum culling](image-url)
2.2.2. Dispatch Algorithm
For each visible tree node passing the viewing frustum test, a dispatch algorithm is employed to select the appropriate tile level along the tile pyramid tree. For example, in Cesium, two parameters are used to determine which level of nodes to be loaded, including the screen space error (SSE), which is calculated according to the geometric error (GE) in the current tile (Equations (1) and (2)), and the preset maximum screen space error threshold (mSSE), which represents the precision required for screen display.

In Equation (1), GE is the geometric error of the current tile, which is the diagonal of the bounding box (BB) in Figure 3a. D represents the closest distance from the viewpoint to the tile center, and k is a constant determined by H and FOV, in which H represents viewpoint height and the field of view (FOV) is the included angle formed by the upper and lower planes of the frustum (Figure 3b).

If SSE is greater than mSSE, the current node’s error is greater than the threshold, and the child node should be loaded. Otherwise, the current node should be loaded.

2.2.3. Analysis of the Data Selection Process for Viewing the Oblique Photogrammetry Model
To summarize, the loading process of the oblique photogrammetry model is shown in Figure 4. First, the root node in the field of view is obtained through frustum culling. Then, the SSE of the current node and mSSE are recursively compared along the tile tree to select the proper tiles for rendering. Finally, the selected tiles are thrown into a loading array and are loaded asynchronously.

![Diagram](image1.png)

**Figure 3.** Parameters for SSE calculation.

If SSE is greater than mSSE, the current node’s error is greater than the threshold, and the child node should be loaded. Otherwise, the current node should be loaded.

![Diagram](image2.png)

**Figure 4.** Load process of the oblique photogrammetry model.

However, problems still persist in this visualization process. In the data organization step, when the volume of data in the oblique photogrammetry model is massive, the entire dataset is organized into the pyramid model with multiple sub-tile trees. This causes two problems: (1) Each time the culling method is applied to the scene, the root nodes of all of the tile trees are traversed and the non-quadtree division level of the upper layer of the tile tree is equivalent to a linked list in terms of indexing efficiency, which decreases the performance and (2) when the user changes the region viewed, a cross-block tile scheduling process needs to load new tile trees of the multi-subtree pyramid models into the memory, which causes unpredictable rendering lag.

In the data selection step, the GE parameter is only related to the size of the tile and is used to decide on the level of tiles in the tree to be selected. However, the size of the tile does not always indicate geometric precision of the tile. The geometric precision of the corresponding level between different tile trees cannot be presented in GE; that is, no uniform geometric standard exists in different datasets. Thus, if the oblique photogrammetry model has multiple sources with different data production standards, improper tiles may be selected using the current GE-based dispatch algorithm, and unified loading and scheduling cannot be achieved.
3. Research Methodology

3.1. Research Framework

To address the problems raised in Section 2.2.3, we conducted this research by improving the visualization process for a current oblique photogrammetry model, i.e., the data organization and a selection step.

In the data organization step, we created a pyramid model for the whole modeling region by deleting all non-quadtrees divided tiles and merging all quadtrees in this region into a unified quadtree. This unified quadtree can help erase cross-block data scheduling by merging all pyramid models in the sub-regions. Then, after passing the frustum culling method step, we proposed a new parameter representing the area of the triangle mesh per pixel, named geometric pixel error (GPE), to replace the SSE and to provide an index that can measure the difference in the fineness of tiles. The GPE was used to filter tile levels for the unified pyramid model emerging from different data sources.

Furthermore, considering that hard disk I/O is the bottleneck of computer program performance, minimizing the operation of reading tile data from the hard disk and loading it into the memory is also an important strategy to improve the visualization effect. Therefore, a node encoding method was raised to encode all nodes in the unified quadtree and to rapidly calculate the parent–child relationships of nodes in an asynchronous tile loading queue. Thus, the redundant loading of tiles of different levels that cover each other could be canceled, helping to reduce data loading time while maintaining a low memory footprint.

The overall technical workflow is shown in Figure 5, which is discussed in detail as follows:

3.2. The Processing of Oblique Photogrammetry Model

OSGB is an OpenSceneGraph binary scene data format storing 3D scene data. The experimental data used in this paper are OSGB files of OrangeBeach, Changsha, China. In order to realize data reorganization and data selection methods in Unreal Engine, four types of attribute values are required, including the corresponding range of the tiles, the texture and geometry of the tiles, the graphic information corresponding to the tiles, and the metadata information needed for indexing. The entire process can be divided into four aspects (as shown in Figure 6).
Therefore, the NGPE in Equation (3) represents the geometric mesh area of the unit pixel.

Upon zooming in, the corresponding surface area of the tile becomes smaller, and the number of pixels in its texture image increases, resulting in different mesh areas.

In each node level, different mesh areas can be used to distinguish between different levels of nodes with different geometric coordinates, vertex coordinates, and texture coordinates. The NGPE parameter defines only by the tile range cannot precisely describe the tile’s precision. However, a tile is composed of several meshes and a texture image, and the NGPE parameter attached, and it only serves as an index) upwards for all subtrees according to the rules of building a quadtree. Finally, a quadtree is merged with the upper layer as the virtual nodes and with the lower layer as the actual nodes, as shown in Figure 8. However, the levels are not correlated with the fineness of the tiles in this quadtree.

![Diagram](image)

**Figure 5.** Technical workflow of visualizing the oblique photogrammetry model.

First, oblique photogrammetry models are divided into blocks according to the model range. Next, the geometry and texture data are extracted. Then, to convert the data to StaticMesh, which is a common data format accepted in UE4, the data are transformed into meshes based on geometric coordinates, vertex coordinates, and texture coordinates. Moreover, the tangent and the lightmap of the static grid body are recalculated. Lastly, the metadata of the nodes are generated to index the data in the files.

### 3.3. Reorganization of the Oblique Photogrammetry Model for a Game Engine

In order to solve the problem of low index efficiency and rendering lag (Section 2.2.3), a new tile reorganization method based on a unified quadtree method is proposed.

Step 1: Delete non-quadtree divided levels so that all tile trees conform to the quadtree data structure (as shown in Figure 7). However, the non-quadtree divided level depends on the amount of data in the corresponding area. Therefore, the depth of different tile trees after deletion is not consistent.

![Diagram](image)

**Figure 7.** The result of deleting a non-quadtree divided level.

Step 2: Reconstruct a unified quadtree by adding virtual nodes (no actual data are attached, and it only serves as an index) upwards for all subtrees according to the rules of building a quadtree. Finally, a quadtree is merged with the upper layer as the virtual nodes and with the lower layer as the actual nodes, as shown in Figure 8. However, the levels are not correlated with the fineness of the tiles in this quadtree.
In this way, the oblique photogrammetry model can be organized by the unified tree in the whole modeling region, and we need new parameters that do not rely on the levels of the tree to more accurately describe the precision of the tiles to run the dispatch algorithm.

3.4. Dispatch Algorithm Using the Ratio of Screen Error to Geometric Error

Since a unified pyramid with inconsistent hierarchical precision is used at the same tree level, the GE parameter defined only by the tile range cannot precisely describe the tile’s precision. However, a tile is composed of several meshes and a texture image, and they can be used to explain tile precision. As the number of levels increases, the corresponding surface area of the tile becomes smaller, and the number of pixels in its texture is similar between adjacent levels and shows an increasing trend across multiple levels. Therefore, the NGPE in Equation (3) represents the geometric mesh area of the unit pixel in each node level and can be used to distinguish between different levels of nodes with different mesh areas.

SGPE in Equation (4) gives the approximate tile area per unit pixel on the display screen for the current viewpoint by projecting the tiles onto the near plane of the viewing frustum. The calculation method of SGPE is shown in Figure 9 in which D is the distance between the viewpoint and the tile, SR is the screen resolution, and FOV represents the field of view of the frustum.

Figure 8. The unified quadtree using virtual nodes.

Figure 9. Calculation of SGPE.
When NGPE is greater than SGPE, the error at the corresponding level of the node is greater than the error displayed on the screen, and the traversal continues. Otherwise, the current node is loaded.

\[
NGPE = \frac{\text{Total area of triangular mesh}}{\text{Corresponding texture area}} \tag{3}
\]

\[
SGPE = \left( D \cdot \tan\left( \frac{\text{FOV}}{\text{SR}} \right) \right)^2 \tag{4}
\]

In this way, a unified standard loading between multi-region tile trees can be realized for the oblique photogrammetry model under different production standards. Moreover, they can be reorganized into a scene tree for loading scheduling, which is of great significance when using multi-source data and establishing a unified spatial reference. For example, Figure 10 illustrates the data for an 18-layer tile and a 20-layer tile. The data with a deeper level have smaller NGPE values and are selected after the 18-layer one when accessing nodes along the quadtree to select the proper level.

**Figure 10.** The data composition of oblique photogrammetry tiles.

### 3.5. Parent–Child Relationship Encoding Quadtree

The quadtree based on recursive decomposition is often used in geographic information system data representation [29] for spatial partition and index [30]. It stores data in a hierarchical tree, which is widely used in 3D digital cities due to its high query efficiency [31].

In current data loading strategies, tiles are pushed onto the waiting list and loaded asynchronously. In this process, as the scope of the view changes, tiles of different levels in the same block are added to the list and loaded into the memory. As the tile level to be loaded and rendered at the current frame is uniquely determined, it takes a significant amount of time to load other levels of the tile during the movement process. Therefore, if we cancel the loading process of other levels from real-time judgment during the loading process, we need to determine the parent–child relationship between the current frame loading level and the level to be loaded in the waiting list and to cancel the loading task of the levels where a parent–child relationship exists. However, the quadtree is often implemented based on single-linked lists or double-linked lists. Therefore, obtaining the
parent–child relationship between the nodes to be loaded by traversing the entire tree with a large number of tiles is time consuming and reduces the rendering performance.

Thus, a binary encoding method built on the parent–child relationship of a quadtree is proposed. All data in the computer are stored as binaries (two states of 0 and 1), and the relationship between nodes is obtained through two-bit operations, in which compiler conversion is not required, and a higher execution efficiency can be achieved compared with other calculations. In this way, the parent–child relationship between the loaded tile and the tile that the current frame requests can be quickly acquired to ensure that the data in the waiting list does not have a parent–child relationship. The following discusses the principle of encoding implementation (Section 3.5.1) and how to combine it with asynchronous loading to reduce memory reads and writes (Section 3.5). Moreover, this method can also further eliminate memory usage by removing redundant tiles that have a parent–child relationship from memory.

3.5.1. Principles of Parent–Child Relationship Quadtree Encoding

As a two-binary-digit pair can represent four states, that is 00, 01, 10, and 11, it can distinguish four child nodes for each node in the quadtree. The root node is encoded as 00, and each node in this tree has a unique code that expends the code inherited from the parent node with a two-binary-digit pair. Each time the tree’s depth is added, the length of the code increases by two bits. In our implementation, INT32 is chosen to store the binary code, which expresses the relationship between quadtree tiles up to 16 layers. A top-down recursive encoding method is adopted, which is shown in Figure 11.

![Figure 11. Parent–child relationship quadtree encoding process.](image)

To obtain the parent–child relationship of each node based on our encoding strategy, we adopted some rules. First, when the effective digits are equal, they are at the same depth of the tree, so there must be no parent–child relationship. Second, if the effective digits are not equal, take the xor operation on the binary codes of any two nodes, which means that all levels of the two nodes are compared: those that are the same are 0; those that are different are 1; then, those remaining shift the resultant code by x bits (x equals 32 minus the smaller effective encoding digits in two comparison nodes), which is equivalent to retaining only the shallower node level comparison results. If the result is not zero, this part is not the same, and no parent–child relationship is present; otherwise, a parent–child relationship exists between the two nodes compared. The overall process is shown in Figure 12. Finally, if the affiliation between the two nodes regarding the length of the effective digits is different, the longer one represents the child node, and the other one is the parent node.

This method can reduce real-time loading tasks and can reduce a large amount of memory usage in real-world applications at the cost of only storing one more INT32 value in the memory for each node and performing a small number of low-overhead binary operations in the CPU.
frames, as shown in Figure 13. nodes (both loaded and unloaded) that were requested to be loaded in the previous part is not the same, and no parent–child relationship is present; otherwise, a parent–child relationship exists between the two nodes compared. The overall process is shown in Figure 12.

3.5.2. Bit Operations to Obtain the Parent–Child Relationship

To obtain the parent–child relationship of each node based on our encoding strategy, we adopted some rules. First, when the effective digits are equal, they are at the same level data, the process of loading, rendering, and unloading is carried out from the current frame requesting a node can be loaded asynchronously. In this way, redundant node or child nodes are found, the corresponding loading tasks can be canceled and the request from the child nodes for this parent node may be raised in the current frame, so the parent node needs to be marked and unloaded after all of its child nodes are loaded to make sure the entire region related to this parent node can be covered constantly; in this way, we can ensure that the data in the memory do not have a parent–child relationship.

Moreover, the aforementioned binary operations are operated between the nodes that have not been loaded in the array and the new node to build their parent–child relationships, as shown in Figure 14a,b. Then, if its child nodes are found, the new required node unloads them after the current requested node is loaded. Otherwise, if its parent node is found, more than one request from the child nodes for this parent node may be raised in the current frame, so the parent node needs to be marked and unloaded after all of its child nodes are loaded to make sure the entire region related to this parent node can be covered constantly; in this way, we can ensure that the data in the memory do not have a parent–child relationship.

In the asynchronous loading strategy, the parent–child relationship between two tree nodes is divided into three categories, i.e., no relationship, parent–child relationship, and child–parent relationship.

When a new node is required to be loaded, the OnLoadArray is looped, and the aforementioned binary operations are operated between the nodes that have been loaded in the array and the new node to build their parent–child relationships, as shown in Figure 14c,d. Then, for the new required node, if its parent node or child nodes are found, the corresponding loading tasks can be canceled and the current frame requesting a node can be loaded asynchronously. In this way, redundant asynchronous loading tasks can be canceled in real-time, reducing the number of memory reads and writes.

(1) When loading continuous levels in the same area, for example, when the camera gradually drops from a height, if the camera moves faster than the rendering speed of the level data, the process of loading, rendering, and unloading is carried out from the coarse to fine levels, significantly increasing the pressure on real-time rendering. The asynchronous loading method based on the parent–child relationship can prevent tiles that should be uninstalled in the current frame from continuing to load and directly cancels the task during the asynchronous process, which improves the rendering efficiency during

Figure 12. Calculating the process of a parent–child relationship.

Figure 13. All nodes requested to be loaded in the previous frame.
fast browsing, saves on GPU rendering, and improves the fluency of the rendering with a smaller CPU calculation cost by the method.

(2) The traditional method of controlling caching is to cache in chronological order regardless of the relationship between different levels. As a result, multiple tile levels are loaded in the same area simultaneously as the cache, which increases the pressure on memory (as shown in Figure 15a). The parent–child relationship-based culling method can ensure that only a single tile level is loaded in the same area to reduce the pressure on memory (as shown in Figure 15b).

(a) Caching strategy based on chronological order.

(b) The child node has been loaded.

(c) The parent node has not been loaded.

(d) The child node has not been loaded.

Figure 14. The asynchronous loading strategy based on the parent–child relationship.

(a) Caching strategy based on chronological order.

(b) Caching strategy based on parent–child relationship.

Figure 15. Caching strategy comparison.

4. Experiment Verification

A platform was developed to visualize the large-scale oblique photogrammetry model based on Unreal Engine. The model rendering result in this platform is shown in Figure 16.

Three sets of control experiments were set up for the abovementioned unified quadtree method and the asynchronous loading–unloading method based on parent–child relationship encoding for quadtree.

The first experiment compared the unified quadtree method. In order to compare the indexing efficiency of the unified quadtree method and quadtree of the region segmentation method (as shown in Section 4.1), the CPU used in this experiment was an inter(R) Core (TM) i7-9750H CPU @ 2.60 GHz.
The other two experiments addressed asynchronous loading–unloading based on parent–child relationship encoding for quadtree. One of the first experiments assessed the efficiency of visualization while comparing the method with asynchronous loading and unloading (as shown in Section 4.2). The other experiment assessed memory usage while comparing the method with the caching method based on chronological order (as shown in Section 4.3). The CPU and GPU used in this experiment were an Intel(R) Core (TM) i9-10,900 KF CPU @ 3.70 GHz and an NVIDIA GeForce RTX 3090, respectively.

![Visualization of the large-scale oblique photogrammetry model based on UE.](image)

**Figure 16.** Visualization of the large-scale oblique photogrammetry model based on UE.

### 4.1. Index Efficiency Comparison

In order to compare the efficiency of traversing between scenes between the region segmentation method and the unified quadtree method, in this experiment, a fixed path from a high field of view to a low field of view was set as the camera trajectory. The loading tiles array has more tiles of the coarse levels to a small number of tiles in the fine levels, it takes 323 $\mu$s for the region segmentation method and 80 $\mu$s for the unified quadtree method to index to the most refined level, and the result of the comparison is shown in Figure 17. As the loading data become more and more refined, the indexing time of nodes in the unified quadtree is reduced, which is helpful in large-scale HLOD oblique photogrammetry models’ rendering.

![Indexing efficiency from a high field of view to a low field of view under a fixed camera path between the region segmentation method and refactoring the virtual quadtree.](image)

**Figure 17.** Indexing efficiency from a high field of view to a low field of view under a fixed camera path between the region segmentation method and refactoring the virtual quadtree.

### 4.2. Visualization Efficiency Comparison

In order to compare the efficiency of asynchronous loading–unloading with and without the parent–child relationship-based culling method, in this experiment, a fixed roaming path from a high field of view to a low one was set as the camera trajectory. This
experiment compared three calculation times for the game thread (the time of CPU game thread in one frame), the GPU thread (the time of GPU render thread in one frame), and the frame thread (the time executing a frame). The frame time was jointly affected by each frame’s CPU thread and GPU thread and was determined by the larger one. From Figure 18, we found that the visualization efficiency bottleneck is on the GPU thread. Using the parent–child relationship-based culling method reduces the massive GPU rendering time by reducing real-time loading and unloading tasks with only a few increases in CPU calculation. As shown in Figure 18, when loading to the finest level, the increased CPU computing time is less than 3 ms; the GPU thread time is reduced by 4.5 ms, which accounts for about 30% of the total time; and the frame time is maintained below 10 ms for most of the time. Since the CPU thread time is always less than the GPU time, the final frame time is reduced by about 30%. The experimental results indicate that the method can obtain better visual results, as shown in Figure 19.

(a) Comparison of the game time  
(b) Comparison of the GPU time

Figure 18. Comparison of the time efficiency between asynchronous loading–unloading with and without the parent–child relationship-based culling method.

Figure 19. Comparison of the frame time.

4.3. Memory Usage Comparison

In order to compare the memory usage of data scheduling using the chronological order and the parent–child relationship-based culling method, in this experiment, the fixed path roamed around in five locations and moved from a higher field of view to a lower one, and the sum of the texture and triangulation memory was used as the overall memory usage value. The memory recording interval was 5 s. The result of the comparison is shown in Figure 20, and the periodic rapid memory decline in this result is caused by the UE’s tracking Garbage Collection (GC) algorithm releasing the memories of unreachable objects periodically (one-minute intervals in this experiment). As seen from the results, the amount of cache gradually increases due to the caching method based on the chronological order maintained multiple levels of data in memory, while the cache of the proposed parent–child
relationship-based culling method has no apparent upward trend. In fixed path roaming, the proposed method reduces the amount of memory cache by 2200 MB, which accounts for more than 50% by contrast.

Figure 19. Comparison of the frame time.

4.3. Memory Usage Comparison

In order to compare the memory usage of data scheduling using the chronological order and the parent–child relationship-based culling method, in this experiment, the fixed path roamed around in five locations and moved from a higher field of view to a lower one, and the sum of the texture and triangulation memory was used as the overall memory usage value. The memory recording interval was 5 s. The result of the comparison is shown in Figure 20, and the periodic rapid memory decline in this result is caused by the UE’s tracking Garbage Collection (GC) algorithm releasing the memories of unreachable objects periodically (one-minute intervals in this experiment). As seen from the results, the amount of cache gradually increases due to the caching method based on the chronological order maintaining multiple levels of data in memory, while the cache of the proposed parent–child relationship-based culling method has no apparent upward trend. In fixed path roaming, the proposed method reduces the amount of memory cache by 2200 MB, which accounts for more than 50% by contrast.

Figure 20. Comparison of memory usages.

5. Conclusions

This paper improved previous methods of large-scale oblique photogrammetry models visualization from three aspects. First, the unified quadtree method improved the indexing efficiency of a large dataset. Second, for the data scheduling stage, the loading scheduling method based on MPE was proposed to solve the unified scheduling problem of the oblique photogrammetry model under different production standards. Third, in terms of data loading and memory management, the parent–child relationship-based culling method and the asynchronous loading strategy based on binary encoding for quadtree were proposed. These methods were realized in a large-scale oblique photogrammetry model visualization platform based on UE, which proves the advantages of our methods regarding the efficiency of real-time visualization and overall memory usage. Our contribution can improve the efficiency of visualizing a digital twinning city.

To improve the visualization performance, oblique photogrammetry models can be further simplified to reduce the rendering pressure. Additionally, analyses and predictions based on these models can be performed, for example, the estimation of potential solar energy.

In future research, in terms of versatility and transferability, compared with the previously mentioned double-detail hierarchical loading method, this asynchronous loading method based on the parent–child relationship can be used with the data structure of quadtrees and octrees, such as map tiles in Building Information Modeling (BIM). Additionally, the combination of visual content with video streaming is another future direction worthy of further research.

Author Contributions: Conceptualization, Yuhao Huo and Anran Yang; methodology, Yuhao Huo; software Yuhao Huo and Yebin Chen; validation, Yuhao Huo and Qingren Jia; formal analysis, Qingren Jia and Anran Yang; investigation, Qingren Jia and Biao He; resources, Biao He and Jun Li; data curation, Qingren Jia and Jun Li; writing—original draft preparation, Yuhao Huo; writing—review and editing, Qingren Jia and Anran Yang; visualization, Yuhao Huo; supervision, Qingren Jia; project administration, Qingren Jia and Jun Li; funding acquisition, Qingren Jia and Jun Li. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China under Grant No. 41871284, 41971362, 42101435 and the Natural Science Foundation of Hunan Province, No.2021JJ40667.

Institutional Review Board Statement: Not applicable.
Informed Consent Statement: Not applicable.

Data Availability Statement: The experiment data are not publicly available due to the data management policy of our laboratory.

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

References