*electronics*

MDPI

*Article*

# Vertex Chunk-Based Object Culling Method for Real-Time Rendering in Metaverse

**Eun-Seok Lee [1] and Byeong-Seok Shin [2,*]**

[1] Department of VR·Game·Application, Yuhan University, Bucheon 14780, Republic of Korea; elflee77@gmail.com
[2] Department of Electrical and Computer Engineering, Inha University, Incheon 22212, Republic of Korea
* Correspondence: bsshin@inha.ac.kr; Tel.: +82-860-7452

**Abstract:** Famous content using the Metaverse concept allows users to freely place objects in a world space without constraints. To render various high-resolution objects placed by users in real-time, various algorithms exist, such as view frustum culling, visibility culling and occlusion culling. These algorithms selectively remove objects outside the camera's view and eliminate an object that is too small to render. However, these methods require additional operations to select objects to cull, which can slowdown the rendering speed in a world scene with massive number of objects. This paper introduces an object-culling technique using vertex chunk to render a massive number of objects in real-time. This method compresses the bounding boxes of objects into data units called vertex chunks to reduce input data for rendering passes, and utilizes GPU parallel processing to quickly restore the data and select culled objects. This method redistributes the bottleneck that occurred in the Object's validity determination from the GPU to the CPU, allowing for the rendering of massive objects. Previously, the existing methods performed all the object validity checks on the GPU. Therefore, it can efficiently reduce the computation time of previous methods. The experimental results showed an improvement in performance of about 15%, and it showed a higher effect when multiple objects were placed.

**Keywords:** occlusion culling; metaverse; virtual reality; computer graphics; GPU-acceleration



**Citation:** Lee, E.-S.; Shin, B.-S. Vertex Chunk-Based Object Culling Method for Real-Time Rendering in Metaverse. *Electronics* **2023**, *12*, 2601. https://doi.org/10.3390/electronics12122601

Academic Editors: Yiyu Cai, Xiaoqun Wu, Qi Cao and Xiao Zhang

Received: 1 May 2023
Revised: 8 June 2023
Accepted: 8 June 2023
Published: 9 June 2023



## 1. Introduction

In recent years, various technologies for building the Metaverse [1,2] have been studied alongside the advancement of computer graphics and virtual reality technology. Metaverse is a term that combines "meta", meaning transcendence, and "universe", referring to the real world, and it provides users with a space where they can engage in various activities in a virtual world and enjoy interactive environments that contain a variety of content. Metaverse is designed to allow users to interact with it, not only on mobile devices and PCs, but also on various platforms such as VR devices and gaming consoles.

To provide more realistic and high-quality graphics in the Metaverse, various three-dimensional real-time rendering technologies are needed, including high-resolution three-dimensional meshes and realistic lighting effects. As the number of objects that need to be visualized on the screen increases, these rendering technologies require high-performance processors, such as high-performance CPUs or GPUs, and large amounts of memory. In the Metaverse, where interactions occur between users and various objects, it is important to allow users to freely place objects of various types without constraints and show them to others. In the case of a digital twin that needs to respond in both the real and virtual worlds, these features are even more necessary. Therefore, as the number of high-resolution objects placed in the world increases, Metaverse platforms on mobile devices, portable gaming consoles, and regular office PCs, which have relatively weak computing power, experience performance degradation and overheating issues. To address these problems, various algorithms are being researched to reduce the overhead of computing operations.

Recently, object culling techniques have emerged as essential technologies in Metaverse platforms, including game engines [3], to accelerate the rendering of various objects in real-time. Object culling is a technique that removes or simplifies objects that are not visible on the screen to minimize unnecessary rendering tasks. For example, view frustum culling (VFC) [4] and occlusion culling [5], shown in Figure 1, are representative object culling techniques. VFC is a method of processing objects that are not visible on the screen but are limited to objects within the frustum. The view frustum is constructed based on information, such as camera position, field of view and aspect ratio, and only the objects within the view frustum are selected for culling. By doing so, unnecessary objects outside the view frustum do not need to be rendered, resulting in improved rendering performance.

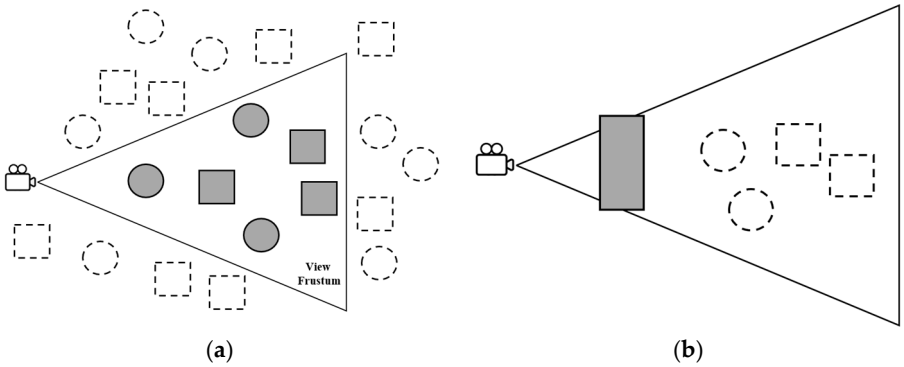



**Figure 1.** Examples of (**a**) VFC and (**b**) Occlusion Culling.

Object culling is a technique used in real-time rendering to improve performance by excluding objects that are not visible to the viewer. Occlusion culling, which is based on the same principle as VFC, selectively excludes objects that are within the viewer's field of view but obscured by other objects, reducing unnecessary computations. To achieve this, the objects within the viewer's field of view are rendered in advance to create a depth buffer, which is used to determine which objects to exclude from rendering. Collision detection [6] or rasterization techniques [7] can be used to determine which objects are not visible.

Object culling can significantly reduce rendering time as the number of excluded objects increases, making it a useful technique in large Metaverse applications where many people are interacting. However, there is a disadvantage that it does not always improve performance. When most objects are not culled, computational time is required to select the objects to cull, in addition to the rendering operations, which can actually decrease rendering performance.

To perform object culling, a common method is to inspect all objects to select the objects to use in the next frame. However, this is highly inefficient, so hierarchical search methods based on bounding volume hierarchies have been proposed. HROC (Hierarchical Raster Occlusion Culling) [8] is a method that performs occlusion culling using hierarchical data structures on the GPU, reducing the computational time required to determine unnecessary objects for rendering. However, this method also has the disadvantage of becoming slower when the number of objects to be processed by the GPU reaches its limit, as it may take longer to perform rendering without object culling.

In this paper, we propose a more efficient approach for performing object culling in large-scale scenes using vertex-chunk. To address the bottleneck issue in the GPU, input data is grouped into chunks and culling is performed at the chunk level on the CPU. This reduces the input data values of the conventional object culling process with minimal computations. Additionally, when creating chunk data, the geometry data is compressed, and unlike previous methods, we do not input bounding boxes of all objects into the GPU rendering pipeline. Instead, each object's bounding box is compressed in vertex format, and only the vertex lists present in the specified area of the chunk map data structure are stored. This partitioning into area units allows for the selection of only the necessary

data for computations, facilitating the real-time assembly of the input buffer in the GPU rendering pipeline and reducing transmission and computational overheads. We also describe a method for efficiently unpacking the vertex-compressed data at the geometry shader stage of the GPU and rapidly selecting the data for culling.

In conclusion, the proposed Vertex-Chunk method is an efficient way to access massive world data compared to existing methods, allowing for the faster selection of objects to cull than previous methods. This method is not limited to occlusion culling but can also be applied to other object culling techniques. In Section 2, we introduce the research related to the proposed method, and in Section 3, we provide a detailed description of the proposed method. In Section 4, we discuss the experiments and results, and in Section 5, we conclude the paper.

## 2. Related Works

Object culling is an essential technique in real-time rendering that efficiently accelerates the rendering process by removing objects that do not affect the final rendering result. With various hardware-based features added to recent GPUs, these techniques are being more effectively utilized. Generally, scenes are most affected by viewing conditions, so the selection process of objects to cull is crucial and dependent on the viewing condition. VFC is a typical example that utilizes the viewing condition. This technique selects only the objects within the field of view to render, effectively culling objects outside the field of view to improve rendering performance. Typically, objects are rendered if their bounding volume falls within the view frustum.

The geometry shader [9] is a GPU pipeline stage [10] that enables the selection and culling of geometric data during on-line processing. Using this stage, the VFC technique [11,12] can efficiently remove objects outside the field of view in a single pass. However, this technique cannot select and remove objects within obstructed areas due to occluders. Occlusion culling is a method of selecting and culling objects located in obstructed areas by occluders.

Recent GPUs perform fragment-level culling effectively by using techniques, such as early depth test (early Z) [7] or Hierarchical-Z(Hi-Z) [13], to remove unnecessary fragment operations on the object's unwanted areas. However, these methods cannot perform culling at the object level, so object operations must be performed through draw calls, which is a disadvantage.

To select objects more efficiently, techniques using depth buffers with a hierarchical structure have been researched [14–16]. These techniques quickly find the obstructed areas by occluders, and select and cull objects located in these areas. This method uses a low-resolution depth buffer to efficiently reduce the amount of computation, solving the problems of existing object-level rendering techniques. However, it requires a two-pass rendering process.

To quickly select objects in a single pass, techniques using bounding volumes are available. These techniques improve the speed of creating the depth buffer by using methods such as Box, Sphere, and K-DOP [12,17,18]. Since they effectively reduce the geometric data of objects using high-resolution meshes, these techniques can resolve the bottleneck that occurs in the rasterization stage when creating the first depth buffer in scenes with many objects.

Methods that use hierarchical structures, such as Bounding Volume Hierarchy [19], have been proposed to efficiently perform occlusion calculations in larger scenes with more objects. Among these methods, research that utilizes temporal coherence [8,20] proposes to update the hierarchy only for objects whose positions have changed since the previous frame, which reduces the cost of updating the hierarchy for objects that have not moved.

With the advancement of game engines [21] used to build the metaverse, various fields have started utilizing object culling techniques. In metaverses based on real-world environments such interior areas [22], acceleration for lighting processing is essential. Therefore, there are ongoing research efforts to integrate occlusion culling techniques with lighting

rendering algorithms. The Particle-k-d tree [23] is a method that performs probabilistic occlusion culling based on ray tracing, which has achieved speed improvements in fixed viewpoints. In the field of lighting, research has also been conducted on methods for culling rays that do not intersect with objects in order to achieve real-time lighting for dynamic entities [24], not just objects.

For molecular-level simulations, such as fluid dynamics, a method has been proposed that divides particles into groups to define occluders for particle-level occlusion culling [5]. This method improves the rendering performance of particle-based rendering by grouping and removing certain particles, instead of performing rendering for all particles. Particles, such as fluids, are challenging to generate accurate geometric data, and occlusion culling is only possible when particle positions align perfectly at the pixel level and each particle is opaque. Performing such calculations for massive particles is inefficient, so grouping them allows for defining occludees for spatial regions, enabling culling and selectively choosing particle data required for rendering more simply and quickly than before.

In the holography domain, performance improvement methods utilizing occlusion culling have also been proposed [25,26]. Computer-generated holography visualizes using point sources as units. These point sources, representing color units in the spatial domain for hologram creation, define occludees in the spatial region through spatial data structures, such as octrees. By reducing calculations for specific regions due to the optical nature of reconstructing computer-generated holographic images, efficient spatial-level operations can be achieved for hologram devices.

In the modern metaverse environment, where realistic images need to be rendered in real-time on mobile platforms, it is advantageous to create accurate occluders that precisely hide objects. Therefore, instead of methods [27] that reduce power consumption by minimizing computations through occlusion culling, an approach [28] has been proposed to automatically place occluders in locations where occlusion occurs frequently. This method places low-resolution occluders in crucial locations, allowing exclusion of high-resolution meshes from rendering calculations, thus efficiently reducing computations.

## 3. GPU Accelerated Hierarchical Object Culling

Object culling is a necessary technique for rendering multiple objects in real-time on Metaverse platforms. However, since additional calculations are required to select objects to cull in the overall process, performance may be lower than with previous methods. To address this issue, the proposed method uses a Vertex-Chunk to reduce the total amount of input data required for culling operations, resulting in more efficient calculations in the graphics pipeline.

The input data consists of objects that make up the three-dimensional world in the content. These data include three-dimensional coordinates and resources, such as shaders or textures, corresponding to materials. Since most objects are rendered individually, culling operations are essential to improve the overall performance of the rendering process.

Figure 2 shows the overall procedure of the proposed method. In the preprocessing stage, a Bounding Volume is obtained for each Object to compress it into a Vertex form. The proposed method uses AABB (Axis-Aligned Bounding Box) [12] to simplify the process. After obtaining the Bounding Volume, it is compressed into a single Vertex. These compressed vertices are then divided into regions and stored as Vertex Chunks. During the rendering stage, only the necessary Chunks are selected and a new input vertex buffer is assembled each frame. This newly assembled buffer is more optimized than previous methods as it contains less data required for the screen space. On the GPU, the restored vertex group is further optimized by applying Visibility checking and VFC using AABB. Then, the fragment shader performs occlusion testing using early depth testing, followed by culling.

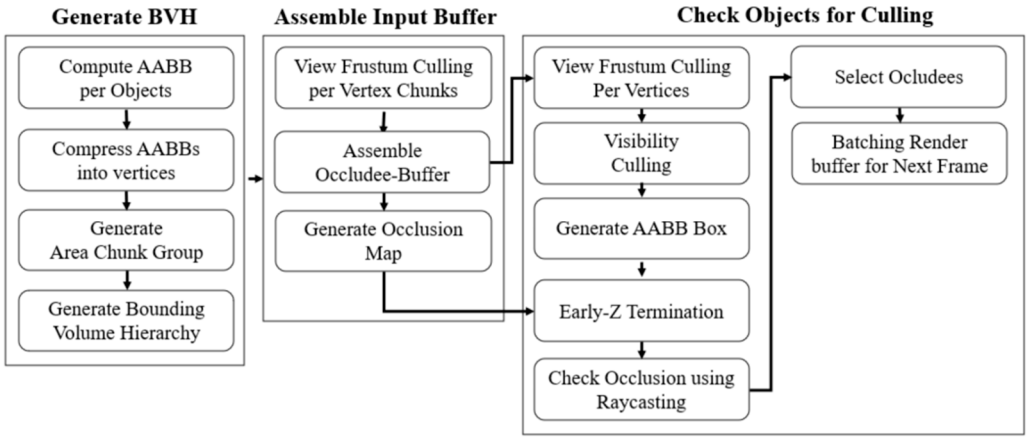



**Figure 2.** Procedure of proposed method.

### *3.1. Object Culling*

To perform object culling, it is important to determine if an object exists in the position where it will be drawn on the screen. The proposed method selects objects to be drawn on the screen using three methods.

- Check if the object is located within the view frustum.
- Check if the object is of a visible size in screen space.
- Check if the object is occluded by an occluder.

The above three methods are performed to check if an object is being drawn on the screen. The first method, VFC, is a widely known acceleration method that determines if an object is located inside the six planes that comprise the View Frustum. This method can be easily computed by using the object's bounding sphere, which can be obtained by drawing a sphere with its center and radius.

The second method measures the size of the object that will be drawn on the screen to determine its visibility. If the predicted size of the object on the screen is smaller than one pixel, it will be culled. This method can also be quickly computed by using the formula for the screen space area $\tau$ when a bounding sphere is drawn on the screen.

$$\tau = \left( r \times \frac{l_s}{2} \times distance\left(c_{obj},\ c_{camera}\right) \times \tan\left(\frac{FOV}{2}\right)\right)^2 \pi \tag{1}$$

The three methods described above are performed to check whether an object is drawn on the screen. The first method, VFC, is a widely known acceleration method that checks whether an object is located inside the six planes that comprise the View Frustum. This method can be easily computed by drawing a bounding sphere around the object using its center and radius.

The second method measures the size of the object that will be drawn on the screen to determine its visibility. If the predicted size of the object is smaller than one pixel, it is culled. The area of the object in the screen space can be quickly computed using the following formula if the bounding sphere is drawn on the screen.

All spheres are assumed to be rasterized into circles in screen space. Using the distance and *FOV* (Field of View) between the viewing camera origin ($c_{camera}$) and the origin of the object ($c_{obj}$), the length of the radius r of the sphere in the screen space can be measured. This can be used to compute the area of the circle with a radius of *r*, which predicts the space the object occupies in the screen space. If the area is smaller than one pixel because the object is very small, culling is performed.

All three methods can perform culling using simple calculations on the GPU by using bounding spheres. However, in the case of the third method, occlusion culling, which uses the latest ROC (Raster Occlusion Culling), it is more advantageous to use an AABB that uses

relatively small geometry because mesh composed of triangles must be rasterized. This method creates an AABB for replace occludes into simple geometry and determines whether they will be drawn on the screen or not by performing the Rasterize phase and using early Z termination. Early Z termination refers to the technique of performing depth testing at an early stage, specifically during the rasterize stage, to determine whether occluded objects can be replaced with simplified geometry and avoid unnecessary rendering.

### *3.2. Data Compression*

In the proposed method, an AABB that can easily determine the object's boundaries is used as a common object boundary so that it can be used in all three culling methods, as described in Section 3.1. To rasterize this bounding box, a mesh composed of 12 triangles with a total of 36 vertices must be used. As the amount of geometry data increases, the overhead related to Object Culling also increases, so the proposed method needs to reduce the data of this mesh.

Figure 3 describes a method of compressing AABB vertices to be used. Generally, AABB is defined as a rectangular parallelepiped whose edges are parallel to the XYZ axes of a three-dimensional space. Therefore, by extracting the maximum and minimum values of the x, y, and z components of the vertices that constitute the object, we can construct an AABB. Let us denote the center of this bounding box as $c$, and let $\vec{r}$ represent the vector from this center to a vertex located in the positive direction. If we obtain vectors of the same size that are symmetric with respect to the remaining octants, given the knowledge of $c$ and $\vec{r}$, we can reconstruct the AABB. So if we save the distance from the center point of AABB, $c$, to the point furthest away in the positive direction from the center point, we can restore the shape of AABB. By saving the position of the center point of AABB in the position attribute and the size of AABB in the normal attribute, such as the gray vertex, we can effectively compress AABB into a single vertex with only position and normal information. This allows for the easy restoration of the mesh as a box shape in the geometry shader stage, when fragment-level calculations of the mesh are required in rasterization stages, such as Raster Occlusion Culling. Therefore, by reducing the amount of input data and operations per vertex in the stage of checking VFC or Visibility, it is possible to restore the data in the Raster stage and use it with a small amount of geometry.

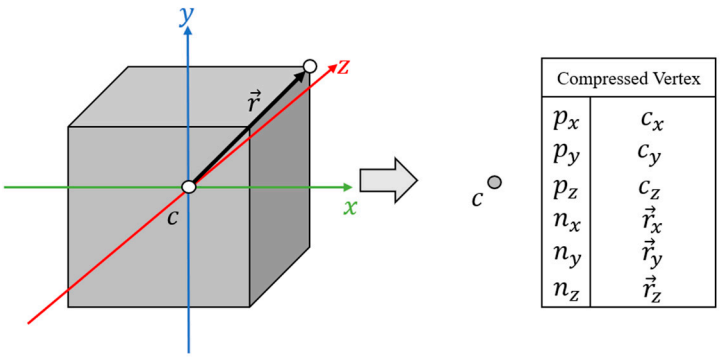



**Figure 3.** A method of compressing AABB into a single vertex.

### *3.3. Vertex-Chunk Generation and Assembling Input Buffer*

Compressed objects can have advantages with minimal geometry, but in the Metaverse, even these compressed geometries can be wasteful. To address this issue, methods have been proposed using Bounding Volume Hierarchy, such as HROC [8], to more efficiently use compressed vertex groups called Vertex-Chunks. BVH is commonly used for objects with hierarchies, such as skeletal meshes, or combinations of different objects that make up a parent object. Assuming a uniform division of an object's area and creating BVH similar to a Quadtree [4], chunks are defined as a bundle of objects located in a divided area. However, this approach presents problems, as shown in Figure 4, where objects

overlapping adjacent areas, such as the gray area in the middle (belonging to Chunk 1), can occur, which can interfere with chunk-level VFC when performing algorithms such VFC. Furthermore, there is an issue of ambiguity in which chunk to include in the compressed vertices of overlapping objects.

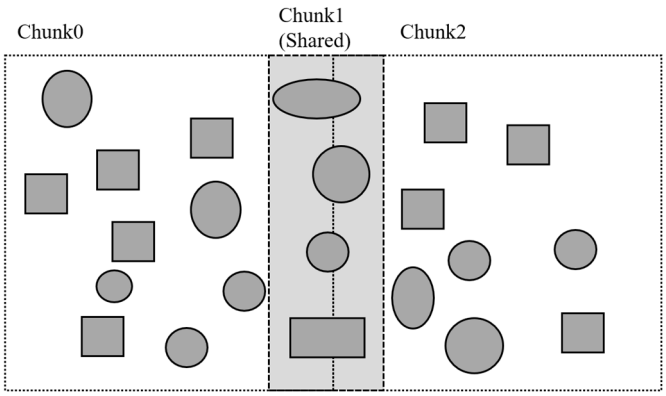



**Figure 4.** The problem that arises when dividing vertex chunks by region.

The proposed method addresses the problem of objects that span multiple regions by separating them into separate chunks defined within a single region or shared by two regions, as shown in Figure 5. The white area contains only vertices that fully fit within the region, while the Shared Chunk stores objects that are shared by both regions. When using these chunks to construct a bounding volume hierarchy (BVH), a hierarchy based on regions can be easily created. This can be particularly useful for applications, such as the Metaverse rendering large worlds, where techniques such as VFC can be efficiently applied in advance. The region selection using BVH can be performed prior to vertex-level culling explained in Section 3.1 to more efficiently reduce the number of vertices required for the culling process.

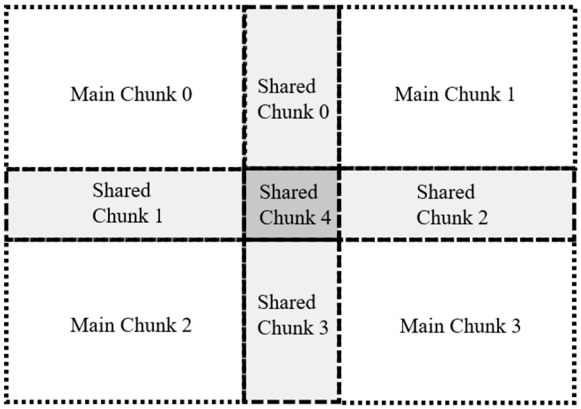


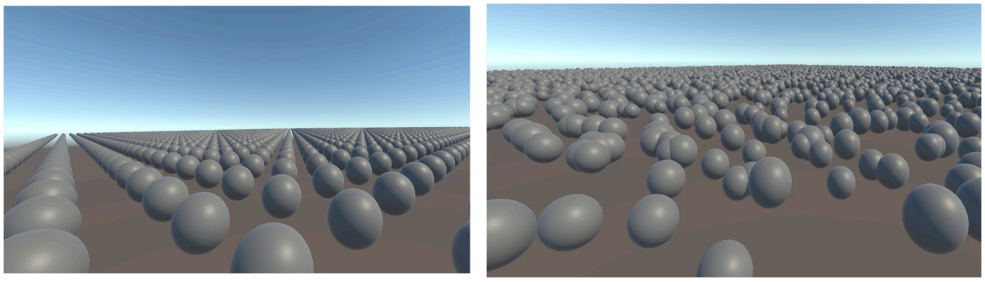

**Figure 5.** How to construct vertex chunk in the proposed method.

## 4. Experimental Results and Discussion

To demonstrate the effectiveness of the proposed method, experiments were conducted to compare it with existing methods. In order to verify real-time rendering of multiple objects in the metaverse, it is crucial to measure the performance of the client process responsible for rendering the screen. Therefore, in the proposed method, the performance of the proposed approach was measured in terms of frames per second (fps) to compare it with other acceleration methods.

The experiments were performed on a popular PC environment with an Intel® Core™ i5-1038NG7 10th Generation (Ice Lake) CPU, 16 GB of main memory, and an Intel® Iris® Plus Graphics processor. DirectX was used as the graphics library on the PC. Two virtual worlds were created for performance evaluation: a "Regular World" with multiple identical

spherical objects evenly distributed, and an "Irregular World" with different-sized and unevenly distributed spherical objects. The results for these worlds are shown in Figure 6. In general, various objects such as cars, buildings, and characters are commonly used in the Metaverse. These objects are typically composed of multiple polygons, and the rendering performance of the GPU pipeline is largely influenced by the number of polygons. Therefore, to construct a test scene in the Metaverse that includes a diverse range of objects, we chose to use spheres, which consist of a higher number of polygons, and arranged them in multiple instances.



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

**Figure 6.** The rendering result videos for the world spaces used in the experiment. (**a**) is for the regular world, and (**b**) is for the irregular world.

Regular world serves as an ideal experimental environment where objects are arranged with a consistent pattern, resulting in a constant number of polygons within the same space. Thus, it represents an optimal scenario for the proposed method of constructing chunks based on spatial units. On the other hand, irregular world features non-uniform placement of objects of varying sizes within the same space. As a result, chunks are constructed in a manner that resembles the real-world configuration of spaces in the Metaverse. Moreover, the overlapping of spheres allows for the formation of diverse shapes, enabling the representation of objects, such as humans, cars, and buildings, which closely resemble the variety of objects found in real-world data.

The experiments were divided into two parts to demonstrate the efficiency of the proposed method. The first part involved measuring the computational complexity required for culling, while the second part involved measuring the actual performance improvement due to culling.

The key aspect of the proposed method is to accelerate the occlusion culling stage by performing VFC and visibility culling [11] on a vertex-chunk basis in large-scale scenes. To demonstrate the efficiency of the proposed approach, it is necessary to compare and analyze it with various existing studies. Among the recent studies in the field of computer graphics, HROC [8], and its foundational study, ROC [10], have shown excellent performance in rendering scenes with large-scale worlds, employing early Z techniques similar to the proposed method. HROC accelerates occlusion culling on GPUs using bounding volume hierarchy and ray-casting techniques, similar to the proposed method. In contrast, the proposed method eliminates vertex chunks in advance through visibility culling and VFC [4]. Therefore, it is necessary to demonstrate that the proposed method can render large-scale scenes faster than HROC and ROC. Additionally, the proposed method is analyzed by comparing it with a test application that applies only visibility culling and VFC, examining whether the proposed method efficiently removes vertex chunks. By comparing these two categories of methods, the efficiency of the proposed approach can be verified when rendering large-scale worlds. The number of chunks was adjusted to ensure that the total number of objects handled by each chunk was less than 100, and the depth of the region for BVH construction was limited to 3 or less.

Tables 1 and 2 measure the execution time for object culling in regular and irregular worlds based on the first experimental results. The proposed method effectively reduces input data by using vertex chunk in advance, but it has the disadvantage of having to create

vertex data every time and update the input buffer. Therefore, in cases such as irregular worlds where data is irregularly clustered and objects to be uploaded sometimes need to be clustered in a specific chunk, there are cases where the performance is not significantly different from algorithms that do not require updates, such as HROC, if the number of vertices increases significantly or if the Chunk is too detailed and the Buffer Assemble time takes too long. However, as shown in the results, the proposed method has decreased the processing time by about 23% compared to the latest model HROC, which showed the fastest speed. It is predicted that the total GPU operations can be efficiently reduced by significantly reducing the input data.

**Table 1.** Computing time required for object culling in regular world.

| | Total Object Num | Number of Vertices | Processing Time (ms) |
|---|---|---|---|
| Proposed Method | 9024 | 2338 | 17 |
| HROC | 9024 | 324,864 | 29 |
| ROC | 9024 | 324,864 | 45 |
| VFC + VC | 9024 | 9024 | 21 |

**Table 2.** Computing time required for object culling in irregular world.

| | Total Object Num | Number of Vertices | Processing Time (ms) |
|---|---|---|---|
| Proposed Method | 9024 | 2703 | 21 |
| HROC | 9024 | 229,832 | 26 |
| ROC | 9024 | 324,864 | 55 |
| VFC + VC | 9024 | 9024 | 31 |

To prove this, Tables 3 and 4 measured the time it takes for rendering in fps. As shown in the table, the proposed method efficiently manages the Object list through Chunk beforehand, even though the computation time to get the Objects to Cull takes longer than HROC, resulting in superior performance in Culling. Ultimately, by using Vertex Chunk to reduce input data and minimize GPU computations, this method showed 15% faster performance than HROC, which uses the same approach with BVH.

**Table 3.** Comparison of improved rendering time through the proposed method in regular world.

| | Total Object Num | Culled Object Num | Rendering Speed (fps) |
|---|---|---|---|
| Proposed Method | 9024 | 7938 | 193 |
| HROC | 9024 | 7938 | 172 |
| ROC | 9024 | 7938 | 97 |
| VFC + VC | 9024 | 5849 | 78 |
| No Acceleration | 9024 | 0 | 12 |

**Table 4.** Comparison of improved rendering time through the proposed method in the irregular world.

| | Total Object Num | Culled Object Num | Rendering Speed (fps) |
|---|---|---|---|
| Proposed Method | 9024 | 8127 | 203 |
| HROC | 9024 | 8127 | 178 |
| ROC | 9024 | 8127 | 91 |
| VFC + VC | 9024 | 6048 | 62 |
| No Acceleration | 9024 | 0 | 11 |

Figure 7 presents the results of FPS measurements in a moving view. The camera view used for the measurements allows vertical movement in a three-dimensional space, rather than a walkthrough. Starting from the ground with scene 1, the camera gradually ascends vertically to encompass the entire world as the scene number increases. As the view moves farther from the ground, the number of objects that need to be displayed in each scene increases. Consequently, the number of culled objects decreases, resulting in an overall decline in FPS. The proposed method exhibits the best performance compared

to existing methods in this scenario. This is attributed to the effective improvement of the existing methods' limitations, which involve performance degradation as the input data increases. The proposed method achieves this by utilizing vertex chunks in the preprocessing stage to efficiently reduce the number of objects required for computation, ensuring optimal efficiency.

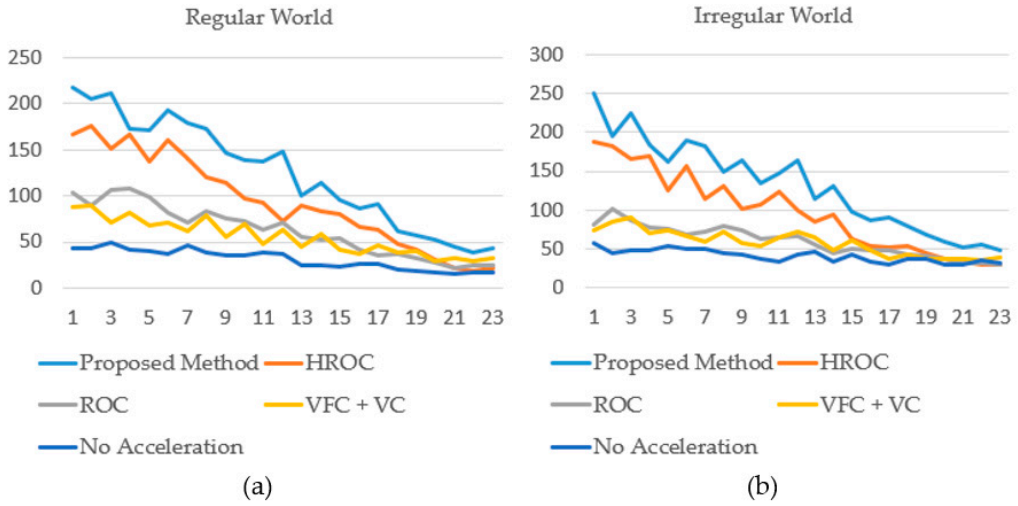



**Figure 7.** (**a**) Video measuring FPS in a moving view of the regular world, (**b**) Video measuring FPS in a moving view of the irregular world. The horizontal axis represents the scene number, and the vertical axis represents the FPS displayed.

In Table 5, we compared the average performance of object placement in the near view, close to the ground where objects are placed, and the far view captured from a high altitude, such as an aerial view. In the near view, chunk-level culling is performed during preprocessing, resulting in a reduced number of polygons being input to the GPU compared to the conventional HROC, and occlusion causes multiple objects to be culled. Therefore, it shows relatively faster rendering speed under these observation conditions.

**Table 5.** Comparison of average FPS with conventional methods in fixed view.

| | **Regular World** | | **Irregular World** | |
|---|---|---|---|---|
| | **Far View (fps)** | **Near View (fps)** | **Far View (fps)** | **Near View (fps)** |
| Proposed Method | 29 | 220 | 37 | 223 |
| HROC | 21 | 192 | 33 | 182 |
| ROC | 23 | 108 | 31 | 92 |
| VFC + VC | 31 | 88 | 36 | 83 |
| No Acceleration | 9 | 12 | 9 | 12 |

However, in the far view of this experiment, more objects come into the field of view at once compared to the near view, and occlusion occurs less frequently. As a result, there is a situation where most objects need to be rendered because there are fewer objects to be culled. For this reason, as shown in Table 5, it is possible to achieve a performance improvement of over 20% compared to HROC in the near view, but in the far view, it shows similar or slower speed compared to the basic methods of view frustum culling and visibility culling. This is because they perform visibility checks for the objects composing the scene, but most of them are not culled. However, unlike the latest techniques, such as HROC or ROC, the method that pre-removes objects using view frustum culling and visibility culling shows a significant performance difference due to the small input data size. This demonstrates the efficient resolution of the issue of computational overhead in cases where occlusion occurs infrequently, which was a problem with conventional HROC.

## 5. Conclusions

In the metaverse, as the world space expands and numerous users concurrently access it, the real-time rendering of a multitude of objects in a wide space has become a crucial requirement. Consequently, reducing the number of polygons has emerged as a highly significant topic in the development and research domains of the metaverse. In this paper, we propose a method of accelerating rendering performance by using a data structure called vertex chunk to efficiently perform object culling for fast rendering processing. This is a new approach to hierarchical raster-occlusion culling that uses GPU-accelerated bounding volume hierarchy to improve performance. This method effectively reduces the data input to the GPU rendering pipeline while performing culling quickly, eliminating the need to upload all data to GPU memory for processing. Therefore, it efficiently reduces the computation required for selecting objects to cull, which results in an improved overall performance. However, the method is not always effective. Similar to HROC and ROC, this method also needs to render most objects when there are not many objects to cull on the screen. Therefore, it shows little or even slower performance compared to the method that only performs view frustum culling and visibility culling. This is because the proposed method involves additional operations for rendering passes and selecting vertex chunks for object culling. If there are no objects to cull on the screen, these operations are unnecessary. Future research will require exploring methods to perform object culling for dynamic objects that require real-time updates by updating chunk data in real-time.

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

**Funding:** This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No.NRF-2022R1A2B5B01001553 and No. NRF-2022R1A4A1033549).

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Data sharing not applicable.

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

## References

1. Dionisio, J.D.N.; Burns, W.G., III; Gilbert, R. 3D Virtual Worlds and the Metaverse: Current Status and Future Possibilities. *ACM Comput. Surv.* **2013**, *45*, 1–38. [CrossRef]
2. Gaafar, A.A. Metaverse In Architectural Heritage Documentation & Education. *Adv. Ecol. Environ. Res.* **2021**, *6*, 66–86.
3. Gregory, J. *Game Engine Architecture*, 3rd ed.; CRC Press: Boca Raton, FL, USA, 2018; ISBN 978-1-351-97427-1.
4. Lee, E.-S.; Lee, J.-H.; Shin, B.-S. Bimodal Vertex Splitting: Acceleration of Quadtree Triangulation for Terrain Rendering. *IEICE Trans. Inf. Syst.* **2014**, *E97-D*, 1624–1633. [CrossRef]
5. Ibrahim, M.; Rautek, P.; Reina, G.; Agus, M.; Hadwiger, M. Probabilistic Occlusion Culling Using Confidence Maps for High-Quality Rendering of Large Particle Data. *IEEE Trans. Vis. Comput. Graph.* **2022**, *28*, 573–582. [CrossRef]
6. System for Collision Detection Between Deformable Models Built on Axis Aligned Bounding Boxes and GPU Based Culling-ProQuest. Available online: https://www.proquest.com/openview/ee52f6ff878bc46c41bc5f5967089c25/1?pq-origsite=gscholar&cbl=18750&diss=y (accessed on 27 March 2023).
7. Kubisch, C. OpenGL 4.4 Scene Rendering Techniques. *NVIDIA Corp.* 2014. Available online: https://on-demand.gputechconf.com/gtc/2014/presentations/S4379-opengl-44-scene-rendering-techniques.pdf (accessed on 27 March 2023).
8. Lee, G.B.; Jeong, M.; Seok, Y.; Lee, S. Hierarchical Raster Occlusion Culling. *Comput. Graph. Forum.* **2021**, *40*, 489–495. [CrossRef]
9. Mukundan, R. The Geometry Shader. In *3D Mesh Processing and Character Animation: With Examples Using OpenGL, OpenMesh and Assimp*; Mukundan, R., Ed.; Springer International Publishing: Cham, Switzerland, 2022; pp. 73–89; ISBN 978-3-030-81354-3.
10. Haar, U.; Aaltonen, S. GPU-Driven Rendering Pipelines. *SIGGRAPH Adv. Real-Time Render. Games Course* **2015**, *2*, 4.
11. Sunar, M.S.; Zin, A.M.; Sembok, T.M.T. Improved View Frustum Culling Technique for Real-Time Virtual Heritage Application. *Int. J. Virtual Real.* **2008**, *7*, 43–48.
12. Assarsson, U.; Moller, T. Optimized View Frustum Culling Algorithms for Bounding Boxes. *J. Graph. Tools* **2000**, *5*, 9–22. [CrossRef]

13. Greene, N.; Kass, M.; Miller, G. Hierarchical Z-Buffer Visibility. In *20th Annual Conference on Computer Graphics and Interactive Techniques-SIGGRAPH '93*; ACM Press: New York, NY, USA, 1993; pp. 231–238.
14. Klosowski, J.Y.; Silva, C.T. *The Prioritized-Layered Projection Algorithm for Visible Set Estimation*; IEEE: Piscataway, NJ, USA, 2000; Volume 6, pp. 108–123. Available online: https://ieeexplore.ieee.org/abstract/document/856993 (accessed on 1 May 2023).
15. Ho, P.C.; Wang, W. Occlusion Culling Using Minimum Occluder Set and Opacity Map. In Proceedings of the 1999 IEEE International Conference on Information Visualization (Cat. No. PR00210), London, UK, 14–16 July 1999; pp. 292–300.
16. Nießner, M.; Loop, C. *Patch-Based Occlusion Culling for Hardware Tessellation*; Computer Graphics International: Poole, UK, 2012; pp. 1–9.
17. Fünfzig, C.; Fellner, D. Easy Realignment of K-DOP Bounding Volumes. *Graph. Interface* **2003**, *3*, 257–264.
18. 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]
19. Zhang, H.; Manocha, D.; Hudson, T.; Hoff, K.E. Visibility Culling Using Hierarchical Occlusion Maps. In *24th Annual Conference on Computer Graphics and Interactive Techniques-SIGGRAPH '97*; ACM Press: New York, NY, USA, 1997; pp. 77–88.
20. Coorg, S.; Teller, S. Temporally Coherent Conservative Visibility (Extended Abstract). In Proceedings of the Twelfth Annual Symposium on Computational Geometry, Philadelphia, PA, USA, 24–26 May 1996; pp. 78–87. [CrossRef]
21. Duanmu, X.; Lan, G.; Chen, K.; Shi, X.; Zhang, L. 3D Visual Management of Substation Based on Unity3D. In Proceedings of the 2022 4th International Academic Exchange Conference on Science and Technology Innovation (IAECST), Guangzhou, China, 9–11 December 2022; pp. 427–431.
22. Oksanen, M. 3D Interior Environment Optimization for VR. *Bachelor's Thesis*; Turku University of Applied Science. Available online: http://www.theseus.fi/handle/10024/788282 (accessed on 2 June 2023).
23. Walloner, P. Acceleration of P-k-d Tree Traversal Using Probabilistic Occlusion. Bachelor's Thesis, University of Stuttgart, Institute for Visualization and Interactive Systems, Stuttgart, Germany, 2022.
24. Xu, Y.; Jiang, Y.; Zhang, J.; Li, K.; Geng, G. *Real-Time Ray-Traced Soft Shadows of Environmental Lighting by Conical Ray Culling*; Association for Computing Machinery: New York, NY, USA, 2022; Volume 5.
25. Wang, F.; Ito, T.; Shimobaba, T. High-Speed Rendering Pipeline for Polygon-Based Holograms. *Photonics Res.* **2023**, *11*, 313–328. [CrossRef]
26. Martinez-Carranza, J.; Kozacki, T.; Kukołowicz, R.; Chlipala, M.; Idicula, M.S. Occlusion Culling for Wide-Angle Computer-Generated Holograms Using Phase Added Stereogram Technique. *Photonics* **2021**, *8*, 298. [CrossRef]
27. Corbalán-Navarro, D.; Aragón, J.L.; Anglada, M.; Parcerisa, J.-M.; González, A. Triangle Dropping: An Occluded-Geometry Predictor for Energy-Efficient Mobile GPUs. *ACM Trans. Archit. Code Optim.* **2022**, *19*, 39:1–39:20. [CrossRef]
28. Wu, K.; He, X.; Pan, Z.; Gao, X. Occluder Generation for Buildings in Digital Games. *Comput. Graph. Forum* **2022**, *41*, 205–214. [CrossRef]