Object level frustum culling based frame rate acceleration method

Le Hoang Minh[1*], Tran Binh Minh[1], Luu Van Sang[1], Hoang The Khanh[2]

[1]Military Information Technology Institute, Academy of Military Science and Technology;
[2]Missile Institute, Academy of Military Science and Technology.
[*]Corresponding author: minh.rti@gmail.com
Received 16 Sep 2022; Revised 5 Dec 2022; Accepted 15 Dec 2022; Published 30 Dec 2022.
DOI: https://doi.org/10.54939/1859-1043.j.mst.CSCE6.2022.28-40

ABSTRACT

Frame rate is an important index to evaluate the performance of simulation systems. The higher it is, the more intuitive and realistic the simulation application will be. The terrain database rendering process in the simulation field takes a lot of time and hardware resources. Therefore, optimizing it leads to a significant speedup of the simulation system. The frustum culling method is very helpful in avoiding computations of things that are not visible. In this method, instead of sending all information to the graphics processing unit, visible and invisible elements will be sorted, and only visible elements will be rendered. This paper presents an effective frustum culling technique at object level. In the proposed method, the rendered region was frustum culled by objects, so the number of objects to be rendered in each frame was significantly reduced. The purpose of the study is to improve the rendering performance of terrain data in the Unity 3D framework. Experimental results have shown that the given method is very competitive.

Keywords: Frame rate; Culling; Simulation; Terrain database; Image render; Unity 3D.

1. INTRODUCTION

1.1. Frame rate in the simulation field

In the simulation field, frame rate (FR) indicates the number of frames the screen displays per second or the refresh rate of the screen. FR is measured by frames per second (FPS) unit and is one of the most important factors that affect to user's experience. In fact, the higher FR is, the smoother the displayed image will be, and the more realistic the user will feel. And vice versa, low FR (with FPS <20) will cause jerky lag, adversely affecting the user experience, and in many cases can cause dizziness and nausea [1].

Normally, FR is divided into the following segments:

- 24 FPS: the FR commonly used in filmmaking is the minimum speed that allows users to feel the movements realistically.

- 30 FPS: this is a common FR, considered the minimum standard on games and simulation to ensure the majority of users won't notice any lag.

- 60 FPS: often considered the ideal FR, the balance between user experience and investment costs for system configuration.

- 120 FPS: this speed can only be achieved on computers with powerful hardware connected to a monitor with a refresh rate of up to 144Hz and a well-optimized program.

The ideal FR value in simulation systems is 60 FPS, but it is difficult to achieve in practice, so any value in the range from 30 FPS and 60 FPS is acceptable.

1.2. Factors affecting frame rate in simulation applications

To understand which factors affect FR in simulation applications, first, we should

know the frame life cycle, where it begins, and what course it takes. Actually, the life cycle of the frame is quite complicated, but in this study, we give a brief description with the main purpose of analyzing the factors affecting the FR. The life cycle of the frame is illustrated in figure 1. In this cycle, at first, the simulation application sends a frame to the Central Processing Unit (CPU), then the CPU calculates and processes the frame and, after that, sends it to the rendering queue of the Graphics Processing Unit (GPU). In the next step, the GPU performs calculations (geometry, texture, shading…), puts them into the render buffer and finally outputs to the monitor to display. In the same way, all the next frames are processed and displayed by the monitor.

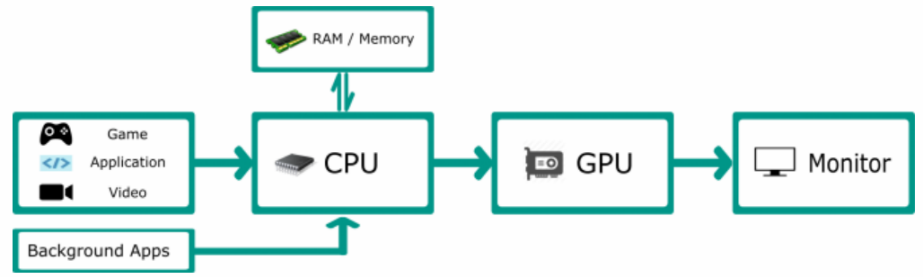


Figure 1. *Life cycle of the frame.*

There are some main factors that affect FR in simulation applications [2]:

(1) Hardware and its quality (CPU, RAM, GPU and monitor)

The device facilities and their qualities greatly affect the FR. A device with a faster CPU, a higher capacity RAM and a higher GPU will perform faster processing and consequently higher FR and vice versa. In addition, a monitor with a low refresh rate might not display all the frames it receives from the GPU in simulation systems with a high FR.

(2) Code and calculation optimization

This factor is greatly related to developers of simulation systems which deal with FR. The code writing style affects the FR directly as the more well written the code is and the smarter the calculations are, the higher the FR can get and consequently can avoid choppy movement.

(3) Content of the frame

The content of the frame affects the FR as the frames with high resolution take relatively much more time for rendering than those with low resolution. In simulation developing, each object has its own scripts, codes and calculations. So, the more objects present in the frame, the more calculations there would be and the more time needed for rendering that frame and consequently this would affect the FR.

(4) Applications running in the background

There may be programs and services running in the background, and these applications will take possession of the device resources and indirectly affect the FR of other applications.

1.3. Common methods to enhance frame rate

Based on the factors that affect FR mentioned above, there are some common methods to enhance FR in simulation applications as follows:

(1) Upgrade hardware

One of the simplest ways to improve the FR is choosing high configuration devices to implement simulation systems with high FR. However, this depends on the user's ability to invest and is not always possible, especially in the case of reusing of available equipment.

(2) Optimize software and database rendering

Optimization in designing, building database and coding would reduce the number of computations and increases the FR. Normally, simulation frameworks or third-party applications provide tools for optimizing the code of simulation applications but the combination with developer's knowledge will be better.

(3) Culling content of the frame

As mentioned above, the more objects present in the frame, the more calculations would be and the lower FR is. Therefore, in simulation systems, decreasing the number of objects in the frame would make FR becomes higher.

(4) Restrict background applications

Turn off the programs and services running in the background when don't need to save as much of the device resources as possible. However, this can only be done by users of simulation systems, not developers.

In this study, we concentrate on the third method (culling content of the frame) by reducing the number of objects in the frame. The rest of this paper is organized as follows: problem and methodology are described in section 2 and section 3 respectively. Section 4 presents experiments, results and discussion. Finally, the conclusions are given in section 5.

2. PROBLEM

2.1. Frame rendering process in Unity 3D

Unity 3D [3] is an advanced framework to build simulation systems, especially in virtual reality (VR) and augmented reality (AR) fields. Due to the high computation cost of rendering real-time animation, performance optimization plays an important role in Unity 3D software development [4]. The main rendering process in Unity 3D is similar to that described in figure 1, for each frame, the specific process at object level is presented in Figure 2 as follows [5]:

(1) CPU

- *Step C1. Create frame object list:*

Analyze the scene to be rendered and make a list of all objects in the frame (list L1) without paying attention to objects that are obscured or out of view.

- *Step C2. Culling:*

Divide the scene into bounding boxes with dimensions determined by Unity 3D. If a bounding box is outside the field of view (FOV), all its objects will be removed from the list L1. If a bounding box is inside or intersects with the FOV, all its objects will be kept in the list L1. In addition, objects also will be removed from list L1 if they are not in the layer the camera chooses to display or be obscured by other objects. The result of this

step is list L2 (called culling data).

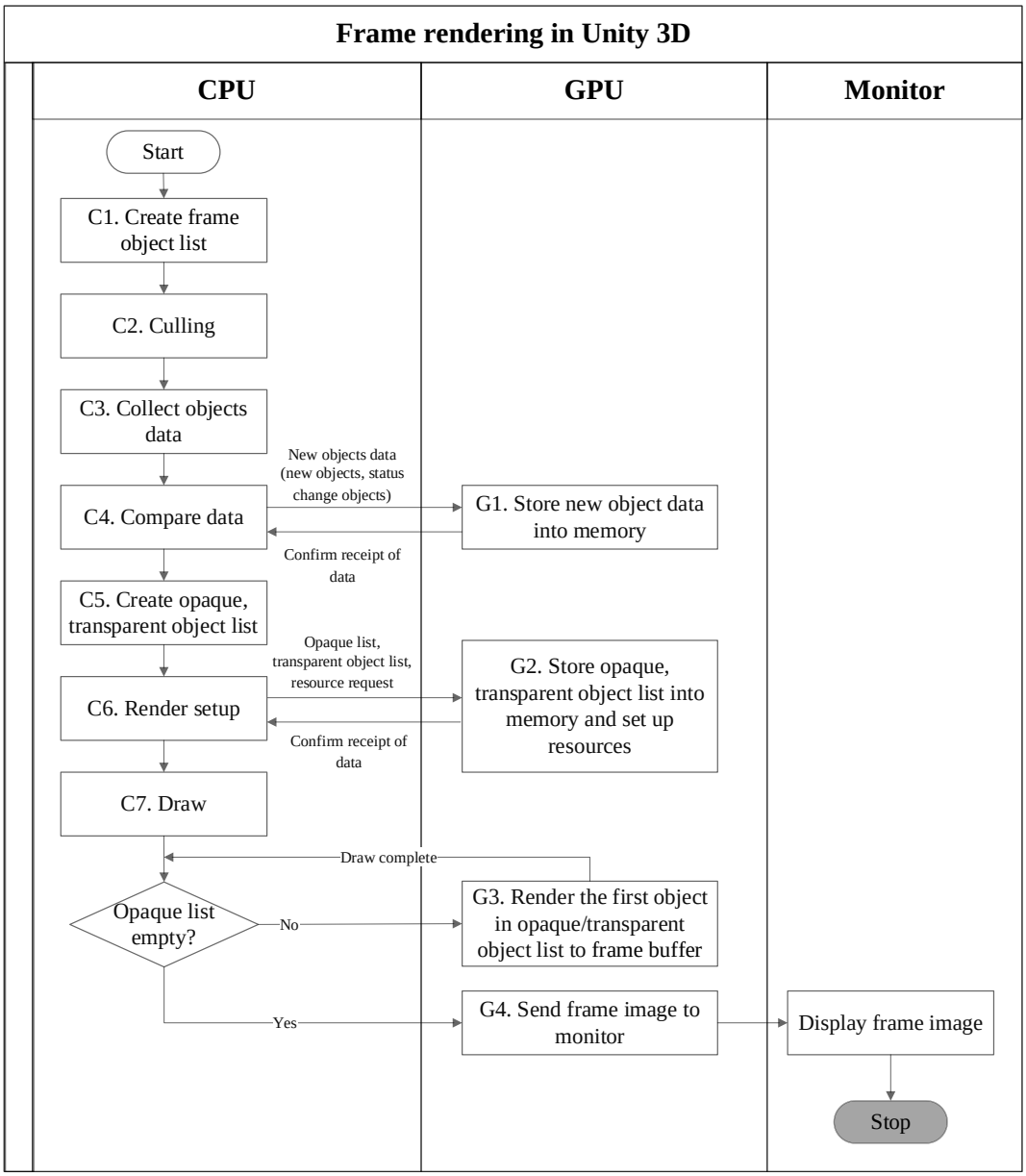


Figure 2. *Frame rendering process in Unity 3D.*

- *Step C3. Collect objects data:*

All information of selected objects in L2 such as mesh (skeleton structure), material including textures, shaders, position, color, mapping,... is collected and stored by the CPU.

- *Step C4. Compare data:*

Collected object data will be compared with the corresponding data of the previous frame (in case the current frame is not the first frame). If there is a change to the data (for example, a new object is added or the state of the object changes), the changing part of data will be send to the GPU.

- *Step C5. Create opaque, transparent object list:*

All objects in L2 will be divided into opaque object list (L3) and transparent object list (L4) based on material, light, distance, etc. Opaque object can obscure objects behind it in the direction of observation and vice versa transparent object can let light through so objects behind it in the direction of observation can be observed. In this step, in addition, information about GPU resources to use for rendering will be calculated. For opaque objects, the closer to the viewpoint will be put in the top of the list and and vice versa for transparent object.

- *Step C6. Render setup:*

The rendered objects will be determined, the index of objects in list L3, L4 and information about GPU usage sent to the GPU.

- *Step C7. Draw:*

With opaque object list (L3) (organized by stack), the CPU sends additional information on the first object in the list and a draw command to the GPU, removes that object from the list. This process will be repeated when the CPU receives the message that the GPU has finished rendering the object and continues until there are no more objects in the list.

For the transparent object list (L4), the execution is similar to the L3 and starts at the time when L3 has been processed completely.

When all the objects in the two lists have been rendered, the CPU sends a finish signal to the GPU to notify that it is ready to move on to the next frame.

(2) GPU

- *Step G1. Store new object data into memory:*

Receive new object data from the CPU (step C4), store in GPU's memory and notify the CPU that data has been received.

- *Step G2. Store opaque, transparent object list into memory and set up resources:*

Receive index of objects in opaque object list, transparent object list and GPU resources information from the CPU (step C5), store in GPU's memory and notify the CPU that data has received. Based on received information, GPU does some setup such as preparing memory, buffer,…

- *Step G3. Render the first object in opaque/transparent object list to frame buffer:*

Render object specified by the CPU to frame buffer.

- *Step G4. Send frame image to monitor:*

After receiving the finish signal from the CPU, GPU send frame image in frame buffer to the monitor to display.

2.2. Analysis

In the frame rendering process of Unity 3D, both the CPU and the GPU must complete all tasks to render the frame. Therefore, if a task takes a long time to complete, it will cause a delay in rendering the frame, thereby reducing the FR of the application.

Look detail at the frame rendering process in Unity 3D, we can see that one of the most effective optimizations is culling at step C2 because the smaller the number of

rendered objects, the less CPU and GPU resources are used and the higher FR can get. Three are three types of visibility culling methods frustum, occlusion, and back-face, respectively [6] as depicted in Figure 3:

- ***Frustum culling method*** (sometimes called view - frustum culling) is used to avoid rendering object that is outside the viewing frustum (VF);

- ***Occlusion culling method*** aim at avoiding rendering primitives that are occluded by some other part of the scene;

- ***Back-face culling method*** avoid rendering geometry that faces away from the viewer rendering geometry that is outside the VF.

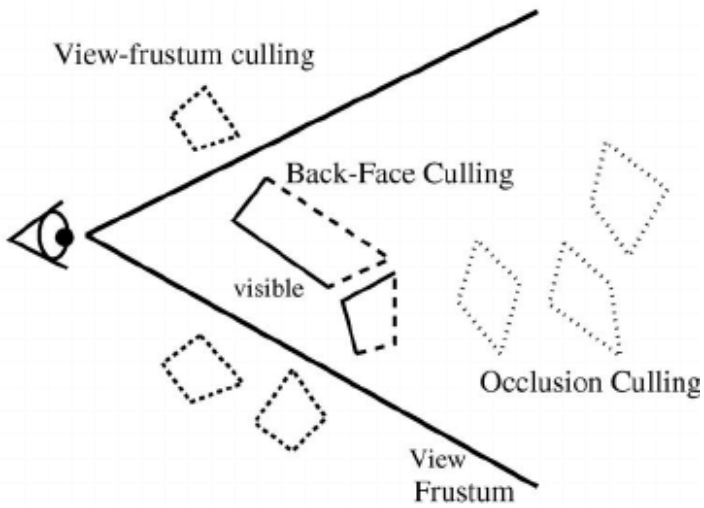


Figure 3. *Types of visibility culling method.*

The latest version of Unity 3D provides tools for all three above culling methods, however, this study focus on the frustum culling method, so we will analyze Unity 3D frustum culling in detail. The main idea of the frustum culling in Unity 3D [7] is illustrated in figure 4. Although the scene space in Unity 3D is three-dimensions (3D) but to make the idea easy to understand, we describe in two-dimensions (2D):

- Divide the scene into bounding boxes with dimensions determined by Unity 3D: in the illustration, we suppose that the grid is divided into 20 bounding boxes (4 rows and 5 columns) numbered by row and column;

- With each bounding box:

+ If a bounding box is outside the VF: all its objects will be removed from the render objects list (culling data). As shown in figure 4, those boxes are 1.1, 1.5 (out of view angle), 4.1 4.2, 4.3, 4.4, 4.5 (out of view distance) and all of their objects (denoted as dashed outline circles) will be removed from the render objects list.

+ If a bounding box is inside the VF: all its objects will be kept in the culling data. As shown in figure 4, those boxes are 2.3, 3.2, 3.3, 3.4 and all of their objects (denoted as filled circles) will be kept in the render objects list.

+ If a bounding box intersects with the VF: all its objects will be kept in the culling data. As shown in figure 4, those boxes are 1.2, 1.3, 1.4, 2.1, 2.2, 2.4, 2.5, 3.1, 3.5 and all of their objects (denoted as solid outline circles) will be kept.

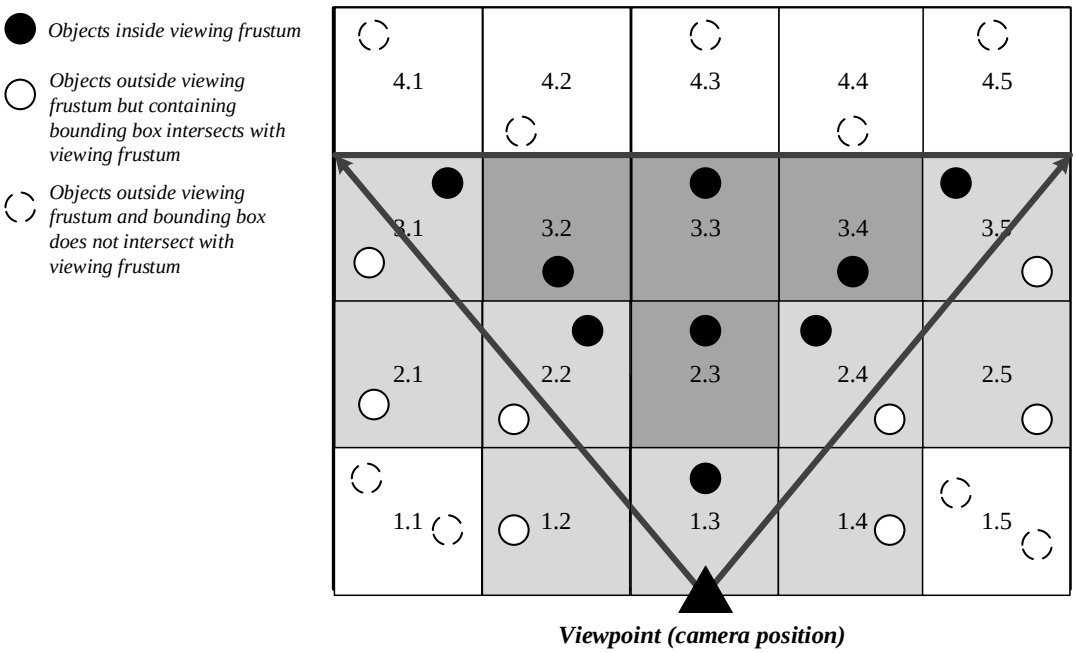


Figure 4. *Frustum culling in Unity 3D.*

As can be seen, the original frustum culling method in Unity 3D takes not only objects inside the VF but also outside (in case of bounding box intersects with the VF). As illustrated in figure 4, only objects denoted as filled circles inside the VF, objects denoted as solid outline circle outside the VF but still taken. This leads to object redundancy, therefore, the FR will be lower and need higher CPU and GPU resources to render. The reason is frustum culling in Unity 3D performed at bounding box level (grid level) and not the object level.

2.3. Related works to solve the problem

To handle the drawback in the Unity 3D culling process, some research works have been suggested and some notable studies include:

- H.C. Batagelo and al [8] proposed an occlusion culling algorithm for densely occluded dynamic scenes based on a regular grid that uses opaque regions of the scene as virtual occluders. Besides the efficiency of representing visible dynamic objects and temporal bounding volumes, the benefits of using regular grids are strengthened by novel methods of view-frustum traversal and occlusion computation based on raster principles. In addition, the algorithm fuses occluders in object space in order to increase occlusion size and is output-sensitive. According to the timing tests, the overhead due to the handling of hidden dynamic objects is very low for most scenes.

- Ziyi Wang and al [9] presented a rendering algorithm for large-scale complex scenes based on Unity. This algorithm is applied to large-scale factory scene rendering in real-time. The main idea of this study was to combine the frustum culling and occlusion culling methods. The algorithm can be divided into two stages: preprocessing (or compiling) and runtime. The preprocessing stage focuses on frustum culling by generation of a database of occlusion from the scene database, and the data of the scene database and the occlusion database are organized by the spatial data structure of the

bounding volume level (the same as in Unity 3D). The runtime stage concentrates on occlusion culling, a LOD algorithm is used to select the appropriate level of detail for the visible objects. Experiments showed that the combination method can guarantee the rendering quality, real-time rendering of large-scale factory scene than a single and have different degrees of improvement in the efficiency of rendering, and the scene is more complex, more can reflect the efficiency of the algorithm.

- Rodriguez and al [10] suggested a dynamic culling using octrees with Unity for virtual reality. An Octree is a tree data structure that divides 3D space into smaller partitions (nodes) and places objects into the appropriate nodes. This allows fast access to objects in an area of interest without having to check every object. This study focuses on occlusion culling of dynamic objects by eliminating unnecessary objects whose view is obstructed by other objects in the forefront. Although it is different from our research direction, this research helps us orient our future development.

3. METHODOLOGY

This study proposes a novel method for accelerating FR in Unity 3D. The main idea is to eliminate all objects outside the VF or FOV of observer. Unlike the origin frustum culling method in Unity 3D, our method is implemented at the object level and consists of only two types of objects which are inside object and outside object.

- Inside object: an object is within range of the VF and chosen to put in the rendering list. Look at Figure 5, the objects in the white color region;

- Outside object: an object is out of range of the VF and eliminated from the rendering list. See the detail in figure 5, the objects in the gray color region.

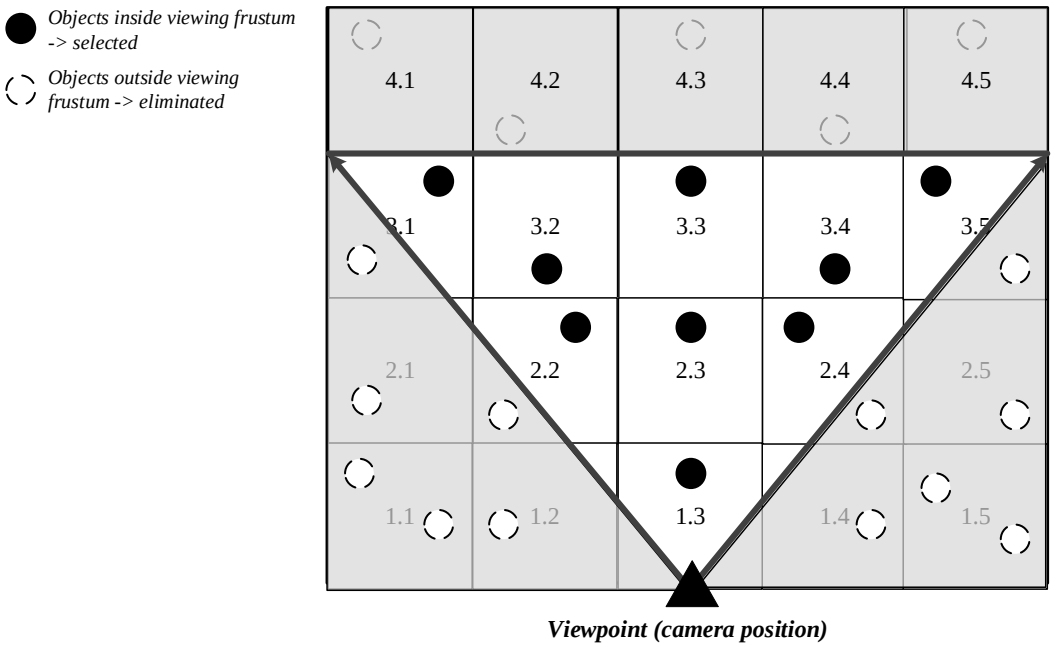


Figure 5. *Frustum culling in this research.*

With this method, it is possible to avoid redundant objects so which significantly reduces the number of objects in the rendering list and therefore leads to lower GPU usage. With CPU usage, the calculation cost at step 2 in frame rendering process will

increase because of the need to process each object, however, the cost of calculation at the following steps will be lower because of the number of objects will be significantly reduced.

To clarify the difference between the origin Unity 3D frustum culling method and our method, we have a summarization in table 1.

Table 1. *Idea comparing between origin Unity 3D frustum culling and our method.*

Case	**Origin Unity 3D frustum culling**	**Our method**	**Note**
	Unity 3D origin	Our method	
a) Objects inside the VF	Put into rendering list	Put into rendering list	Same
b) Objects outside the VF			
- *Objects contained in bounding box which intersect with VF*	*Put into rendering list*	*Eliminate from rendering list*	*Difference*
- Objects contained in bounding box which does not intersect with the VF	Eliminate from rendering list	Eliminate from rendering list	Same

The implementation process of proposed method as follows:

(1) Bounding box division

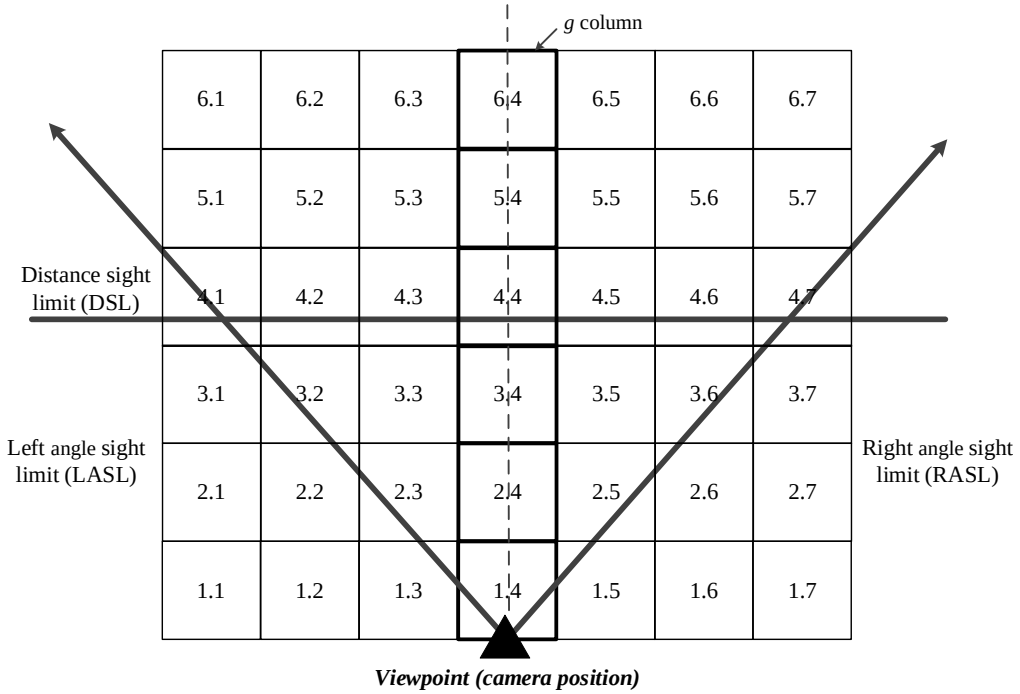


Figure 6. *Illustration for the proposed method implementation.*

Divide the scene into bounding boxes (same as in Unity 3D), and suppose that we have a bounding box grid of *M* row x *N* column with *K* bounding boxes (in figure 6, *M* =

6, N = 7 and *K = 42*). Number bounding boxes from 1 to *M* by the view direction (from near to far) and columns from 1 to *N* by the view angle (from left to right).

(2) Visibility evaluation

Based on the viewpoint, determine which bounding box is inside or intersects with the VF by distance sight limit (DSL).

- Determine the index of column (denoted *g*) which align with the viewpoint (column 4 in figure 6).

- Perform iteration by row index from row 1:

+ Calculate the farthest distance from the viewpoint to the bounding boxes in *g* column:

+ If the farthest distance is less than DSL: Add all bounding boxes of the current row to the list of bounding boxes inside the VF (*BLi*) and move to the next row (rows 1, 2, 3 in figure 6);

+ If the farthest distance is greater than DSL: calculate the nearest distance from the viewpoint to the current bounding box. If it is less than DSL (row 4 in figure 6) then add all bounding boxes of the current row (denoted *h*) to the list of bounding boxes those intersect with the VF (*BLc*) and stop the iteration. So, we do not need to consider rows with an index greater than *h* (rows 5, 6 in figure 6).

(3) View angle evaluation

Based on the viewpoint, determine which bounding box is inside or intersects with the VF by the angle sight limit (ASL).

a) Consider columns on the left of *g* column

- From column *g - 1* back to column 1:

From row 1 to row *h* (the row that intersects with the VF in the previous step):

+ If the left angle sight limit (LASL) intersect with the current bounding box then move the current bounding box from *BLi* to *BLc* and switch to the left bounding box (in figure 6, they are 1.3 in row 1, 2.2 and 2.3 in row 2, 3.2 and 3.1 in row 3, 4.1 in row 4);

+ If LASL does not intersect with the current bounding box, then delete the current bounding box and all bounding boxes on the left of the current bounding box from *BLi* list (in figure 6, they are 1.1 and 1.6 in row 1, 2.7 in row 2).

b) Consider columns on the right of *g* column

- From column *g + 1* to column *N*:

From row 1 to row *h* (the row that intersects with the VF in the previous step):

+ If the right angle sight limit (RASL) intersect with the current bounding box then move the current bounding box from *BLi* to *BLc* and switch to the right bounding box (in figure 6, they are 1.5 in row 1, 2.5 and 2.6 in row 2, 3.6 and 3.7 in row 1, 4.7 in row 4);

+ If LASL does not intersect with the current bounding box then delete the current bounding box and all bounding boxes on the right of the current bounding box from *BLi* list (in figure 6, they are 1.2 and 1.1 in row 1, 2.1 in row 2).

(4) Objects taking

- Bounding boxes in *BLi* list: take all objects inside them to the object list to be rendered.

- Bounding boxes in *BLc* list: take only objects inside the VF to the object list to be rendered.

The calculation of the intersection of line and region, region and region, and objects in the region using conventional geometric algorithms.

4. EXPERIMENTS, RESULTS AND DISCUSSION

4.1. Experiments

To evaluate the performance, experiments were conducted with the following conditions:

- Hardware: Computer with an Intel Core i7 7800X CPU, 16GB of RAM and GTX 1080Ti graphic card;

- Unity 3D framework: version 2020 3.32;

- Tools for performance evaluation: Profiler Window, Rendering Statistic and Frame Debugger.

- Terrain database: one kilometer square with 30,000 tree type objects (random planted);

- Viewpoint: camera is placed in the middle, 8.9m high, straight angle.

4.2. Results and discussion

In order to illustrate the performance of our proposal, with Unity 3D original method and the given method, we implemented ten times and recorded the CPU usage, the GPU usage and the FR value. Results are presented in table 2.

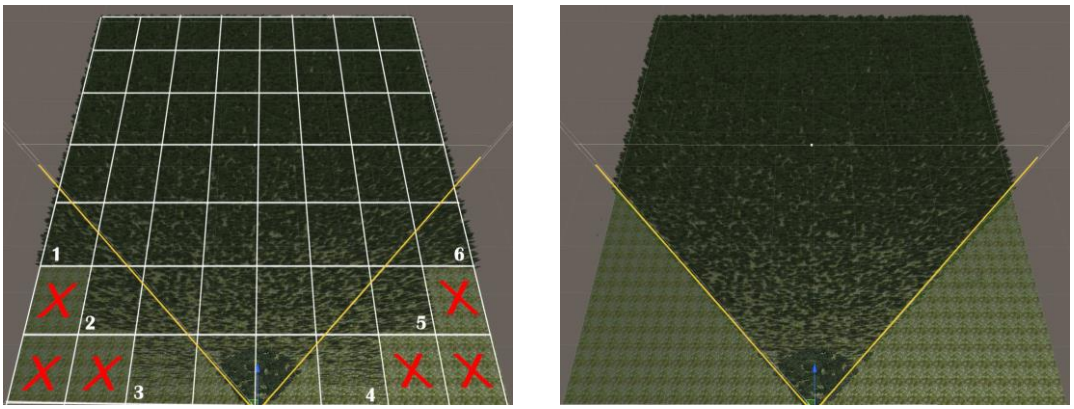
Table 2. *Performance evaluation.*

Times	**Unity 3D original method**			**Our method**		
	CPU usage (ms)	GPU usage (ms)	FR (FPS)	CPU usage (ms)	GPU usage (ms)	FR (FPS)
1	36.0	18.0	27.6	24.8	9.1	40.1
2	35.8	17.9	27.4	24.6	9.0	39.9
3	36.4	18.2	27.9	25.2	9.3	40.4
4	36.6	18.4	28.1	25.4	9.5	40.6
5	35.7	17.8	27.5	24.5	8.9	40.0
6	36.3	18.3	28.0	25.1	9.4	40.5
7	36.0	17.9	27.6	24.8	9.0	40.1
8	35.9	18.0	27.6	24.7	9.1	40.1
9	36.2	18.2	27.9	25.0	9.3	40.4
10	35.9	17.9	27.7	24.7	9.0	40.2
Average	**36.1**	**18.1**	**27.7**	**24.9**	**9.2**	**40.2**

The obtained results showed that, in the case of the experimental terrain database, FR in Unity 3D original method was 27.7 FPS on average. FR in our method was 40.2 FPS on average, which is significantly higher than the original method and up to standard.

For visualization, we took a screenshot of the rendering results and described in figure 7. It is clear that in the image *(a)*, objects in cross marked bounding boxes are not rendered because the bounding boxes do not intersect with the view culling, however,

objects in bounding boxes numbered from 1 to 6 are still rendered and displayed in image because the bounding boxes cross the view culling. In our new method, all objects outside the view culling were eliminated from culling data and therefore did not show on the image *(b)*.



a) Unity 3D original method *b) Proposed method*

Figure 7. *Experimental results.*

5. CONCLUSIONS

In this paper, we presented an advanced method to accelerate FR to enhance the performance of terrain database rendering in the Unity 3D framework. Based on the analysis of weaknesses in the Unity 3D process, we improved the frustum culling by execution at the object level. The experiments have shown that the method has good performance and efficiency. The research results have been applied in KC.BM.04 [11], the KC-KT.04/19 [12], and the ĐTVCN.01.22/CNTT [13]. In the future, we will combine this method with other occlusion culling and back-face culling techniques to make the culling process more efficient not only in terrain database rendering (compile stage) but also runtime stage of simulation applications built in the Unity 3D framework.

REFERENCES

[1]. B. Wang and P.-L. P. Rau, *"Effect of vibrotactile feedback on simulator sickness, performance, and user satisfaction with virtual reality glasses",* In International Conference on Human-Computer Interaction, Springer, pp. 291-302, (2019).
[2]. https://developingschool.com/a/1/what-is-frame-rate-and-factors-affecting-it/.
[3]. https://unity.com: Unity Real-Time Development Platform | 3D, 2D VR & AR.
[4]. F. Nusrat, F. Hassan et al, *"How Developers Optimize Virtual Reality Applications: A Study of Optimization Commits in Open Source Unity Projects"*, IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp. 473-485, (2021).
[5]. https://unity.com/how-to/real-time-rendering-3d.
[6]. Cohen-Or, Daniel & Chrysanthou, Yiorgos & Silva, Cláudio, *"A Survey of Visibility for Walkthrough Applications"*, Proceedings of SIGGRAPH, (2001).
[7]. https://docs.unity3d.com/Manual/OcclusionCulling.html.
[8]. Batagelo, H. C., & Wu Shin-Ting, *"Dynamic scene occlusion culling using a regular grid"*, Proceedings XV Brazilian Symposium on Computer Graphics and Image Processing, (2002), doi:10.1109/sibgra.2002.1167122.
[9]. Ziyi Wang, Meng Tan, Gang Liu, *"Research on Real Time Rendering Algorithm for Large Scale Complex Scenes based on Unity"*, International Journal of Science, Vol.4, No.4,

pp.14-18, (2017).
[10]. Rodriguez, *"Dynamic occlusion culling using octrees with Unity for virtual reality"*, The University of Texas at San Antonio, (2017).
[11]. Binh Do Viet, *KC.BM.04*, Military Information Technology Institute, (2017).
[12]. Khanh Hoang The, *KC-KT.04/19*, Missle Institute, (2019).
[13]. Minh Tran Binh, *ĐTVCN.01.22/CNTT*, Military Information Technology Institute, (2021).

TÓM TẮT

Phương pháp tăng tỷ lệ khung hình dựa trên lọc hiển thị ở mức đối tượng

Tỷ lệ khung hình là một tham số quan trọng để đánh giá hiệu năng của các ứng dụng mô phỏng, nếu càng cao thì càng đem lại sự trực quan và chân thực cho các hệ thống mô phỏng. Trong lĩnh vực mô phỏng, quá trình sinh cảnh dữ liệu địa hình chiếm nhiều thời gian và tài nguyên phần cứng, do đó, tối ưu hóa quá trình này sẽ làm cho cả hệ thống mô phỏng được tăng tốc đáng kể. Phương pháp lọc hiển thị theo khung nhìn rất hiệu quả trong việc giảm tính toán các đối tượng không được hiển thị trong khung hình. Trong phương pháp này, chỉ các đối tượng được hiển thị và không được hiển thị sẽ được phân loại và chỉ có đói tượng hiển thị mới được gửi đến cạc đồ họa để kết xuất đồ họa. Bài báo đề xuất một phương pháp lọc hiển thị khung hình đến mức đối tượng. Trong phương pháp đề xuất, vùng kết xuất đồ họa được lọc theo đối tượng, do đó, làm giảm đáng kể số lượng đối tượng cần kết xuất đồ họa trong mỗi khung hình. Mục đích của nghiên cứu là cải thiện hiệu năng kết xuất đồ họa dữ liệu địa hình trên nền tảng Unity 3D. Các kết quả thực nghiệm cho thấy, phương pháp đề xuất đạt được kết quả tốt so với khi sử dụng phương pháp gốc.

Từ khóa: Tỷ lệ khung hình; Chọn lọc; Mô phỏng; Cơ sở dữ liệu địa hình; Kết xuất đồ họa; Unity 3D.