Research on Unity Scene Optimization Based on Fast LoD Technique

Performance Comparison on Android Mobile Platform

Xin Yang
College of Computer
Qinghai Normal University
Xining, China
loongxin@yeah.net

Jindong Liang
College of Computer
Qinghai Normal University
Xining, China
1356673614@qq.com

*Ping Xie
College of Computer
Qinghai Normal University
Xining, China
* Corresponding author: 272984903@qq.com

Yanping Wu
College of Computer
Qinghai Normal University
Xining, China
1291029148@qq.com

***Abstract*—With the completion of Unity scene development, it is often necessary to publish the scene on all platform for different user communities to operate. However, due to the insufficient hardware performance of Android mobile devices, in order to provide players with a smoother operation experience, developers will optimize the scene before final release. This article elaborates on the process of optimizing models in a certain scene through fast LoD technique based on QEM algorithm. It demonstrates that simplifying mesh grids of models can effectively improve the smoothness of scene operation and reduce device resource consumption. Through comparative analysis, it is concluded that the fast LoD technique can achieve higher optimization efficiency and better optimization effects than common LoD technique, and it has brought better performance and experience improvement to scene operation.**

Keywords-Android mobile terminal; QEM algorithm; mesh simplification; LoD techniques; Unity performance optimization

I. INTRODUCTION

Some large-scale virtual reality content or application designs can make players feel immersive. The design of objects, characters, trees, flowers and plants in the scene is just like in the real world, and players will be fascinated by it. This is because application design achieves the extreme in both scene object modeling and scene rendering [1]. But at the same time, it also brings many drawbacks to application during operation, such as high device resource consumption, 3D dizziness caused by slow scene operation, excessive volume of application installation package, and so on. Some applications [2], in order to cater to more comprehensive players, will be published on all platforms, such as PC and mobile devices. Especially after publishing Unity project [4] on Android mobile devices, due to the inherent hardware performance issues of them, the experience after scene publishing will be much lower than that of PC devices. Therefore, it is very important to optimize the scene after application development and before official release. The optimization of scene models is mainly achieved through LoD multi-levels of detail technique, but common LoD technique has problems such as high implementation complexity, large workload, and inconvenient management and maintenance. The fast LoD technique based on QEM mesh simplification algorithm proposed in this article can efficiently optimize models in the scene. While improving LoD optimization efficiency, it brings better performance and experience improvement to scene operation.

This work is supported by The Key R&D and Transformation Project of Qinghai Province under Grant 2022-SF-165.

II. RELATED TECHNOLOGIES

The essence of LoD optimization [8] is actually the process of converting mesh [3] of an object from one form to another by controlling distance between camera and the object. Specifically, when the distance between camera and the object is closer, the mesh will have more vertices, edges and triangles. As the distance between camera and the object increases, vertices, edges and triangles of the mesh gradually decrease, and eventually disappear within camera's range. The key techniques involved in LoD optimization process mainly include LoD, vertex clustering, edge collapse, QEM mesh simplification algorithm [5], DrawCall, and so on.

A. LoD

Levels of Detail (LoD) is a means of flexibly displaying the precision of object models by controlling the distance between camera and objects. Under normal circumstances, the human eyes cannot see distant objects (such as mountains or vegetation) very clearly, and in virtual reality scenes, the camera represents the human eyes. The farther the camera is from the object, the less detail enters the scene rendered by the camera. On the contrary, the closer the camera is to the object, the more details will enter the scene rendered by the camera. If there are fewer scene elements that can be rendered in the camera at once, the virtual reality application will operate smoother and consume fewer device resources. LoD achieves

2023 3rd International Symposium on Computer Technology and Information Science (ISCTIS) | 979-8-3503-2538-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISCTIS58954.2023.10213077

the classification of object models, and display details in the scene mainly through lighting, textures and triangles.

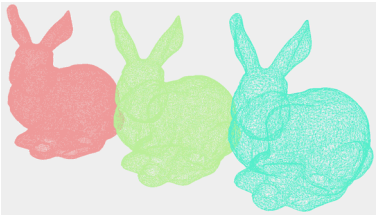


Figure 1. Comparison of Mesh Details at Different LoD Levels

TABLE I. COMPARISON OF THE NUMBER OF MESH VERTICES AND TRIANGLES AT DIFFERENT LOD LEVELS

LoD Levels	*Vertices*	*Triangles*
LoD0	69664	34834
LoD1	34832	17418
LoD2	15060	7532

Among them, the number of triangles in a model determines the amount of scene details that the camera will render. Three different precision models can be prepared for the same object: high, medium, and low. These three models have different segmentation values set in the modeling software, which sequentially reduces the precision of the model. When rendering with a Unity camera, the object is closer to the camera, it displays high resolution and looks very clear. After being away from a certain distance, the object displays a medium mode, slightly blurring. If the low mode of the object is displayed at a long distance, and even the object will disappear directly from the scene, this is very consistent with actual situation of observing objects in the real world. When there are many objects in the scene, LoD can greatly improve the smoothness of scene operation, ensuring a good operation experience [9]. As shown in Fig. 1, the LoD0 level model mesh in the scene has higher accuracy and priority, with more vertices, edges, triangles and details. However, the LoD1 level model mesh has been simplified due to its distance from the camera, resulting in fewer vertices, edges, triangles and details. The LoD2 level mesh becomes simpler, even after a certain distance from the camera, it will disappear directly from the camera's range.

Comparison of the number of vertices and triangles of the same object at the LoD0, LoD1 (precision reduced to 50 percent), and LoD2 (precision reduced to 20 percent) levels is shown in Table 1.

B. *Vertex Clustering*

The methods for model mesh optimization can be roughly divided into two categories, namely vertex clustering and edge collapse.

Vertex clustering [10], also known as vertex merging. Its main idea is to use a cube bounding box to wrap the entire model, and then divide the bounding box into many small cube regions.

Finally, vertices that fall in the same region will be merged. Vertex clustering can be processed starting from the top down or bottom up tree hierarchy. As shown in Fig. 2, in model A and model B, a new green vertex will be generated at the center point of the small cube by clustering n ($n \geq 1$) vertices located in the same small cube area.

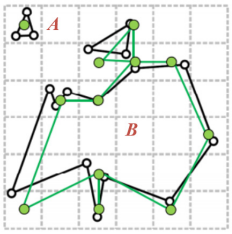


Figure 2. Vertex Clustering Processing of Model.

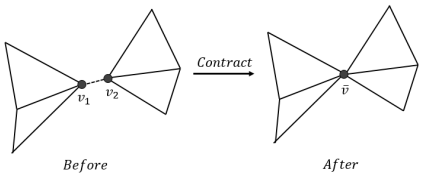


Figure 3. Edge Collapse Processing of Model.

The principle of vertex clustering is relatively simple and runs fast, but it can easily cause the triangles of key parts of a model to cluster into an edge or even a vertex, resulting in the clustered model being unable to maintain its previous shape, and thus the model mesh may be changed.

C. *Edge Collapse*

Due to the drawbacks of vertex clustering methods, the mainstream approach in mesh optimization is edge collapse technique. Edge collapse is a method of achieving incremental simplification by collapsing triangular edges to simplify the mesh, specifically by combining two vertices v_1 and v_2, merge into a new vertex $\bar{v}$, as shown in Fig. 3. It can be seen that one collapse can reduce two triangles, three edges, and one vertex. In the edge collapse algorithm, through continuous iteration, the model mesh finally can be simplified to the number of target triangles we expect after simplification.

If we want to avoid the problems of vertex clustering, we must find the minimum cost required to remove specified edges. The cost here refers to collapse cost between v_1 and v_2, also known as the *error*. Calculate collapse cost can take the following two factors into consideration. The first is the effectiveness of edges, during the process of edge collapse simplification, if the length of edges (v_1, v_2) is less than a certain threshold, then the edge is considered an effective edge. The existence of edges that meet this requirement does not affect the overall mesh topology of a model, so they can be collapsed. The second is that during the collapse process, the curvature around a vertex will change. In theory, the smaller the curvature change, the flatter the region the vertex is located in, and the edges related to the vertex can be collapsed. But the coordinate of the new vertex $\bar{v}$ generated after collapse need

to be located as soon as possible to ensure the mesh topology will not be affected.

D. QEM Simplification Algorithm

The QEM simplification algorithm [6] is an algorithm that simplifies 3D model meshes, and its core idea is based on the iterative contraction between v_1 and v_2, in order to find the point with the smallest contraction cost during iteration process, is defined as the target vertex $\overline{v}$ after contraction. This algorithm proposes an edge collapse algorithm based on *quadratic error matrix* as the measurement cost, which has fast calculation speed and high simplification quality. The workflow of this mesh optimization algorithm is explained as follows.

1) Calculate their respective Q matrices for all initial vertices.

2) Select all valid edges in a model mesh, and select edge which connects lines between v_1 and v_2, or an edge with a connecting line between v_1 and v_2, and distance less than a certain threshold, denoted as (v_1, v_2).

3) For each effective edge (v_1, v_2), Q_1 and Q_2 are the symmetric error matrices of vertices v_1 and v_2, respectively.

4) Sort all edges in a queue from smallest to largest based on the weight value of contraction cost $\Delta\overline{v}$.

5) Remove the edge with the smallest collapse cost $\Delta\overline{v}$ at the top of the queue each time, and delete v_1 and v_2, replacing them with a new vertex $\overline{v}$.

6) Repeat steps 1-5 until the number of vertices is less than threshold, or all collapse costs $\Delta\overline{v}$ are greater than certain value, then stop the collapse algorithm. At present, the mesh optimized by edge collapse is the target.

E. DrawCall and SetPassCall

During a rendering process, Unity engine first prepares data and then notifies the GPU for processing, which is called a DrawCall. This operation requires processing every object in the scene, not only for GPU rendering, but also for Unity engine to reset the material Shader. Therefore, the number of DrawCall per frame during the processing is a very important performance indicator, and for Android mobile operating systems, Unity recommends it should be controlled within 600 as much as possible.

Unity has a built-in technique called DrawCall Batching, as the name suggests, which enables batch processing of multiple objects simultaneously in a single DrawCall. As long as these objects have same transformation properties and materials, the device GPU can place them into the same DrawCall group and batch process them in exactly the same way.

A Pass semantic block in a Shader script is a complete rendering process, and a Shader can contain one or more Pass semantic blocks. Every time the GPU runs a Pass, a SetPassCall is generated, which can be understood as the number of times the complete rendering process is executed.

Figure 4. Unity Scene to Be Optimized.

III. Optimization Process

A. Optimization Idea

The fast LoD technique proposed in this article can automatically add LoD Group component to the currently selected object through running script. At the same time, it can automatically calculate mesh grids of different LoD levels for the current object based on the QEM algorithm, and save the calculated mesh grids as independent files in project path. Developers can flexibly set generated LoD levels' parameters, and if you are not satisfied with current generated LoD levels, you can delete them with just one click, and the mesh grids files saved in project path will be deleted synchronously, thereby improving the management efficiency of LoD Group.

B. Advantages of Fast LoD Technique

The fast LoD technique based on QEM simplification algorithm has the following advantages.

1) Able to quickly and automatically generate meshes of different LoD levels in Unity scenes to be optimized.

2) The source model only needs to be a high-precision one, reducing the workload of modeling for artists.

3) The goal of mesh optimization is objects in the scene actually, without making changes to original models, thus protecting original model resources.

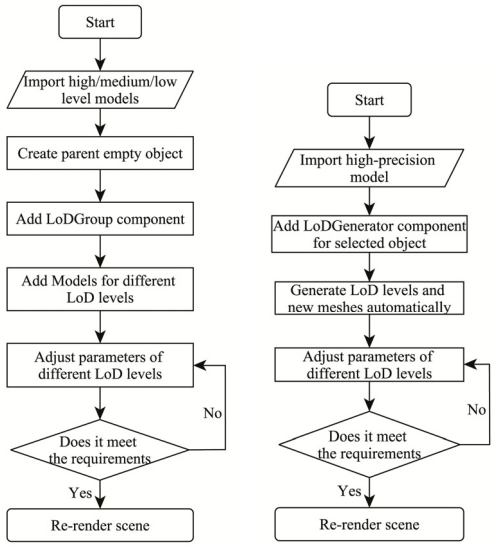
4) The generated mesh files with different LoD levels can be automatically saved, and can be managed flexibly.

This article optimizes a scene with dreamy features, and publishes the optimized scene to Android mobile platform for detailed performance evaluation. The Unity scene to be optimized is shown in Fig. 4.

C. Comparison of LoD Implementation Processes

The comparison of the implementation process between common LoD technique and fast LoD technique is shown in Fig. 5(a) and Fig. 5(b).

From the comparison between Fig. 5(a) and Fig. 5(b), it can be seen that the implementation steps of fast LoD technique are simpler, and required LoD levels can be automatically generated, and subsequent maintenance and management of them is also very convenient.



(a) Common LoD (b) Fast LoD

Figure 5. Common LoD and Fast LoD Implementation Processes.

TABLE II. EXPERIMENTAL HARDWARE ENVIRONMENT PARAMETERS

Hardware Type	***Parameters***
CPU	Cortex-A78/A55(2.0GHz)
Memory	8GB
Hard Disk	512GB
Graphics Card	ARM Mali-G610 MC6
OS	Android 11

IV. PERFORMANCE EVALUATION

A. Experimental Setup

This article mainly compares FPS [7] frame count, device resource consumption (including CPU usage, GPU usage, memory usage, rendering situation, etc.), and installation package size after the Unity scene export in the case of unoptimized and optimized through common LoD and fast LoD techniques. The experimental platform is Android mobile terminal, hardware environment parameters are shown in Table 2.

B. Performance Evaluation Topology

In order to ensure stability during the performance evaluation process, Unity development host should install the desktop version evaluation tool *UPR Desktop*, and then connects to the Android terminal after installing the application through a USB cable with the Android Debug Bridge (ADB) wired method. This is a connection method for debugging apps in Android terminal, while ensuring the reliability of the connection between Android terminal and development host. However, Android terminal needs to enter developer mode and enable USB debugging function. The performance evaluation topology is shown in Fig. 6.

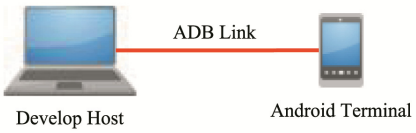


Figure 6. Performance Evaluation Topology.

C. Evaluate Device Resource Consumption

Create a new evaluation project in the Unity UPR platform, use *UPR Desktop* to capture performance data during operation of Android app, and evaluate the consumption of device resources within 1000 frames without optimization, using common LoD and fast LoD techniques.

D. Evaluate Smoothness of Scene Operation

Add a script for counting the number of frames during the operation of the scene, and analyze the changes in the average number of frames without optimization and using two LoD optimization techniques. The higher the number of frames, the smoother the scene operates on Android mobile terminal.

E. Comparison of Installation Package Size

Export the scene to Android platform and compare the size of installation package without optimization and with two types of LoD optimization techniques. This determines the amount of storage space occupied by the app installed in Android mobile terminal. Therefore, the smaller the optimized installation package, the better.

V. RESULTS AND ANALYSIS

A. Memory Usage

Reserved memory is actual physical memory allocated by Unity engine during project running, while used memory is the specific memory size in the reserved memory occupied during scene running. As shown in the Fig. 7, it can be seen that the reserved memory before optimization and after optimization are basically the same, approximately 672.35MB. For used memory, after optimization, its value significantly decreased, approximately 2MB lower than before.

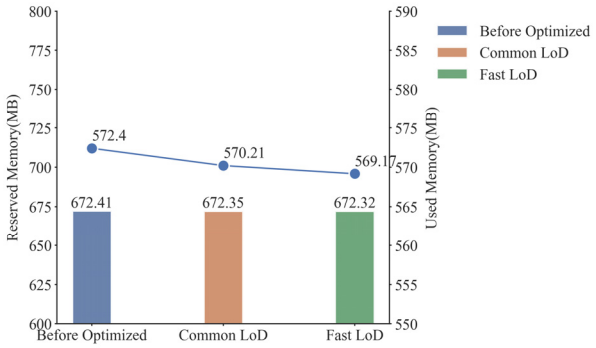


Figure 7. Comparison of Memory Usage.

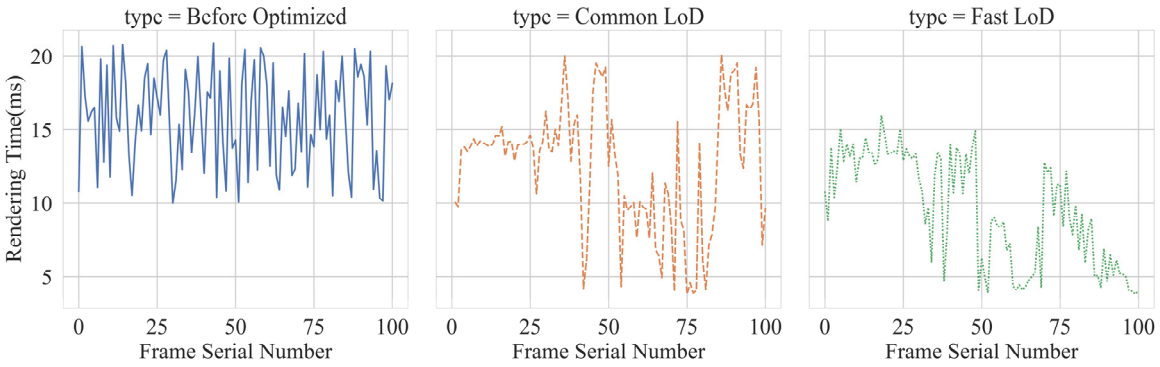


Figure 8. Comparison of Rendering Time.

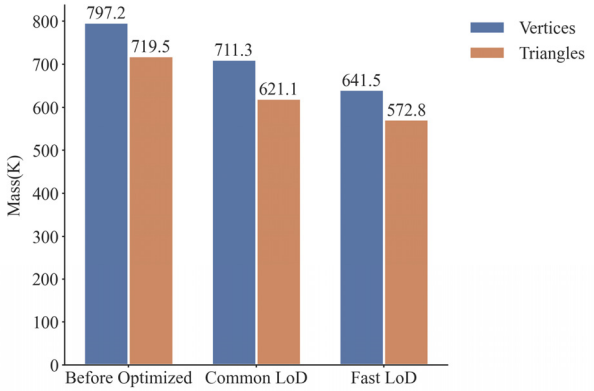


Figure 9. Comparison of Vertices and Triangles.

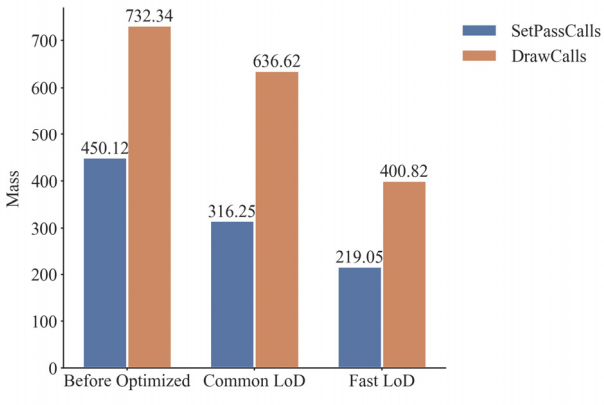


Figure 10. Comparison of DrawCalls and SetPassCalls.

B. CPU Performance

This article only discusses how models optimization can improve scene operation experience. The more important parameter in CPU time is rendering time. Using frame serial number to draw a line chart of rendering time, as shown in the Fig. 8, it can be concluded that the optimized value is significantly lower than that before optimization, and the rendering time is the smallest after fast LoD optimization.

C. Number of Vertices and Triangles

During the optimization process, the meshes of models in the scene were simplified through QEM algorithm. Therefore, after optimization, the average number of vertices and triangles in the scene will be significantly reduced, the comparison is shown in Fig. 9.

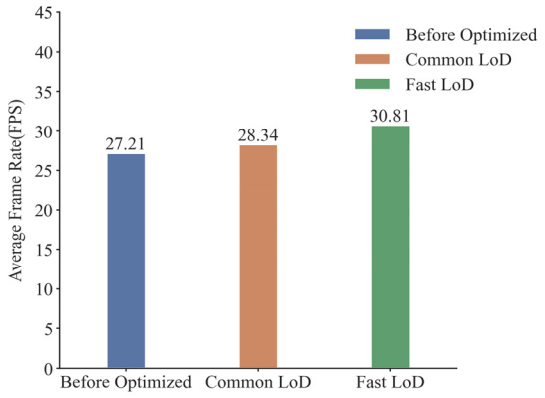


Figure 11. Comparison of Average Frame Rate.

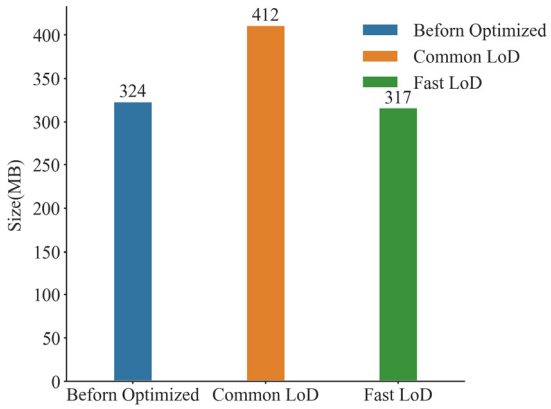


Figure 12. Comparison of Package Size.

D. Graphic Processing DrawCalls and SetPassCalls

After fast LoD optimization of the scene, due to different distances between different objects in the range of camera, the number of DrawCalls and SetPassCalls handled in one rendering process is different. As shown in the Fig. 10, it is obvious that the average values of DrawCalls and SetPassCalls after optimization have a significant downward trend, and the number of DrawCalls and SetPassCalls during the rendering process is even smaller after using fast LoD technique optimization.

E. Smoothness of Scene Operation

The smoothness of scene operation is mainly reflected by the average number of frames (FPS). As shown in the Fig. 11, before optimization, the scene has a little jam. However, after optimization by common LoD technique, the average frame rate of scene operation has been improved to a certain extent. After optimization by fast LoD technique, it can be seen that the scene frame rate has reached 30 frames. For Android mobile terminals, the scene operation is very smooth.

F. Installation Package Size

The package size of the scene before optimization is 324MB, as shown in the Fig. 12. When optimize the scene through common LoD technique, the need to create different levels of lower precision models for the same object in the scene increases the number of model resources. Therefore, the exported package size will increase to 412MB. However, if fast LoD technique is used for optimization, instead of increasing the number of model resources, only mesh files are generated based on the current model. Mesh files belong to the category of materials and basically do not occupy disk space. Therefore, the optimized scene's installation package size is reduced by 7MB compared to the unoptimized one.

VI. Conclusion

The fast LoD technique based on QEM algorithm can indeed achieve higher optimization efficiency and effects than common LoD technique, include the size of package exported from the scene and the performance of scene operation on Android mobile terminal. However, there are still some drawbacks to fast LoD technique. For example, although models with different accuracy levels do not need to be manually created, the parameters adjustment of LoD levels still need to be manually completed. Therefore, compared to automatic LoD technique, the optimization efficiency still needs to be improved in the future. In addition, fast LoD technique can adjust LoD settings for individual objects on-demand in the scene. Compared to automatic LoD technique that sets global LoD settings in the scene, it has higher flexibility, when different objects have different requirements for LoD settings. Later, the textures, animations, audios, UIs and physical system in the scene can be further optimized, so that the scene can get a better experience in Android mobile terminal.

Acknowledgment

We would like to thank the anonymous reviewers who have helped to improve the paper. This work is supported by The Key R&D and Transformation Project of Qinghai Province under Grant 2022-SF-165.

References

[1] M. Li, and F. Han, “Overview of Virtual Reality Technology,” Software Guid, No. 06, pp. 142-144, 2010.

[2] J. Y. Liu and J. K. Yu, “Research on Development of Android Applications,” 4th International Conference on Intelligent Networks and Intelligent Systems (ICINIS). Kuming, China, pp.69-72, Dec. 2011.

[3] H. Hoppe, “Progressive Meshes,” ACM SIGGRAPH Computer Graphics, vol. 30, pp. 99-108, Jan. 1996.

[4] P. Xie and X. Yang, “Introduction to Unity AR/VR Development Case Study,”. China Science and Technology Press, Xining, Qinghai, China, 2023.

[5] J. Fu, Z. S. Yu, and J. L. Zhang, “Mesh simplification algorithm based on quadratic error matrix,” Journal of Hangzhou Dianzi University, vol.33, No.04, pp. 43-45, 2013.

[6] M. Garland and P. S. Heckbert, “Surface simplification using quadric error metrics,” ACM SIGGRAPH Computer Graphics, pp. 209-216, 1997.

[7] T. Wright, E. Boria, and P. Breidenbach, “Creative player actions in FPS online video games: Playing Counter-Strike,” Game Studies, vol.2, No.2, Dec. 2002.

[8] X. G. Wei, X. Tang, and Q. Zhang, “Research on Key Technologies of Realistic Virtual Reality,” Modern Electronic Technology, vol.42, No.09, pp.111-114, 2019.

[9] Y. Y. Sun, “Research on Key Technologies of Virtual Reality Complex Scene Optimization Based on CAVE System,” Changchun University, 2021.

[10] L. K. Lim, and T. S. Tan, “Model simplification using vertex-clustering,” Symposium on Interactive 3D Graphics DBLP, pp.75-ff, 1997.