Mechatronics and Automation Technology
J. Xu (Ed.)

doi:10.3233/ATDE221162

A Method to Optimize the Performance of Three-VRM

ZhiXiang TAO[a,1], Yang HUANG[b], Xiufang QIU[a] and JingWANG[b]
[a] *School of Economics And Business College Guangdong Polytechnic Of Industry And Commerce, Guangzhou, China*
[b] *Faculty of English Language and Culture Guangdong University of Foreign Studies, Guangzhou, China*

Abstract. This paper proposes a simple and efficient VRM format display optimization scheme. The optimization effect is significant. The display of avatars has played a pivotal role in the development of all types of interactive media. VRM is an easy-to-create, share, and copyright-protect virtual image file format. VRM is an easy-to-create, share, and copyright-protected avatar format, but its common browser-based display solution, three-vrm, suffers from serious performance problems. In this project, we propose several performance optimization schemes, each of which is experimentally analyzed and verified, and finally propose a simple and efficient optimization scheme for the display of the VRM format, which achieves significant results. This technical report contains four main sections: the first provides an introduction to the background of the project and a brief comparison of the virtual image implementation options that exist on the Internet today to discover the direction and significance of the project. The technical and system features and innovations used in the project are also covered. The second part describes the research process, intermediate results, and phased conclusions of the project. In the third part, the principles of the implementation and the core code of the final solution in the project are explained and sorted out in detail. The fourth part discusses and summarises the limitations of the system and areas for improvement.

Keywords. Avatar, WebGL, render optimization.

1. Introduction

In 2010, when people talked about interactive media, they mostly thought of web-based Flash mini-games. But with the proliferation of portable smart devices, the number and variety of interactive media have exploded. In such rapid development, the generation and display of virtual images is a very important technique. To simplify the process of generating avatars, the VRM format was created. The VRM format can be imported into different development platforms and displayed in different software environments. Its rendering on the WebGL side generally uses the three.js + three-vrm scheme. However, in practice, three-vrm has performance problems, which to a certain extent hinders the wider use of VRM format applications in browsers. This project aims to propose an efficient optimization to address the performance issues when rendering skinned character actions in three-vrm.

[1] Corresponding Author, ZhiXiang TAO, School of Economics And Business College Guangdong Polytechnic Of Industry And Commerce, Guangzhou, China; E-mail: 1433142067@qq.com.

1.1. Rationale and Research Gap

Skinned character model display technology dates back to the early 21st century and has experienced rapid development in character modeling and its animation technology, driven by the games industry and the animation industry represented by Disney. From simple skeleton binding to IK control, hair systems, and GPU skinning, character animation techniques have made it possible to create fictional avatars. The advent of VRM technology has been a major step forward in making character modeling accessible to the general public. Previously, the rendering of skinned animations required the execution of many matrix transformations, which often had to be run in native applications to rely on heterogeneous computing hardware such as GPUs. Nowadays, browser-based applications are also becoming available with significant computing power and GPU access performance, making it possible to display character animation in the browser.

1.2. Features and Innovations

In this project, the performance of the skinned model display of the three-vrm project was tested and debugged to identify performance bottlenecks. Three performance optimization solutions were then proposed for the bottlenecks, implemented, and tested individually. The result was a 400% improvement in the rendering efficiency of the three-vrm model by optimizing the model at the skeletal structure level. The solution is straightforward to implement.

2. Related Work

In 2019, a character model format based on the glTF format [1], the VRM format, emerged. It aims to provide a convenient vehicle for VR devices to create and display virtual images, making full use of the extensibility of the glTF format, storing information such as character models, bones, and expressions in a single file, and integrating functions such as skeletal springs and physical collisions. What's more, along with the VRM format comes the Void Studio software for creating VRM files [2]. Using this software, users can visually and simply select the type of five senses required, adjust the body attributes and finally generate a usable VRM file directly. This innovation greatly simplifies the process of creating virtual images and makes it possible to apply more high freedom virtual images.

Due to previous practical experience with relative projects, Three.js was chosen for this project to experiment with. To import and display avatars in Three.js, third-party plugins were introduced for this project, including the MMDLoader and Three-VRM libraries. MMDLoader is an MMD format loading plugin that comes with the Three.js example folder, which provides the ability to read and parse MMD file formats. Three-VRM is a VRM model format plugin for Three.js, developed and maintained by Pixvi. It not only provides the ability to read and parse VRM format models but also retains information about the joints, expressions, spring bones, physics, etc. that are unique to VRM models. Using the Three-VRM library, web developers can customize animation values and control model animations in real-time, enabling web-side motion capture and other features.

3. Procedures

3.1. Performance Test

Firstly, performance tests were carried out against existing virtual image display solutions. The test cases were written using the Three.js framework, referencing two third-party components, MMDLoader and Three-VRM.

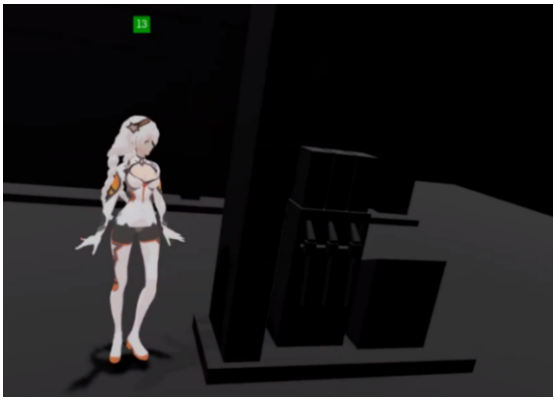

Figure 1. MMD rendering test.

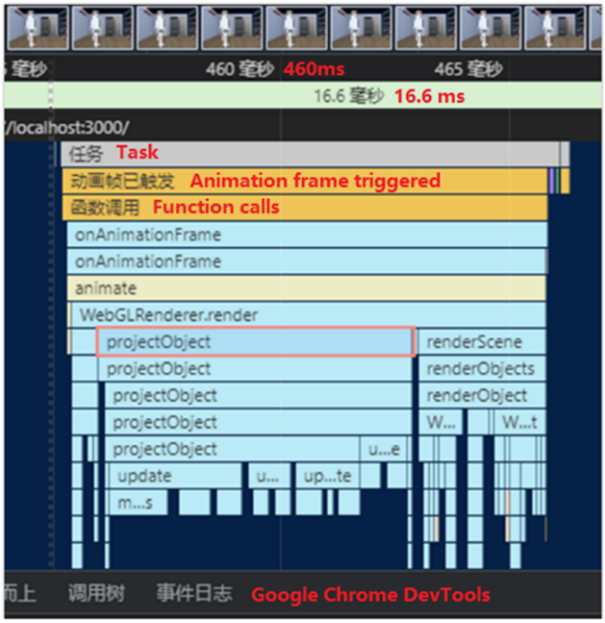

Figure 2. VRM render performance profiling.

The MMD model shown above (figure 1) was tested to maintain a frame rate of 90fps on a desktop platform with 25 animated instances drawn simultaneously [3], while on the VR all-in-one platform it could only barely reach around 12fps (figure 2).

The performance of the desktop platform has been unsatisfactory in the tests on the VRM format models. With just one model displayed, the desktop browser was only able to maintain a rendering frame rate of around 60fps. After analyzing the performance of the call tree in the browser Profiler, the performance bottleneck for VRM model rendering was the projectObject function. The execution time of this function accounted for around 70% of the time per frame. After a more in-depth analysis of this function, we were able to further locate the problematic code.

3.2. Locate Bottleneck

Once the problem function was found, a deeper understanding of the causes of the performance issues was sought. The process of displaying the skinned animated model in Three.js is divided into the following steps: first, the basic information about the model (vertices, slices, shaders), etc. is loaded. The pose of the skeleton for the current frame is then solved by interpolation based on information such as animation tracks, keyframes, and renderer time. Due to the hierarchical structure of the model skeleton, the absolute pose of the skeletal points is solved from the pose of each skeleton, which requires a recursive bottom-up pose transformation, starting from the root joint [4]. The skeletal pose solution is a depth-first search-like process in which the absolute pose of each joint point is recorded as the skeleton is traversed and later weighted according to the vertex weights to determine the final position of each vertex of the skin model (figure 3).

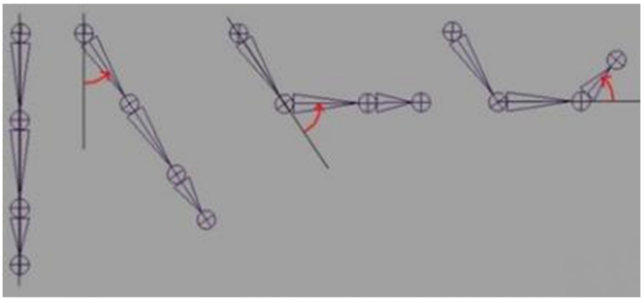

Figure 3. Skeleton transform.

Based on Three-.js, the Three-VRM library intervenes in the VRM model pre-processing process to maintain the functionality specific to the VRM format. From the loading stage, in addition to loading the model's basic information using the glTFLoader, Three-VRM also implements a part that reads and parses additional information about the model. This includes the model's skeleton and expression definition, the spring skeleton function, the physics simulation function, etc. With the JSON plain text storage of glTF files, this information can be easily read out using Javascript's own JSON processing tools. After reading the above information, Three-VRM corresponds the model bones to the additional binding information, generating a special binding structure for each bone to facilitate the subsequent dynamic setting of the joint angles. As Three-VRM takes over the change of the bone joint values, the animation calculation is no longer performed by Three-.js here but is passed to Three-.js for recursive transformation and skinning operations after Three-VRM has set up the bone information. The hotspot function multiplyMatrices found in the performance tests is the function used in the recursive transformation phase.

3.3. Experiments on Solutions

3.3.1. WASM Code Optimization

Firstly, we tried to rewrite the performance hotspot function multiplyMatrices in C (figure 4) and compiled it into a WASM module for Javascript calls (figure 5). As the function performs the multiplication of two matrices, it did not encounter too many obstacles during the migration to C.

```
void mul_mat(float *ae, float *be, float *te)
{
    const float a11 = ae[0], a12 = ae[4], a13 = ae[8], a14
= ae[12];
    const float a21 = ae[1], a22 = ae[5], a23 = ae[9], a24
= ae[13];
    const float a31 = ae[2], a32 = ae[6], a33 = ae[10], a34
= ae[14];
    const float a41 = ae[3], a42 = ae[7], a43 = ae[11], a44
= ae[15];
    // following code omitted
```

Figure 4. Part of the function code.

```
let start = performance.now();
        for (let i = 0; i < runs; i++) {
            multiplyMatrices(
                [1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2
,1.2,1.2,1.2,1.2,1.2,1.2],
                [1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2
,1.2,1.2,1.2,1.2,1.2,1.2]
            );
            multiplyMatricesWASM(
                [1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2
,1.2,1.2,1.2,1.2,1.2,1.2],
                [1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2,1.2
,1.2,1.2,1.2,1.2,1.2,1.2]
            );
        }
        let end = performance.now();
```

Figure 5. Part of the test code.

```
multiplyMatrices for 1e7 times cost 435.3999999985099ms
multiplyMatricesWASM for 1e7 times cost 4851.699999999255ms
```

Figure 6. Test results.

From the analysis of the test results we found that although WASM bytecode is parsed and executed more quickly than Javascript, it has better performance optimization for hotspot code due to the JIT optimization used for Javascript in modern browsers. On the other hand, WASM function calls have a greater performance disadvantage due to context switching and memory copy overhead (figure 6). If the performance benefits of WASM are to be fully exploited [5], the entire maths library of Three.js would have to be rewritten to eliminate the performance overhead of cross-language calls, and the effort would be more than worth it.

3.3.2. Model File Optimization

We have tried to optimize the model files. This was done in several ways: reducing the number of model triangles, reducing the number of model bones, reducing the quality of model mapping, and merging models and maps.

With these adjustments, the file size has been reduced by about 20% compared to the original file without affecting the visual appearance (figure 7). The smaller file size will facilitate the transfer in a network environment [6].

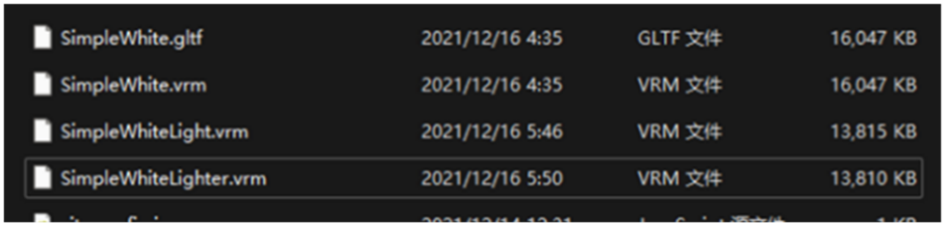

Figure 7. Optimized model file.

Unfortunately, the performance of the optimized model file was tested to show no significant difference from the performance before the optimization, and the hotspot code still severely slowed down frame times. The optimization of the model only had the effect of saving network traffic [7].

3.3.3. Model Loader Optimization

As the optimization of the model files did not have the desired effect, we suspected that there were inefficiencies in the code implementation itself and conducted a review of the code in Three.js and Three-VRM. The code review mainly covered the glTFLoader, and VRMLoader sections, with a focus on how the third-party libraries handle the loading and transformation of character bones. Upon examination, the glTFLoader section that the Three-VRM library relies on to load skinned bones contains the following code (figure 8):

```
    // in function buildNodeHierarchy
    }

                        mesh.bind( new THREE.Skeleton( bones,
boneInverses ), mesh.matrixWorld );

                    } );
                    return node;
```

Figure 8. Problematic code.

To solve the problem, we created a general modified glTFLoader file for Three-VRM, using the same skeletal object to bind all the sub-objects of the VRM model, to ensure that only one full-body skeletal transformation is required per frame to draw all the sub-objects. After this, a performance comparison test was carried out.

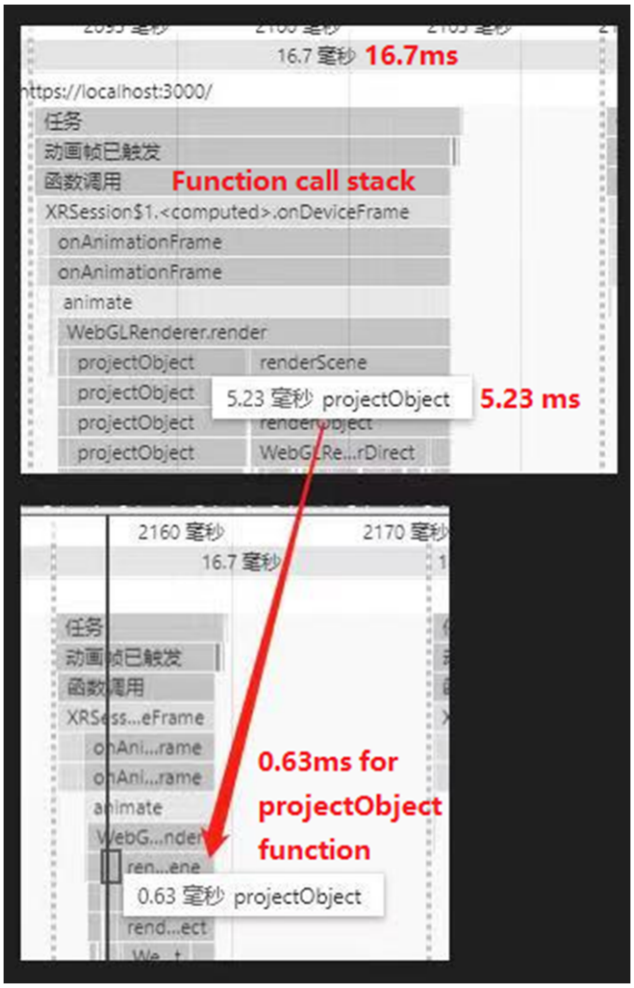

Figure 9. Comparison test.

Before the skeleton merging optimization, the total execution time of the project object function was 5.23 milliseconds. After the optimization, the execution time of the projectObject function was reduced to 0.63 milliseconds using the same model and action settings, an increase in computational efficiency of over 800% (figure 9).

4. Results

After experimenting with different solutions, we finally adopted the optimization of merging skeletal objects. By modifying the model loader source code, we succeeded in making the rendering of VRM models on the browser side significantly more efficient. In subsequent tests, even after increasing the number of avatars to four and rendering the room scene simultaneously, the rendering time per frame was well below 16.6ms (60fps), reaching a usable level (figure 10).

Figure 10. Additional test.

5. Discussion & Conclusions

This project proposes a performance optimization solution for virtual image rendering in the VRM format in a browser environment by identifying the problems of a specific technology in an application scenario. It has a high practical value and can provide more possibilities for web application development. The results of this project can be combined with more complete project development in the future to promote research in the field of web-side scene editors and game editors.

References

[1] Fang C. glTF format: A in-depth review. ZhiHu. 2019 May 17.
[2] VRM Applications. VRM. 2022 Feb. 28.
[3] Unity Inc. How do Honkai Impact 3 make cartoon effects? Bilibili. 2018 June 1.
[4] Bermudez, Luis. Overview of Inverse Kinematics. Medium. 2020 May 31.
[5] Threads and Atomics in WebAssembly. GitHub. 2022. Mar..
[6] File System Access API - Web APIs | MDN. MDN. 2022 Feb. 18.
[7] Lin H X. Large scene optimization for WebGL. ZhiHu. 2020 July 3.