Design and Implementation of WebGL Model Rendering Engine

Xiao Liu¹, Tingting Wu²,³, Wenjian Xu⁴, Qing Liu⁵. *

¹ School of Intelligent engineering technology, Jiangsu Vocational College of Finance and Economics, Huaian 223000, China
² School of Geomatics, Anhui University of Science and Technology, Huainan 232001, China
³ Coal Industry Engineering Research Center of Mining Area Environmental and Disaster Cooperative Monitoring, Anhui University of Science and Technology, Huainan 232001, China
⁴ Huaian Langsheng Media Co., Ltd, Huaian 223000, China
⁵ School of Mechanical and Electrical Engineering, Huainan Normal University, Huainan 232038, China

*Corresponding Author

Abstract

This thesis conducts an in-depth study and implements a model rendering engine based on WebGL. It elaborates on the basic concepts, characteristics of WebGL, and its application prospects in the field of real-time graphics rendering. The implementation methods are discussed, and a design scheme of the model rendering engine based on WebGL is proposed. When introducing the background and basic principles of WebGL, it is pointed out that it is a technology for high-performance graphics rendering in the browser using JavaScript. Based on the OpenGL ES 2.0 standard, it is achieved by embedding its context in the HTML5 Canvas element, and has advantages such as cross-platform, high performance, easy learning and use. In addition, the design scheme of this engine is proposed, including key technologies such as model loading and lighting calculation. Common rendering engine frameworks and tools are also introduced to help developers efficiently build WebGL applications.

Keywords

WebGL; Model Rendering Engine; Rendering Pipeline; WebGL Framework.

1. Introduction

With the development and popularization of Internet technology, the demand for online graphic rendering is gradually increasing. Traditional graphic rendering technologies rely on specific software and hardware, and the development and deployment are complex¹. However, the emergence of WebGL technology has broken these limitations. It is based on the OpenGL ES 2.0 standard and embeds the context in the HTML5 Canvas element to achieve high-performance graphic rendering in the browser, allowing developers to use similar interfaces to achieve smooth and realistic 3D image effects²-³.

Because of the flexibility and cross-platform nature of WebGL technology, it is widely used in fields such as games, virtual reality, augmented reality, architectural design, and data visualization⁴. Users can directly interact with 3D models in the browser without the need for additional plugins or programs⁵-⁶.

With the progress of WebGL technology, more developers pay attention to the implementation and optimization of model rendering engines based on WebGL. These engines utilize their functional features to efficiently load and render complex 3D models, providing a realistic
visual experience and supporting a variety of functions and optimizations, allowing developers to flexibly control the rendering process to meet different needs.

1.1. The Basic Principles of WebGL

The basic flowchart of WebGL is shown in Figure 1. Its principle is: (1) GPU acceleration. WebGL relies on the computer's Graphics Processing Unit (GPU) to accelerate graphics rendering. The GPU has strong parallel computing capabilities and is suitable for processing a large amount of graphics data. WebGL interacts with the GPU, hands over the 3D model data to the GPU for processing and rendering, and the results are displayed in the web browser; (2) Shader programs. WebGL uses shader programs to define graphics rendering. Shaders are small programs running on the GPU, controlling vertex processing, lighting calculations, and pixel rendering, etc. WebGL mainly uses vertex shaders to process the vertex positions and attributes of the model, and fragment shaders to process pixel colors and lighting effects; (3) Buffers and textures. WebGL uses buffers to store and manage model data, which can efficiently transfer data to the GPU for processing. It also supports textures. Image data can be used as texture mapping to enhance the visual effect; (4) Rendering pipeline. WebGL processes and renders graphics through the rendering pipeline, including stages such as vertex processing, lighting calculation, pixel rendering, and final output to the screen. WebGL controls each stage by defining shader programs and setting the rendering state to achieve the desired graphics effect and interaction.

![Basic Flow Chart](image)

**Figure 1.** Basic Flow Chart

1.2. Characteristics of WebGL

WebGL has many advantages. It is based on the OpenGL ES standard, which is convenient for developers to create graphic effects for web applications by using familiar functions and syntax; it supports all major modern web browsers, achieving cross-platform and cross-browser operation, and the code can be used on various devices and browsers without modification once it is written; it utilizes the user's computer GPU for hardware acceleration, significantly improving performance, and being able to realize complex 3D scenes and effects in web applications; it can run in supported browsers without the need for additional plugins, simplifying the usage and deployment process; as an open standard led by the Khronos Group, it has extensive community support and developer participation, providing a large amount of documentation, examples, and tools, lowering the threshold of learning and development.
2. Design Scheme

2.1. Rendering Engine Process

This system is mainly composed of a model loading module, a model simplification module, and the WebGL rendering engine’s rendering core module. The model loading module and the model simplification module are responsible for generating the model document tree, and these data structures will be passed to the rendering core module, which is then responsible for performing the actual rendering operation. The rendering core module uses its internal components such as lighting, materials, shaders, and rendering classes to complete the interaction between the model data and the GPU shader variables, thereby realizing the rendering function of the model[7-8]. The basic flowchart of the complete code of the rendering engine is shown as follows in the figure:

![Figure 2. Basic Flow Chart of the Rendering Engine](image)

2.2. Process of the Model Loading Module

The model loading module mainly includes three core modules: the file loading module, the file parsing module, and the generating document tree module. The file parsing module is further subdivided into six submodules, including the parse accessor module (ParseAccessors), the parse buffer view module (ParseBufferViews), the parse image information module (ParseImages), the parse material information module (ParseMaterials), the parse mesh data module (ParseMeshes), and the parse texture module (ParseTextures).

![Figure 3. Schematic Diagram of the Specific Process of the Model Loading Module](image)

2.3. Implementation of the Model Simplification Module

The basic principle of the model simplification algorithm: first, calculate the merge cost and mean deviation of vertices, and implement the merge operation for vertices with small merge cost and deviation within the specified range. After all vertices have completed this simplification process, the remaining Face and Vertex class objects are stitched back into new mesh objects, and the original mesh objects in the document tree are replaced. Figure 4 illustrates the flow of the model simplification algorithm.

![Figure 4. Flowchart of Model Simplification Algorithm](image)
2.4. Design of the Lighting Module

The lighting module is an important module of the rendering engine. If no lighting is added to the scene, any three-dimensional characteristics of the rendered model cannot be seen. The lighting module developed and designed based on the WebGL rendering engine contains three types of lights: ambient light, point light, and parallel light.

2.4.1. Principles of Ambient Light Calculation

In the fragment shader, the pixel color under the effect of ambient light is the sum of the \( C_{ambient} \cdot n \) represents the number of ambient lights, and its specific calculation method is shown in Equation (1):

\[
C_{pixel} = \sum_{i=0}^{n} C_{ambient}
\]  

(1)

2.4.2. Point Light Source Calculation Principle

The color calculation of each pixel under the effect of the point light source in the fragment shader is as shown in Equation (2) below:

\[
C_{pixel} = \sum_{i=0}^{n} C_{point} \cdot \cos \theta = \sum_{i=0}^{n} C_{point} \cdot (N \cdot L)
\]

\[
= \sum_{i=0}^{n} C_{point} \cdot \left( N \cdot (P_{light} - P_{geometry}) \right)
\]  

(2)

The above formula: \( n \) represents the number of point lights, \( C_{point} \) represents the color value of the point light, \( \theta \) represents the angle between the incident ray and the normal
vector, $N$ represents the normal vector of the current pixel, $L$ represents the direction vector of the light ray, $P_{light}$ represents the position of the light, $P_{geometry}$ represents the position vector of the pixel. The difference between the two vectors indicates the direction vector of the light. In the actual implementation process of the shader code, $L$ must be vector normalized. The GLSL language provides a built-in function for vector normalization called "normalize". This built-in function returns a vector with the same direction but a length of 1.

### 2.4.3. Parallel Light Source Calculation Principle

The color calculation method of each pixel under the effect of the parallel light source in the fragment shader is as shown in Equation (3) below:

$$ C_{pixel} = \sum_{i=0}^{n} C_{direction} \cdot \cos \theta = \sum_{i=0}^{n} C_{direction} \cdot (N \cdot L) $$

The above formula: $n$ represents the number of parallel light sources, $C_{direction}$ represents the color of the parallel light source, $\theta$ represents the angle between the normal vector of the pixel and the direction of the parallel light, $N$ represents the normal vector of the pixel, and $L$ represents the direction vector of the parallel light at the pixel.

### 2.5. Material Module Design

The material information of the model is of great significance for model rendering. The Material and the lighting jointly determine the displayed effect of the model on the screen. The Material class of the rendering engine is mainly used to handle the material data of the model. The related UML class diagram of the material module is as follows:

![Figure 5. Relevant UML Class Diagram of the Material Module](image)

### 3. Conclusion

This paper conducts a comprehensive and in-depth research and design implementation of the model rendering engine based on WebGL. By analyzing the technical principles of WebGL, a feasible design scheme is proposed, and key modules such as the lighting module and the
material module are elaborated in detail. In the actual development process, although many technical challenges are encountered, through continuous exploration and optimization, ideal results are achieved. However, WebGL technology continues to evolve, and the optimization and improvement of the model rendering engine are endless. In the future, we will continue to pay attention to the latest trends in the industry, further enhance the performance and functionality of the rendering engine, expand its application fields, and bring users a more realistic, smooth and wonderful visual experience.

Acknowledgments

The Project was Supported by Coal Industry Engineering Research Center of Mining Area Environmental and Disaster Cooperative Monitoring (Anhui University of Science and Technology) (NO. KLAHEI202308), and the project of Intelligent Recognition Method of Rock Formation Based on Multi-source Borehole Data (NO. 2023HX181).

References