Applied Mathematics, 2016, 7, 701-708
Published Online April 2016 in SciRes. http://www.scirp.org/journal/am
http://dx.doi.org/10.4236/am.2016.77064

Architecture and Implementation of 3D Engine Based on WebGL

Xinliang Wei[1,2], Wei Sun[1,2], Xiaolong Wan[1,2]

[1]SUN Yat-sen University, Guangzhou, China
[2]Key Laboratory of Information Technology (Ministry of Education), SUN Yat-sen University, Guangzhou, China
Email: weixinl@mail2.sysu.edu.cn, sunwei@mail.sysu.edu.cn

Received 10 March 2016; accepted 25 April 2016; published 28 April 2016

Abstract

As the progress of 3D rendering technology and the changes of market demand, the 3D application has been widely used and reached as far as education, entertainment, medical treatment, city planning, military training and so on. Its trend is gradually changed from client to web, and so many people start to research the 3D graphics engine technology on the web. WebGL and HTML5 rise in recent years and WebGL solves two problems of interactive 3D application on the web perfectly. Firstly, it implements the interactive 3D web application by JavaScript without any browser plug-in components. Secondly, it makes graphics rendering using the underlying graphics hardware, which is united, standard and cross-platform OpenGL interface. However, it is very difficult for 3D application web programmer to understand multifarious details. Therefore, a 3D engine based on WebGL comes into being. The paper consults the existing 3D engine design idea, architecture and implementation experience, and designs a 3D graphics engine based on WebGL and Typescript.

Keywords

WebGL, Design Idea, Architecture of 3D Engine

1. Introduction

As the progress of Internet industry, technologies of web are more and more important. Any one of the essential elements of an attractive web program is graphics. However, with the increasing complexity of the web program, the traditional two-dimensional graphics have been unable to meet the needs of the program. Thus, the interactive 3D graphics application for web program comes into being.

But the early technology is not mature, such as a simple web interactive 3D graphics program achieved by Java Applet, not only need to download a huge support environment, but also the picture is very rough and the performance is also very poor. The main reason is that the Java Applet doesn't use the acceleration function of graphics hardware directly, even though we install a graphics card with high performance, it still can't play any role in the interactive 3D graphics rendering.

Later, Adobe's Flash Player browser plug-in and Microsoft's Silverlight technology have emerged, and become the mainstream of web interactive 3D graphics technology gradually. Different from Java Applet, these technologies are directly used the graphics programming interface offered by operating system, and call the acceleration function to achieve high performance graphics rendering. However, there are some problems in these two solutions. First of all, they are achieved through the browser plug-in, which means that for different operating system or browsers, you need to download different versions of the plug-in, and for a special operating system or browser, there may not be a corresponding version of the plug-in. Secondly, for different operating system, these two technologies need to call different graphics programming interface. These two points are insufficient, which largely limit the application of web interactive 3D graphics program.

However, WebGL appears to solve the above two problems. Firstly, it uses JavaScript to achieve Web interactive 3D graphics programming, without any browser plug-in. Secondly, it makes graphics rendering using the underlying graphics hardware, which is united, standard and cross-platform OpenGL interface.

This means that no longer need to use any browser plug-ins, just using HTML and JavaScript, we can make a good web interactive 3D graphics applications that performance no less than Flash, Silverlight and so on. The application can be operated in the same way on any platform, which is a revolutionary change.

However, the development of interactive 3D graphics program based on WebGL is very complicated and time-consuming. It needs to pay close attention to the details of a large number of ground floor. To solve these problems, some developers will make some of the common features of the interactive 3D graphics package into a software package, which provides convenience for other developers to develop similar procedures. This software package is often referred to the 3D graphics engine or 3D engine.

We usually compare the 3D engine to the car's engine. It is well know that the heart of a car is the engine, which determines the stability and performance of the car. The sense of operation, speed and other closely related indicators are closely related to the engine. The 3D engine is the same as the car's engine. The user experience from the interactive 3D graphics program, interactive experience, etc. are closely related with the 3D engine. The 3D engine plays the role of the engine in the 3D graphics program, and the program is tied up with all elements, and coordinating them. It is because the 3D engine is very important for the development of interactive 3D graphics program. Therefore, the research has not stopped in the academia and industry. It has accumulated a lot of valuable experience, including the basic structure of 3D engine and the theory of computer graphics and many software resources. These experiences provide a theoretical support and reference for the development of 3D engine based on WebGL. In view of the fact that the 3D engine based on WebGL is still in the basic stage, this paper designs and implements a 3D engine based on WebGL and Typescript. It verifies the feasibility of WebGL application in the development of interactive 3D graphics program, and provides convenience for other developers to develop Web interactive 3D graphics program.

2. The Design Idea of 3D Engine

In this paper, the design idea, architecture and implementation experience of a variety of 3D engines are introduced, and their advantages and disadvantages are analyzed in order to be used in the design of their own.

The first object model is based on derived class. In the traditional design, we generally use the "derived" to describe the relationship between the object. Subclass derived parent class to obtain the function of the parent class. In the design of the game, it will be based on the needs of the game itself to add a variety of functional support for the game, such as rendering, collision, rigid body, and particle system, etc. These generic functions must be implemented in the base class to provide services for a variety of derived classes. This results that the base class of game object has become very large and bloated, that is difficult to use and maintain [1].

The second object model is based on component. The main idea is all the basic functions that provided to the game object are made into "component" independently. A specific game object can combine the function modules when it needs. All functions are no longer interfaces in the parent class, and become a child object instance, providing service to the game object. This can not only ensure the reusability of the functional code, but also increase the whole object system's modularity and flexibility. The main advantage of this approach is the reduction of the coupling between classed. The dependencies between components still exist, but the responsibilities of the components are more clear.

The advantage of the component-based approach is bellow:

1. Component structure has good scalability. Adding some new behavior to the 3D graphics program is very simple, as long as you create a new component.

2. Responsibility of each class is clear and code is clear too.

The shortcomings of component-based approach are mainly shown in the increase of the indirect relationship, which will add some additional system overhead to the class inheritance structure of hard encoding.

In the popular 3D engine, Cocos engine is implemented based on the object model of the derived class, and the unity engine is realized by using the object model based on the component. Based on the design idea of two kinds of engines, the engine of this paper adopts the method of object model based on component. For objects in 3D graphics program, such as scene, camera, light, geometric texture, material effect, etc. will use the method of object model based on component. The architecture and implementation of a 3D engine in the next section of the specific reference.

3. Architecture and Implementation of 3D Engine

3.1. Engine Architecture

Each 3D engine is very similar to the basic module, according to the main points of the function: the rendering module, system module, control module, data storage module, game interface module and game plug-in module. What's more, runtime engine architecture can be subdivided into target hardware, device drivers, operating system, third-party SDKs and middleware, platform independence layer, core systems, resource manager, rendering engine, profiling and debugging Tools, collision and physics, animation, human interface devices, audio, online multiplayer/networking, gameplay foundation systems and game-specific subsystems, etc. [2].

In this paper, by the basic content of game engine architecture, as well as the characteristics of HTML5 and JavaScript, the design of the engine architecture is defined as **Figure 1**.

As shown in the graph, the engine class is the core of the 3D engine, and the main rendering functions of WebGL are encapsulated in the engine class. Objects in 3D game can have different components, and different components are independent of each other.

The engine class encapsulated main rendering functions and some basic WebGL properties. The process of rendering happened in the engine class and if the functions of WebGL are encapsulated in the class, it will be more convenient to call those properties and functions. Objects in 3D game may have basic components such as

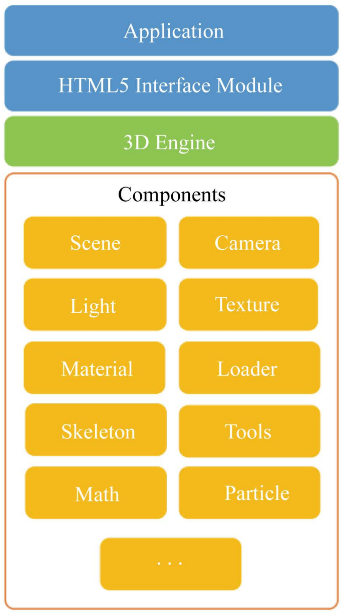

Figure 1. Engine architecture.

Texture, Material, and Skeletonetc.

3.2. Comparison of Engine

3.2.1. Vertical Contrast

Before the WebGL standard has not been published, the existing engine is based on DirectX or OpenGL graphics library, which is to limit the diversity of the platform. WebGL is a 3D drawing standard, which allows JavaScript and OpenGL ES2.0 to be combined together. WebGL can provide 3D hardware accelerated for HTML5 canvas, so web developers can not only make browser display 3D scene and model more smoothly with the graphics system, but also create a complex navigation and data visualization. Obviously, it is convenient to develop 3D graphics application or game on browser by WebGL, and solve the problem of multi-platform extensions.

3.2.2. Horizontal Contrast

There're many open source 3D engine based on WebGL, such as Threejs, Babylonjs, PhiloGL, SceneJS, CopperLicht. Each engine frame has its own characteristics. Next, we use Threejs and Babylonjs as a reference for the engine of this paper to do the corresponding contrast.

As can be seen from **Tables 1-3**, the engine of this paper was not yet mature in the rendering quality and model loading, but other aspects of the performance were as good as Babylonjs and Threejs. In particular, the generation of dynamic shadows and the function of ray tracing, made the 3D scene on the screen more realistic.

Those tables above are created by comparing these three engines. Comparative content consisted of rendering effect, basic function, and real sense enhancement. We did a lot of comparative experiments and recorded the result of experiments.

3.2.3. Implementation of Augmented Reality

Dynamic shadow

The key to implementing dynamic shadow was that the scene would update attributes of objects' shadow,

Table 1. Rendering effect contrast.

	The engine of this paper	Threejs	Babylonjs
Rendering speed	Fast	Fast	Fast
Rendering quality	Better	Best	Best

Table 2. Basic function contrast.

	The engine of this paper	Threejs	Babylonjs
Shadow mapping	Better	Better	Better
Light rendering	Best	Best	Best
Texture mapping	Best	Best	Best
Model loading	Better	Best	Best
Environment mapping	Better	Better	Better
Particle system	Better	Better	Better

Table 3. Real sense enhancement contrast.

	The engine of this paper	Threejs	Babylonjs
Dynamic shadow	Best	Best	Best
Ray tracing	Best	Best	Best
Light characteristics	Best	Best	Best

such as position, size and so on. In the engine of this paper, two properties were set for objects in the scene, one was *castShadow*, the other was *receiveShadow*. *castShadow* indicates whether the object will have a shadow over another object. *receiveShadow* indicates whether the object will receive the shadow cast by other objects on it. In addition, we should set the shadow properties of the light. The shadow properties of the light included *castShadow*, *onlyShadow*, *shadowBias*, *shadowDarkness*, *shadowCascade* and so on [3].

We used shadow maps technology to render the scene if we set the shadow properties of objects and light. To render a scene using a shadow map, we draw the scene as usual from the point of view of the camera. For each vertex of every triangle, we calculate its position in *light space i.e.*, in the same "view space" that was used when generating the shadow map in the first place. These light space coordinates can be interpolated across the triangle, just like any other vertex attribute. This gives us the position of each fragment in *light space*. To determine whether a given fragment is in shadow or not, we convert the fragment's light-space (x, y)-coordinates into texture coordinates (u, v) within the shadow map. We then compare the fragment's light-space z-coordinate with the depth stored at the corresponding texel in the shadow depth map. If the fragment's light-space z is farther away from the light than the texel in the shadow map, then it must be occluded by some other piece of geometry that is closer to the light source—hence it is in shadow. Likewise, if the fragment's light-space z is closer to the light source than the texel in the shadow map, then it is not occluded and is not in shadow. Based on this information, the fragment's color can be adjusted accordingly [4]. The shadow mapping process is illustrated in **Figure 2**.

Ray tracing

Ray tracing is actually a reflection of the real scene, such as the reflection of the glass, the reflection of the mirror, the reflection of the metal and so on. Therefore, we need to set up the corresponding material, add reflectivity, refractive index, etc. [5]. The steps of ray tracing algorithm used by this paper are as follow:

1. The camera's film is divided into discrete grids (*i.e.*, pixels), and our goal is to determine the color value of each pixel. It is shown in **Figure 3**.

2. For each pixel, a ray is traced from the camera position, pointing to the pixel. As shown in **Figure 4**.

3. For this beam of light, to determine whether the object in the scene and the intersection. If the intersection, then go to step 4; otherwise, fill the background color to the current pixel, go back to step 2, continue processing the next pixel.

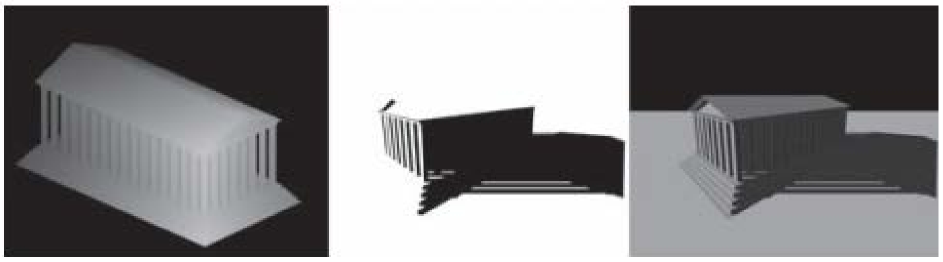

Figure 2. The far left image is a shadow map—the contents of the z-buffer as rendered from the point of view of a particular light source. The pixels of the center image are black where the light-space depth test failed (fragment in shadow) and white where it succeeded (fragment not in shadow). The far right image shows the final scene rendered with shadows.

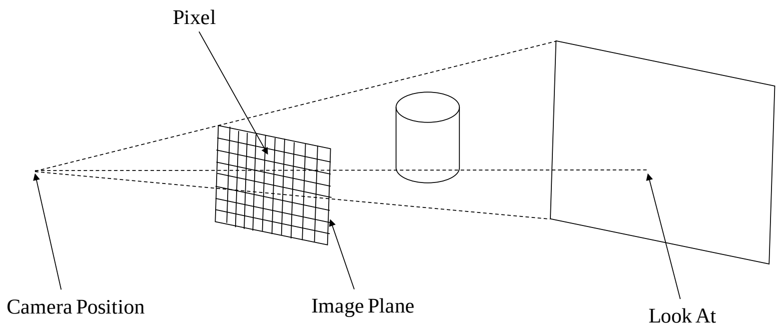

Figure 3. The camera's film is divided into discrete grids (*i.e.*, pixels).

4. If the rays and objects intersect, the color value of the intersection point of the object surface is calculated. The color value of the pixel is the color value of the pixel. As shown in **Figure 5**.

a) Firstly, check the contribution value of each light source at the intersection point. Tracking a new light to the light source, used to determine the intersection point is all lit, part of the light is not lit, but also determine the shadow [6].

b) If the object's surface has a reflective nature, calculate the initial light's reflection, then track the reflection light, go to step 3.

c) If the surface of the object has a refractive property, the initial light is calculated to reflect the light, and then track the refraction of light, go to step 3.

d) Finally, according to the surface properties (reflectance, refractive index), and different types of light calculated by the color value, we can determine the intersection of the color value, that is, the color value of the current pixel point [7].

5. Go back to step 2 and continue with the next pixel point. This process is repeated until the pixel point is traversed.

4. Conclusions

In this paper, the engine was designed by the object model based on component, improving the engine's scalability. It's used the new technology of TypeScript, HTML5 and WebGL.

So far, this paper had completed the basic functions of the engine, including the core rendering, scene components, camera components, material texture components, basic geometric components, mathematical components and so on. The performance was very close to Threejs and Babylonjs, especially in the real scene of enhancement. The corresponding screenshots were as follows (**Figures 6-9**).

The engine of this paper was still developing, and there're many place to supplement and perfect, such as model loading, scene editing, audio, collision detection and so on. The author of this paper would improve other parts of the engine, and hoped that through this article, other developers would get to know the knowledge of architecture of 3D engine based on WebGL.

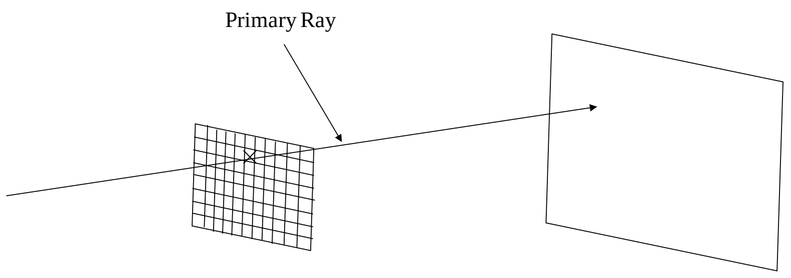

Figure 4. For each pixel, a ray is traced from the camera position, pointing to the pixel.

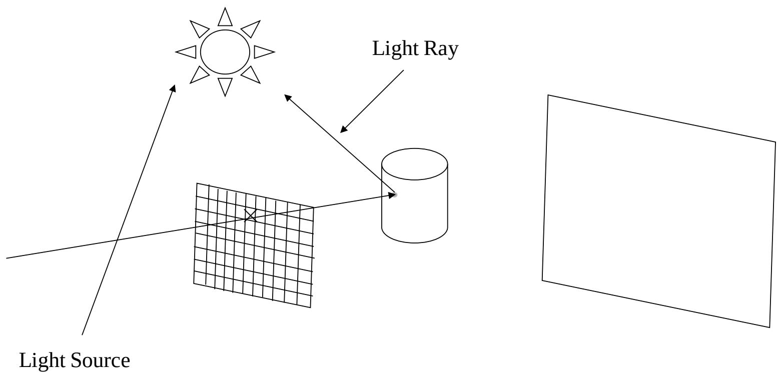

Figure 5. If the rays and objects intersect, the color value of the intersection point of the object surface is calculated. After the above steps, the object surface will be based on the characteristics of the material to reflect the scene.

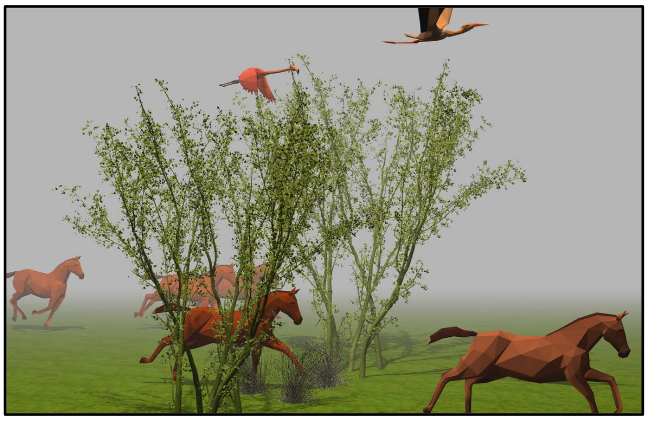

Figure 6. Combination of deformation animation and natural scene.

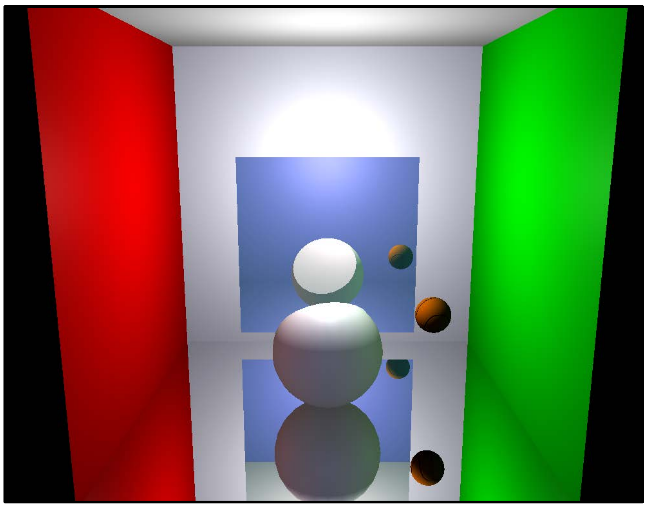

Figure 7. Animation and the real scene of enhancement.

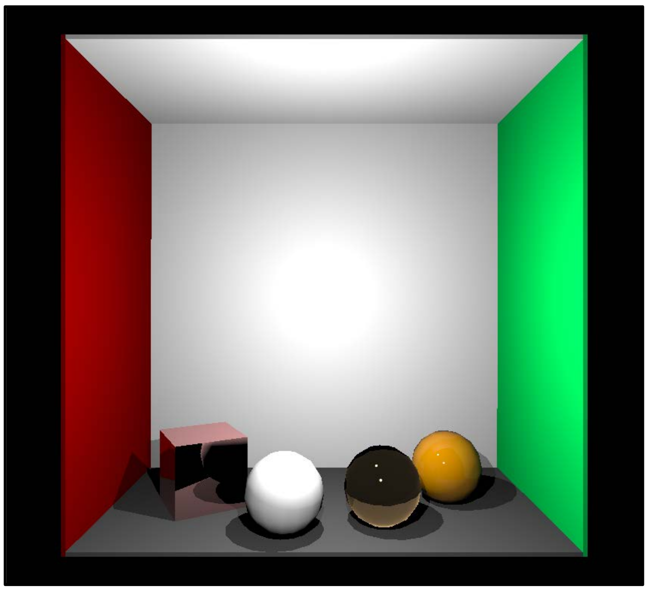

Figure 8. The true scene of ray tracing.

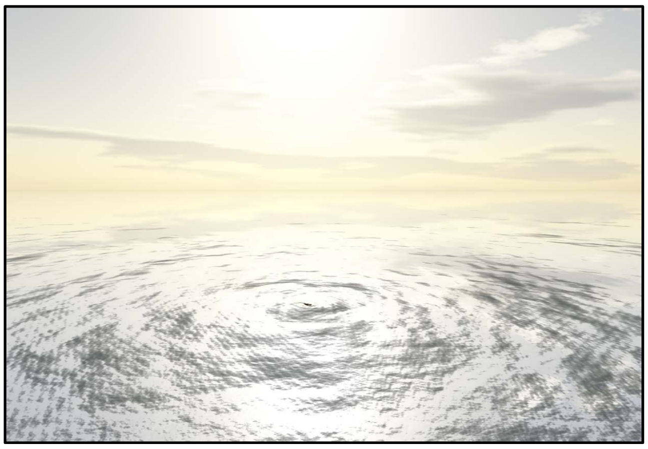

Figure 9. Simulated natural sky and water.

Acknowledgements

Thanks to support of Chinese National Natural Science Foundation and Key Laboratory of Information Technology (Ministry of Education) in SUN Yat-sen University.

References

[1] Eberly, D.H. (2001) 3D Game Engine Design: A Parctical Approach to Real-Time Computer Graphics. Morgan Kaufmann.

[2] Liu, Y.J., Chen, W.B. and He, X.M. (2010) 3D Graphics Engine Technology Research and Implementation. *IEEE*, 697-700.

[3] Shiratuddin, M.F. and Thabet, W. (2002) Virtual Office Walkthrough Using a 3D Game Engine: Special Issue on Designing Virtual Worlds. *International Journal of Design Computing* (*IJDC*), **4**, 4.

[4] Eberly, D.H. (2006) 3D Game Engine Design. 2nd Edition, Morgan Kaufmann.

[5] Hook, B. (1995) Building a 3D Game Engine in C++. Wiley, Hoboken, 122-130.

[6] Chipman, R.A. (2014) The Polarization Ray Tracing Calculus. Eprint Arxiv.

[7] Arikan, M., Preiner, R. and Wimmer, M. (2016) Multi-Depth-Map Raytracing for Efficient Large-Scene Reconstruction. *IEEE Transactions on Visualization & Computer Graphics*, **22**, 1-1.