International Journal of u- and e- Service, Science and Technology
Vol.8, No. 11 (2015), pp.139-150
http://dx.doi.org/10.14257/ijunesst.2015.8.11.14

Research and Application of Collision Detection in Virtual Roam based on WebGL and HTML5

Li Ruizhi[1], Xu Huahu[2], Gao Honghao[3], Yan Yingmin[3], Bian Minjie[1] and Xu Jiepin[2]

[1]School of Computer Engineering and Science, Shanghai University, 200444, Shanghai, China
[2]Shanghai Shang Da Hai Run Information System Co., Ltd
[3]Computing Center, Shanghai University, 200444, Shanghai, China
liruizhi@shu.edu.cn

Abstract

With the development of the Internet technology, people's life and work are more and more dependent on the Internet. Remote rendering techniques permit streaming of high-quality 3D graphics to apply onto a wide range of smart devices, and recent years have also seen many solutions of Web3D applications. Web3D is a kind of technology, which can perform the 3D visualization on the Internet, and virtual roam can present information in a far more intuitive and interactive way to people. So, Web3D technology is the foundation of virtual roam technology. Today, Web3D technology and virtual roam technology has become the research focus of computer technology. This paper presents the basic concept and characteristics of Web3D, and designs a solution concerning the virtual roam by combining WebGL and HTML5. This solution has characteristics of good compatibility, GPU acceleration and free plug-ins. Meanwhile, this paper aims at researching collision detection in the virtual roam and presenting a fast collision detection and correction algorithm.

***Keywords**: Web3D, Virtual Roam, Collision Detection, WebGL, HTML5*

1. Introduction

1.1. The Present and Problems of Virtual Roam

Virtual roam is an important part of Virtual Reality (VR). With the development of the Internet technology, virtual roam is considered as a new interactive way and has attracted more and more attention. The Web3D technology is the foundation of virtual roam technology. The Web3D technology can be traced back to the VRML whose full name is a virtual reality modeling language [1]. VRML came into the world in the 1990s, and was officially released as an international standard in December 1997. Without considering the compressed script code, the huge texture mapping data and the slow speed of the Internet led to the poor promotion that VRML has got. The VRML Organization was officially renamed its name as The Web3D Organization. The Web3D Organization established the Extensible 3D (X3D), which is considered as the new standard, and completed conversion from the VRML to X3D in 1998 [2]. X3D is combined with the developing computer technology such as XML, Java, *etc*. This way can improve 3D computing ability, rendering quality and the speed of transmission [3]. For now, a competition with the theme of Web3D is underway. And there are so many companies joining in this competition and launching their own solutions. For example, Unity3D [4], Java3D [5], Adobe Stage3D [6], Microsoft Direct3D [7], *etc*. However, the format and method in each solution are different, and the plug-in is also an unsolved problem. Almost every manufacturer specifies criteria that only apply to their own plug-ins, moreover these plug-ins' size range from several K to several M. When the network bandwidth is not in ideal condition, this will inevitably affect the part of the people using the Web3D.

In summary, the existing Web3D solutions almost have the weaknesses of low development efficiency, poor compatibility, depending on the CPU, and needing to install the plug-ins. To solve the problems mentioned above, the Chapter 2 introduces the main technologies of virtual roam, and proposes a new solution.

1.2. The Present and Problems of the Collision Detection

Collision detection is one of the most important geometric questions, with diverse applications in areas such as computer graphics, virtual roam, animation, robotics, computer games, etc. Many collision detection algorithms have been proposed in recent years [8-9]. Spheres and axially aligned bounding boxes (AABBs) [10] allow the simple tests. Oriented bounding boxes (OBBs) [11] and discrete orientation polytopes (k-DOP) [12] can fit more tightly. Space-Time Bounding Boxes (STTB) [13] can pay attention to the position of the object in the 3D space. Since virtual roaming is an online Web3D application, which is based on the Internet and browser, so how to detect the collision quickly and correct it are the most important problems of virtual roam.

The Chapter 3 presents a suitable algorithm for virtual roam in Web3D scenes. This algorithm combines the advantages of spheres and Space-Time Bounding Boxes, and also can detect from many directions in the real 3D space.

2. Technologies of Virtual Roam

In this chapter, in order to solve the problem of compatibility, depending on the CPU and installing the plug-ins, this paper puts forward a design of virtual roam based on HTML5, WebGL, and Three.js.

2.1. HTML5

HTML5 is a core technology, markup language of the Internet and used for structuring and presenting content for the World Wide Web [14]. As of October 2014, the fifth revision of the HTML standard of the World Wide Web Consortium (W3C) is completed. The previous version, HTML4, was standardized in 1997. For the applications of Web3D technology, many new syntactic features were added to HTML5. These include the new <video>, <audio> and <canvas> elements, as well as the integration of scalable vector graphics (SVG) content, etc. With the help of HTML5, more and more amazing 3D visual effects can be observed in the browser without plug-ins. At the same time, another goal of HTML5 is to create a unified world of the Internet, whether it is a notebook, desktop, or smart phone should be very convenient to access HTML5 websites in the browser. This is a better solution to solve the compatibility of Web3D technology.

2.2. WebGL

WebGL (Web Graphics Library) is a JavaScript API for rendering interactive 3D computer graphics and 2D graphics within any compatible web browser without using plug-ins [15]. WebGL is developed by the Khronos Group that is composed of many developers from Google, Opera, Mozilla, Apple, *etc*.

The traditional Web3D solutions, such as Java3D, Direct3D, almost not only need support from the third-party rendering engine, but also need to install plug-ins. These traditional Web3D solutions have a common drawback that is unable to use the GPU, so it is difficult to directly render the complex 3D scenes in the browser. WebGL can solve these problems perfectly. WebGL can be almost completely compatible with all browsers. WebGL is based on the OpenGL ES 2.0, so it can use the GLSL to provide API, which can use GPU through the browser. WebGL elements also can be mixed with other HTML elements, so the online virtual roam can be realized in the browser in this way. This is a suitable solution for the problem of hardware acceleration and plug-in.

2.3. Three.js

Considering the virtual roam in Web3D scenes needs simplicity and generality, so the Three.js framework is quoted in this solution. Three.js is a lightweight cross-browser JavaScript library/API used to create and display animated 3D computer graphics on a Web browser [16]. If the developer was mainly dependent on WebGL source, the efficiency of the development would be very low. By using the Three.js, developer can easily use the concept of object-oriented to design and develop. Meanwhile the Three.js is written by using JavaScript, which has very good compatibility in all browsers and can easily develop in the second time. By using JavaScript, customized services can be carried out according to customer needs.

The combination of HTML5 and WebGL creates a new opportunity for the development of Web3D [17]. This paper designs the virtual roam that has the advantages of good compatibility, free plug-in, and depending on the GPU. The basic elements such as lights, materials, geometries, camera, textures and animations, also can be added into Web3D scenes. Now, this chapter has solved problems about how to design and build virtual roam in Web3D scenes. Then the Chapter 3 introduces how to detect collision and correct in virtual roam.

3. Collision Detection and Correction

Before this paper formally introduces the collision detection algorithm, let firstly explain how Web3D can render in 2D computer screen. Every Web3D scene requires a point of view from which the people will be viewing it. Web3D scenes typically use a camera, an object that defines where (relative to the scene) the people is positioned and oriented, as well as other real-world camera properties such as the size of the field of view, which defines perspective (*i.e.*, objects farther away appearing smaller) [18-19]. The camera's properties combine to deliver the final rendered image of a 3D scene into a 2D viewport defined by the computer windows or canvas.

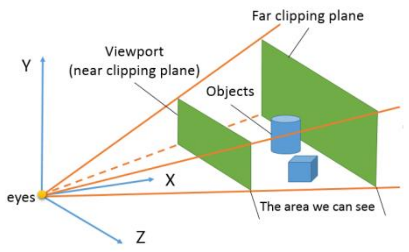


Figure 1. Depicts the Core Concepts of the Camera, Viewport and Projection

From Figure 1, the blue cube and the blue cylinder are the objects in Web3D scenes, and these two green planes define the boundaries of a subset of the 3D space, known as the view frustum. Only objects within the view frustum are actually rendered to the screen. The near clipping plane is equivalent to the viewport, where you will see the final rendered image.

3.1. Collision Detection

The illusion of movement in a virtual roam that People feel is caused by the change of camera's position and orientation So collision detection often occurs between the camera and exhibitions or walls in virtual roam. From Figure 1, the distance between eyes and viewport may be a little far, but in fact the distance is less than one relative unit.

In this chapter, the sphere box is used to surround the camera, which is also like eyes to observe the virtual world. From Figure 2, the red sphere box is not solid but has wireframe, which contains many points. These points can be useful in the algorithm of collision detection and correction. The Three.js provides the controls that people can look around without changing the sphere box.

Assume the position of camera is (x_0, y_0, z_0), and one vertex on the sphere is (x_1, y_1, z_1). A plane comes through the point(n_1, n_2, n_3), the normal vector of this plane is (v_1, v_2, v_3). The radius of sphere is r. This paper uses a space straight line, that formed by the position of vertex and camera, to judge whether the vertex collides with exhibitions. The following is the process of collision detection:

① According to the two coordinate points (the position of camera $(x_0, y_0, z_0)$and the vertex (x_1, y_1, z_1) on the sphere box), a space straight line is obtained and get the equation:

$$\frac{x-x_0}{x_1-x_0} = \frac{y-y_0}{y_1-y_0} = \frac{z-z_0}{z_1-z_0} = t \tag{1}$$

② According to equation (1), get the answer equation:

$$\begin{cases} x = t * (x_1 - x_0) + x_0 \\ y = t * (y_1 - y_0) + y_0 \\ z = t * (z_1 - z_0) + z_0 \end{cases} \tag{2}$$

③ According to the coordinate point (n_1, n_2, n_3) and the normal vector (v_1, v_2, v_3), get the plane's equation:

$$v_1 * (\, x - n_1) + v_2 * (\, y - n_2) + v_3 * (\, z - n_3) = 0 \tag{3}$$

④ According to equation (1), (2), (3), get the equation:

$$t = \frac{-(v_1 * x_0 + v_2 * y_0 + v_3 * z_0)}{v_1 * (x_1 - x_0) + v_2 * (y_1 - y_0) + v_3 * (z_1 - z_0)} \tag{4}$$

⑤ If t has a real answer, it proves that the line and the plane have an intersection point, go to⑥. Otherwise, they are parallel with each other, then it needs a new point from the sphere box, go to⑦.

⑥Obviously, only one intersection point does not prove that the collision occurs between the sphere box and exhibitions. According to equation (2) and t, intersection point(x_c, y_c, z_c) can be calculated. Then the distance from the camera to the intersection point, also can be calculated by the equation:

$$d = \sqrt[2]{(x_0 - x_c)^2 + (y_0 - y_c)^2 + (z_0 - z_c)^2} \tag{5}$$

If $d \le r$, the collision occurs at the point that it needs to correct the position next. The Chapter 3.2 discusses how to correct the collision. If $d > r$, the collision is not occur at this point, then it needs a new point from the sphere box, go to①. When all points are detected, it is clearly that there is a collision between sphere and exhibitions or not.

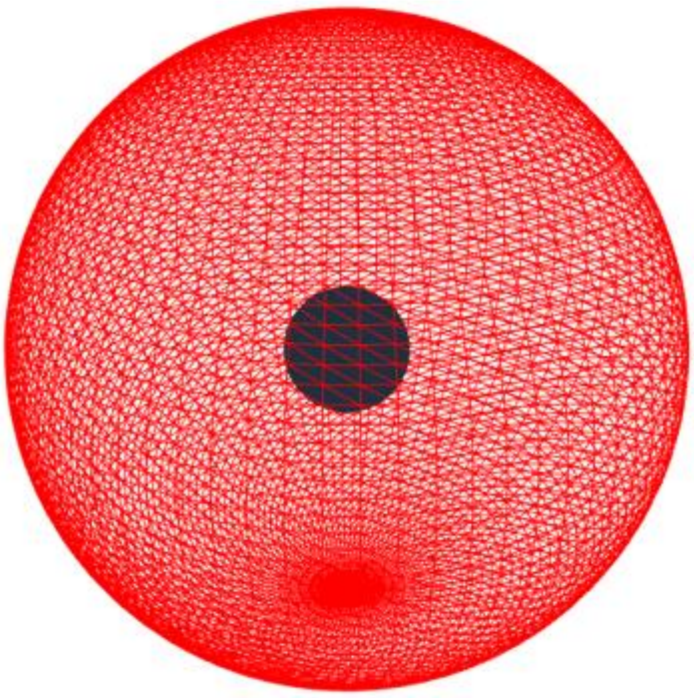


Figure 2. The Blue Sphere is Camera and the Red Sphere is the Sphere Box

In summary, the accuracy of collision detection depends on the number of vertices on the sphere box. If the number is huge, it will certainly influence the speed of algorithm. The Chapter 4 discusses the influence of the vertices' number to collision detection. When all the collision points are found, the Chapter 3.2 will fix the position of the camera to ensure that the sphere box do not collide with other objects.

3.2. Collision Correction

This chapter proposes a method of collision correction. The Chapter 3.1 has just mentioned that if the vertex is detected the collision, then it needs to be used the algorithm of collision correction.

From Figure 3, the blue cube can be regarded as an exhibition, which is an object in Web3D scene. Assume the green point A is one vertex on the sphere box, and the green point B is the new position of point A after the movement. The red point is the new position after using the correction algorithm. However, how can get the position of the red point, and the following is the process of collision correction:

①Assume the position of the point A is (x_0, y_0, z_0), and the position of the point B is (x_1, y_1, z_1), get the direction of movement, equation:

$$\begin{cases} diffX = \frac{x_1 - x_0}{|x_1 - x_0|} \\ diffY = \frac{y_1 - y_0}{|y_1 - y_0|} \\ diffZ = \frac{z_1 - z_0}{|z_1 - z_0|} \end{cases} \tag{6}$$

②Calculate the angle θ_1 between the moving direction and the Y axis, equation:

$$\tan\theta_1 = \frac{\sqrt[2]{(x_1 - x_0)^2 + (z_1 - z_0)^2}}{y_1 - y_0} \tag{7}$$

③Calculate the angle θ_2 between the moving direction's projection on the x-z plane and the X axis , equation:

$$\tan\theta_2 = \frac{x_1 - x_0}{y_1 - y_0} \tag{8}$$

④Assume that dis is the depth that the point A enter into the exhibition (the black line in Figure3). According to equation (6), (7), (8), modified value can be calculated by the equation:

$$\begin{cases} updateX = dis * \sin\theta_1 * \cos\theta_2 * diffX \\ updateY = dis * \cos\theta_1 * diffY \\ updateZ = dis * \sin\theta_1 * \sin\theta_2 * diffZ \end{cases} \tag{9}$$

When a vertex is fixed, it may cause collision between other vertices and exhibitions. So before using the collision correction, the dis value is chosen to determine the sequence to use the algorithm of collision correction. When the sphere box completely leaves all exhibitions, this algorithm has finished its job.

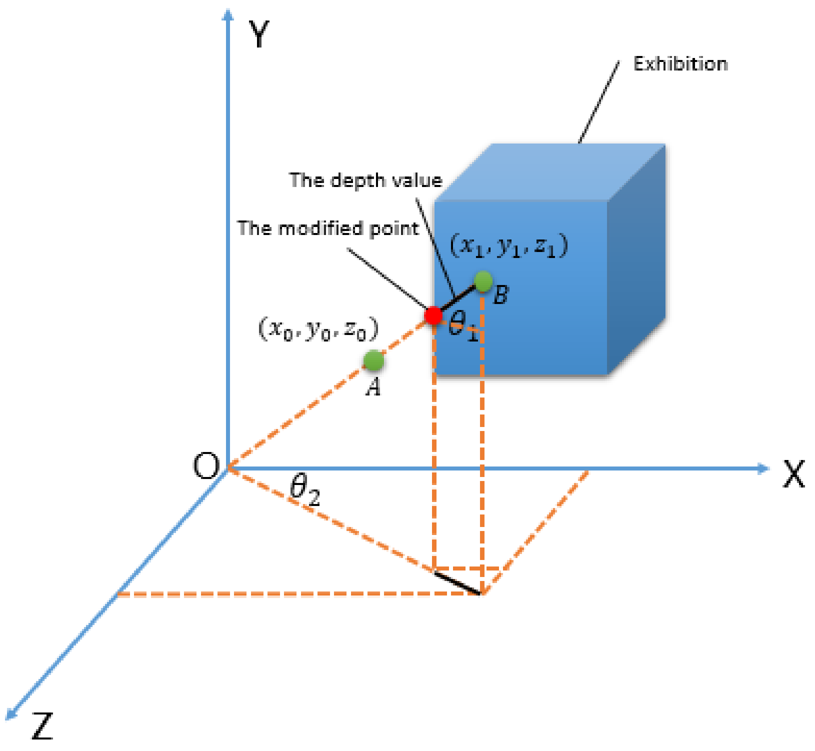


Figure 3. Analysis Diagram of Collision Correction

In summary, this chapter discusses the algorithm of collision correction. The main idea of this algorithm is how to calculate the right position that all the vertices can just leave the exhibitions. The Chapter 4 also did some experiments to verify the feasibility of this algorithm.

4. Experimental Results

Three.js is a lightweight cross-browser JavaScript library/API, so this algorithm is achieved by the language of JavaScript. The experiments were done on an Intel Core i7-3630 2.4GHz PC with 8GB main memory and NVIDIA GEFORCE 650M Graphics. In experimental Web3D scene, there are a JSON model (see Figure 4), two cuboid models (see Figure 5) and the sphere box, which has the radius of 20 relative units. This paper defines three types of collisions, which are head-collision, side-collision and jump-collision, to test this algorithm. From Figure 6, these pictures represent the state before and after the collision between the sphere box and exhibitions.

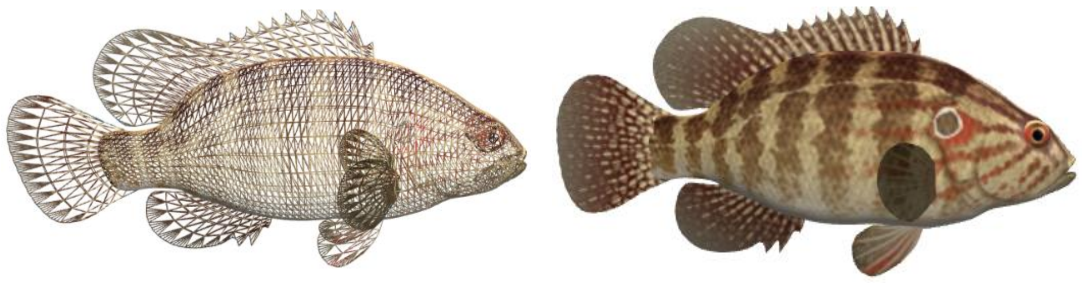


Figure 4 The JSON Model. The First One Shows the Wireframe of the Model and the Second Shows Model with Material

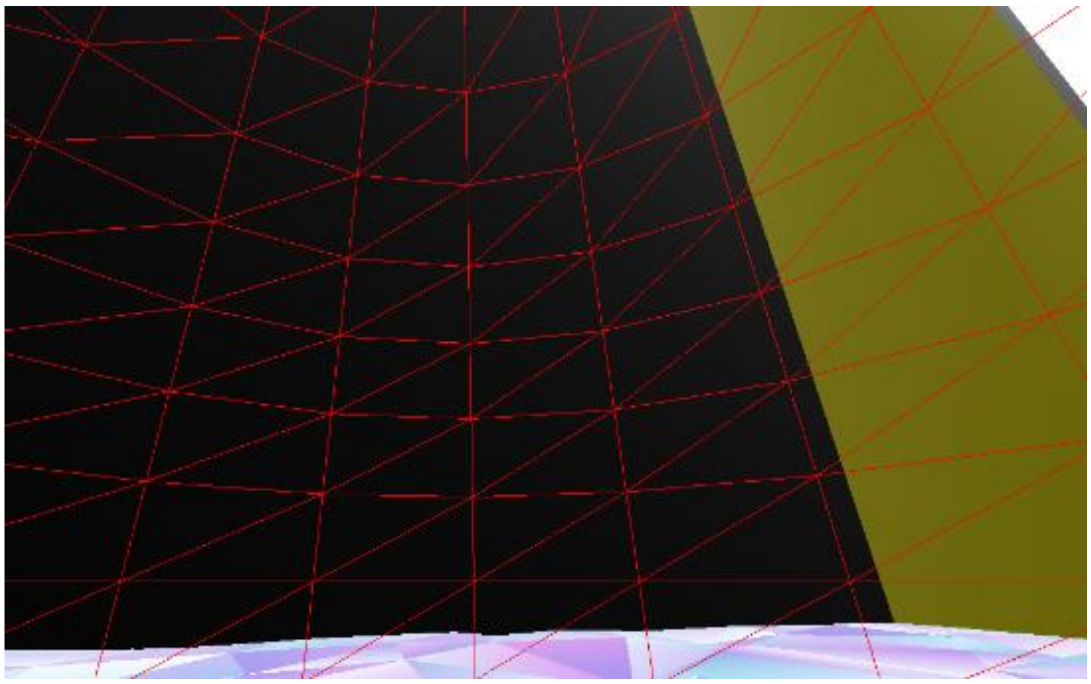


Figure 5. The Cuboid Models (size: 500*500*100)

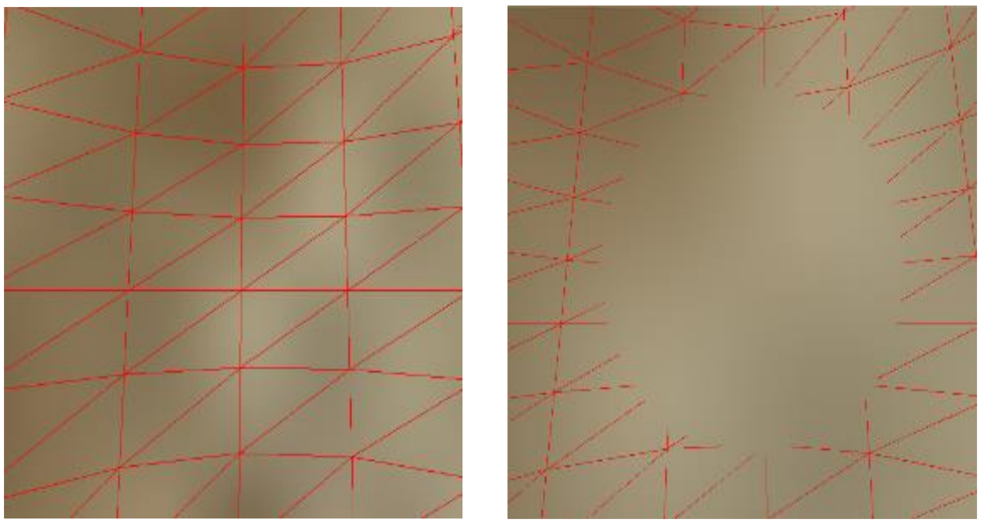


Figure 6. Before Collision and After Collision

In Experiment 1, this test focused on testing the collision between the sphere box, which contains different number of points, and cuboid model. Each kind of collision was tested one hundred times. The program counted the points, which already into the exhibition, to make the line chart (see Figure 7). From Figure 7, it shows that the number of points on the sphere box is lager, the more points should be detected, and jump-collision needs to detect more points than others. Whether head-collision or side-collision, it only has a certain moving direction in x-z plane, but jump-collision has a moving direction in the 3D space.

By counting the failure times, the histogram was finished. From Figure 8, it shows that the number of points on the sphere box is lager, the failure times is less.

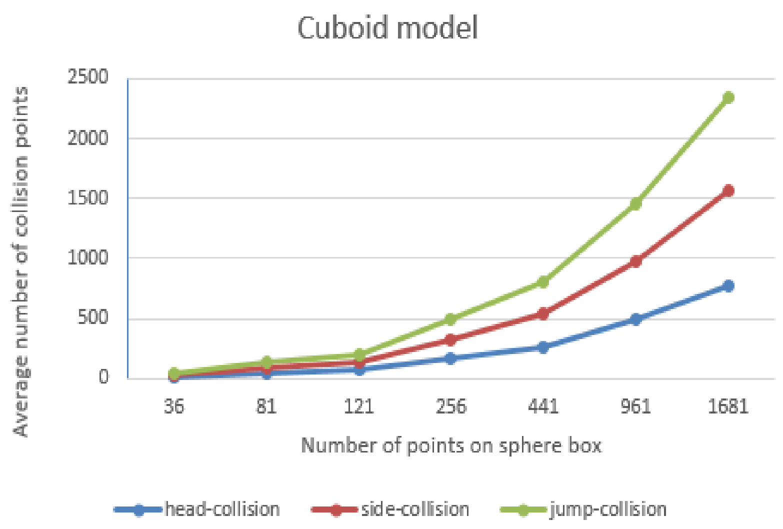


Figure 7. The Average Number of Sphere Box Need to Detect (Cuboid Model)

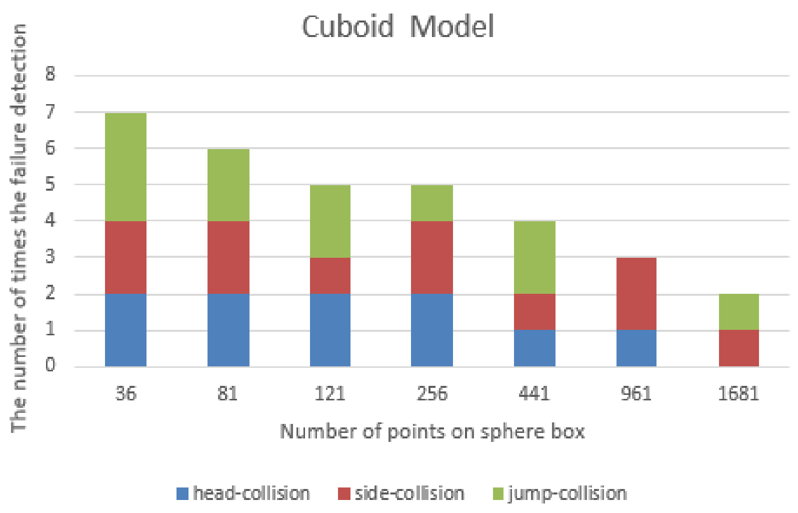


Figure 8. The Chart of Failure Detection (Cuboid Model)

In Experiment 2, this test focused on testing the collision between the sphere box, which contains different number of points, and JSON model. Each kind of collision was also tested one hundred times. According to the test data, the Figure 9 and the Figure 10 were completed.

By comparing Figure 7 and Figure 9, it shows that the points that collide with JSON model are more than cuboid model's points. From Figure 8 and Figure 10, whether cuboid model or JSON model, it shows that the number of points on the sphere box is lager, the failure times is less. But the JSON model's failure times are a little higher than cuboid model's. Since the cuboid is regular, the sphere box can be collided and corrected more easily.

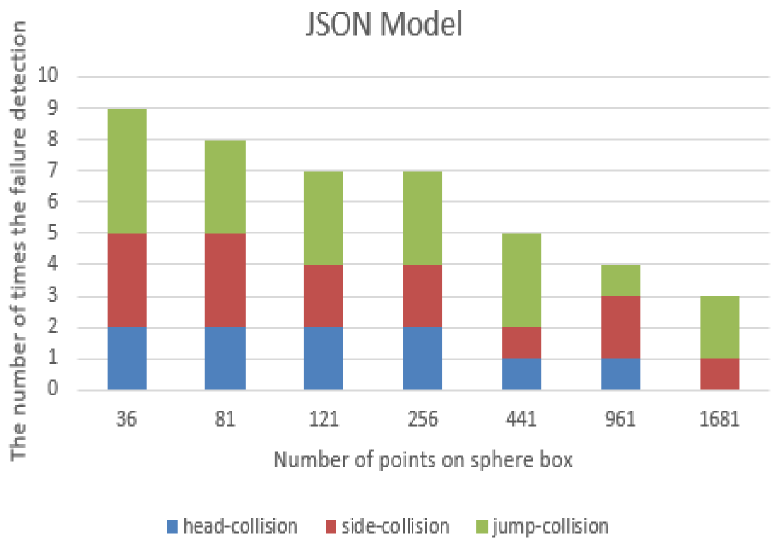


Figure 9. The Average Number of Sphere Box Need to Detect (JSON Model)

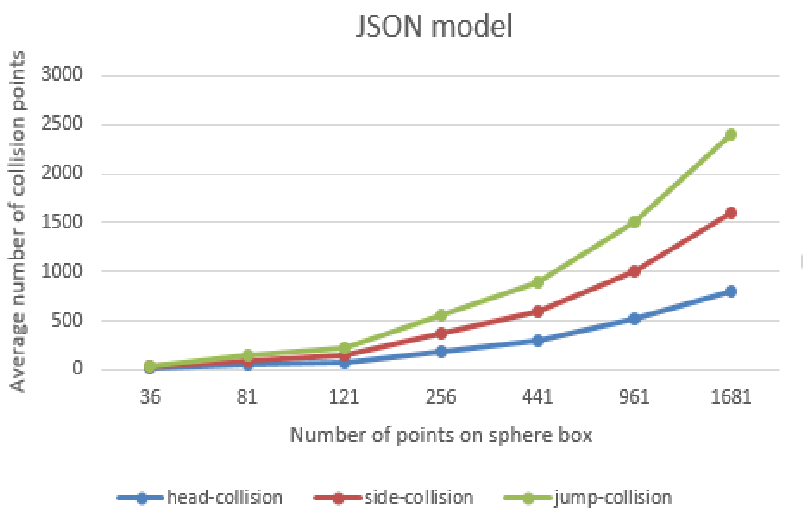


Figure 10. The Chart of Failure Detection (JSON Model)

In summary, this chapter uses two experiments to discuss the influence of the number points on the sphere box. Because of the browser's limited resources and the experiment results, this paper suggests that the number of points on the sphere box should be less than 400. This paper also suggests using a transparent and suitable cuboid to surround the JSON model. In that way, the number of failure times can be reduced. The chapter 5 shows a virtual roam example that people can walk on the ship in the Web3D scenes.

5. Verification Example

This chapter introduces a verification example, which uses HTML5, WebGL (Three.js) and the algorithm of collision detection and correction. Virtual roam in Web3D scene is designed with the advantages of good compatibility, free plug-in, depending on the GPU. The main part of this example's Web3D scene is a cargo ship (see Figure11). People can use keyboard and mouse to walk and look on this ship (see Figure12-15). In real life, people seldom have a chance to see the real one, but virtual roam give people a chance to experience it.

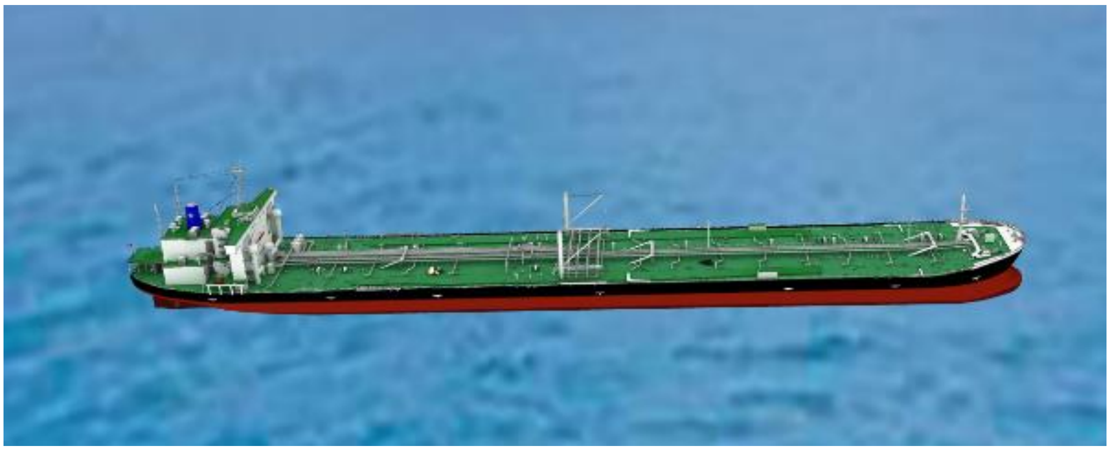


Figure 11. The Example of Web3D Scene

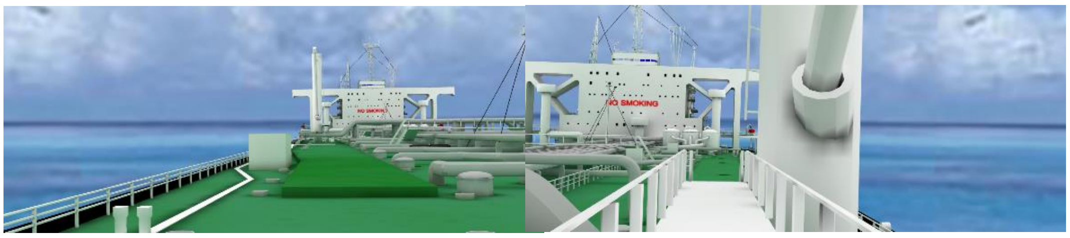


Figure 12, 13. Virtual Roam on the Ship (1-2)

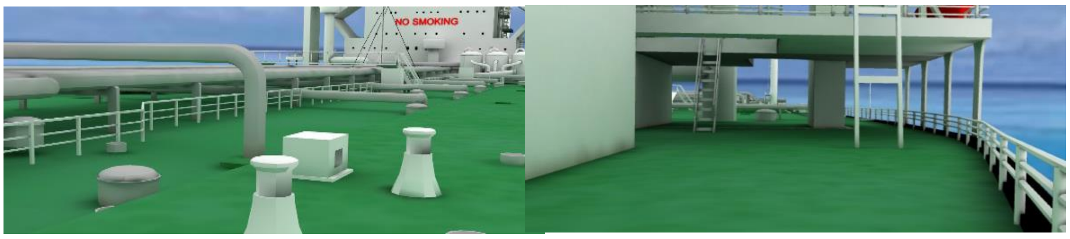


Figure 14, 15. Virtual Roam on the Ship (3-4)

In order to help people get a better experience in virtual roam, hot point that can provide more information and interactions is also designed.

6. Conclusions

Web3D is a practical, useful, professional, and comprehensive technology, which plays a more and more important role in the development of the Internet, and will gradually replace the 2D graphics technology. It is sure that Web3D technology will bring a revolutionary change on the Internet applications in future. This paper uses HTML5, WebGL (Three.js) and the algorithm of collision detection and correction to design a solution of virtual roam in Web3D scenes. In chapter 5, verification example has the advantages of good compatibility, free plug-in, depending on the GPU. Virtual roam not only can give people the immersive feeling in the Web3D scenes, but also can be easily operated through the browser.

In future work, authors plan to focus on the research of the animation, path navigation and new interactions in the virtual roam.

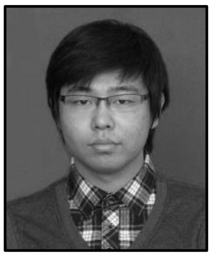
Acknowledgments

This paper is supported by Foundation of Science and Technology Commission of Shanghai Municipality under Grant No. 14590500500, Natural Science Foundation of Shanghai under Grant No. 15ZR1415200, and Young University Teachers Training Plan of Shanghai Municipality under Grant No. ZZSD13008

References

[1] Web3D Consortium, “The virtual reality modeling language”, International Standard ISO/IEC, **(1997)**, pp. 147-721.
[2] “Visualizing Information Using SVG and X3D: XML-based Technologies for the XML-based Web”, Springer Science & Business Media, **(2005)**.
[3] Web3D.X3D; URL: http://www.web3d.org/x3d/what-x3d, **(2013)**.
[4] Unity.Unity3D; URL: http://www.unity3d.com, **(2013)**.
[5] Oracle.Java3D; URL: https://java3d.java.net/, **(2008)**.
[6] Adobe. Stage3D; URL: http://www.adobe.com/devnet/flashplayer/stage3d.html, **(2011)**.
[7] Microsoft. Direct3D; URL: https://msdn.microsoft.com/en-us/library/windows/desktop/ee663274, **(2007)**.
[8] M. Lin and S. Gottschalk, “Collision detection between geometric models: A survey”, Proc. of IMA conference on mathematics of surfaces, **(1998)**.
[9] P. Jiménez, F. Thomas and C. Torras, “3D collision detection: a survey”, Computers & Graphics, vol. 25, no. 2, **(2001)**, pp. 269-285.
[10] X. Zhang and Y. J. Kim, “Interactive collision detection for deformable models using streaming AABBs”, Visualization and Computer Graphics, IEEE Transactions, vol. 13, no. 2, **(2007)**, pp. 318-329.
[11] S. Gottschalk, M. C. Lin and D. Manocha, “OBBTree: A hierarchical structure for rapid interference detection”, Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. ACM, **(1996)**.
[12] J. T. Klosowski, M. Held and J. S. B. Mitchell, “Efficient collision detection using bounding volume hierarchies of k-DOPs”, Visualization and Computer Graphics, IEEE Transactions, vol. 4, no. 1, **(1998)**, pp. 21-36.
[13] P. M. Hubbard, “Collision detection for interactive graphics applications”, Visualization and Computer Graphics, IEEE Transactions, vol. 1, no. 3, **(1995)**, pp. 218-230.
[14] W3C.HTML5 Specification; URL: http://www.w3.org/TR/html5/, **(2009)**.
[15] Khronos. WebGL specification; URL: http://www.khronos.org/registry/webgl/specs/latest/1.0/, **(2011)**.
[16] R. Cabello, Three.Js: URL: http://threejs.org/, **(2010)**.
[17] S. Jourdain, U. Ayachit and G. B. Paraviewweb, “A web framework for 3d visualization and data processing”, IADIS international conference on web virtual reality and three-dimensional worlds. **(2010)**.
[18] A. Evans, M. Romeo and A. Bahrehmand, “3D graphics on the web: A survey”, Computers & Graphics, vol. 41, **(2014)**, pp. 43-61.
[19] T. Parisi, WebGL: up and running, “O'Reilly Media, Inc.", **(2012)**.

Authors



Li Ruizhi, he is a Master candidate in graduate students in School of Computer Engineering and Science Shanghai University. His main research is about 3D visualization and computer vision. He works at the Information Technology Office of Shanghai University during the study of Master.