EPJ Web of Conferences **68**, 00035 (2014)
DOI: 10.1051/epjconf/20146800035

Comparison of Physics Frameworks for WebGL-Based Game Engine

Resa Yogya and Raymond Kosala[a]

School of Computer Science, Bina Nusantara University, Jakarta, Indonesia

Abstract. Recently, a new technology called WebGL shows a lot of potentials for developing games. However since this technology is still new, there are still many potentials in the game development area that are not explored yet. This paper tries to uncover the potential of integrating physics frameworks with WebGL technology in a game engine for developing 2D or 3D games. Specifically we integrated three open source physics frameworks: Bullet, Cannon, and JigLib into a WebGL-based game engine. Using experiment, we assessed these frameworks in terms of their correctness or accuracy, performance, completeness and compatibility. The results show that it is possible to integrate open source physics frameworks into a WebGL-based game engine, and Bullet is the best physics framework to be integrated into the WebGL-based game engine.

1 Introduction

The growth of internet is very fast, people can access internet easily nowadays. This goes the same for computer games. There are some kinds of games that people can play via their internet browser, which are often known as browser-games [1]. One of the most popular technologies that are used to develop browser games is Flash [2]. Flash games offer interactive gameplay but the drawback is that the player has to download the plugin first before he or she can play the game.

Recently, physics based casual games often get high ratings, such as Angry Birds [3] and Cut the Rope [4] for example. There are many more games that use physics as one of its features. It is true that using physics does not guarantee that the game will be successful. However by simply featuring physics, the gameplay can be richer and more appealing to the users. Based on this fact, it can be said that physics plays an important role in games.

Recently, a web rendering technology called WebGL [5] was introduced. This technology is similar to OpenGL, but it can run in internet browsers. The advantage of using this technology is that people do not need to download the plugin first to run WebGL application, which looks very promising for deploying games on website.

People will no longer need to download plugin to play the game. Furthermore, it is cross-platform so there will be no additional work to port the game into desired platform, for instance the internet browsers.

To develop the games rapidly, the existence of game engine is required. However, since WebGL technology is still new, currently there is no dedicated game engine that uses this technology. Fortunately, there are already some frameworks for WebGL. These frameworks can be used to develop a game engine, but maybe not all of them are suitable to be used for the game engine.

Therefore, the aim of this paper is to do a comparative study of existing physics frameworks for WebGL-based game engine. In this paper, we specifically compare three physics frameworks: cannon.js, bullet.js, and jiglib.js. The scope of this study is as follows. The first is to do compatibility testing of each framework with the game engine. The second is to test the correctness of rigid body (box & spheres) physics. The third is to compare the completeness of physics features. Finally, is to do performance testing of each framework in actual application.

To accomplish this, a prototype game engine will be created. The game engine consists of rendering engine that uses WebGL, core engine, and physics engine that uses the frameworks that will be tested. For the experiment, we prepare test cases that include performance test, compatibility test, correctness test, and completeness observation. After the engine is deployed on the internet browsers, then test cases will be carried out. Finally, the result of each physics framework can be obtained and analyzed.

2 Preliminaries

WebGL [5] is a cross-platform 3D graphics library for web that makes use of HTML5 canvas element. One major advantage of WebGL compared to other web rendering technology, such as Flash and Microsoft

[a] Corresponding author: rkosala@binus.edu

Silverlight, is that WebGL is plug-in free, which allows the user to run the application without having to install additional software/plug-in. The stable release of this technology was released after February 2011, which is not really new but currently this technology is still not a W3C (World Wide Web Consortium) standard.

WebGL was developed by Khronos Group, the organization that develops the OpenGL, therefore there are many similarities between OpenGL and WebGL. More specifically, WebGL is based on OpenGL for Embedded System 2.0 (OpenGL ES 2.0), which in turn is a stripped down version of OpenGL 2.0 that allows OpenGL to run on embedded platforms.

At the moment, most major internet browsers already support WebGL [6]. Mozilla Firefox 4.0++, Opera 12 and Google Chrome already support WebGL by default, however in Safari, it is disabled by default so the user will have to enable it manually. Currently Internet Explorer does not support WebGL. Some mobile user can use WebGL but there may be a slight incompatibility due to their hardware capability.

According to Gregory [7], game engine is software that is extensible and can be used as the foundation for many different games without major modification. Some examples of game engine are Unreal Engine [8], Irrlicht [9], Unity3D [10].

Physics engine, the main focus of this research, is one of the components of game engine that is responsible for managing and handling all physics related functions. In general, physics engine that is used in game engine is often adapted from commercial physics engine developed by the third party. Two examples of popular commercial physics engines are NVIDIA PhysX [11] and Havok [12]. The alternative would be to develop physics engine based on exiting physics frameworks. We differentate between Physics framework and Physics engine. Physics framework is a library that provides low level physics functions, while Physics engine provides a higher level of interface to the user.

Physics engine or physics framework must include two main functionalities: collision detection and collision response/handling. Up until this moment, there are two popular physics theories, which are Newtonian physics and rigid bodies. Newtonian physics is based on Newton's laws of motion, while rigid bodies assume that the objects are solid and not deformable. Rigid body physics becomes popular because it greatly simplifies the calculation required and gives acceptable result.

Some advanced features of physics are ragdoll physics [13], soft body dynamics, cloth physics, hair physics, fluid dynamics, water surface simulation. Rag doll physics. Ragdoll physics is usually used for dead people animation where the bodies goes limb. Soft body dynamics is like rigid bodies dynamics but for deformed objects. One of popular soft body dynamics implementation is spring [14].

Since WebGL API is written in javascript, theoretically all frameworks that are javascript based can be used as the physics engine. For this research, the frameworks that will be used as research objects are only the open source ones, so the results can be analyzed further by examining their code structure. There are few javascript based physics frameworks out there, but for this research we only experiment with three physics frameworks. Our criteria for choosing the frameworks are the popularity among game developer and the ability to model physics in 3D. The frameworks that are chosen are the following.

The first one is Bullet [15], which is originally written in C++ but recently there is third party software called kripken/emscripten [16] that can port it into javascript. Bullet is one of the well known open source physics frameworks among game developers, and used on film industry as well. Nevertheless, its performance after ported into javascript is not fully known yet.

The second one is JigLibJS [17], which is another open source physics frameworks that can be used for WebGL. It is already in ported into javascript format so there is no need to port the code first. Based on the demo, this framework shows decent result but this framework seems to be computationally intensive.

The final one is Cannon.js [18], which was written from scratch, and is claimed to be light weight. There seems to be lack of documentation of this framework at the moment, however the demo shows its capability on handling rigid bodies physics. This framework is interesting because it claims it is light weight, which is a big plus in development aspect.

3 Game Engine Architecture and Experiment Design

Figure 1 shows our game engine architecture. In this architecture, there are some components but the most important one is the core game engine. This game engine was developed using some existing physics frameworks that were tested in this research. After the game engine is developed, test applications can be generated and finally run on web browsers. To facilitate the testing, a user interface for the game engine, which can be seen from Figure 2, was developed.

The game engine in this project is composed of three main components: core engine, rendering engine, and physics engine. Core engine is responsible for managing memory and acts as the main controller of any other components. Rendering engine is responsible for displaying the view to the user, and in this game engine, the rendering engine is using WebGL technology. Lastly, the main focus in this research is the physics engine. The physics engine is developed using physics frameworks that will be tested in this research.

To facilitate the research, the physics engine will provide general interface to the physics frameworks, so the user can simply use the interface function and choose which framework that will be used instead of directly using the functions that are provided by the frameworks. The advantage of this approach is that the user will not need to change the code if he/she wants to change the physics frameworks that will be used. After the game engine is developed, an application/game can be developed and deployed in the web browsers.

For the experiment, some simple test applications will be developed and run in the web browser to test the

physics frameworks. The test includes performance test, compatibility test, correctness test, and completeness observation. One application will include the performance and correctness test for every framework. For the compatibility and completeness observation, each framework will be tested in separate test applications. Because every framework has varying features, they cannot be tested in same application. These test applications will be run on internet browsers.

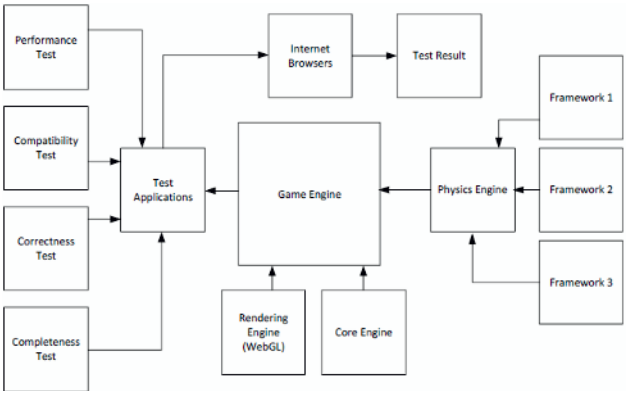

Figure 1. The Prototype Game Engine Architecture.

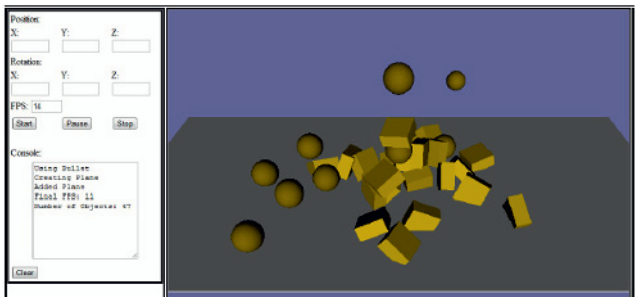

Figure 2. The Game Engine User Interface.

Performance Testing. To test the physics framework's performance, a test application that can generate a physics object continuously was developed. When the application run, there will be no objects yet. The application will generate a physics object every one second interval and the FPS (Frame per Second) rate is monitored. If the FPS rate is dropped into less than 12, the test will be stopped.

Correctness Testing. For this test, first the test application is run. There will two objects inside this application, one of them is static on the ground and another one is in the air. The position of these objects will be updated from time to time until they finally collide with each other. When two objects collide, the collision will be handled. Finally, the velocities of the objects are recorded to be analyzed for its correctness by using a relevant physics formula later on.

Completeness Observation. For the completeness observation, first, additional functions that are provided by the physics frameworks are listed. After that, each of those functions will be tested whether it can be used or not. The test result will be recorded and it will be used to determine the completeness of the physics frameworks. Since there is no limit to completeness, the completeness will be determine based on commonly used physics features only. Those features are: plane, sphere, box, capsule rigid body, ray cast, constraint, ragdoll, cloth, soft body, water surface physics.

Compatibility Test. In this test, first an interface function will created inside the game engine code. This function will wrap the physics functions from the physics frameworks. After the function is integrated, compile time check will be carried out to test whether there is any conflicting code or not. The error will be recorded and the function will be removed if there is an error. If there is no compile time error, run time checking will be carried out. Again, any error will be recorded, and the test will be concluded.

4 Experiment Results

For the experiments, the following is the hardware and software specification.

Processor : Intel(R) Core(TM) i7 CPU @ 1.20 GHz
RAM : 4 GB
VGA adapter: GeForce GT 335M 1GB memory
Operating System: Windows 7 SP 1 64-bit

As mentioned before, currently Mozilla, Chrome, Safari and Opera browsers support WebGL. However, Opera was removed at the later stage of this research because the current rendering engine cannot be run on Opera. Another removal is the Safari browser as only the Mac version of Safari browser that is able to run WebGL. Therefore only two browser platforms were used: Mozilla Firefox 12.0, and Google Chrome 19.0.1084.52 m.

Performance Test result. The results of this test were obtained by calculating the average of maximum number of objects before the FPS dropped into 12 and can be seen from Table 1.

From Table 1, we can see that the performance was better in Google Chrome rather than in Mozilla Firefox. This is to be expected because Chrome can interpret javascript language faster that Firefox and since all physics code are written in javascript Chrome has advantage over this. Statistically, Chrome can perform up to 300% faster than Firefox.

For the performance of the physics frameworks itself, overall, cannon.js performance was a little bit faster compared to bullet.js. If we look carefully from the result, cannon.js can perform faster in calculating sphere rigidbodies compared to bullet.js based on the result from scenario 1 and 3. For box rigidbodies, it seems that both cannon.js and bullet.js performance is similar; cannon.js performs better in Firefox, and bullet.js performs better in Chrome based on result from scenario 2 from table above. By average, cannon.js performs 13.88% faster than bullet.js.

Performance wise, both physics framework can be used as the physics engine for game in WebGL, however based on the result above, it seems that it is not favorable

to deploy a game with a lot of physics objects in Firefox web browser.

Table 1. The Performance Test Summary

Platform	Scenario #	Framework	Average maximum number of objects
Firefox	1	cannon.js	18.6
Firefox	1	bullet.js	15.2
Chrome	1	cannon.js	48.2
Chrome	1	bullet.js	40.6
Firefox	2	cannon.js	26.4
Firefox	2	bullet.js	22.6
Chrome	2	cannon.js	50
Chrome	2	bullet.js	52.4
Firefox	3	cannon.js	23.2
Firefox	3	bullet.js	18.4
Chrome	3	cannon.js	54
Chrome	3	bullet.js	52

Correctness Test result. From Table 2 and 3, we can see that bullet.js has more accurate physics. Cannon.js seems to perform well in box handling, but not on sphere. Bullet.js can handle both box and sphere very well. The error values from both framework are acceptable (less than 1) except for sphere in cannon.js. The sphere in cannon.js does not rotate where it should rotate. We also found that the physics simulated by Bullet.js is stable, in the sense that they have the same result no matter how many times the tests were carried out. However cannon.js was a bit unstable in the sense that the results varied on each attempt. This maybe because the cannon.js was not using continuous collision detection, so the collision point varied on each run. On the other hand, bullet.js was using continuous collision detection so the collision points were always the same; therefore the result was always the same.

Completeness Observation result. From the result in Table 4, it seems that bullet.js fulfill all requirements for commonly used physics in game, except cloth, water surface, and softbody physics. This is because Bullet.js has been developed for some time already, while cannon.js was still new. So it is to be expected that bullet.js has more complete features. Even so, cannon.js provides most basic features of physics that should be sufficient for simple games development.

Compatibility Test result. Based on the result on Table 5, only two out of three frameworks are compatible with our game engine. Note that the asterisk sign (*) in Table 5 indicates that there is no actual function (API) provided by the framework, but the problem can be solved by creating a function, in the game engine, that accesses the variable directly.

Both bullet.js and cannon.js had no trouble in compatibility, except that the cannon.js setting for the rigid body rotation was still not working properly. The most likely reason for this is that there was a bug in the framework. Overall, bullet.js worked quite well and cannon.js was still lack of some APIs but still compatible to be used. However, jiglib.js was not compatible at all with our game engine.

To summarize, the framework compatibilities are as follows.

- Bullet.js : 100 % (15 out of 15 functions)
- Cannon.js : 86,67 % (13 out of 15 functions)
- JibLib.js : 0 % (0 out of 15 functions)

Table 2. The error on the box body objects

Framework	Platform	Average error					
		vx	vy	vz	ωx	ωy	ωz
cannon.js	Firefox	0.7132	0.4	0	0	0	-0.6476
cannon.js	Chrome	0.9242	0.4	0	0	0	-0.7936
bullet.js	Firefox	0.41	-0.128	0.001	0.001	0.004	-0.191
bullet.js	Chrome	0.41	-0.128	0.001	0.001	0.004	-0.191

Table 3. The error on the sphere body objects

Framework	Platform	Average error					
		vx	vy	vz	ωx	ωy	ωz
cannon.js	Firefox	-1.6256	-0.8	0	0	0	-3.1
cannon.js	Chrome	-2.0474	-0.8	0	0	0	-3.1
bullet.js	Firefox	-0.819	0.255	-0.003	0.001	-0.015	-1.673
bullet.js	Chrome	-0.819	0.255	-0.003	0.001	-0.015	-1.673

Table 4. The completeness observation result

Feature	Framework	
	bullet.js	cannon.js
Plane	√	√
User defined gravity	√	√
Sphere Rigidbody	√	√
Box Rigidbody	√	√
Capsule Rigidbody	√	X
Raycast	√	X
Ragdoll physics	√	X
Constraint	√	X
Cloth Physics	X	X
Softbody dynamics	X	X
Water surface physics	X	X

Table 5. The compatibility test result

Functions	Framework		
	bullet.js	cannon.js	jiglib.js
Instantiate World	√	√	X
Set Gravity	√	*√	X
Instantiate Plane	√	√	X
Instantiate Rigidbody	√	√	X
Instantiate Sphere	*√	√	X
Instantiate Box	√	√	X
Set Position	√	*√	X
Set Rotation	√	X	X
Simulate Physics	√	√	X
Set Velocity	√	*√	X
Set Angular Velocity	√	X	X
Get Position	√	*√	X
Get Rotation	√	√	X
Set Friction	*√	√	X
Set Restitution	*√	√	X

Based on the evaluation above, it can be said that bullet.js was the best physics framework in this research due to its accuracy, completeness and compatibility. Cannon.js was better in term of performance compared to bullet.js, and this framework shows a good potential if it is updated regularly in near future, especially bug fixes update and API update. It was unfortunate that JigLib.js could not be tested because it could not run in our game engine due to its incompatibility.

Conclusion

In this paper, we have shown that there are few open source physics frameworks that can be used as the component of WebGL-based game engine with acceptable accuracy and performance. This is very promising considering the physics frameworks that we tested were still lack of features and there were still many things that can be improved. WebGL game engine development might be slow due to lack of documentation and APIs from the frameworks. Another finding from this research is that Google Chrome seems to perform best in running WebGL application compared to Firefox browser.

In brief, it is recommended to use bullet.js if accuracy is critical, and use cannon.js if accuracy is not the main issue. As a reference, some of the game genres that usually need more accuracy are fighting, racing, and most FPS games. The games that require less accuracy include puzzle, RPG, and RTS games. From our observation, it seems that bullet.js is capable to be used for making racing games or simple fighting games, while cannon.js might be better to be used in puzzle or simple RPG games because cannon.js is lighter in term of computation cost.

Some possible future work include using capsule rigid body for testing, implementing more physics frameworks, and adding more scenarios for testing the correctness, especially testing objects with initial angular velocity, gravity and friction enabled.

References

1. D. Schultheiss, Long-term motivations to play MMOGs: A longitudinal study on motivations, experience and behavior, page 344. DiGRA (2007)
2. Statowl.com, Web Browser Plugin Market Share / Global Usage. (Retrieved April 2010).
3. Angry Birds, http://www.angrybirds.com/. (Retrieved February 2012).
4. Cut the Rope, http://www.cuttherope.ie/. (Retrieved February 2012).
5. Khronos Group, WebGL - OpenGL ES 2.0 for the Web, http://www.khronos.org/webgl/. (Retrieved April 2012).
6. A. Deveria. Can I use WebGL? http://caniuse.com/webgl. (Retrieved April 2012).
7. J. Gregory, *Foundations in Game Engine Architecture*. Massachusetts, ch. 1, pp. 3-55. (A K Peters, Ltd., United States of America, 2009)
8. UDK iOS Games. http://udk.com/mobile. (Retrieved April 2012).
9. Irrlicht Engine - A free open source 3D engine. http://irrlicht.sourceforge.net/. (Retrieved April 2012).
10. UNITY: Unity 3 Engine Features. http://unity3d.com/unity/engine/. (Retrieved April 2012).
11. PhysX - GeForce. http://www.geforce.com/hardware/technology/physx. (Retrieved May 2012).
12. Havok. http://www.havok.com/. (Retrieved May 2012).
13. G. Mulley and M. Bittarelli. Ragdoll Physics. http://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S07/final_projects/mulley_bittarelli.pdf. (2007)
14. J. Gästrin, Physically Based Character Simulation– Rag Doll Behaviour in Computer Games. Royal Institute of Technology, Stockholm. (2004)
15. Game Physics Simulation. http://bulletphysics.org/wordpress/. (Retrieved April 2012).
16. kripken/emscripten Wiki · GitHub. https://github.com/kripken/emscripten/wiki. (Retrieved April 2012).
17. JigLibJS. http://www.jiglibjs.org/. (Retrieved April 2012).
18. S. Hedman. http://schteppe.github.com/cannon.js/. (Retrieved April 2012).