Evaluation of Physics Frameworks for Building Web-Based Games

Resa Yogya
Binus University International
Bina Nusantara University
Jakarta, Indonesia

Raymond Kosala
Binus University International
Bina Nusantara University
Jakarta, Indonesia

***Abstract*— Recently, WebGL technology has shown a lot of potential for developing games. Since this technology is still relatively new, there is still much potential in the game development area that has not been explored yet. This paper explores the development of a game engine made with WebGL technology that integrates some physics frameworks for developing web-based 2D or 3D games. Specifically, we integrated three open source physics frameworks, which are Bullet, Cannon, and JigLib, into a WebGL-based game engine. We assessed these frameworks using some experiments, 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 a WebGL-based game engine.**

Keywords—physics frameworks; webgl; game engine; browser games

I. INTRODUCTION

The growth of the internet is very fast, and people can access the internet through mobile phones easily nowadays. The same thing goes for computer games that are played through the internet. Browser games are the kinds of games that people can play through their internet browsers [1]. One of the most popular technologies that are used to develop browser games is Adobe Flash, often just called Flash. Games developed using Flash technology offer interactive gameplay, but the drawback is that the player has to install the web browser plugin first before he or she can play the game.

In the recent years, physics-based casual games have often got high ratings, such as Angry Birds [2] and Cut the Rope [3]. Physics are also used as one of the features in many games recently, including the very popular Pokemon Go [4]. 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.

Also in recent years, a web rendering technology called WebGL [5] was introduced. This technology is similar to OpenGL, but it can run on internet browsers. The advantage of using this technology is that people do not need to install the plugin in the web browser to run the WebGL application, which is very promising for deploying games on a website. Furthermore, it is cross-platform so there will be no additional work to port the game into the desired platform, for instance the web browser.

The existence of a game engine is required to develop games efficiently. Currently, there are several WebGL-based game engines, for instance in [6]. WebGL-based game engines can be developed using some already available frameworks, for instance physics frameworks. However, not all of these physics' frameworks are suitable to be used for WebGL-based game engines.

Therefore, the aim of this paper is to do an evaluation about some existing physics frameworks that can be used to develop a WebGL-based game engine. In this paper, we specifically evaluate three physics frameworks: cannon.js, bullet.js, and jiglib.js. We performed the following four types of experiments to evaluate the physics frameworks. The first is to do compatibility testing of each framework with the game engine. The second is to test the correctness of box and sphere-shaped rigid body physics. The third is to compare the completeness of physics features. Finally, performance testing of each framework in the actual application was conducted. A prototype game engine and four test applications were created to carry out the experiments. The game engine consisted of a core engine, a rendering engine that uses WebGL, and physics engine that uses the frameworks that will be tested. After the test applications are loaded on the internet browsers, then test cases can be carried out. For the experiments, we have prepared several test cases that include performance testing, compatibility testing, correctness testing, and completeness observations. Finally, the test results of each physics framework can be obtained and evaluated.

The rest of this paper is organized as follows. In the next section, the relevant background and technologies related to WebGL and game engines will be explained. Next, the design of the game engine architecture, testing the application user interface, and test cases are presented. Then the experiments results and and evaluation are presented. Finally, the findings, their implications and future research directions are discussed.

II. BACKGROUND

A. WebGL Technology

WebGL [5] is a cross-platform 3D graphics library for the world-wide web that makes use of a HTML5 canvas element.

One major advantage of WebGL compared to other web rendering technologies, such as Flash and Microsoft Silverlight, is that WebGL is plug-in free, which allows the user to run the application without having to install additional software/plug-ins. The version 1.0 of this technology was released on March 2011 [7] and currently version 2.0 of the WebGL is still under development.

WebGL was developed by Khronos Group, the organization that develops OpenGL. Therefore, there are many similarities between OpenGL and WebGL. More specifically, WebGL 1.0 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 [7].

At the moment, most major internet browsers already support WebGL [8]. From Figure 1 and Figure 2, the browser support of WebGL has grown from around 45 percent in April 2012 to around 87 percent in July 2016. In April 2012, Mozilla Firefox 4.0++, Opera 12 and Google Chrome already supported WebGL by default. However, in Safari, it was disabled by default, so the user had to enable it manually, and Internet Explorer did not support WebGL. Some mobile users could use WebGL but there may be slight incompatibility issues due to their hardware capability. Now in July 2016, only Opera Mini does not support WebGL.

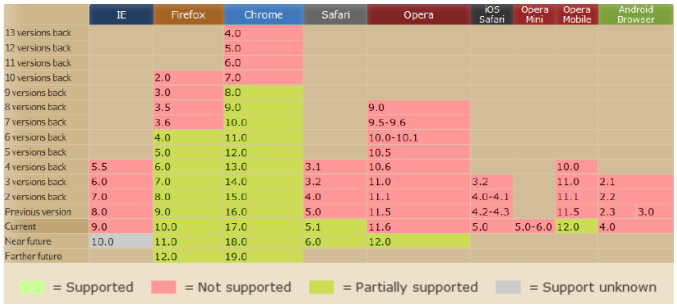

Fig. 1 WebGL support by browsers in April 2012 (Source: caniuse.com)

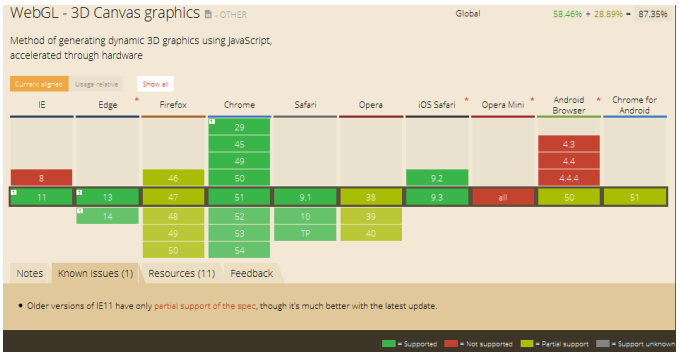

Fig. 2 WebGL support by browsers in July 2016 (Source: caniuse.com)

B. Physics Frameworks

According to Gregory [9], a 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 [10], Construct 2 [11], Unity [12].

A physics engine is one of the components of a game engine that is responsible for managing and handling all physics-related functions. In general, a physics engine that is used in a game engine is often adapted from a commercial physics engine developed by the third party. Two examples of popular commercial physics engines are NVIDIA PhysX [13] and Havok [14]. The alternative would be to develop a physics engine based on existing physics frameworks.

We differentiate between physics frameworks and physics engines. A physics framework, the main focus of this research, is a library that provides low level physics functions, while a physics engine provides a higher level of interface to the user. A physics engine or physics framework must include two main functionalities: collision detection and collision response or 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 objects are solid and not deformable. Rigid body physics has become popular because it greatly simplifies the calculations required and gives acceptable results.

Some advanced features of physics are ragdoll physics [15], soft body dynamics, cloth physics, hair physics, fluid dynamics, and water surface simulation. Ragdoll physics is usually used for dead people animation where the bodies go limp. Soft body dynamics is like rigid bodies dynamics but for deformed objects. One of the popular soft body dynamics implementations is 'the spring' in animating dead bodies' rigidity and lack of flex [16].

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 a 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 developers and the ability to model physics in 3D. The frameworks that are chosen are the following.

The first one is Bullet [17], which was originally written in C++ but recently there is third party software called kripken/emscripten [18] that can port it into JavaScript. Bullet is one of the well-known open source physics frameworks among game developers and is used in the film industry as well. Nevertheless, its performance after being ported into JavaScript is not fully known yet.

The second one is JigLibJS [19], which is another open source physics framework that can be used for WebGL. It is already ported into JavaScript format so there is no need to port the code first. Based on the demo, this framework shows decent results, but this framework seems to be computationally intensive.

The final one is Cannon.js [20], which was written from scratch, and is claimed to be lightweight. There seems to be a lack of documentation of this framework at the moment.

However, the demo shows its capability of handling 'rigid body' physics. This framework is interesting because it claims it is lightweight, which is a big plus from the development aspect.

C. Physics for Rigid Body Collision

When two objects collide, those objects will spin and their speed is changed. How big their speed is changed and how fast they spin can be calculated through a physics formula, specifically from an Impulse-Momentum formula. In this paper, the physics engine will be created based on Impulse-Momentum theory. More detail about this theory is described below.

Impulse is a force that acts over a short period of time [21]. From this definition, we can write a formula: impulse is equal to the average of force multiplied by delta time. However, this formula is not applicable for computer games because we do not know the delta time. For this reason, we need to use another formula that does not require time. That impulse formula is as follows:

$$J = \frac{-(1+e)\left((v_a - v_b)\bullet n + (r_a \times n)\bullet \omega_a - (r_b \times n)\bullet \omega_b\right)}{1/m_a + 1/m_b + (r_a \times n)\bullet([I_a]^{-1}(r_a \times n)) + (r_b \times n)\bullet([I_b]^{-1}(r_b \times n))} \quad (1)$$

Where, J = impulse (scalar), also denoted as Λ, e = coefficient of restitution of an object, v = linear velocity (vector), n = unit surface normal vector, r = distance from center of mass to collision point, ω = angular velocity (vector), m = mass, and *[I]* = inertia tensor.

The coefficient of restitution is a scalar value that tells us how much of the incoming energy is dissipated during the collision [22]. This value depends on the material of two colliding objects. For example, a basketball has a coefficient of restitution of 0.75 against a hard-wooden floor.

Moment of inertia is a quantitative measure of the radial distribution of the mass of a body about a given axis of rotation [21]. Inertia tensor is actually just a matrix containing the value of moment of inertia from three axis: *x,y,z* (denoted by *Ixx*, *Iyy*, *Izz*). The general formula for calculating Moment of Inertia is:

$$I = mr^2 \quad (2)$$

Where, I = moment of inertia, m = mass of rigid body, r = distance from center of mass to the collision point. The concept for moment of inertia is simple. However, for calculating the moment of inertia of a rigid body, the formula is different depending on the form of the rigid body and which axis it is facing. Some common rigid body forms are cube, rectangle, sphere, cone, and cylinder. For uniform objects like spheres or cubes that we use in this paper, the value of moment of inertia from any axis is same. Therefore, we only need to calculate moment of inertia from one axis for these objects.

After we calculate the impulse, we can finally use the Impulse-Momentum formula to calculate the linear and angular velocity of the rigid body after collision. The formula used to calculate linear and angular velocities after collision is as follows:

$$m_1 v_{1\,After} + m_2 v_{2\,After} = m_1 v_{1\,Before} + m_2 v_{2\,Before} \quad (3)$$

If the second object is static, e.g. the ground, the formula can be written as follows:

$$m_1 v_{1\,After} = m_1 v_{1\,Before} + \Lambda \quad (4)$$

$$m_2 v_{2\,After} = m_2 v_{2\,Before} - \Lambda \quad (5)$$

$$I_1 \omega_{after} = I_1 \omega_{before} + \Lambda(r_1 \times n) \quad (6)$$

$$I_2 \omega_{after} = I_2 \omega_{before} - \Lambda(r_2 \times n) \quad (7)$$

$$\Lambda = \Lambda n \quad (8)$$

Where, m = mass, V_{before} = velocity before collision (vector), V_{after} = velocity after collision (vector), Λ = impulse (vector), Λ = impulse (scalar), also denoted as J, n = unit surface normal vector, I = moment of inertia (scalar), ω = angular velocity (vector), and r = distance from center of mass to collision point.

III. GAME ENGINE ARCHITECTURE

Figure 3 shows the 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 was developed, test applications can be generated and finally run on web browsers. A user interface for the game engine and test applications, which can be seen from Figure 4, was developed to facilitate the testing.

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

To facilitate the research, the physics engine was developed to provide a 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 does not need to change the code if he/she wants to change the physics frameworks that will be used. After the game engine was 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 testing, compatibility testing, correctness testing, and completeness observations. One application will include the performance and correctness test for every framework. For the compatibility and completeness observations, each framework will be tested in

separate test applications. Because every framework has varying features, they cannot be tested in using one test application. These test applications will be run on internet browsers.

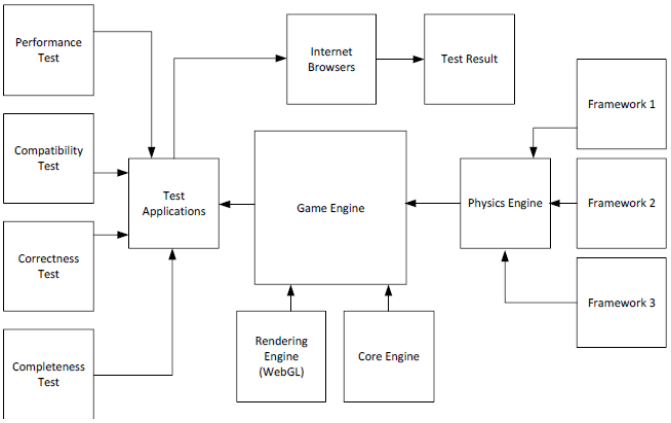

Fig. 3 The Prototype Game Engine Architecture.

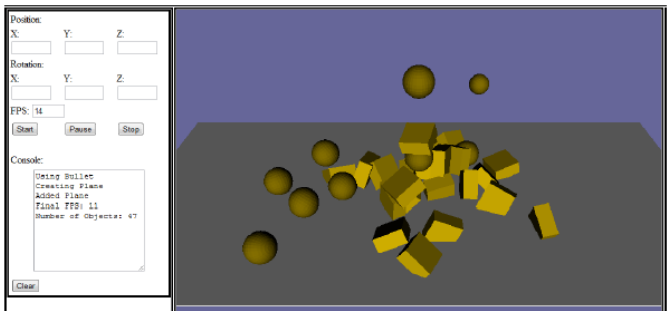

Fig. 4 The Game Engine User Interface.

IV. Experiments Design and 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

At the time of the experiment, Mozilla, Chrome, Safari and Opera browsers supported WebGL. However, the Opera browser 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 is able to run WebGL. Therefore, only two browser platforms were used: Mozilla Firefox 12.0, and Google Chrome 19.0.1084.52 m.

A. Design of Experiments

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

Performance Testing. To test the physics framework's performance, a test application that can generate physical objects continuously was developed. When the application ran, there were no objects. The application generated a physical object every second interval and the frames per second (FPS) rate of the scene was monitored. If the FPS rate dropped to less than 12, the test was stopped.

We created three test scenarios for performance testing by creating scenes that contained a lot of physical objects that collided with each other so the application forced the CPU to calculate heavy physics computations. The heaviness of this computation can be monitored by looking at the FPS (frames per second).

Scenario #1. A spheroid rigid body was generated every one second until the FPS dropped to 12 or less. In this test, the gravity of the world was y = -10. The specification of the rigid body was as follows: mass = 0.5kg, radius = 2m, start location x = 0 if the object number was even and x = 1 if the object number was odd, start location y = 20, start location z = 0, no initial velocity. There was also a ground/plane located at y = 0. We assume there was no friction involved in this test.

Scenario #2. A box rigid body was generated every one second until the FPS dropped to 12 or less. In this test, the gravity of the world was y = -10. The specification of the rigid body was as follows: mass = 1kg, width = 2m, length = 2m, height = 1m, start location coordinates (0, 20, 0), no initial velocity. There was also a ground/plane located at y = 0. We assumed there was no friction involved in this test.

Scenario #3. A box and spheroid rigid body were generated alternatively every one second until the FPS was dropped to 12 or less. In this test, the gravity of the world was y = -10. The specifications of the rigid bodies were as follows. Box with mass = 1000g, width = 2m, length = 2m, height = 1m, start location coordinates (0, 20, 0), no initial velocity. Sphere with mass = 500g, radius = 2m, start location coordinate (0, 20, 0), no initial velocity. There was also a ground/plane located at y = 0. We assumed there was no friction involved in this test.

Correctness Testing. In this test, the accuracy of the physics for collision handling were recorded. For this test, first the test application was run. There were two objects inside this application. One of them was static on the ground and another one was a falling object in the air. The position of the falling object was updated from time to time until it finally collided with the object on the ground. When the two objects collided, the collision was handled. Finally, the velocity of the falling object was recorded and analyzed for its correctness by using a relevant physics formula. This test was done five times on each browser, and the errors or deviations from the physics formula were averaged. We chose the sphere-box collision because it covers many formulas such as box inertia, sphere inertia, and rigid body collision.

For this test, the test scenario was as follows: There were two rigid bodies, one box and one sphere. The specifications of

the box were: mass = 1kg, width = 4m, length = 4m, height = 6m, x = 10, y = 5, z = 0. The specification of the sphere was: mass = 0.5kg, radius = 1m, x = 7, y = 8, z = 0. The box was static (no velocity) while the sphere moved 10m/s toward the x axis. No gravity and friction were involved in this test. The velocity after the collision was recorded and analyzed for correctness. From the scenario above, by using the previous formulas, the end result after collision was as follows. The box velocity (x,y,z): (2.1, -0.4, 0), box angular velocity (x,y,z): (0, 0, -1.3), sphere velocity (x,y,z): (5.8, 0.8, 0), and sphere angular velocity (x,y,z): (0, 0, 3.1).

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

B. Experiment Results

Compatibility Test result. Based on the result on Table I, only two out of three frameworks were compatible with our game engine. Note that the asterisk sign (*) in Table 5 indicates that there was 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 lacked some APIs but was still compatible to be used. However, jiglib.js was not compatible at all with our game engine, so we did not use it further in our experiments.

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)

Performance Test result. The results of this test were obtained by calculating the average maximum number of objects before the FPS dropped to 12. From Table II, we can see that the performance was better in Google Chrome rather than in Mozilla Firefox. This is to be expected because Google Chrome can interpret JavaScript language faster than Mozilla Firefox and since all physics code were written in JavaScript, Google Chrome had the advantage over this. In our evaluation, Google Chrome performed up to 282% faster than Mozilla Firefox.

For the performance of the physics framework itself, overall, the performance of cannon.js was a little bit faster compared to that of bullet.js. If we look carefully from the results, cannon.js performed faster in calculating spheroid rigid bodies compared to bullet.js based on the result from scenario #1 and #3. For box rigid bodies, it seems that both cannon.js and bullet.js performance was similar; cannon.js performed better in Mozilla Firefox, and bullet.js performed better in Google Chrome based on results from scenario #2 in Table II. On average, cannon.js performed 13.88% faster than bullet.js.

Performance wise, both physics frameworks can be used as physics engines for games in WebGL. Based on the results, it was not favourable to deploy games with a lot of physics objects in a Mozilla Firefox web browser.

TABLE I. 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

TABLE II. 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 III and IV, we can see that bullet.js has more accurate physics. Cannon.js seemed to perform well in box handling, but not on spheres. Bullet.js can handle both boxes and spheres very well. The error values from both frameworks were acceptable (less than one) except for spheres in cannon.js. The spheres in cannon.js did not rotate when they should have rotated. We also found that the physics simulated by Bullet.js were stable, in the sense that they had 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 may be 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 results in Table V, bullet.js fulfilled all requirements for commonly used physics in game, except cloth, water surface, and soft body physics. This is because Bullet.js has been developed for some time already, while cannon.js was new. So, it is to be expected that bullet.js would have more features. Even so, cannon.js provided the most basic features of physics that should be sufficient for simple games development.

TABLE III. THE ERRORS ON 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 IV. THE ERROR ON SPHEROID 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 V. 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

Based on the results of the four experiments 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 with bullet.js, and this framework showed good potential if it was updated regularly in the near future, especially bug fix updates and API updates. It was unfortunate that JigLib.js could not be tested because it could not run in our game engine due to its incompatibility.

V. CONCLUSION

In this paper, we have shown that some open source physics frameworks can be used as a component of WebGL-based game engine with acceptable accuracy and performance. [23] This is very promising considering the physics frameworks that we tested still lacked features and there were still many things that could be improved. Another finding from this research is that Google Chrome seems to perform best in running WebGL applications compared to Mozilla 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 observations, 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 includes 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,” DiGRA, 2007, pp. 344.
[2] Angry Birds, [Online]. Available: http://www.angrybirds.com/.
[3] Cut the Rope, [Online]. Available: http://www.cuttherope.ie/.
[4] Pokemon Go, [Online]. Available: http://www.pokemongo.com/
[5] Khronos Group, “WebGL - OpenGL ES 2.0 for the Web,” [Online]. Available: http://www.khronos.org/webgl/.
[6] Game engines and tools, The Mozilla Developer Network, [Online]. Available: https://developer.mozilla.org/en-US/docs/Games/Tools/Engines_and_tools
[7] C. Marrin. "WebGL specification." *Khronos WebGL Working Group*, 2011.
[8] A. Deveria. “Can I use WebGL?” [Online]. Available: http://caniuse.com/webgl.
[9] J. Gregory, “Foundations in Game Engine Architecture.” Massachusetts, A K Peters, Ltd., United States of America, 2009, ch. 1, pp. 3-55.
[10] Unreal Engine. [Online]. Available: https://www.unrealengine.com/.
[11] Construct 2. [Online]. Available: https://www.scirra.com/construct2.
[12] Unity. [Online]. Available: http://unity3d.com/unity/.
[13] PhysX - GeForce. [Online]. Available: http://www.geforce.com/hardware/technology/physx.
[14] Havok. [Online]. Available: http://www.havok.com/.
[15] G. Mulley and M. Bittarelli. “Ragdoll Physics.” [Online]. Available: http://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S07/final_projects/mulley_bittarelli.pdf. 2007
[16] J. Gästrin, “Physically Based Character Simulation–Rag Doll Behaviour in Computer Games.” Royal Institute of Technology, Stockholm. 2004
[17] Game Physics Simulation. [Online]. Available: http://bulletphysics.org/wordpress/.
[18] kripken/emscripten Wiki · GitHub. [Online]. Available: https://github.com/kripken/emscripten/wiki.
[19] JigLibJS. [Online]. Available: http://www.jiglibjs.org/.
[20] S. Hedman. [Online]. Available: http://schteppe.github.com/cannon.js/.
[21] D. M. Bourg, “Physics For Game Developers.” O'Reilly Media, Inc., 2002.
[22] C. Hecker. "Physics, part 3: Collision response." Game Developer Magazine, 1997, pp: 11-18