WebGL-Based Game Engine

Ritesh Grandhi, Bandi Vamsi Krishna Reddy, Varshan Guntupalli and P. Natarajan

Abstract Web-based games and simulations are becoming popular because they are easily accessible and are mostly hardware independent. This gives scope for improvements for the field of game development from a Web perspective. The Game Engine of this paper is proposed and developed to meet the increasing needs of a more straightforward way to develop Web-based 3D games and other visualizations. The Game Engine of this paper is made using certain existing tools such as three.js. The Game Engine gives flexibility to users by combining physi.js objects and three.js objects and introduces new classes of objects and also supports particle systems providing good control to developers without compromising performance. The Engine also features a HTML5-based editor that lets developers to give more prominence to game logic. This also streamlines the development process by splitting up the tasks among various streams of development.

Keywords WebGL · HTML5 · Three.js · Physi.js · JavaScript GPU · Fps · Game engine · Shaders · Real time

R. Grandhi (✉) · B. V. K. Reddy · V. Guntupalli · P. Natarajan
School of Computer Science and Engineering, VIT University, Vellore 632014, India
e-mail: ritesh.grandhi@gmail.com

B. V. K. Reddy
e-mail: bandivamsi95@gmail.com

V. Guntupalli
e-mail: guntupallivarshan12345@gmail.com

P. Natarajan
e-mail: pnatarajan@vit.ac.in

H. S. Saini et al. (eds.), *Innovations in Computer Science and Engineering*,
Lecture Notes in Networks and Systems 32,
https://doi.org/10.1007/978-981-10-8201-6_2

1 Introduction

Game Engines are used to build and develop 3D or 2D games. Each Game Engine varies in different aspects because each Game Engine is made to support certain features and their architecture is constructed as stipulated [1].

Web-based games have been limited by hardware and did not support GPU acceleration. However, there is more hardware support for Web apps today than ever before. This is a good thing for Web-based games as they can now make use of features like multi-threading, GPU-accelerated graphics.

A Game Engine that runs entirely from a browser lets users to work on their Web-based game or visualization in a flexible editor and avoids the whole install to run problem with traditional software. The Game engine uses three.js to make use of WebGL and acts as the rendering framework for the engine and Ammo.js to seamlessly integrate physics into any project.

2 Key Technologies

2.1 WebGL

WebGL (Web Graphics Library) is a JavaScript API which is used to render interactive 3D graphics and 2D graphics in any Web browser [2], which is compatible. WebGL can be integrated completely into all the Web browser standards as well as GPU acceleration in the usage of image processing and physics and effects as part of the Web page canvas [3]. HTML elements can be mixed with other WebGL elements, and other parts of the page or page background can be composited.

2.2 Html5

Html5 provides supports for 3D graphics in Web applications especially, the infamous WebGL. It is already known that the goal of Html is to create a unified network interface design platform. Hence, Html5 can provide seamless workflow, when it comes to 3D Web visualization needs.

2.3 Three.js

Three.js is an open-source 3D graphics tool built on WebGL using JavaScript [4]. WebGL is very complex, which makes the development duration longer and taking

up most of the manpower. Using three.js makes our implementation much easier and is a very sensible choice.

2.4 Bullet Physics

Bullet Physics is a physics simulation engine used for digitally simulating physics for three-dimensional objects. Its main functions include collision detection, soft body dynamics, and rigid body dynamics [5].

2.5 Physi.js and Ammo.js

Ammo.js is a JavaScript port of Bullet Physics that lets users to make use of the features of Bullet Engine in any JavaScript-based environment [4]. Physi.js uses Ammo.js and inherits from three.js objects to create Physi.js objects for which physics can be simulated in a three.js scene.

3 Architecture of the System

Understanding how Game Engines work is essential for planning and designing a game engine. There are a number of tools and frameworks necessary to develop a game engine [1]. Components of a Game Engine generally include as follows.

Renderer. This is where geometric data is converted into images by using techniques like rasterization. A 3D game engine uses 3D render libraries like OpenGL, Vulkan API. Renderers also take care of various post-processing effects such as fog, depth of field, and color correction.

Physics Engine. Physics engines simulate physics in a digital environment, either 3D or 2d, based on various approximation algorithms and real physics laws to produce fastest possible results.

Scripting and Logic. The users who make use of game engines will be able to assign logic to their games by making use of their scripting API.

Editor. Every game engine has an editor to let them design the levels of their game and control visual aspects of the game. The editor features the user interface to give them control over the 3D or 2D world depending on the engine.

Artificial Intelligence. It plays a crucial role in advanced 3D engines and provides services like path finding, goals, rules.

The engine described in this paper uses various frameworks like three.js and Physi.js to bring each component into play for the game engine to function properly (Fig. 1).

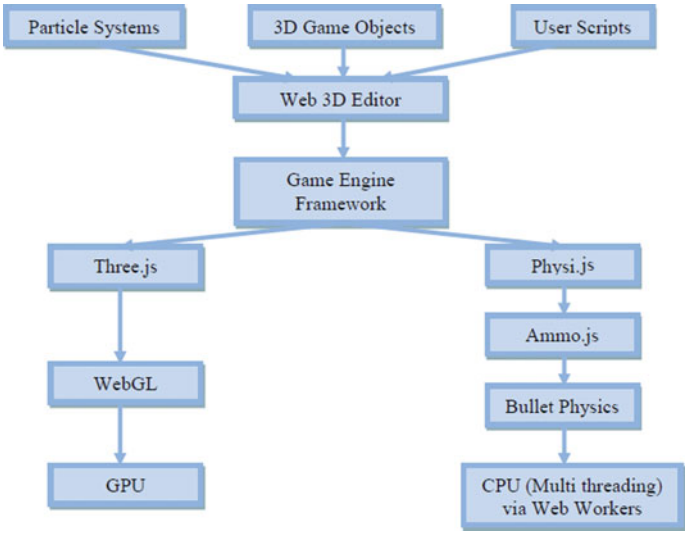

Fig. 1 System architecture showing the levels involved in the internal working of the game engine

3.1 Rendering

In general, game engines make use of the system resources like GPU and CPU to convert geometric data into pictures by using libraries like OpenGL, Vulkan. The main function of these libraries is to take geometric representations of three-dimensional points, lines, and polygons as input and draw pixels across the screen. This is done by using various efficient algorithms such as line drawing algorithms, texture mapping algorithms.

WebGL is a graphics library for rendering 3D images in a Web browser. It uses system resources just like other libraries [2], but it is very similar to OpenGL. Three.js is used to overlay WebGL for flexibility [4].

3.2 Physics

The game engine supports physics simulations with the help of Physi.js. Physi.js does an amazing job at simulating physics in a three.js scene. It is based on the Bullet Physics Engine and is a more flexible extension of Ammo.js, which is a JavaScript port of Bullet Physics engine itself. Physi.js can be seamlessly integrated into any three.js projects. Physi.js supports collisions for major primitives in 3D shapes such as cube, cone, cylinder, plane, capsule, convex, and concave objects.

3.3 Game Engine Framework

The main purpose of every Game Engine is to combine all aspects and components required to develop games. Such a Game Engine needs to provide an easier way to access these features for the developer than already present [1]. This simplification is very important for reasons like optimization and faster development speed. The Game engine described in this paper provides such an API to the developers. The API provides a simple object-oriented approach to creating scenes and adding objects into the scene. Other features of the engine include dynamic lighting, particle systems, a clean and simple HTML-based user interface, and a user-friendly 3D Editor. All these features are internally controlled by the Game Engine Framework.

4 Design of the System

4.1 Game Object

A Game Object is an engine-specific object that contains parameters, which acts by itself and also act as a part of a 3D scene in a game engine. One of the main parameters of a Game Object is the Transform. Transform stores the position, rotation, and scale data of a 3D object. The visual properties of object geometry are controlled by its material. A material decides how light reacts with the object with the help of its shader. The Game Object inherits from the Physi.js Mesh. All physics related properties can be controlled directly by using the synthesized game object. The Game Object also has a nature property of being static or not.

4.2 Particle System

In general, particle systems are a robust way to represent complex effects like smoke and dust, explosions, fire. Particles generated from particle systems just need a position attribute and visual properties such as color or texture. They typically occupy a single draw call which is a feature very crucial in the efficiency of games and other real-time applications. Particle systems have certain properties that control the properties of each particle. The properties include velocity vector, lifetime of a particle, its age and its target direction vector. In addition to the above attributes, the class inherits from the Game Object class present in the Game Engine API which the emitter object of this class behaves like a Game Object.

4.3 3D Editor

The 3D Editor is where the user interacts with 3D scene and all objects of the scene [1]. Control over objects in the scene is given to the user with the help of transform gizmos. The properties that can be controlled include Transform properties, Physics properties, and Material properties of the object. If the object is a particle system emitter, its particle system settings such as particle size, emission rate, and particle gravity influence can also be altered.

4.4 User Scripting

The Game Engine already provides API that is fully utilized by the 3D Editor. Users can also use the API provided to create game logic of their own. Game Engines can only provide features to make the development process easier. The Game Object and particle system API can be used and inherited by classes constructed by the developers.

5 Results and Discussion

The result is a functional 3D Game Engine with a flexible API. This engine has been tested on a midrange laptop that has an I5 3rd generation processor, 4 GB RAM, 2 GB VRAM GPU. This has been tested in the Google Chrome Web browser, which is the browser that has full support of features provided by three.js and WebGL. The following is a bar graph showing the performance analysis of the Game Engine at different numbers of objects in a scene.

The Engine performed at an average frame rate of above 60 fps and a minimum of 50 fps, when there are more than 40 active Game Objects (Fig. 2).

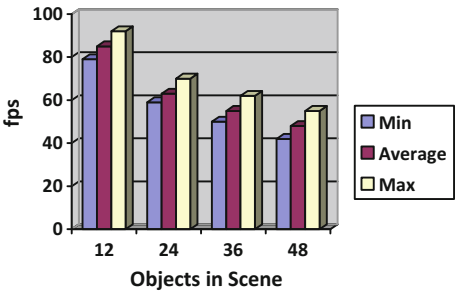

Fig. 2 Performance analysis measured in fps against the number of game objects in a scene

6 Conclusion and Future Work

With all the requirements met, the Game Engine is usable by developers to develop Web-based games. Although there can be more features that can be added, the current features of the Game Engine can meet all the requirements of any physics-based game or simulation project that can be published on the Web.

There are plans to improve the Game Engine and add more features to it, to meet the demands of modern Game developers. The engine may include support for various other simulations such as force fields, interactive physics, along with soft body dynamics, more shader support, and PBR support.

References

1. Xingliang Wei, Wei Sun, Xiaolong Won, M.S.: Architecture and Implementation of 3D Engine Based on WebGL. Applied Mathematics. 7, 701–708 (2016).
2. Simran Bhatti, Vandana Tayal, Pooja Gulia, M.S.: 3D development with WebGL. International Journal for Research in Applied Science & Engineering Technology. Volume 2 Issue XI, November (2014).
3. Alun Evans, Marco Romeo, Arash Bahrehmand, Javi Agenjo, Josep Blat, M.S.: 3D Graphics on the Web: a Survey. Computers and Graphics. 41, 43–61 (2014).
4. Rovshen Nazarov, John Galletly, C.: Native browser support for 3D rendering and physics using WebGL, HTML5 and Javascript. BCI'13 September 19–21, Thessaloniki, Greece (2013).
5. Adrian Boeing, Thomas Braunl, C.: Evaluation of real-time physics simulation systems. Proceedings of the 5th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia, Perth, Western Australia (2007).