An Evaluation of the Suitability of WebGL on Smartphones

TAREQ K. FADEL

Supervised by

Dr. HARIN SELLAHEWA

A Master of Science (MSc.) Thesis in Innovative Computing

November 2014

Department of Applied Computing

University of Buckingham

Table of Contents

Dedication

I dedicate this thesis to my parents and wife in appreciation for their unbounded support, love and encouragement.

Tareq

Abstract

With the introduction of Web Graphics Library (WebGL), modern web browsers were able to render computer graphics in two dimensions (2D) and three-dimensions (3D) without the need of third-party plugins such as Adobe Flash. WebGL allowed the use of hardware accelerated rendering giving web developers direct access to the graphics-processing unit (GPU) through a JavaScript Application Programming Interface (API). Browsers on the latest smartphones have recently adopted this technology. This has opened up a wide range of possibilities with many uses and applications. WebGL is used commercially to provide the end user with quick easy access to rendered scenes and environments and is used widely with mapping interfaces, rich Internet applications (RIA) and 3D games.

This thesis will research the performance and suitability of WebGL on modern smartphones and compares the results with the performance on their desktop counterparts. The study compares the performance on HTML5 compliant modern browsers on Android, Apple iOS and Microsoft Windows Phone platforms. The browsers that were used for the test are Google Chrome, Apple Safari, Opera, Opera Mini, Mozilla Firefox and Microsoft Internet Explorer.

The study investigates the uses, features and capabilities of WebGL and how suitable the technology is for smartphones. The thesis will analyse the performance of WebGL on four different mobile devices, desktop PC and Apple MacBook Pro laptop. The study compares the performance of the browsers on the mobile devices with desktop equivalent browsers. Research is performed on different benchmarks and benchmarking methods, and selects suitable third-party benchmarks for gauging WebGL performance. The benchmark tests compare and analyse different frameworks and libraries in WebGL and how well they perform on mobile devices. The study focuses on libraries such as three.js, babylon.js, Construct, TDL.js, GLMath.js and Pixi.js.

In general, the study has found that the performance on desktop equivalent browsers outperforms the browser performance on smartphones. However, with the newer generation of smartphones the performance is comparable and provides a stable usable state for commercial applications with WebGL on mobile devices.

The availability of WebGL on smartphone browsers makes WebGL development an attractive platform and API for web developers for delivering rich web-based applications designed specifically for mobile use.

Acknowledgements

I would like to express my gratitude to my supervisor Dr. Harin Sellahewa for his support, guidance and expertise. Dr. Harin has provided invaluable feedback and remarks throughout the learning process of this master thesis.

Furthermore I would like to thank my wife, Phoebe, for her continued support and patience during this journey. I would like to thank my parents, who have shown appreciation and support throughout the entire process. I will be forever grateful for their love.

Declaration

I Tareq Fadel declare that this thesis and the work presented are my own and have been produced by myself as the result of my own original research.

Glossary

Adobe Flash Player a plugin developed by Adobe Systems Inc. for the display and render of Flash based media .swf produced by Adobe Flash and ActionScript.

Adobe Shockwave Player a plugin developed by Adobe Systems Inc. for the display and render of Shockwave Media produced by Adobe Director.

AMD (Advanced Micro Devices, Inc.) is a company based in California, U.S.A. that designs and develops computer based chipsets and processors.

Android an open-source operating system developed by Google Inc. for mobiles and tablets.

API (application programming interface) is a set of rules and protocols that provides developers an interface of programming to the main functions of a third-party development tool or library.

AS3 (ActionScript 3) is an object-oriented programming language that is based on ECMAScript developed by Adobe Systems Inc.

CERN (European Organization for Nuclear Research) a European research institute for researching particle physics.

CSS (Cascading Style Sheets) a style sheet language that allows the format of HTML documents.

DirectX a Microsoft based computer graphics library for developing 2D and 3D computer graphics for the Windows based platforms.

FPS (frames per second) is a term used for the measuring of moving graphics. Measured by how many frames are rendered or displayed in a given second.

GLSL (Graphics Language Shading Language) a high-level syntax based language based on C for the development of shaders on the vertex graphics pipeline.

GPU (graphics processing unit), the central processor designed specifically for the processing of graphics based instructions. A parallel based processor that is efficient in dealing with matrix calculations.

HTML (HyperText Mark-up Language) a web standard for the Internet. A language based on mark-up tags.

HTML5 is the fifth version of HTML. HTML5 allows the development of web-based applications with rich media through the use of JavaScript.

HTTP (HyperText Transfer Protocol) A protocol designed for the transfer of HyperText Mark-up Language across networks and devices.

iOS is an operating system for Apple mobile and handheld devices.

JavaScript is a dynamic function based programming language for the web.

JIT (Just-in-time compilation) is a methodology or technique that modern applications use to perform compilations to source code in real time or just before execution.

Mobile app is a term used for an application that is installed and resides on a smartphone. Mobile apps tend to be smaller in size than desktop applications with a smaller subset of features.

OpenGL (Open Graphics Library) is an open-source cross platform graphics library for the development of two-dimensional and three-dimensional computer graphics.

OpenGL ES (OpenGL for Embedded Systems) is a subset of OpenGL designed specifically for embedded systems such as mobile phones, tablets and other low-powered handheld devices.

Plugin a piece of software that adds extra functionality to a browser

RAM (Random Access Memory) is a form of memory storage for the computer that is volatile. RAM is very fast at reading and writing from memory compared to non-volatile memory storage.

RIA (Rich Internet Application) is a web-based application that shares many characteristics to an installed based application but resides on a remote server and is accessed through a browser.

Silicon Graphics Workstation a machine designed and built by Silicon Graphics Incorporated (SGI) allowing the rendering and building 3D graphics. One of the first commercial computers designed for the rendering of 3D computer graphics.

Smartphone (or smart phone) is a cellular mobile phone with additional computing power that operates a computer with its own operating system and has access to WWW and email. Smartphones have a browser, email client and the ability to install additional software.

Unity is a games development engine and Integrated Development Environment (IDE) for the development of computer games across different platforms, desktop, mobile and web.

VBO (Vertex Buffer Object) a feature of OpenGL and WebGL that allows the upload of vertex based properties such as vector to the display for non-immediate render.

W3C (The World Wide Web Consortium) is an international standard organisation for the World Wide Web.

WebGL (Web Graphics Library) a JavaScript API based on OpenGL allowing the rendering of 2D and 3D graphics in a HTML5 compliant browser without the need of plugins.

WWW (World Wide Web) is a system of linked networks with HTTP as its backbone. The web pages are interlinked to each other through hyperlinks.

XML (Extensible Mark-up Language) a mark-up language that follows a set of rules. Allows developers to build their own custom tags. XML files are both human and machine-readable.

Table of Figures

Chapter 1: Introduction

Computer graphics has rapidly changed and developed over the latter half of the 20th century through to the 21st century. Research and development in the field of computer graphics has led to some remarkable progression, most notably, the latest advancements in computer graphics today, WebGL (Web Graphics Library). This technology has given access to the rendering of computer graphics in two dimensions (2D) and three dimensions (3D) directly in a web browser without the need of a third-party plugin. Direct access to the hardware provides hardware-accelerated graphics rendering with performance on par to native graphics. This has provided web developers the power of rendering high quality graphics through the use of a JavaScript API giving direct access to the graphics-processing unit (GPU). With WebGL the display of rich Internet applications (RIA) full with animated interactive graphics as part of the web experience all native to the browser.

On the other hand the progression of computers, personal computers and devices processing and displaying the computer graphics has also rapidly progressed and developed over the years. The relationship between the devices that rendered and displayed the computer graphics to the end user and the technology used to render the computer graphics has not always been smooth sailing. One of the reasons for this is that computer graphics are very processor intensive and in many cases require a dedicated processor to handle the graphics render to the display monitor.[1]

There has also been a rapid development of the devices that can render the computer graphics. From the first Silicon Graphics workstations to the latest handheld smartphones, the hardware and software have advanced many milestones. WebGL has only recently entered the foray of smartphones. With modern HyperText Mark-up Language version 5 (HTML5) browsers on smartphones are now able to render and display WebGL. However, the performance and usability of WebGL has not been extensively studied. WebGL has been available on desktop browsers since 2011 [2] and has progressively improved since its release. Now that WebGL is available on smartphones it is important to establish the usability of the technology on this platform and how well it performs.

1.1 Problem definition

The aim of this thesis is to conduct a series of tests to evaluate the usability of WebGL on smartphones. Running a series of benchmarks to test the 3D capabilities of smartphones will be the basis of this thesis. The smartphones that will be used for the test are Apple iPhone 6, Google Android LG Nexus 5, Google Android Samsung Galaxy Nexus and Windows Phone Nokia Lumia 520. The benchmark suite will be testing both the hardware and software of the device. The study's main focus is to evaluate the suitability of WebGL on modern smartphones through a series of browsers both native and third party. The research undertaken will compare the results to their counterpart, a desktop personal computer (PC) and analyse the results to determine its suitability.

The benchmarks used in this test are a collection of benchmark tests that are used to test the capabilities of the graphics rendering in WebGL through a browser. Some of the benchmarks are provided by the open source community and others by third-party sites and WebGL frameworks.

The information gained from performing this research has been gathered from books, blogs, articles, research papers, studies, workshops, conferences, online courses, meetings and other theses in WebGL.

Chapter 2: Background

In this chapter, I discuss the history of HTML and how it had developed to the latest release of HTML5. How computer graphics had evolved over time on the web platform and how WebGL had become one of the components of HTML5. In this chapter I also discuss how the web had moved from static web pages to rich Internet based applications and the use of computer graphics both two-dimensional and three-dimensional had increased. Further, I discuss how this move of web usage is moving towards mobile with the development of smartphones.

2.1 A History of HTML5 and WebGL

In a world where computers and devices are connected globally around the world, the World Wide Web (WWW) has provided us with a platform of communication, idea sharing and innovation. It has provided us with information resource, entertainment, methods of fast communication and many more. With the fast moving pace of how web technologies have advanced we have moved from an age of having static websites to an age where applications have become part of our everyday lives. To be more specific, apps or smaller forms of an application that target a specific task.

Twenty-five years ago the WWW was born and the HyperText Transfer Protocol (HTTP) [4] was established. The idea behind this was to connect different computers from different locations, regardless of age, operating system or software across a common protocol. A protocol that can be understood and interpreted by any device that connects to it. To this day this mission is still achievable with the multitude of devices, operating systems and software available. All devices that are connected to the WWW can communicate with each other even though there had been some challenges on the way.

Figure 1 Yahoo Page in 1995 [3]

In the labs of European Organization for Nuclear Research (CERN), Tim Berners-Lee [4] had authored a mark-up language that is simple, effective and is a plain-text language that is to be transferred from one device to another with no compilation. The language would be interpreted or 'parsed' by the client receiving the plain-text file and convert it client-side to a comprehensible readable document to the user. The language was named HyperText Mark-up Language (HTML) [5]. To this day, HTML is the most widely used language on the WWW; in some cases it is the backbone of the web. Nearly every website or web application that exists is written in HTML.

In the early days of HTML, it existed as a means of transferring documents that were minimally formatted. The web has evolved so much since then that what we expect from our experience on the web is a plethora of multimedia, interactivity and high social interaction. HTML is an open-source standard and unfortunately, as standards take time to develop depending on how the

technology is used in the industry, they tend to play catch up. Therefore, the technology is always years behind of what the industry currently is. HTML as a mark-up language had not changed much up until 2009 [6] where they had added additional tags to the language, but structurally it stayed the same. The web was dramatically changing and how the web was being used was changing. When mobile devices started to seriously contend in the web space, a major overhaul of the language was needed.

Figure 2: A sample web page from 2009 [7]

HTML as a language had developed over the years from the early release of version 1 and it was not till version 2.0 of HTML that it was standardised in November 1995 [8]. Over the years there have been many iterations and draft versions of the language and as the web has evolved, the standards for the language also needed to evolve. The web had evolved as a medium for sharing and transferring ideas. Ideas that not only were in text format but there came many formats of expression. The usage of images had increased on the web and advancements in multimedia had started to take place off-line. It was a matter of time before multimedia; the usage of moving graphics, animation, video and sound would start to become the norm as a form of communication across the web.

Third-party developers had started to develop plug-ins to the browser as extensions to cater for these demands. These plugins were not part of the HTML standard and nor were they part of the operation or function of the browser that displayed the HTML. The plugins would essentially extend the functionality of the browser to cater for additional features that the intended browser did not. Some plugins allowed the use of video playback, audio and web animation, whilst others were pushing the boundaries and catering for 3D content.

Figure 3: 3D content using Flash Plugin in the browser [9]

One of the most popular plugins today is the Adobe Flash plugin. Initially the plugin allowed the playback of vector based web animations and with later releases the ability to include interactivity sound and in an even later release, video. Due to the ease of accessibility to the plugin and the wide usage of content that was developed for the plugin, it had grown in popularity. At its peak Adobe Flash had reached a 99% penetration in desktop and laptop personal computers (PC) [10]. There had been attempts by other third-party developers in developing plugins that allowed the render of 3D but none of them had achieved critical mass to be deemed successful. Adobe had produced a plugin that rendered 3D, Adobe Shockwave Player but this never really took off and it installs declined as Adobe Flash gained popularity. Adobe eventually added 3D to Flash naming the technology Stage3D [11] but, unfortunately, due to the lack of Adobe Flash on mobile and tablet systems the technology did not really take off.

In the meantime, the standards of HTML had been catching up with the industry and a lot of the features used in the industry for the web had a huge demand for a standard to be developed for them. HTML started to include other standards that were related to the core language, like Cascading Style Sheets (CSS) and eXtensible Mark-up Language (XML). HTML was ready for the next step of including a lot of rich media elements as part of a standard.

This is where the concept of the next iteration of HTML5 [12] had started to take place, but this time, rather than reacting to the market and building the standard. The World Wide Web Consortium (W3C) have been developing a platform for the next age of the web and proactive planning and collaborating with the industry in what would be suitable for tomorrow's technology. For a lot of the features that were not available in HTML but had appeared in other third party technologies like Adobe Flash, these technologies were implemented as part of HTML5.

As mentioned earlier, HTML existed to aid the transference of documents but now we use applications. HTML5 is here to allow us to build applications and in doing so, JavaScript which had always been seen as an add-on language is now an integrated part of HTML and is the core building block of developing applications in HTML5.

Another aspect of HTML5 is the `<canvas>` tag. The `<canvas>` tag allowed developers to treat a web page as a 'canvas' to freely paint pixels on the screen with whatever colours they so wish and not be restricted to the browser's compilers method of rendering. With the work of the Khronos Group in conjunction of W3C they had looked at taking the two-dimensional (2D) canvas concept to the third dimension and introduce 3D rendered graphics directly in the browser with no additional plugins.

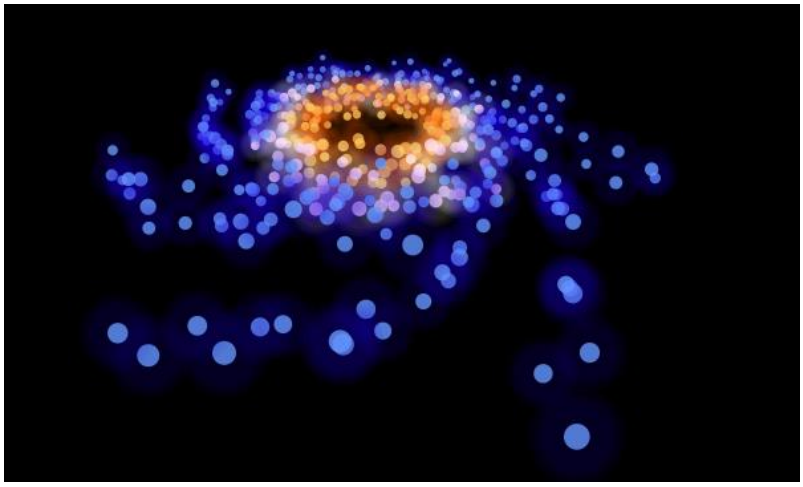


Figure 4: 3D Canvas Galaxy [13]

In order to do this would mean the creation of a new 3D rendering language or adapt an existing one. The issue with creating a new language is convincing developers to learn this new language.

Adopting an existing language is easier to convince developers to start developing with rather than in learning a new language.

WebGL[2] was developed as the medium for allowing developers to build 3D environments within the HTML5 canvas tag using JavaScript and Open Graphics Language (OpenGL). WebGL is closer to OpenGL Embedded Systems (ES) OpenGL ES than OpenGL is.

The web had always been a 2D landscape for web developers to build and develop the sites that exist on the web. If we look at how other mediums had adopted and inherited the 3D realm. Computer games and video games had all been two-dimensional and now almost every game is three-dimensional. Even film and TV have now adopted the third dimension and seeing how interactive and accessible the web platform is we can see that there is definitely room for this.

3D visuals and web technologies have been a keen interest of mine and seeing both these technologies come together in an open standard and being at the forefront of this new cutting edge technology is an exciting development. I have been following the progress of WebGL since 2009 and have seen it change as it has gone through the drafts to its current stable release. Although WebGL is based on OpenGL ES where 'embedded systems' and integrated devices like smartphones and tablets were in mind, its main usage and development had been on desktop. Now that smartphones and tablet devices have enough power to run and render 3D visuals we have started to see WebGL take place on smartphones.

Figure 5: WebGL 3D Eagle [14]

Web browsers for a long time were only able to render text and bitmap graphics and only with the advent of HTML5, it has given developers the ability to pixel paint directly to the screen

through a canvas. Prior to HTML5, the only way to render graphics to a screen through a web-based medium was through a plugin. There had been many iterations and variations of plugins that allowed you to display rendered graphics, both 2D and 3D, static and animated but by far the most popular is Adobe Flash. Due to its lightweight plugin and easy install, it had a high penetration to the market with 98% market share at its peak. Adobe Flash had provided developers a mean of creating graphics through the web to deliver rich media, high interaction, multimedia, games and video.

WebGL had started as a series of experiments at Mozilla and Opera and a JavaScript API based on OpenGL ES 2.0 that exposes the HTML5 canvas element.[15]. This enabled the use of 2D and 3D rendered scenes rendering in real-time through a browser with no plugins. WebGL had initially appeared for HTML5 compliant browsers for desktops but recently, we have started to see WebGL support in browsers on smartphones.

2.2 The advancement of smartphones

Technology had advanced leaps and bounds in both hardware, software and usability in smartphones. As HTML5 went in to draft, developers and manufacturers started to implement HTML5. WebGL initially featured in desktop browsers Google Chrome, Opera, Firefox and Safari and then later Internet Explorer but they were not yet released on mobile browsers. Later releases of the browser versions for smartphones started to feature WebGL. Initially it appeared on the BlackBerry Playbook tablet and later was featured in a Google Chrome beta release on the Android platform. WebGL is now available on Windows 8.1 mobile and with the release of iOS 8 on Apple iPhone.

Figure 6: Timeline of HTML and WebGL

One of the reservations of using WebGL smartphones is due to the high-load in CPU cycles and in particular the GPU cycles. Smartphones have been rendering and delivering rendered 2D and 3D graphics via their built in development environments natively. For example, the iOS platform allows developers to build applications and render graphics in both 2D and 3D in Objective-C through the Cocoa API. This had allowed developers the ability of developing games and applications that would run natively as an installed application on the device. There was no way of running real-time rendered applications through the browser. The browsers of the smartphones were only able to render text and bitmap images and recently SVG (scalable vector graphics) images.

Refinements and optimisations in the HTML5 working draft specification, the WebGL specification and the optimisation of the browser allowed the improvements in performance in WebGL. A comparison of natively run render and WebGL render has found that on all counts the natively developed application always outperforms the equivalent render via WebGL. The reason for this is that there is an additional layer it would need to go through. The overhead of the browser, the operating system before it gets direct access to the GPU. In addition to the extra layers, WebGL is a JavaScript API where JavaScript is not a compiled language like C, but an interpreted language that is left to the browser to compile. There have been some major advancements in the JavaScript engines like the V8 found in Google chrome that can execute JavaScript [16] at speeds much faster than previous engines.

Chapter 3: Literature Review

This chapter provides an overview of the various aspects of WebGL and the research and studies that have been done on the topic. WebGL is a relatively new subject and there has not been much research in this area, but there has been extensive research in the field of graphics performance and benchmarking. There have been studies in the general performance of WebGL but this has focused mainly on desktop-based browsers. Specifically, the area of WebGL and smartphones has yet to be touched upon and there is little to none research or literature conducted in this particular field.

The chapter will focus on three main topics. The first will be looking at WebGL as a technology and the research conducted on it and its uses. The second will be exploring the techniques of benchmarking graphics performance and the analysis of these results. Finally, the focus will be on the research that has been done on WebGL performance on desktops and smartphones.

3.1 WebGL

The concept of WebGL is to provide rendered graphics to the end user through the means of a browser. The graphics that WebGL provides can be in two dimensions (2D) or three dimensions (3D). The purpose of WebGL is that this can be done without the need of any plug-ins and is native to the browser. The inception of WebGL started 20 years ago [17, p. xxi] during the release of the first version of OpenGL (an open-source library and API for graphics rendering). Not much had changed in OpenGL until the release of OpenGL 2.0 [18] in 2004. There had been some fundamental changes to the language from using a fixed rendering pipeline to a dynamic rendering pipeline. Graphics Language Shading Language (GLSL) was also introduced as part of the Application Programming Interface (API). GLSL had provided a framework for programming the vertex shading and fragment shading giving the developer more refined control to the rendered scene. Khronos Group Inc. [19] released a specification for OpenGL ES designed specifically for mobile devices in 2004. OpenGL ES is a trimmed version of OpenGL. The Khronos Group had then released WebGL a modified version of OpenGL ES version 2.0 to be used with HTML5 to render 2D and 3D graphics through a browser. [20]

3.2 Uses of WebGL

There have been many uses of WebGL as web applications in both the commercial and scientific sectors. One of the first uses of WebGL has been the Google Labs experiment - Google Body (now Zygote Body) [21] - a rendering of the human body in 3D, allowing the user to zoom in on different organs for further analysis. One of the most popular uses of WebGL was in game development, as it provided a means of delivery of rendered 3D graphics to the end user through a browser without the means of a plugin. The alternative was to use a plugin and the most commonly used plugin had been Adobe Flash [22]. A few of the game engines that render to WebGL are Unity, Construct2, PlayCanvas, Pixi.JS, KickJS and Babylon.js [23]–[28].

WebGL is not only popular among game developers but also in web applications. One of the most popular uses of WebGL in a web application was Google Maps released 24th October 2011 [29]. It allowed the user to browse through the Google Maps with a faster response time and allowed seamless integration with Google Earth viewer directly into the browser without the need of a plugin. Google has also been using Adobe Flash to render the immersive street view features but with the introduction of MapsGL (a name that was given by Google when integrating WebGL with Google Maps) it has been possible to provide the same immersive experience without the use of a plugin. WebGL has also been used as part of medical research for medical volume rendering [30]. The research entails the use of WebGL on smartphones and discusses the methods and the viability of the technology. During the time the paper was written, the only mobile platform that had supported WebGL was the Android platform. The paper states that some light benchmarking was done on the different browsers on the platform on two devices - a tablet and a smartphone. The browsers that were used were Google Chrome, Mozilla Firefox and Opera. From the tests that were performed, Google Chrome outperformed the other browsers. Therefore, the rest of the tests were conducted on Chrome. There has been advancement in browser technology since the writing of the paper and the inclusion of both Apple iOS and Microsoft Windows Phone to the range of devices that support WebGL.

3.3 WebGL on smartphones

WebGL is based on OpenGL ES an embedded system technology and as OpenGL ES was originally developed to be used on mobile systems and smartphones this was a natural evolution

of the language. Unfortunately, although WebGL was officially released in March 2011, there has been little progress on the development of WebGL on smartphones. The main focus of usage of WebGL was on desktop platforms through WebKit based browsers akin to Google Chrome and Apple Safari. The development team at Mozilla and Opera had also been working on developing improvements in the rendering of WebGL. This had advanced further development of WebGL on smartphone devices. Due to the lack of sufficient computing power on smartphones in both CPU and GPU power there has been little development in this field. Only until recently WebGL has started appearing on mobile platforms. WebGL's first appearance on a mobile device was the BlackBerry PlayBook featuring a rich HTML5 browser that was capable of running WebGL content [31]. WebGL then made its way on to Google Chrome in a beta release on the Android Platform as announced by Brandon Jones, a Google Developer on his Google Plus blog post [32]. A few months later Google had released a full stable release of Google Chrome for the Android platform that includes WebGL. In regards to the iOS platform, WebGL was not released until the release of iOS 8 with the timed release of the iPhone 6. iOS 8 was released with Safari 8 that allowed and rendered WebGL content. This was released to both the OS X 10.10 Yosemite desktop operating system and the iOS platform, this was first announced at the World Wide Developer Conference (Apple) June 2014 [33]. This was then officially released out of beta on the 19th September 2014 with the release of the iPhone 6 and iPhone 6 Plus [34].

WebGL on smartphones has always been a challenge as graphics rendering in general has always been a demanding task for processors and dedicated processors, GPUs are required for rendering the graphics pipeline [35]. The rendering of graphics has a high consumption of power, which does not perform well with low-powered devices i.e. smartphones. There is another factor that must be taken into consideration, which is that high-powered consumption will quickly drain battery power and can cause the devices to overheat. Once there have been advancements in the optimisation of hardware chipsets for both the CPU and the GPU on battery-powered mobile devices, it could be possible to have WebGL on these devices.

There has been some early testing of WebGL on smartphones, one study that was discussed earlier on the volume rendering on smartphones and tablets [30]. There was also some early research on the capabilities of using WebGL on smartphones, in particular iOS in 2012 [36]. This was quite revolutionary in 2012 when it was announced at the Web3D conference. The main

reason was that WebGL was not supported by iOS at the time when the paper was written. The research was a proof of concept of the viability of the technology of representing a 3D talking head that can interact with the human user via voice commands.

3.4 Graphics Performance Analysis

The analysis of graphics performance on the GPU is done primarily via a benchmark or benchmark suites [37]. A benchmark is used here as a standard point of measure where the benchmark is run on different devices using the same parameters and the results from the different hardware is compared. When it comes to analysing GPUs for their effectiveness in rendering 3D objects and scenes, the GPUs may suffer from internal pipeline bottlenecks [35], which may hinder or affect the result. The benchmarking of GPUs is used for the fine-tuning and optimising the development of future GPUs. There are two internal clocks that control the clock speed of both the GPU processor and the memory. The clock speeds are set by the manufacturer to a value for maximising performance whilst reducing bottlenecks [38]. GPUs are designed and optimised to run threads in parallel, which makes them ideal for graphics rendering as multiple threads are executed in tandem.

There are various different methods of measuring the performance of graphics and one of the most common techniques is the measurement of the "frame retirement time" [35]. Frame retirement time is the time it would take to for the graphics to go through the whole graphics pipeline stage to have the graphic displayed on screen ready for the next frame to render. This measurement can be measured as how many frames are rendered in a given second "frames per second" (FPS), or translated to the time it took for that one frame to fully render to screen. There are six stages that the graphics pipeline would need to complete to render the object on screen. Input Assembler -> Vertex Shader -> Clipping -> Raster Operation -> Pixel Shader -> Output Merger. The Input Assembler stage tends to be the biggest culprit in terms of causing the delay in the frame retirement time.

3.5 JavaScript Performance

When it comes to computation and compilation of programs, JavaScript is a dynamic programming language that is a Just In Time compilation system. "Just-In-Time (JIT) or dynamic compilation, which refers to translation that occurs after a program begins execution" [39].

JavaScript has not always been a JIT compiler but web browsers from Microsoft Internet Explorer 8 and older would compile the code into byte code prior to execution [40]. This had huge overhead in terms of time for processing web applications that for the likes of complex or large web applications this was not suitable. Microsoft had introduced JIT compilation to their new Internet Explorer 9 IE9 engine, Chakra, to meet the demands of the more complex web applications that exist today [41].

Today, most modern web browsers in particular the ones that are most commonly used, such as, Safari, Chrome, Firefox, Opera and IE9+ are JIT compilers and are also referred to as HTML5 compliant browsers. As catalogued in HTML5test.com where an index score is given to every browser version, testing the features of HTML5 for compliancy with HTML5 and browsers IE8 and older had scored 43 or lower and the current range of browsers have an index score between 376 and 512 [42]. A good comparison of performance between a JIT browser and static compiled browser akin to IE8 can be seen in Ratanaworabhan and Livshits paper titled JSMeter: comparing the behaviour of JavaScript benchmarks with real applications [40] where the results are quite significant. There have been dramatic improvements in the JavaScript engines of web browsers and the performance of these browsers has increased as with the phenomenal performance of the V8 engine in Google Chrome. [16] [43]

Browsers vary dramatically in their performance [44] and this is due to the underlying engine that is powering the browser and compiling and executing the JavaScript code that it is receiving. As mentioned earlier, the four big contenders in Internet browsers are Microsoft Internet Explorer, Apple Safari, Mozilla Firefox and Opera (iPhone and Android stock browsers have been excluded as they both have been discontinued in favour for Safari and Chrome respectively) [45]. Microsoft use their own proprietary layout engine Trident with the JavaScript engine Chakra since IE9 [41]. Safari, Opera and Chrome use the open-source WebKit engine [46], [47], although Chrome uses a forked version of WebKit called Chromium [48] running V8 as their JavaScript engine. Firefox uses an open-source layout engine Gecko, SpiderMonkey as their JavaScript engine [49] and asm.js as their low-level JavaScript engine.

3.6 WebGL Performance

The first aspect before investigating the performance of WebGL is looking into which browsers can currently support WebGL. Although WebGL was introduced by the Khronos Group in the first quarter of 2011 as a final specification, the adoption of the technology has been somewhat slow [2]. The two earliest adopters of the technology was Firefox and Chrome, version 4 and version 8 respectively. Even then, the adoption was buggy and very experimental. It was not until version 18 of Google Chrome that it had its first full-fledged adoption of WebGL [50]. Today, WebGL is more widespread amongst browsers but there are still some problems. Currently the browsers that support WebGL are the following: Internet Explorer 11, Firefox 33, Chrome 38, Safari 8, Opera 25, iOS Safari 8, and Chrome for Android 38. Although the list looks near enough conclusive, unfortunately Firefox, Safari, Opera and Chrome for Android only have partial support to official Khronos WebGL specification 1.0.2 released 1st March 2013 [20]. This can have an impact on performance or even running some of the benchmark tests. The Khronos Group developed a WebGL conformance test runner that will run a series of tests to test if the current browser adheres to the specifications set out on the WebGL specification 1.0.2 and will flag up any errors that arises [51]. The browsers that partially support WebGL unfortunately do not pass all the tests set out by the conformance test.

Apart from determining which browsers can support WebGL, fully or partially, it is important also to determine how well the browsers can effectively run any WebGL script they come across. WebGL has many different libraries and can be used in many different ways. The performance of WebGL is mostly determined by how well the browser can effectively run the JavaScript code but there are other factors that can impact the performance of WebGL in the browser.

As WebGL is effectively an OpenGL ES JavaScript API, the performance speed of the browser is crucial for the success in having decent render times. WebGL also relies heavily on mathematical calculations, specifically with matrices and these calculations need to be calculated at fast enough speeds to not cause a bottleneck and slowdown during the rendering pipeline of the graphics. Google Chrome's V8 [44] engine seems to have out performed every other JavaScript engine in the majority of tests conducted from the studies and articles reviewed. This indicates that when choosing a browser based on the execution speed of JavaScript, Google Chrome is the browser of choice. The JavaScript in WebGL adds a layer of complexity as there is

the graphics rendering pipeline and the need to deal directly with the GPU. Unfortunately, there have not been many tests in the performance of JavaScript with WebGL and further studies in that field are needed.

Hoetzlein [52] performed different tests comparing the performance of WebGL, WebGL with Vertex Buffer Objects (VBO) and Adobe Flash with ActionScript 3 (AS3). The tests conducted for Flash were compared with BitmapData and sprites. The tests were performed on different browsers and the results were dramatically different depending on which browser and which framework was used to deliver the graphics through the web platform. From the study, it appears that the choice of the browser had a large impact on the performance of the rendering of the graphics. The tests had been performed in 2012 and in only a space of two years there has been a dramatic improvement in browser technology in terms of the execution speeds of JavaScript and the performance and rendering of web graphics. One surprising result is that the WebGL test with VBO had outperformed Flash and every other framework apart from native rendering. The difference between the native OpenGL rendering and WebGL with VBO is minimal. This in some ways proved that WebGL is a viable platform for the delivery of web graphics through a browser and can be done without the use of any third-party plugins or add-ons.

Hoetzlein had stated that the study indicates that further tests on other platforms would need to be performed to give a larger picture and the effect of the usage of WebGL on mobile browser platforms had not been mentioned or tested. As the usage of smartphones has significantly increased over the last couple of years as well as the development of WebGL on smartphones, a further study in this field is required.

Another paper explored the performance differences of JavaScript Physics frameworks. Whilst this study was not focused on WebGL but on JavaScript physics libraries this is still relevant to WebGL for a couple of reasons. WebGL performance is hindered in a lot of cases by the processing and compilation of JavaScript and in many cases can be the bottleneck to the performance of WebGL. A direct comparison of the performance of WebGL graphics rendering to native graphics rendering is discussed in the Hoetzlein paper [52]. Moreover, in a lot of cases during the development of a WebGL application a physics engine is also used. This is most common during the development of 2D and 3D HTML5 web-based games. In Yogya and Kasala's paper 2014 [53] a detailed study of various different physics engines was performed.

The conclusion of the paper is that bullet.js outperforms the other engines tested. The display and method of testing had become useful in conducting my own tests for the benchmark and the tests of the various different WebGL JavaScript engines that exist.

David Catuhe, the lead developer behind the BabylonJS WebGL framework ran a benchmark test using the Train Demo on four browsers, Chrome, Internet Explorer 11, Firefox and Opera [54]. As can be seen from the benchmark results that Internet Explorer had outperformed the other browsers. The problem with running a benchmark with one test or with one framework is that it may favour one browser or platform to another. To create a fair and objective benchmark, a series of tests with different frameworks on different browsers would need to be created. David Catuhe had outline that during the development of BabylonJS, the development team had worked on outlining and removing the JavaScript bottlenecks to keep the render of the scene at 30 FPS or higher as highlighted in the article.

An article published at Tom's Hardware [55] explores the performance difference between Chrome and Firefox. The review looks at two different benchmark tests, the first benchmark testing the particles developed by the Khronos Group and the other, an aquarium render scene built in WebGL. In both tests Chrome outperformed Firefox. However, this test is not conclusive and further tests would need to be done to provide a better understanding of how WebGL performs on specific browsers.

Unity [23] a games developing platform have been busy developing their upcoming new integrated development environment (IDE) Unity 5, which is due out on release. One of its core features is the ability for the developer to release the game on WebGL, in addition to releasing the game for iOS, Android, Windows, Mac and Xbox 360 as well as many other platforms. This is a new addition to their host of features on the platform. I have been following the development of the Unity platform closely and have attended a few Unity user group meetings in London, where they have demonstrated their future releases and discussed their upcoming features. Unity is widely used for games development specifically for mobile platforms due to the ease of developing on multiple platforms with one code base. Currently it is possible to output the game on to the web as a platform but the user would need to download and install the Unity plugin [56], which in most cases is not desirable. Browsers on embedded platforms (such as mobile and tablets) and browsers on systems without administrator privileges would not be able to install the

plugin limiting the number of users who would be able to play the game. Unity does allow the deployment of the game to Adobe Flash. Therefore, by having a Flash-based player plugin installed in the browser the game would run but due to lack of demand, this feature was deprecated and is being removed from the next release of Unity, Unity 5. [57].

One of the advantages of being able to deploy from Unity to WebGL is that little to no additional development from the WebGL side would be required as this is handled by the engine. This would therefore increase the number of titles and applications that are deployed using this technology. Prior to the release of Unity 5, the development team had published their findings on the performance of WebGL on different browsers [58] and the results had been promising. What the author had deduced is that the performance on WebGL is about 50% of what it is on native code. The results had also showed that Firefox had performed exceptionally well with WebGL due to the underlying asm.js JavaScript engine. “Asm.js is a subset of JavaScript designed to be the assembly language of the web” [59] as it runs at a very low level the performance from asm.js, especially in WebGL is faster than traditional JIT compilation.

The analysis used in the report details by Unity Technologies on the performance of WebGL compares the browsers. The author assigned Firefox the value of 1 as a relative measure of performance to the other browsers. This gave the reader the indication of relative performance from a single base point. This may be useful when conducting my own studies and choosing one browser and one platform as a base measure for comparison, as the measurement of vertices, objects or even FPS is arbitrary. The benchmark was a series of benchmark tests, testing different facets of WebGL performance; in some cases there was evidence that WebGL had outperformed native code. This had been a good study and had reinforced the idea and concept that a suite of tests testing different aspects of WebGL is more informative than conducting one given test. Although the author only tested it on one platform and by conducting the tests on different platforms with different hardware may have well given different results. Out of curiosity, I had tried running the Unity benchmark suite on smartphone WebGL browsers but unfortunately the tests could not run as the tests had not been optimised or tested for mobile use.

3.7 WebGL Best Practices

Determining how well a browser or platform performs from a benchmark is one measure but considerations into other aspects have to be taken into account as well. As WebGL is a JavaScript API and working with good practices when writing JavaScript will naturally enhance the performance. Making sure the code is optimised, refactorised [60] and with efficient memory usage and garbage collection will inherently produce faster FPS by making sure that the JavaScript execution is not the reason for the bottleneck. In general when writing JavaScript, it is best practice to follow the guidelines set out in Douglas Crockford's book "JavaScript the Good parts" [61] in using object-oriented programming techniques, closure, efficient memory management, properly structured and efficient architecture.

Mozilla had published on their developer network an article on WebGL best practices [62]. Unfortunately, not all WebGL that exists would have been developed with these best practices in mind and by doing so, may very well cause a delay in the execution of the code and the JavaScript in the JIT compiler will take longer than it could have been. By doing so this will cause a bottleneck at the JavaScript execution prior to the graphics-rendering pipeline. This will therefore have an impact on the efficiency of the animation and render in WebGL and will impact the frame rate.

Some of the examples of best practices when writing WebGL is querying the browser in question to see if it supports the extensions and features that are to be developed with. As implementing a feature that does not exist may throw up errors, exceptions or make the JavaScript underperform. Other techniques are "rendering at a lower resolution then using CSS to upscale the render scene". These best practices become more critical when developing for mobile browsers due to the low-powered GPUs that the smartphones are built with. Smartphones come in a wide range of devices, specifically on the Android platform and catering for all the different devices can be challenging.

3.8 WebGL Mobile Performance.

When it comes to WebGL on mobile phones or in particular smartphones, there has not been much research done in this field. This is mainly due to the fact that WebGL had only recently started to appear on smartphones and Apple adopting the technology on to their iOS platform on

the 19th September 2014 for their iPhone 6 launch [63]. There had been some studies of usage of WebGL on mobile platforms as discussed earlier but research in the field of performance of WebGL on smartphones was hard to come by. All the research that I have found has been conducted on the performance of WebGL had been on desktop-based machines including laptops. This is one of the motivations that had inspired me to conduct this research.

There had been one small study by the development team of Scirra (a games development company specialising in WebGL) where they had tested WebGL on smartphones [64] There were two tests conducted, one based on a simulated 2D game and an artificial test, testing the capabilities of the rendering of the browser by maximising the number of sprites rendered on the screen. The study compares both canvas rendering and WebGL rendering in an all tests. In all cases the performance of WebGL had outperformed canvas rendering and the results had produced positive results even on mobile. Although this was a small study, this proves that the concept of WebGL on smartphones is viable but it was in its early stages when the study was performed, as only Android was tested and the only browser that could render WebGL had been the beta version of Google Chrome. Today, there is a collection of released browsers that officially support WebGL on different platforms that are available to test on.

3.9 Tests & Benchmark Review

In this section I will discuss and review the benchmarks and reporting tools of WebGL. The reports and benchmarks that have been chosen are a wide range of benchmarks from popular frameworks and libraries of WebGL. These benchmarks and reports have been used in the past to provide results on how well WebGL has performed on desktop-based browsers. However, to my knowledge, no published work has been conducted on the performance of WebGL on smartphone browsers from different platforms. Instead of focusing on one framework, benchmark or report, a multitude of different benchmarks have been chosen to give a generalist idea of the performance of WebGL on smartphones and testing whether WebGL is a usable technology on handheld devices. Four report tools were chosen and eight benchmarks for the benchmark suite.

The first of these reports is a report that represents a spinning cube. It is a simple report just to test if the browser supports the most basic of features in WebGL. [65] http://get.webgl.org/. The

second report http://doesmybrowsersupportwebgl.com/ [66] goes into a bit more detail and queries the browser on the features and extensions that are supported by the browser. This is useful when developing in WebGL as the complexity of the render can be scaled to meet the capabilities of the browser. The results from this test would be useful to compare when analysing the results from the different devices and whether the browser gives different results from the same device. Following on from the previous report, the following report "WebGL Report" delivers a similar report but displays it in how the pipeline is laid out with slightly different parameters [67] http://webglreport.com/. Another report ScirraMark [68] published by Scirra, a games development engine company produces a report testing the browser on different merits for suitability for HTML5 gaming. One of the factors it tests for is WebGL but it looks into other aspects like HTML5 Web Audio, Web Real-time Communication (WebRTC), plus many other factors. http://www.scirra.com/labs/scirramark/.

There are a total of eight benchmarks that have been chosen as part of the benchmark suite. In this section I will be looking at the eight benchmarks and the reasons to why they had been chosen as part of the study. The benchmarks will focus on different WebGL frameworks independently. The frameworks that will be investigated are the following: Construct2 by Scirra, Babylon.js, three.js, TDL.js, GLMath.js, Pixi.js, mjs, GLMatrix, Google closure, and Sylvester. Each of benchmarks are either open-source tests published on github or are developed by the development team for testing the framework in question.

3.10 Literature Review Conclusion

From the literature that was reviewed, many have focused on one particular framework or library. This is useful in its own right; however, it does not really answer the question of how WebGL performs in general regardless of what framework, device or browser that is used. Moreover, it does not address the topic of using the browser on a mobile device. Some of the studies reviewed have only performed one benchmark test, which is adequate for a small study, but the results may be skewed in favour or biased towards one benchmark. To give a fair analysis of the performance of WebGL, a multitude of different tests would need to be conducted. In addition, each of the tests would need to be performed multiple times and a mean average of the results recorded.

In general the studies that have been conducted have been tested on different browsers but only on one platform. Looking forward, the performance of WebGL will vary on different hardware. This is specifically the case when exploring mobile as the hardware, as this varies dramatically from low-end budget phones to high-end flagship devices.

There are other factors to consider when benchmarking for mobile devices. These factors include environmental such as temperature, the overheating of the device, ensuring there is no protective case and a fresh install of the operating system. Other factors that may impact the performance of WebGL in everyday use, such as, applications running in the background and/or other tabs that may interfere with the performance.

After reviewing and analysing the existing studies in the area of graphics and WebGL performance, I have greater clarity on how to conduct graphics performance benchmark tests. I have also concluded that little research has been conducted in the field of WebGL on smartphones and would like to explore this specific topic further.

Chapter 4: Approach

Initially, when scoping the project and the feasibility of the study, I had intended on producing a render of an animated scene with the use of WebGL. At the early stages of the project my knowledge and technical details of the technology was limited. The original plan was to study the library and technology and produce a rendered scene that will work on mobile devices and produce a benchmark report of the performance of the test on the device.

After studying the various literatures that exists and discovering the complexities of the language that this may not be the correct approach to tackling the problem. The main issue with producing an animated rendered scene in WebGL is that it would be produced in one particular way or library and after the extensive research, I had soon discovered that there are different libraries for programming and building a rendered scene. Each of these libraries performs differently and some are better than others for particular features. Most of the development platforms that existed for WebGL had a predefined benchmark for testing the performance of the technology.

After reviewing the literature and discovering how other computer graphics performance testing was performed by benchmarking, the best method or the most objective technique of evaluating WebGL is to conduct a study on the different technologies that exist. This would mean that I would have to write and build a benchmark in each of the frameworks that exist and test it extensively with optimisations to each of the code ensuring that the code that I had developed for the benchmark is not causing an impact on the performance.

I had then researched the different frameworks and the benchmarks that had been used in previous studies, blog articles and research and collated them for this study. A lot of the existing benchmarks had been built particularly just for testing one framework and have been built for testing the performance of WebGL on desktop machines. Each of the curated benchmarks chosen for this study had been extensively tested on mobile devices to ensure the success of the study. In addition to collating the appropriate WebGL performance benchmarks, a collection of reporting tools on the features and capabilities of WebGL on the browser would be required to aid the analysis of the results. So further research had undergone to collate the various different third-party WebGL reporting tools. This was done by querying the graphics renderer of the features it

had supported and if it did support these particular features, what limits or parameters it did support. I had felt this is necessary as the hardware of the devices had differed greatly and this may have an impact on performance.

4.1 Changes from Initial Plan

During the development of the study, there had been some key advancement in the field of WebGL on smartphones. Windows Phone had been updated to Windows Phone 8.1 to include Internet Explorer 11 on mobile devices that now support WebGL and Apple releasing iOS 8 featuring WebGL support for the operating system allowing Safari, Google Chrome and Opera Mini browsers to utilise the WebGL feature built into the latest release of the WebKit engine. The timing of the study and the release of these new technologies on the devices had aided the study and research to be conducted successfully.

Initially the study was to just look at Android and iOS operating systems on mobile. For the purposes of the study, a beta version of iOS 8 was to be acquired but with the official release this was no longer needed and with the release of Windows Phone 8.1, the Windows phone was included.

Due to time and budget constraints the project had to be limited to a selection of four mobile devices. In an ideal solution, a larger study would be conducted to include all smartphone devices that are currently in the market to give a proper evaluation of WebGL on current mobile devices. However, for the purpose of this study, a range of devices covering different release dates, price ranges and platforms have been chosen to give a general overview of devices in existence. Project managing this project has been challenging, due to all the changes to the original plan of the project and the timing of the studies to coincide with the platform release dates.

Having additional resources and time for this study, I would have set out to build a reporting tool with a mobile optimised benchmark for each of the existing WebGL libraries and frameworks and conducted a full investigation on each of these platforms. However, this study focuses on third-party benchmarks and reporting tools provided by the developers of the frameworks or other third parties.

4.2 Getting to know WebGL

Prior to conducting the actual study, an understanding of WebGL was required. To achieve an adequate level of knowledge on the subject I had attended a Massively Online Open Course (MOOC) by Udacity [69] delivered by Eric Haines titled, Interactive 3D graphics. This has given me a solid foundation of the technology and how to tackle the problem of the study. The course focuses on just one framework, three.js but the concept behind building and developing in WebGL is similar across different frameworks. They all share the same underlying library of WebGL that is based on OpenGL ES and Shading Library GLSL through the JavaScript API. I have a good grounding in JavaScript programming, and based on my previous experience, this has helped me gain a better understanding of WebGL and how it operates. As well as the online course, I have referred to Tony Parsi's book, WebGL Up and Running [70]. I have amassed a collection of WebGL books for reference, but I have mainly utilised the vast amount of tutorials and publications that reside on the World Wide Web, which has significantly furthered my understanding of the technology. The usage of WebGL had dramatically changed over the years. The methods and techniques have developed over time and published on blog articles by other developers.

I had regularly attended WebGL Workshop held by [71] Carl Bateman where I had met seasoned developers of WebGL. On occasions there have been talks from researchers, publishers and developers of WebGL showcasing their work and the methodologies used to achieve their results. It has given me an opportunity to get in touch and speak with individuals and organisations that are actively using the technology as a medium of delivery in advertising, commerce and entertainment.

Chapter 5: Methodology

To test the suitability of WebGL on smartphones a series of tests will need to be performed. The tests are a series of benchmarks that will test various different aspects of WebGL. The performance will be determined by how effective these tests are run. A comparison study between how WebGL currently performs on desktops and laptops versus to how they perform on smartphones will be conducted.

5.1 Benchmark Suite

The benchmark suite is comprised of a collection of reporting tools testing the capabilities of WebGL in the browsers and a collection of benchmarks that test the performance of WebGL using different techniques and frameworks.

All reports and benchmarks are measured after a fresh restart from the device with no other applications or tabs running (unless required by the specific test). In addition, the display of the screen is kept awake for the duration of the test and the benchmark is not switched or minimised during the test.

5.1.1 Reports

Each of the following reports were performed on the devices with a full charge of battery and after a full restart and a one minute wait to allow any background services to complete.

Four reports were conducted on each smartphone. Each of the reports was conducted twice on each browser on each smartphone to ensure that the result of the report is consistent.

The reports that were conducted were as follows:

5.1.1.1 Report 1: Get WebGL

http://get.webgl.org/

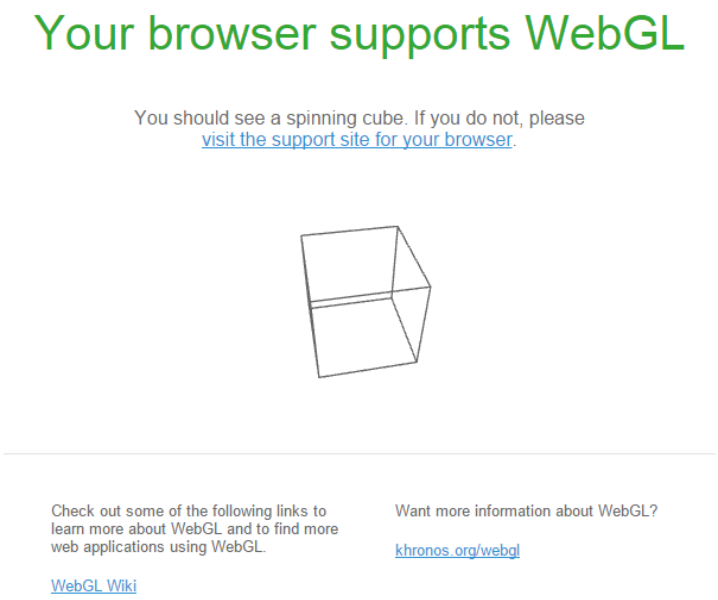


Figure 7: Screenshot of the Get WebGL Report [65]

This initial report is a simple report to test if WebGL can actually run on the device and the browser. This is done by showing a spinning cube. If a spinning cube is represented then the browser and the device can run WebGL.

5.1.1.2 Report 2: Does My Browser Support WebGL

http://doesmybrowsersupportwebgl.com/

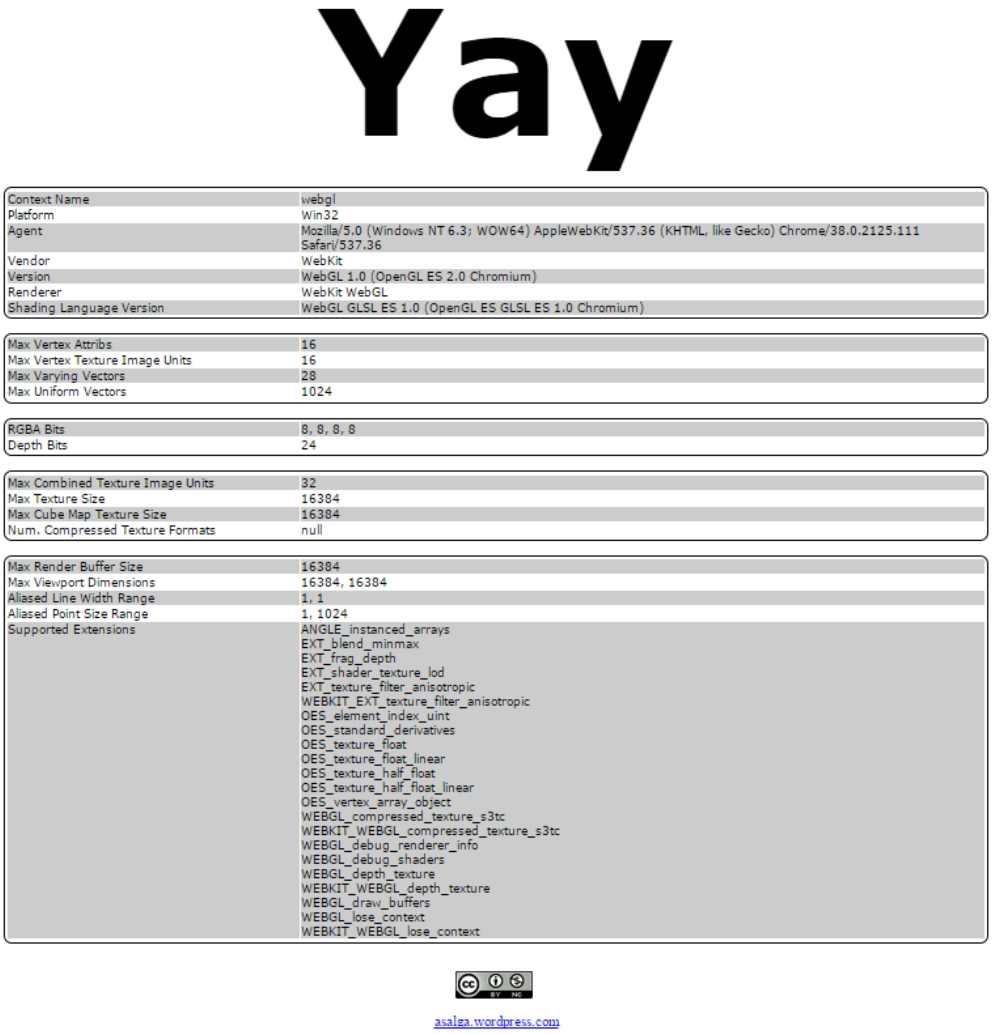


Figure 8: Screenshot of the Does My Browser Support WebGL Report [66]

[65] This second report is similar to the first report, but in addition it shows additional information of what is supported by the browser and the device. The report queries the browser and the device for parameters for the running of WebGL. If the report deduces that the browser and the device can effectively run WebGL a detailed report is generated. The report consists of five parts; General Meta information, Vertex Shader details, bit info, Texture info, and additional details.

Below is a list of details that are collected:

Context Name, Platform, Agent, Vendor, Version, Renderer, Shading Language Version, Max Vertex Attributes, Max Vertex Texture Image Units, Max Varying Vectors, Max Uniform Vectors, RGBA Bits, Depth Bits, Maximum Combined Texture Image Units, Maximum Texture Size, Maximum Cube Map Texture Size, Number Compressed Texture Formats, Maximum Render Buffer Size, Maximum Viewport Dimensions, Aliased Line Width Range, Aliased Point Size Range and Supported Extensions

As you can see this report is a very detailed report and any differences that arise from the different devices and browsers will be highlighted in the analysis.

5.1.1.3 Report 3: WebGL Report

http://webglreport.com/

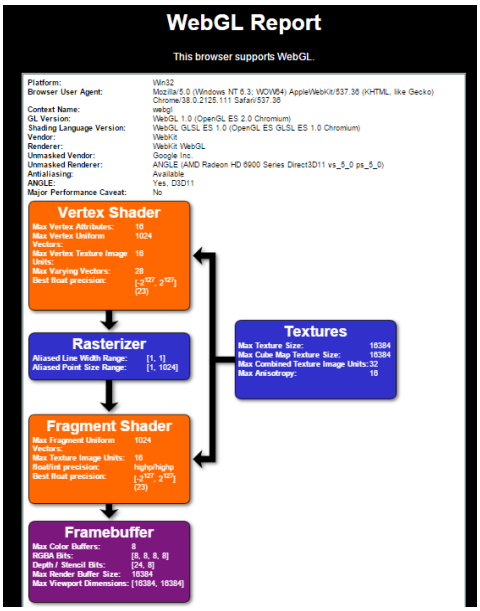


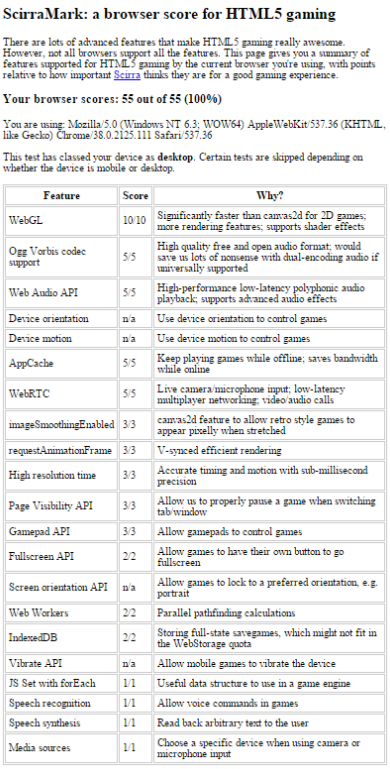
Figure 9: Screenshot of the WebGL Report [67]

This report is similar to the above report but looks into other additional fields. This report will also highlight the unmasked Vendor and the Unmasked Renderer of WebGL, looking at the GPU that is rendering the graphics for the browser. It will check to see if anti-aliasing is available and if there is any performance caveats and it will look at the best float point precision for the vertex shader. It will also gather details on the Fragment Shader looking at the maximum fragment

uniform vectors, maximum texture image units and the float integer precision. Again, a comparison between the reports, between the browsers and the devices will be made and the differences will be identified. A lot of the detail in this report is duplicated from report 2 “Does my browser support WebGL” and only the information that is not duplicated will be analysed.

5.1.1.4 Report 4: ScirraMark browser score for HTML5 gaming

http://www.scirra.com/labs/scirramark/



ScirraMark: a browser score for HTML5 gaming

There are lots of advanced features that make HTML5 gaming really awesome. However, not all browsers support all the features. This page gives you a summary of features supported for HTML5 gaming by the current browser you're using, with points relative to how important Scirra thinks they are for a good gaming experience.

Your browser scores: 55 out of 55 (100%)

You are using: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.111 Safari/537.36

This test has classed your device as **desktop**. Certain tests are skipped depending on whether the device is mobile or desktop.

Feature	Score	Why?
WebGL	10/10	Significantly faster than canvas2d for 2D games; more rendering features; supports shader effects
Ogg Vorbis codec support	5/5	High quality free and open audio format; would save us lots of nonsense with dual-encoding audio if universally supported
Web Audio API	5/5	High-performance low-latency polyphonic audio playback; supports advanced audio effects
Device orientation	n/a	Use device orientation to control games
Device motion	n/a	Use device motion to control games
AppCache	5/5	Keep playing games while offline; saves bandwidth while online
WebRTC	5/5	Live camera/microphone input; low-latency multiplayer networking; video/audio calls
imageSmoothingEnabled	3/3	canvas2d feature to allow retro style games to appear pixelly when stretched
requestAnimationFrame	3/3	V-synced efficient rendering
High resolution time	3/3	Accurate timing and motion with sub-millisecond precision
Page Visibility API	3/3	Allow us to properly pause a game when switching tab/window
Gamepad API	3/3	Allow gamepads to control games
Fullscreen API	2/2	Allow games to have their own button to go fullscreen
Screen orientation API	n/a	Allow games to lock to a preferred orientation, e.g. portrait
Web Workers	2/2	Parallel pathfinding calculations
IndexedDB	2/2	Storing full-state savegames, which might not fit in the WebStorage quota
Vibrate API	n/a	Allow mobile games to vibrate the device
JS Set with forEach	1/1	Useful data structure to use in a game engine
Speech recognition	1/1	Allow voice commands in games
Speech synthesis	1/1	Read back arbitrary text to the user
Media sources	1/1	Choose a specific device when using camera or microphone input

Figure 10: Screenshot of the ScirraMark Report [68]

This report was developed by Scirra, a games engine developer that has a product on the market Construct2 that one of its outputs is to HTML5 which includes WebGL and one of their platforms that they deliver to is WebGL as well as other platforms. The ScirraMark is a general test and one of its components is WebGL. However, as WebGL is used a lot with games a series of tests are done with features of browsers that are associated or used with WebGL. With regards to this series of tests, the interest lies in the WebGL part but taking into consideration the other features provides an indication of how effective the browser is. The test provides a summary as a percentage or a score out of 55.

5.1.2 Benchmarks

In addition to the reports generated, a series of benchmark tests are performed on the browsers running on the various devices. A collection of benchmarks using different frameworks and libraries and from different sources was to remove the bias from one benchmark to another. This is to provide an objective view of the performance of WebGL on these devices and the browsers.

Each of the benchmarks was performed twice on each browser and device and a mean average of the two results was recorded. The tests were performed after a restart with a one-minute delay to allow any background services to complete. The tests were all performed on a fully charged device. The device was left on a flat surface until the test had been completed so that there was not any interference from any of the other sensors or heat transfer from the hands to the heat sink of the device. Each of the devices had the screen timeout maxed or turned off to not allow the screen to dim or turn off during the test.

The benchmarks were initially run after restarting with no apps or tabs in the background and the results were recorded. For further analysis, more tests were conducted by adding additional stress to the device's hardware and software by having software running and/or extra tabs running in the background. For each device and browser, a series of tests was first done with a full stress test by loading five applications in the background with five tabs. The five applications that ran in the background were the following: Facebook, Twitter, YouTube, Camera and a Default Mail Application. The choice for these applications is that these are either default applications that are with the device and Facebook, Twitter and YouTube are three popular social media apps that can be found on all popular mobile platforms. For the desktop equivalent tests, there are no equivalent apps and the closest apps chosen were the following. VLC Media Player running Big Buck Bunny test video as MP4 file format. [72], Microsoft Word 2013, Microsoft Excel 2013 and Microsoft Outlook 2013 on Windows and Microsoft Word 2011 & Microsoft Excel 2011 on Mac and Apple Mail on Mac, Windows Photo Viewer on PC and Preview on Mac loading one picture, colorbands.png from the Lagomi LCD Test page. [73].

5.1.2.1 Benchmark Test 1: BabylonJS Train Demo

http://www.babylonjs.com/index.html?TRAIN

Figure 11: Screenshot of the Babylon JS Train Demo [74]

In this benchmark we are running a demo animation built by the BabylonJS WebGL game development platform. This benchmark has been used by other benchmark tests [54] to perform the performance of WebGL. The test comprises of a train going around a train track and display of the FPS. The value of the FPS is recorded on the 10th second by a measure of a timer. This is to allow the renderer to have cached the objects and pre-rendered, and to measure the effectiveness of the platform of delivering the render of the scene at a desirable frame rate. The demo for the train scene has many parameters that can be adjusted but all the parameters had been left on default.

5.1.2.2 Benchmark Test 2: Aquarium

https://webglsamples.googlecode.com/hg/aquarium/aquarium.html

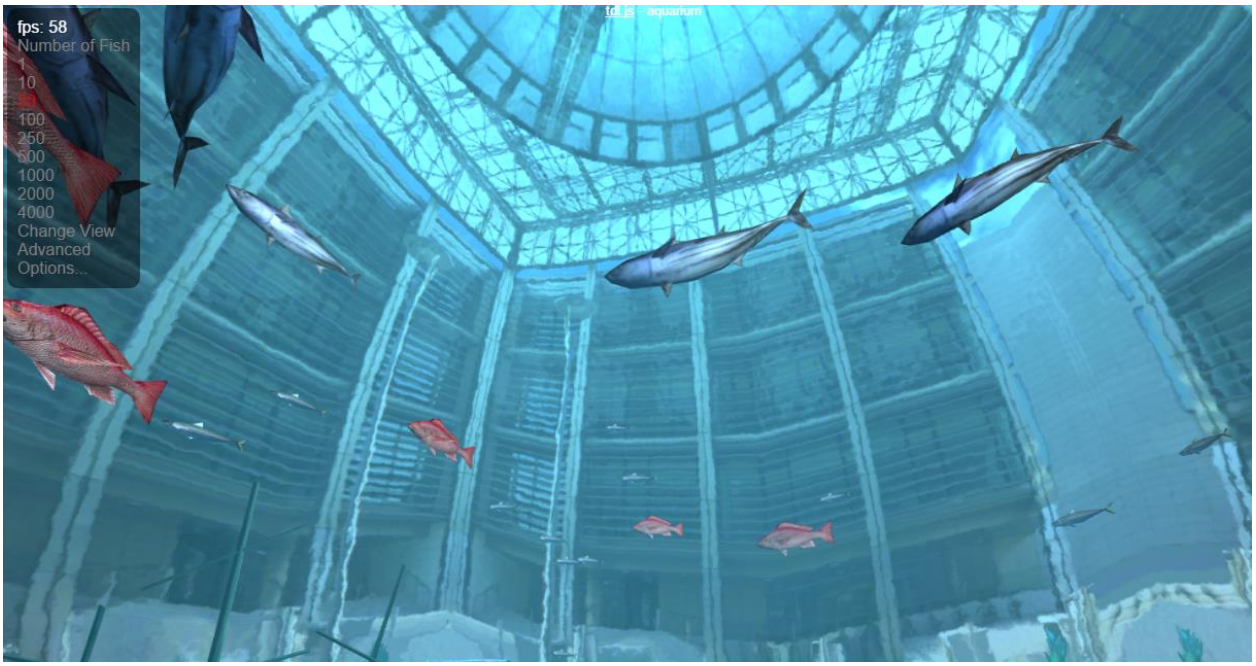


Figure 12: Screenshot of the Aquarium Benchmark [75]

This benchmark tests the performance of WebGL in the browser by populating the browser with 3D fish. The benchmark uses a TDL low-level WebGL library. The test allows the user to increase the number of rendered 3D fish to the display. The dials are set at 1, 50, 100, 250, 500, 1000, 2000, and 4000. The FPS is measured for each of the renders for 50 objects and above. Similar to the previous test, a timer for ten seconds is performed on the test before the recording of the displayed onscreen FPS to allow any models and the scene to have fully loaded and cached in the rendering pipeline. The tests have many other variable parameters but they have all been left at default apart from the number of fish displayed simultaneously. This test is performed on each device and on each of the respective browsers.

5.1.2.3 Benchmark Test 3: WebGL-Bench

https://webgl-bench.appspot.com/

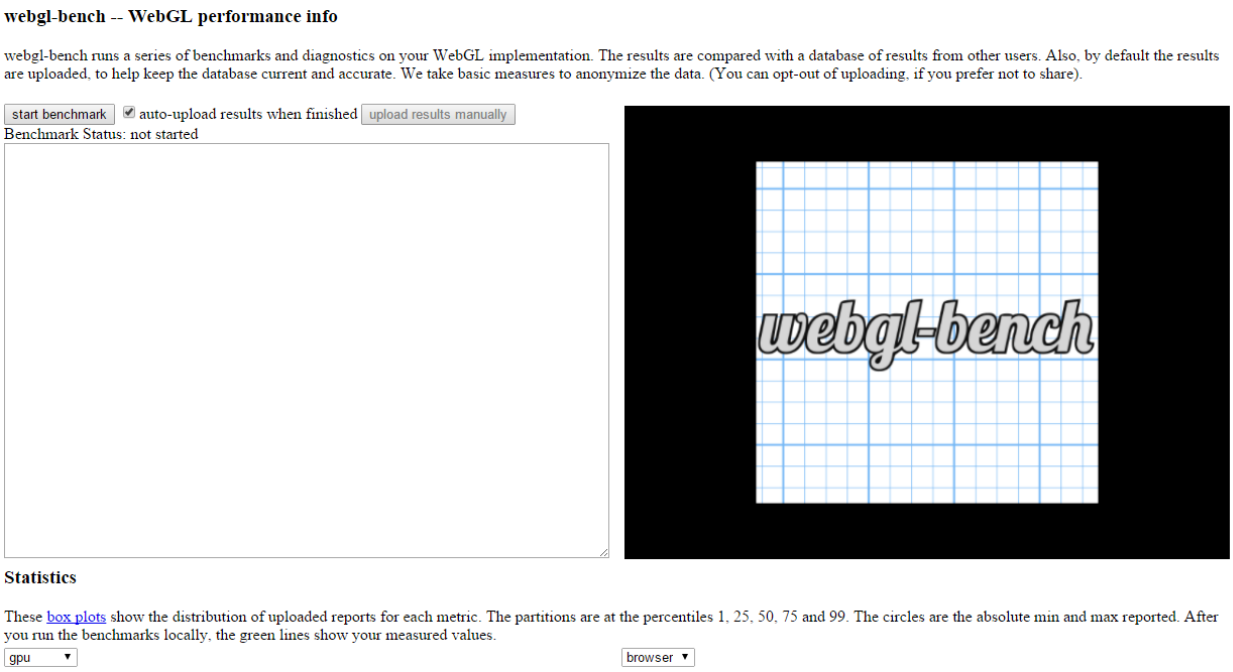


Figure 13: Screenshot of the webgl-bench Benchmark [76]

"WebGL-bench runs a series of benchmarks and diagnostics" [76]. The benchmark produces a detailed report at the end of the test. The benchmark tests each of the specified extensions of WebGL that are supported for the browser. In addition, it will produce a metric index result for the test of unique textures at 32bit and 64bit. These values are recorded and are compared between the different browsers and devices for further analysis.

5.1.2.4 Benchmark Test 4: WebGL - Performance Benchmark

http://luic.github.io/WebGL-Performance-Benchmark/ [77]

This benchmark is a series of three consecutive tests. The benchmark was developed for Remi Arnaud from Advanced Micro Devices (AMD). Byron Kropf and Charles Lui developed the test. The purpose of the test is to provide a more accurate measure to measure the graphics performance of WebGL on a device. Most benchmarks that exist tend to measure the frame rate, but the frame rate alone is not, singularly, an effective measure of performance. These three tests

measure the amount of work that the device rendering the tests produce. The tests use the three.js library to render the objects on screen. In addition to using the WebGL three.js library the benchmark also displays a graphical analysis of the measurement of the changes in frame rate during the test and a report at the end. The report and the graph are displayed by using the d3.js and the real-time display of the frame-rate is provided by stats.js

The first of the three tests is the Pixel Fill Rate with Z-buffering disabled. This test is complex and demanding on the GPU. This is done by sending objects through the pipeline in quick iterations that are drawn and redrawn in each frame. It will come to a point where the GPU will find it difficult to render the number of objects all in one frame and may delegate this to the following frame by breaking the job up. For simplicity, the objects that are drawn are flat two-dimensional coloured rectangles that fill the entirety of the screen. This is repeated with progressively increasing the number of objects for each frame. This test is run twice, once with the Z-buffer disabled and the other with the z-buffer enabled. The z-buffer, or also known as depth buffering is a technique that graphic rendering engines use in three dimensions to maximise the number of objects that can be drawn on the screen at the same time. By only rendering the objects that are visible from the viewpoint of the camera. For example, if we have three objects that are one in front of another the Z-axis, where the object closest to the camera is blocking the view of the other two objects then only the front object is rendered. This saves processing time from the GPU and allows more objects to be drawn on the screen as the objects that are not visible are not drawn.

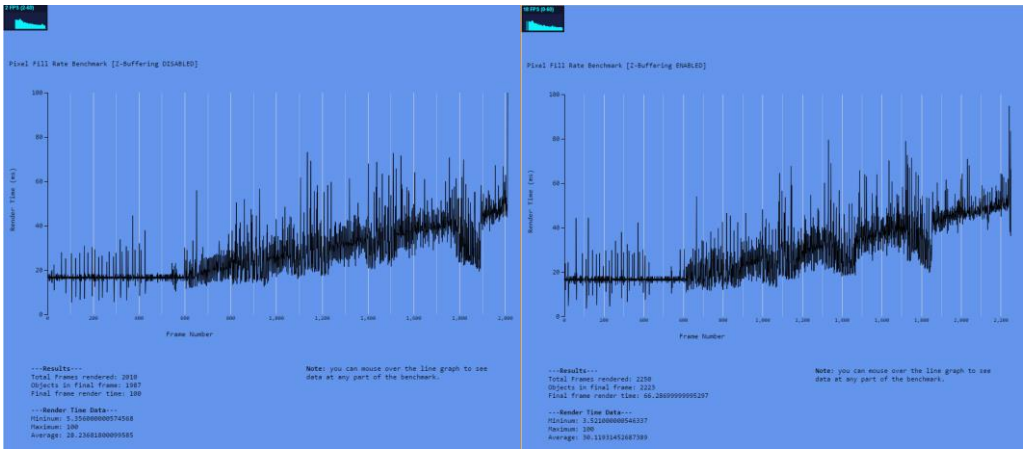


Figure 14: Screenshot of the Z-Disabled and Z-Enabled Graph Report of the WebGL performance benchmark [77]

The first of these two tests is done by having the z-buffer disabled therefore causing the graphics renderer to render all objects regardless if it is viewable from the camera or not. Then this test is performed again but this time it is with the z-buffer enabled allowing it to maximise the performance. The results can then be compared with each other and with other devices and this will give you a measure of how effective the graphics renderer is at using the z-buffer as well as comparing the results with one device and another and one browser and another.

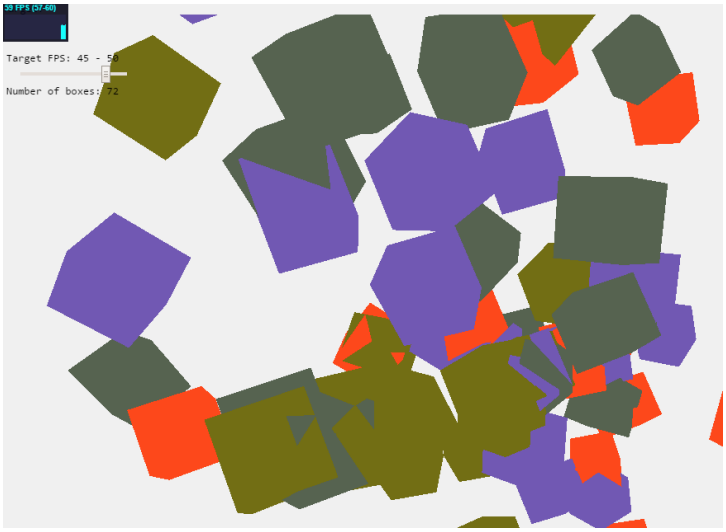


Figure 15: Screenshot of the Cubes Benchmark of the WebGL performance benchmark [77]

The third test measures how many objects can be displayed, rendered and animated simultaneously on screen to effectively keep the frame rate between 45 and 50 FPS. The

benchmark will progressively increase the number of objects, in this case cubes constructed of 12 polygons (two polygon triangles per side) until the frame rate has reached a range between 45-50, if the frame rate drops below 45 the program will decrease the number of objects and if it is above 50 it will increase the number of visible objects. As this test will give different values as it will fluctuate the measure of cubes on screen. The maximum number of objects that the renderer, GPU and browser can ever display effectively to achieve the desired frame rate is recorded. Due to the architecture of GPUs on smartphone devices and due to the low power consumption there has been heavy optimisation and power management built in to the devices. When the processor and/or GPU reach a certain temperature, the clock speed of the GPU will scale down to allow the processor to cool down. This is so that the device does not overheat and cause the device to crash. This can have an impact on the performance of the tests and this was noticeable in this particular test hence why the highest value was recorded.

5.1.2.5 Benchmark Test 5: Scirra - WebGL and Canvas

http://www.scirra.com/demos/c2/renderperfgl/

http://www.scirra.com/demos/c2/renderperf2d/

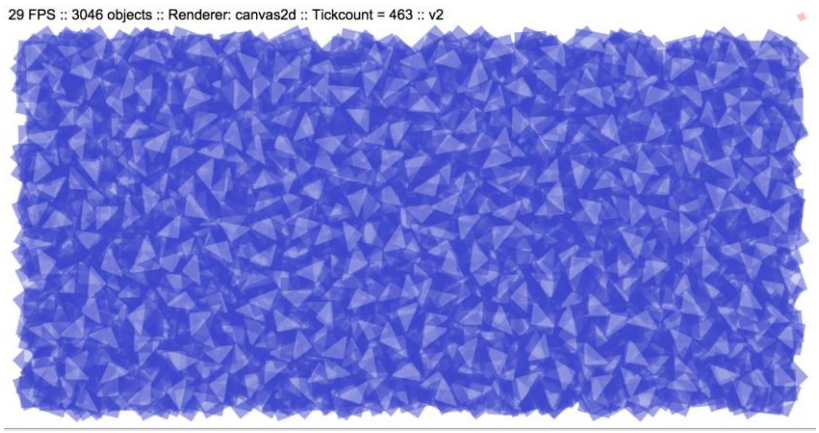


Figure 16: Screenshot of the Canvas 2D Scirra Object Benchmark [78]

Scirra, a games engine development company, developed this benchmark. This is a benchmark that complements report 4 discussed earlier. The benchmark contains two parts. The first part tests the browser's capabilities in rendering WebGL [79] and the other benchmark is identical to the first but renders using the HTML5 `<canvas>` tag [78] as opposed to using WebGL technology. This is a good test from a comparative point of view, as this will show how well the

browser is at rendering WebGL compared to drawing the object using the `<canvas>` tag. The way the benchmark works is that it progressively adds triangles (polygons) on the screen in a 2D format, layering the triangles on top of each other. A measure of how many triangles is recorded. This is progressively added until the frame rate reaches 30 FPS. When it reaches 30 FPS it will reduce the number of objects on screen. As with other benchmarks, the value of the objects on screen will fluctuate and to have a fair test between the various different devices, I allowed the renderer to reach the 30 FPS then a timer is started and after exactly ten seconds the recorded value of the objects rendered on the screen is recorded. This is done twice to counter the fact for any mishaps or miscalculations and a mean average of the two is taken. The other value that the benchmark displays is the tick counter and is just a measure of how long it took to get to the goal of 30 FPS. Both tests were conducted, the WebGL and the canvas test in exactly the same manner and the results recorded for each browser on each device.

5.1.2.6 Benchmark Test 6: PixiJS - BunnyMark

http://www.goodboydigital.com/pixijs/bunnymark/

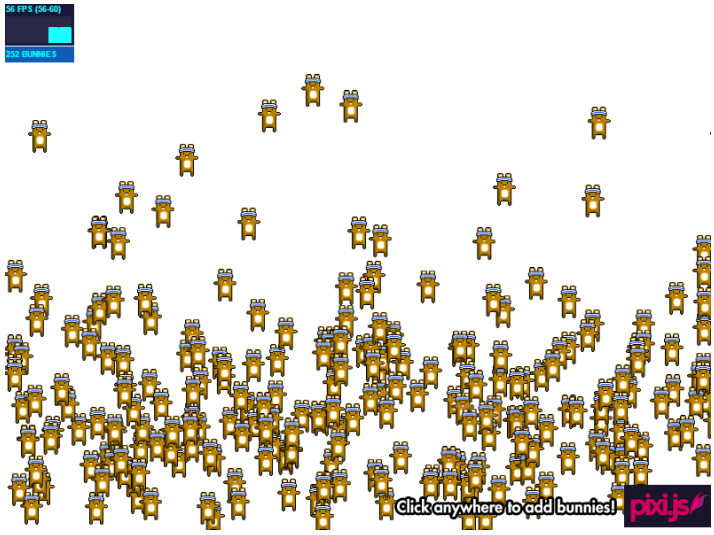


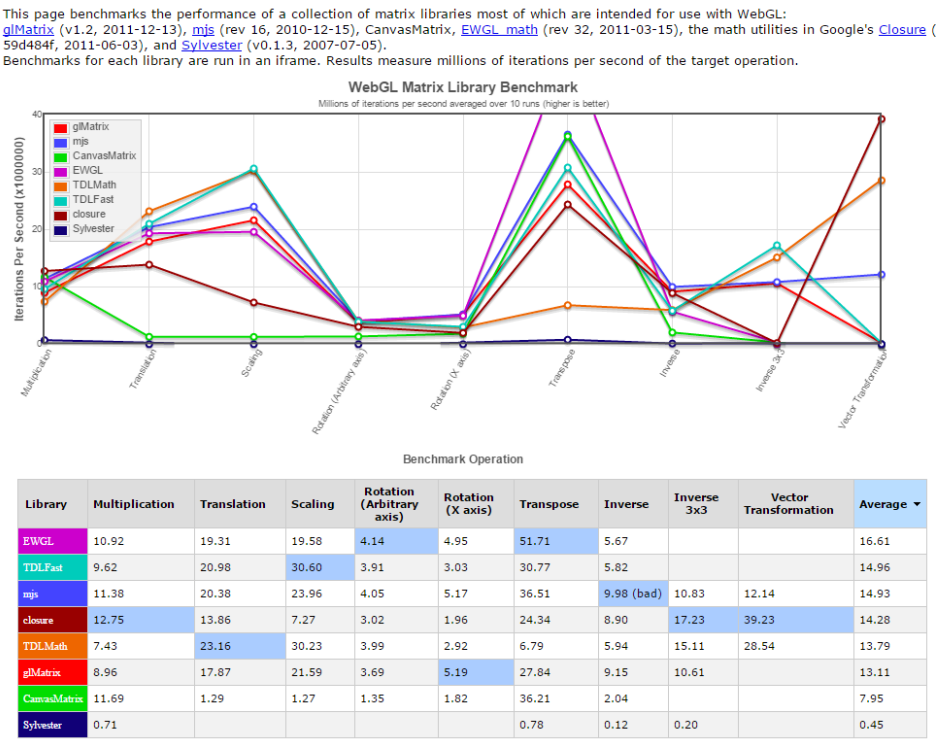
Figure 17: Screenshot of the Pixi.js BunnyMark WebGL Benchmark [80]

This benchmark tests only the 2D capabilities of the browser, and in this benchmark is using the PixiJS framework to render the results on the screen. The benchmark [80] was developed by the PixiJS developers. The benchmark allows you to add additional two-dimensional objects or sprites on the screen, in this case ‘bunnies’. The tester can add as many as needed by clicking on the screen and upon doing so the benchmark will add a collection of bunnies to the render scene.

There is a measure of the counter on the top left that is using stats.js from the three.js library that is measuring the frame rate. The benchmark was to add as many bunnies as possible on the device until the frame rate had dropped to 30 FPS. As soon as it had dropped to the specified frame rate the number of bunnies were recorded. This measure will be used to measure the effectiveness of the browser and device in rendering two dimensional objects using the PixiJS framework.

5.1.2.7 Benchmark Test 7: WebGL - Matrix Benchmark

http://stepheneb.github.io/webgl-matrix-benchmarks/matrix_benchmark.html



Library	Multiplication	Translation	Scaling	Rotation (Arbitrary axis)	Rotation (X axis)	Transpose	Inverse	Inverse 3x3	Vector Transformation	Average
EWGL	10.92	19.31	19.58	4.14	4.95	51.71	5.67			16.61
TDLFast	9.62	20.98	30.60	3.91	3.03	30.77	5.82			14.96
mjs	11.38	20.38	23.96	4.05	5.17	36.51	9.98 (bad)	10.83	12.14	14.93
closure	12.75	13.86	7.27	3.02	1.96	24.34	8.90	17.23	39.23	14.28
TDLMath	7.43	23.16	30.23	3.99	2.92	6.79	5.94	15.11	28.54	13.79
glMatrix	8.96	17.87	21.59	3.69	5.19	27.84	9.15	10.61		13.11
CanvasMatrix	11.69	1.29	1.27	1.35	1.82	36.21	2.04			7.95
Sylvester	0.71					0.78	0.12	0.20		0.45

Figure 18: Screenshot of the WebGL Matrix Benchmark [81]

This benchmark [81] test is unique and different to the other tests previously conducted. Where the other tests are tested on different frameworks and in most cases test the three-dimensional rendering capabilities of the browser and the GPU. It does not take into account or the benchmark does not focus on the execution of the compilation of the JavaScript required to render the scene. JavaScript being a JIT compiler and by having a slow compiler in the browser can dramatically affect the performance of the rendering of WebGL. WebGL as a technology and programming language relies heavily on the use of matrices and the calculation of matrices. The

libraries that the benchmark compares are the following EDGL, TDLFast, mjs, closure, glMatrix, TDLMath, CanvasMatrix and Sylvester. These libraries are not exclusive to WebGL but they are indicative of how WebGL will perform on the various browsers and devices. The benchmark performs the same matrix calculations on each of the frameworks for features that are commonly used in graphics rendering and animation. The calculations that are tested are multiplication, translation, scaling, rotation (through the arbitrary and X axis), transpose, inverse, inverse 3x3 and vector transformation. An average index score is given to each library for comparison. In addition, a line chart is plotted comparing the various different libraries and the calculations tested. This test is not conclusive in its own right, as it does not test the rendering capabilities of the graphics engine. Also there are many other WebGL libraries that are used commercially and popular within the WebGL sphere that are not included as part of this test.

5.1.2.8 Benchmark Test 8: WebGL - three.js - Benchmark

http://threejs.org/examples/webgl_performance_doublesided.html

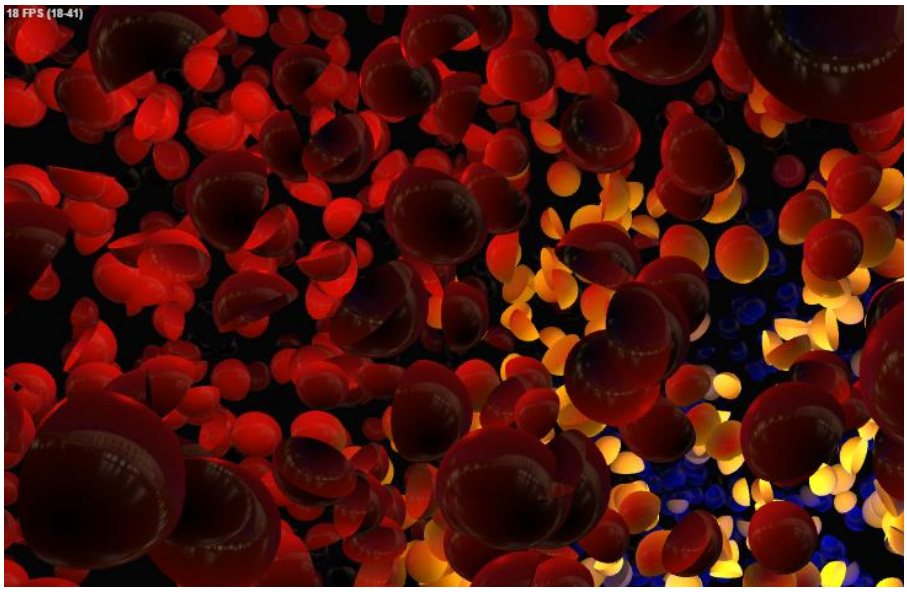


Figure 19: Screenshot of the three.js WebGL Double-sided Performance Benchmark [82]

This benchmark [82] tests one engine in particular and it tests the three.js framework developed by Robert Caballo also known within the community as his alias MrDoob. The benchmark focuses on testing an extensive list of features within the framework. three.js is one of the most popular frameworks used today for the display and rendering of web graphics in particular 3D graphics. The features that this benchmark tests are the following 3D geometry rendering,

animation, point lighting, reflection, 3D object image texturing and specular Phong shading. By all means this is not a conclusive test of all the features of three.js library but is indicative of how it will perform within the browser.

The test that was conducted did not take into account the animation aspect of the benchmark. The frame rate of the still render was recorded as with other tests at exactly after ten seconds to allow the graphics pipeline to have cached the 3D objects and scene fully allowing the frame rate to have stabilised. This test was performed on each device and each browser with different conditions.

5.2 Devices tested

A wide range of devices and platforms had been used to conduct the tests. The tests were performed on desktops, laptops and smartphones consisting of different hardware and platforms with varying operating systems. This was to illustrate the fact that WebGL is used on a wide array of devices as the technology is web-based and the control of how or where it is used can be difficult if not impossible.

The devices tested mainly fall into two categories. Although, one of the devices is a laptop, this will still fall under the desktop category, as it is not a cellular mobile device. The other category is smartphones. Although the study focuses on the suitability of WebGL on smartphones, the study needed a comparison of benchmarks of the same or similar tests on desktops. The purpose of choosing the desktops as a platform for performing the tests on is to compare how WebGL should be ideally performing, and to provide a benchmark of comparison in terms of how WebGL could be performing if it was on par to the desktop versions.

The first device used for testing was a traditional desktop tower PC with an Intel quad-core Xeon processor and a desktop paced PCI-Express graphics card and 8GB of random access memory (RAM). The software installed is Microsoft Windows 8.1 updated with the latest patches and installed with the most popular browsers, Internet Explorer 11, Google Chrome, Mozilla Firefox, Apple Safari and Opera. All the browsers had been updated to their latest versions. The machine was freshly formatted and installed with no other applications in the background so to not interfere with the results of the test.

The other desktop device is a MacBook Pro with 8GB RAM updated to the latest OS X operating system Yosemite 10.10. This had been a fresh install of the OS with the browsers required for the test installed and updated to the latest version.

For the other group of devices being the smartphones, a range of different manufacturers, operating systems and platforms had been chosen to give the study a wide range of devices that reflect the actual usage of WebGL. As WebGL is a web technology that is provided through the means of the Internet and via a browser it is difficult to control the end-user devices that the technology is run on. The device range from low-end devices to high-end devices with ranging price points and release dates. The tests were performed on three mobile platforms; Apple iOS, Android and Windows phone. Ideally, a test would be done on every smartphone on the market but due to time constraints a selection of four devices were chosen.

First of these devices is the Apple iPhone 6. The reason this device was chosen is that iOS is the number one mobile platform used in the UK and the iPhone 6 is Apple's latest flagship device from all iOS devices. Prior to the release of the iPhone 6, WebGL was not supported on Apple devices. Only with the release of iOS 8 and Apple Safari 8 that WebGL is now officially supported. The Safari browser is based on the WebKit engine that does support WebGL. However, this feature was not turned on or available by default and can only be activated by a developer flag on versions prior to version 8. Second to the iPhone is the latest flagship Android phone installed with the stock KitKat version of Android version 4.4.4 on the LG Nexus 5. For comparison a Samsung Galaxy Nexus I9250, Google's flagship smartphone two years prior was also chosen. Lastly a budget smartphone from the Nokia Lumia range, the Lumia 520 but with the latest operating system of Windows Phone 8.1 with Internet Explorer 11 which now supports WebGL.

The browsers chosen for the devices had been the default stock browsers that comes with the devices operating system upgraded to the latest version available. For Windows platforms, it was Internet Explorer 11. Apple products stock browser is the Safari browser 8 on the iPhone. Unfortunately it was not possible to match all browsers on all platforms, for example, Internet Explorer is not available on Apple products. Safari is available on the desktop Windows platform but only up to version 5, which does not support WebGL. Google Chrome was used on all platforms apart from Windows Phone due to unavailability. The selection of these browsers [45]

was chosen as they are the most popular browsers in usage as displayed on StatCounter. Opera was installed on all browsers apart from the Windows Phone and iPhone. Opera Mini was used on the iPhone as the full-fledged Opera was not available on the platform but it was the most suitable alternative. Unfortunately, with the Windows Phone there were no other browsers on the market to conduct the tests on so only the stock browser was used.

With regards to Firefox, this can be found on all platforms apart from Windows Phone 8 and Apple iPhone 6. Unfortunately, Apple has some restrictions about the release of different browser engines to the stock engine that is built into the operating system, that being WebKit that both Safari and Google Chrome share. So in reality both Safari and Google Chrome should be comparatively similar. Mozilla refused to release Firefox on the Apple platform due to the restrictions imposed by Apple [83].

5.2.1 Specifications:

http://www.gsmarena.com/

In this section a list of hardware and software specifications of all the devices used for the benchmark suite. The manufacturers of the devices provide these specifications. Also listed here is the browser versions used for the test.

5.2.1.1 Device 1: Desktop - Personal Computer (Microsoft Windows):

 [84]

Specifications:	**Web Browsers:**
Microsoft Windows 8.1 64-bit	Internet Explorer version 11.0.9600.17351
CPU: Intel Processor Xeon CPU E3-1230 V2 @ 3.30 GHz	Google Chrome version 38.0.2125.104 m
GPU: Radeon AMD HD 6950 with 2GB of VRAM / Driver Catalyst version 14.9	Apple Safari 5.1.7 (7534.57.2)
	Mozilla Firefox 33.0.1

GPU Release Date: 15 December 2010

RAM: 8GB RAM

Display: 1600 x 1050 (1,764,000 pixels)

Opera 25.0.1614.50

5.2.1.2 Device 2: Laptop - Apple MacBook Pro (15 inch, Late 2011)

[85]

Specifications:

Release Date: 24 October 2011

OS X Yosemite 10.10

CPU: Intel Core i7 2.2Ghz

GPU: AMD Radeon HD 6750M 512MB of VRAM

RAM: 8GB of 1333Mhz RAM

Display: 1440 x 900 (1,296,000 pixels)

Web Browsers:

Apple Safari 8.0 (10600.1.25)

Google Chrome version 38.0.2125.104

Mozilla Firefox 33.0.1

Opera 25.0.1614.50

5.2.1.3 Device 3: Android Smartphone - Samsung Galaxy Nexus I9250 16GB

[86]

Specifications:

Release Date: November 2011

Android version 4.3 JellyBean

CPU: Dual-core 1.2 GHz Cortex-A9

GPU: PowerVR SGX540

RAM: 1GB

Display: 1280 x 720 (921,600 pixels)

Browsers:

Google Chrome 38.0.2125.102

Opera 24.0.1565.82529

Mozilla Firefox 33.0

5.2.1.4 Device 4: Android Smartphone - LG Nexus 5

[87]

Specifications:

Release Date: November 2013

Android 4.4.4 KitKat; Nexus 5 Build/kTU84P

CPU: Qualcomm MSM8974 Snapdragon 800

GPU: Adreno 330

RAM: 2GB

Display: 1080 x 1920 (2,073,600 pixels)

Browsers:

Google Chrome 38.0.2125.102

Mozilla Firefox 33.0

Opera 24.0.1565.82529

5.2.1.5 Device 5: Apple Smartphone - Apple iPhone 6 Model MG4F2B/A 64GB

 [88]

Specifications:

Release Date: 19th September 2014

iOS 8.1 (12B411)

Chipset: A8

CPU: Dual-core 1.4GHz Cyclone

(ARM v8-based)

GPU: PowerVR GX6450

(quad-core graphics)

Display: 750 x 1334 (1,000,500 pixels)

Browsers:

Apple Safari

Google Chrome 38.0.2125.59

Opera Mini 8.0.5

5.2.1.6 Device 6: Windows Phone - Nokia Lumia 520 8GB

 [89]

Specifications:

Windows Phone 8.10.12393.890

Release Date: April 2013

CPU: Qualcomm MSM8227 Dual-core 1GHz

GPU: Adreno 305

Display: 480 x 800 (384,000 pixels)

Browser:

Internet Explorer 11

Chapter 6: Results and Analysis

In this chapter I discuss the results collated from the reports and benchmark tests. Each report and benchmark are analysed independently as the underlying WebGL frameworks operate different to each other and each benchmark is compared for performance on the different devices and browsers. An analysis comparison is done between the devices to distinguish which frameworks and benchmarks are better suited for which browsers and/or devices.

6.1 Simple test run

The first test is to check to see if WebGL is compatible at the most basic level in the browser. This was conducted on all the browsers on all the devices and they had all passed with an exception of a couple of browsers. The browsers that failed the test were Safari on the Windows platform and Google Chrome on the Galaxy Nexus. The reason for this is Safari on the Windows platform is still on Safari version 5.1.7. Apple had released WebGL to Safari at version 8. I suspect that Apple will eventually update Safari on the PC platform to version 8, but for this study, Safari does not support WebGL. The reason why the test failed on Chrome on the Galaxy Nexus is that Google had stopped supporting updates to the Galaxy Nexus [90] and it is still on Android 4.3 JellyBean. Although, the version of Google Chrome is up to date as with other systems, the browser does support WebGL but this has been disabled by the underlying operating system. Overall, this is good progress as it is clear that the current and future releases of devices will have support and will continue to support WebGL going forward. Discontinued products may never receive the update to support the technology and this may be due to hardware limitations.

6.2 Features Test

After initially identifying which browsers on what devices can run and support WebGL, a further test had needed to be conducted to provide additional information on the level of support the browser has. WebGL had been in continuous development since its release and in addition to the development of the standard specification by the Khronos Group, browser manufacturers have been adding their own feature-set to the existing specification, providing features that go beyond the original specification. The two engines that had been providing this is WebKit that is the

underlying engine for Chrome, Safari and Opera. Mozilla have also been tweaking and enhancing the support of WebGL for their underlying browser engine. A couple of the reports (specifically report 2 and 3) have provided the results for interpretation.

A full comparison of the features can be found in the appendix.

6.2.1 Features Comparison

From interpreting the table of features test, I have selected some commonly used parameters with WebGL that may impact on the quality of the render. The comparison was made between all the browsers and the devices. The five parameters chosen to compare are RGBA Bits, Depth Bits, Max Combined Texture Image Units, Aliased Line Width Range and Supported Extensions.

Below I will explain what each of these parameters is and why they are useful as a comparison test between the devices rendering them.

RGBA Bits: This parameter looks at the number of bits used for each of the colour display channels, Red Green and Blue with A for Alpha or transparency. Traditionally, computer graphics use an eight bit depth for each channel giving a true colour spectrum at 24-bit depth when using the red, green and blue and combining this with an eight bit depth for transparency provides the renderer with a 32 bit depth image. To be able to provide this level of quality, the results that are to be expected from this test is eight bit depth per channel. And as expected, in all devices and all browsers this was the case.

Depth Bits: This value is the depth of the colours used Red Green and Blue giving a true colour representation at 24 bits, which is what is expected and this is the case between all browsers on all devices that supported WebGL.

Max Combined Texture Image Units: This parameter was inherited from the OpenGL library. Textures are used widely within 3D rendered graphics, as it is possible to ‘dress’ a wireframe with a textured pattern or image to provide a sense of realism. The textures used can be stored and queried for later use. This storage space is defined by the units within this parameter. The unit’s values differed widely between the devices. The Nexus 5 had 32 units, which compared well with its desktop counterpart. Most devices supported 24 units but the iPhone 6 lagged behind providing only eight units, as did the Galaxy Nexus. When comparing the browsers on

each device, they all provided the same number of units across the device that I suspect is a hardware limitation and not software specific.

Aliased Line Width Range: This parameter is also inherited from OpenGL. This provides a range of line widths that can be used for the rendered objects. By default the line width is one pixel and the anti-aliasing of the line is disabled but this can be enabled if required. Providing a larger range provides greater flexibility and if anti-aliasing is enabled, greater processing power to render the details of the blurring of the edges of the lines. This parameter contains two numeric values where the first is one and the other is dependent on the device and the browser. The larger the number the better the support. The iPhone and the Galaxy Nexus both had an upper limit of 16 whilst the Nexus 5 only had eight. The Mac OS X offered a range with an upper limit of 64 whilst Windows 8.1 only offered one as an upper limit. There are methods of post-processing that can be enabled by the means of extensions that can give the illusion of higher width ranges which may negate the need to have a larger range.

Number of Supported Extensions: The report listed each of the extensions that each browser supported. Each browser supported a varied and different number of extensions. The report had listed the extensions and for this part of the study a simple count of the number of extensions used for each browser. Some of these extensions were debug extensions while others were engine specific but overall it is an indication of how well supported WebGL is in the browser. (see the red bar in figure 20).

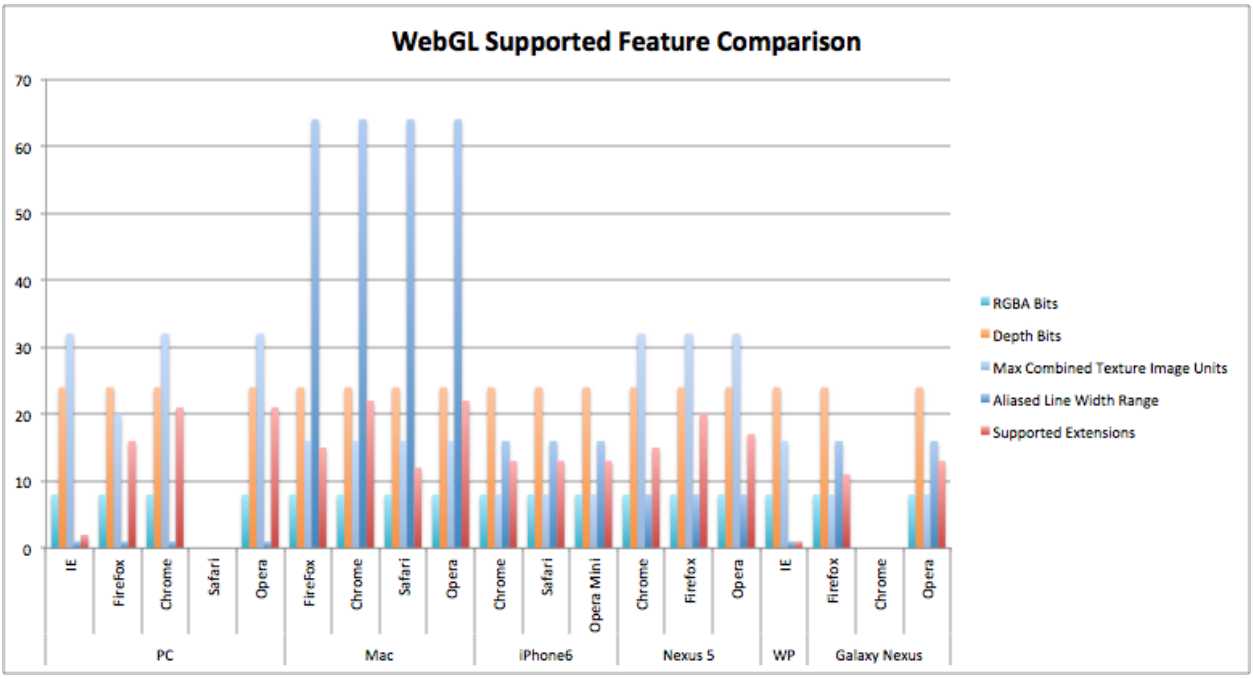


Figure 20: Chart comparing some of the WebGL features across different smartphones and browsers

6.2.2 Max Vertex & Varying Vectors Comparison

In this section, it explores the differences between three parameters in WebGL. These parameters are inherited from OpenGL like many other parameters. The three parameters investigated are maximum vertex attributes, maximum vertex texture image units and maximum varying vectors. For each of these attributes, the study looks at the maximum upper limit of support for each of those features. First, an explanation of what each of these features are.

Max Vertex Attribs: This parameter is key for the efficiency of rendering of WebGL. WebGL relies heavily on the vertex shader to produce shaders for the rendered 3D object. This attribute allows other parts of the graphics pipeline to communicate directly with the vertex shader. All devices both desktop and smartphone produced a value of 16 when queried apart from the Galaxy Nexus. The Galaxy Nexus was one of the first smartphones to experiment with the rendering of WebGL plus its graphics processor is the weakest out of all the ones tested which may be the contributing factor.

Max Vertex Texture Image Units: Similar to the Max Combined Image Units parameter examined earlier but this focuses on the textures particularly for the vertex shader alone. Once again a higher value the better. The Galaxy Nexus and iPhone only scored eight whilst all others

scored 16. Surprisingly though, Firefox on the PC platform only produced a value of four and Windows Phone rendered a value of zero.

Max Varying Vectors: This parameter details the complexity of how the Vertex Shader can deal with matrices and their calculations. For this study, a higher number indicates a more sophisticated rendering process for the vertex shader. The results that returned from this query varied from 32 to six. The iPhone 6 only produced a value of eight whilst Mac Safari, Chrome and Opera rendered a result of 32. The Nexus 5 had a value of 16 and Windows phone had a value of six.

A comparison of the maximum vertex attributes, maximum image units and maximum varying vectors across all browsers and in comparison between the desktop and the mobile browsers and in some features there is a drop in the number of maximum units as this can be seen in the maximum varying vectors. In comparison to the maximum vertex attributes which is the same across the board apart from the browsers that do not support it, Safari PC and Chrome Galaxy Nexus. Also the maximum vertex attributes on the oldest of the smartphones, the Galaxy Nexus seems to have half the features of all modern phones. From this, it can be clear that modern smartphones are clearly developing the hardware and the accompanying software to support such features as WebGL.

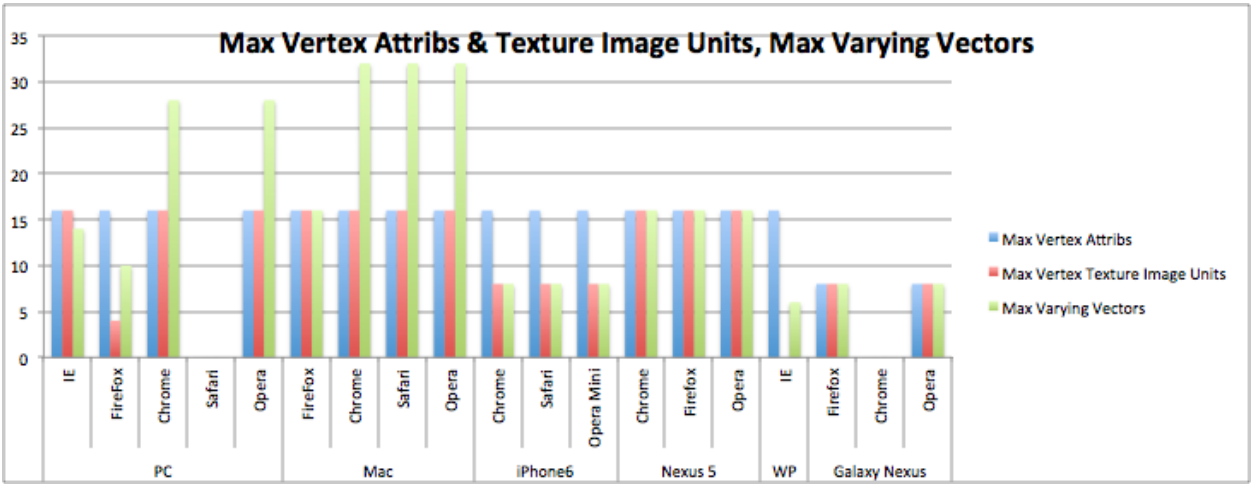


Figure 21: Chart comparing features relating to the Vertex Shader

6.2.3 Comparison of the Maximum Uniform Vectors across devices and browsers

To evaluate the differences between the results obtained in this query first a definition of the uniform vectors needs to be established. Uniform vectors are widely used and is a means of communication between the vertex shader and the graphics pipeline. By having a high number of uniform vectors, the GPU is able to parallel process multiple vectors simultaneously to render graphics at a higher rate. This is useful when using lighting and other shaders that utilise reflection. Graphics cards on PCs have been using 1000s of vertex shaders to achieve high quality rendering. The Chromium engine in Chrome (which is based on WebKit) directly translates the graphics-rendering pipeline of WebGL to DirectX, the native graphics-rendering pipeline for Windows operating system. This can be seen on the chart as Chrome and Opera both produce exceptionally high values of 1024.

In this example, looking at the maximum uniform vectors, there is a clear distinction in the number of uniform vectors that the hardware can support in comparison to desktop and mobile. The exception is Firefox on the PC, which is similar to the Nexus 5. Surprisingly, the iPhone 6 (the most recent of the smartphones) has the same number of maximum uniform vectors as the Galaxy Nexus on Chrome and Firefox. The analysis from this is that mobile phones are about a quarter in terms of the vertex shader capabilities to desktop PCs. Although iPhone 6 is the latest of the devices released its uniform vectors is half of what it is on the Nexus 5 at 256.

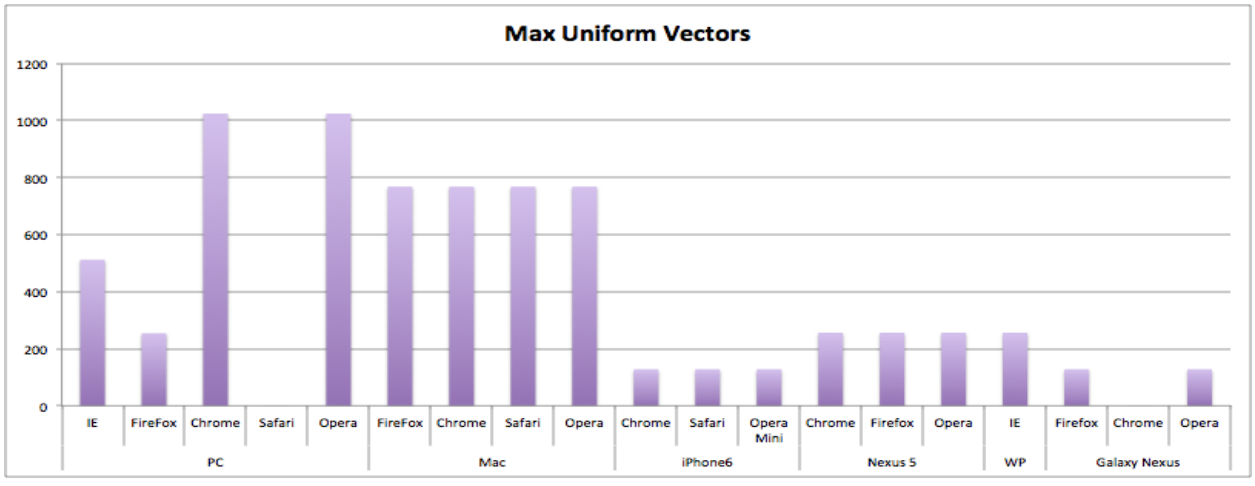


Figure 22: Chart comparing the maximum number of Uniform Vectors

6.3 Performance Testing

Once the feature set has been identified the next step is to test the actual performance of WebGL on the different devices. Each of the frameworks were tested independently and compared across the different devices and their browsers.

6.3.1 BabylonJS Test

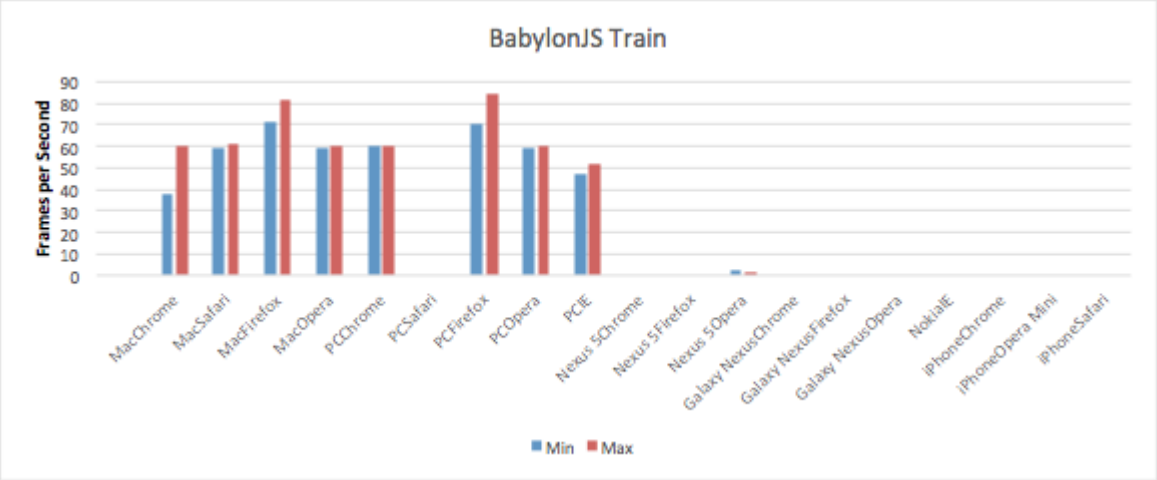


Figure 23: Chart comparing the minimum and maximum frame rates of the BabylonJS Train Demo.

One of the first benchmark tests conducted was on the BabylonJS framework. The library has different demo examples on their site and I chose the Train demo as this was used by Microsoft to demonstrate the performance of Internet Explorer 11 on release showcasing WebGL. The train demo was also used in other studies as a benchmark. This test did not produce any valid results on any of the smartphones as indicated in the chart. In a lot of cases the test would not even run or it would crash prior to loading. Firefox on the Galaxy Nexus had even caused the entire phone to crash and restart. This may not be an attributing factor to the framework of BabylonJS but to the demonstration used. The Train demo is a complex, detailed render and primarily built for desktop usage. But this is an indication that what is built for desktop may not be suitable for mobile and a separate render scene may need to be developed and optimised for mobile use.

6.3.2 Aquarium Test

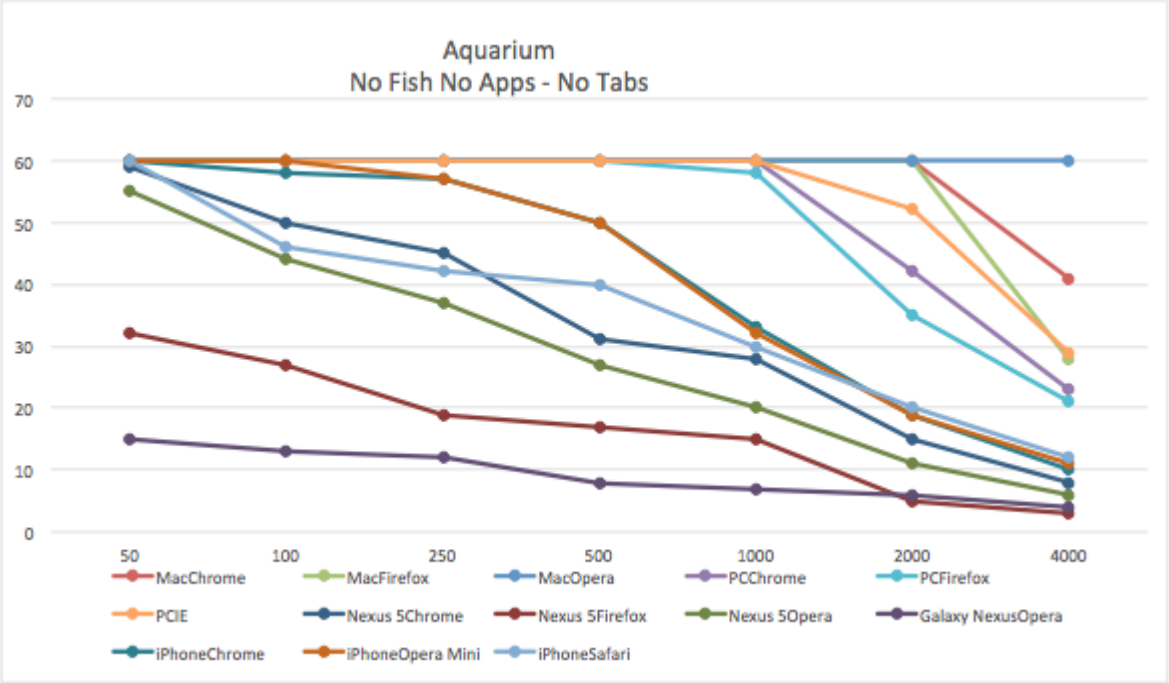


Figure 24: Chart comparing the frame rate as more objects are rendered on the screen simultaneously

This benchmark progressively adds a number of objects to the render scene and at each point the FPS is measured. The performance of the browsers across all devices performed differently. As the number of objects scaled to 2000 or 4000 the behaviour of the browser under strain had changed.

The browser that was un-phased by the number of fish on screen at any one time was Firefox browser for PC as it maintained a consistent frame rate of 60 FPS throughout. Looking at the 500 mark for the number of objects, the browser that comes in at top is the iPhone with all three browsers. The graphics chip in the new A8 chip is perfect at rendering graphics and this can be seen clearly in the results. Chrome and Opera Mini top the list, followed by Safari. This is surprising as Safari is the default browser for the operating system, yet it underperforms to the other two browsers on the same platform. Second comes the Nexus 5 with Chrome topping the list on that device and Firefox underperforming. The graph indicates that there is a large difference in performance on a small number of objects on the scene but when the number of

objects increases to a large number such as 4000 objects simultaneously they converge, providing similar results. This may indicate how the browser's engine interprets the underlying WebGL.

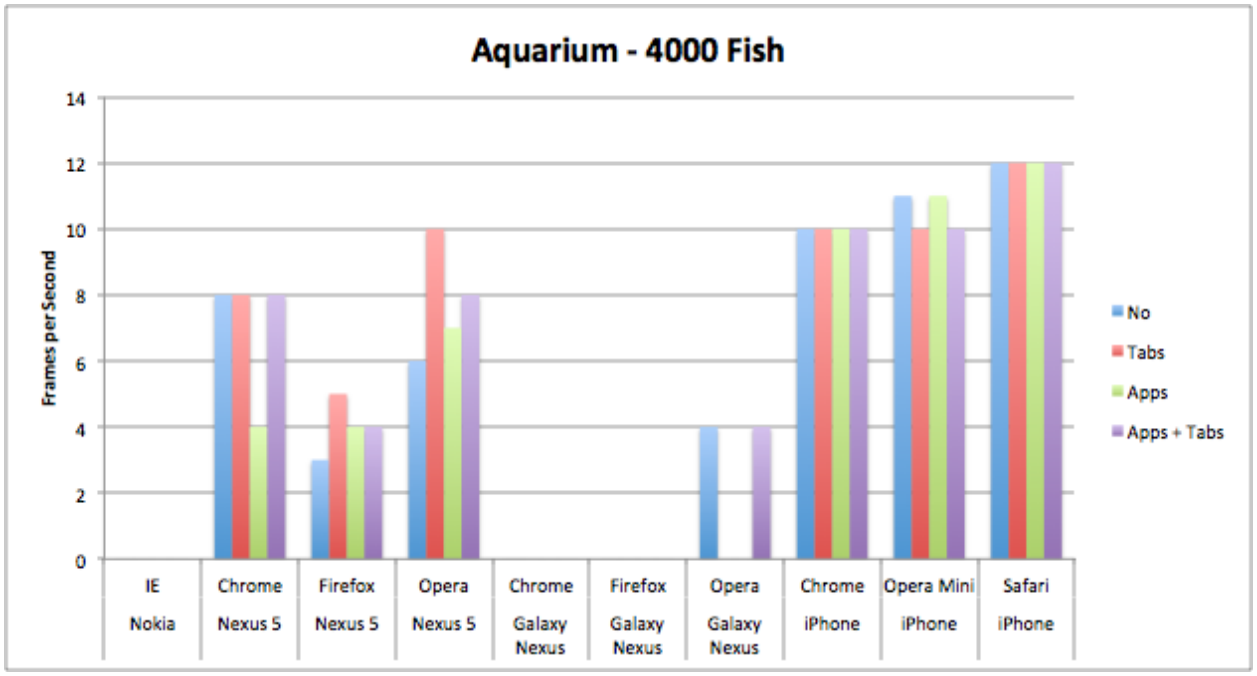


Figure 25: Chart comparing the frame rate for 4000 objects as they are rendered on the screen simultaneously. Comparison with tabs and/or apps.

A further test was conducted with this benchmark to determine if having apps and/or tabs run in the background would impact the performance. The comparison was measured with 4000 fish test. On the iPhone 6 in all browsers there is very little change if the system is loaded with apps in the background, having tabs running in the background or both. This indicates that the iOS browser is very efficient at managing the memory allocation and resources when the app enters the background transition state [91]. In comparison to the Nexus 5 where the results were varied when the tests were performed under different conditions. Chrome produced the same results but when there were apps in the background it had underperformed, but surprisingly when there were apps and tabs it performed just as well when there was nothing in the background. This may be attributed to the memory garbage collection of the underlying operating system and if the processor was busy shutting down or re-allocating resources at the time when the benchmark was performed this may have affected the result. The differences can be seen in the other browsers too such as Firefox and Opera on the Nexus 5.

6.3.3 WebGL vs. Canvas

Prior to using the library available in WebGL to render 3D graphics, developers had access to the `<canvas>` tag in HTML5 to render the graphics by 'painting' the pixels to the screen. Unfortunately, this has a big impact on performance and when rendering anything of medium to high complexity this is not advised. The Scirra framework provided a benchmark for comparison of both WebGL and Canvas by conducting the same test but using the two different methods.

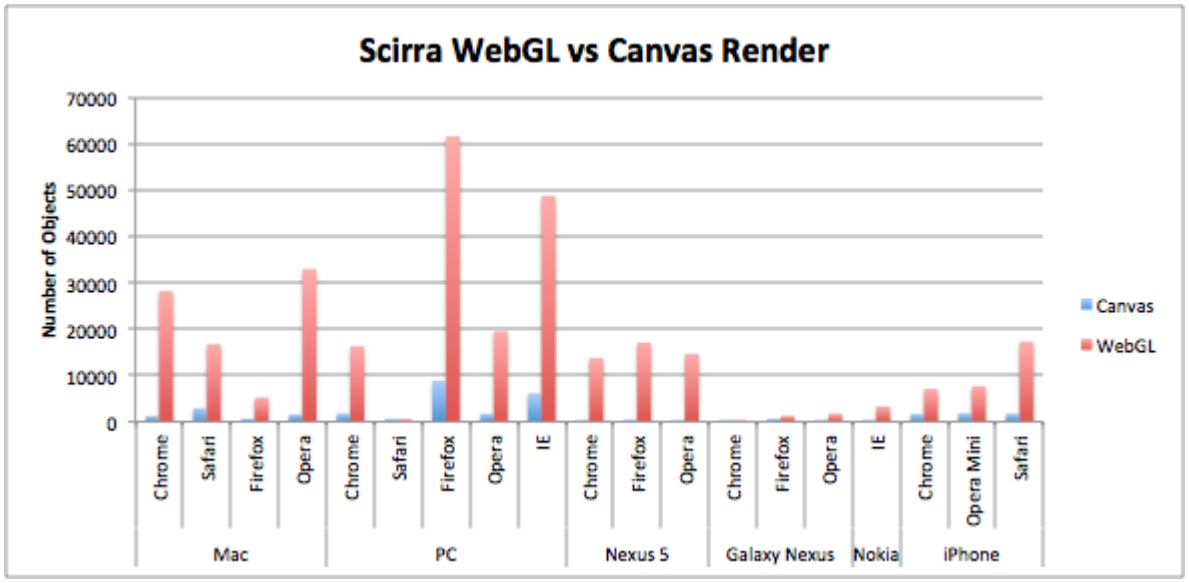


Figure 26: Comparison of the Scirra Benchmark using Canvas vs. WebGL

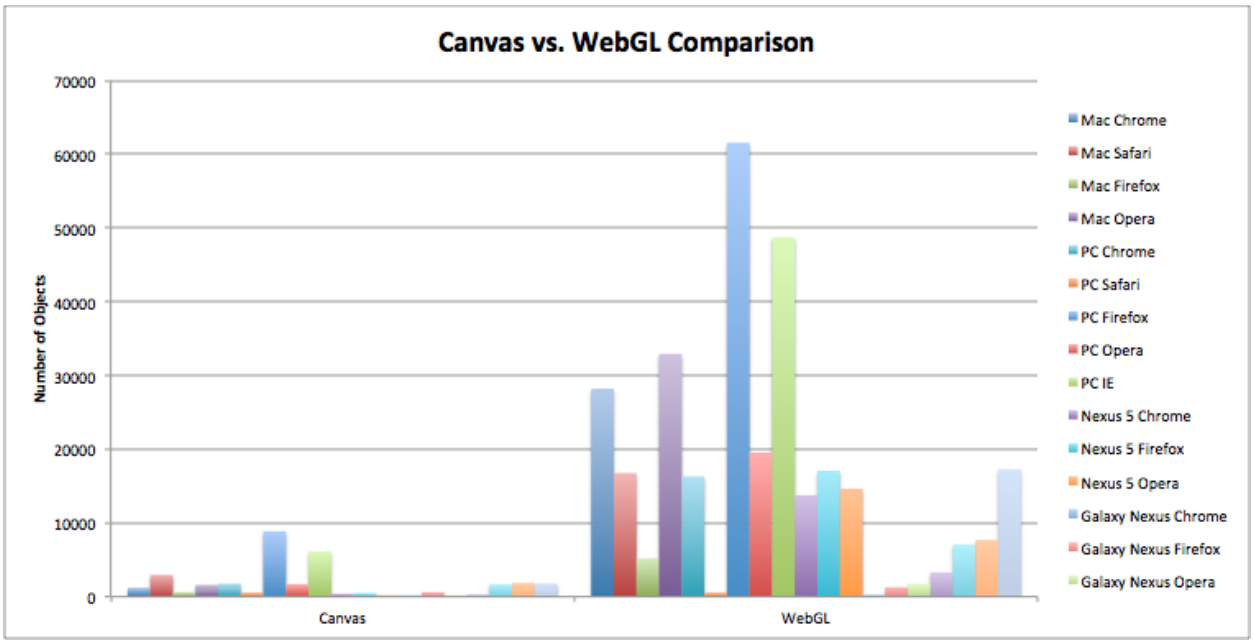


Figure 27: Comparison of the Scirra Benchmark using Canvas vs. WebGL with browsers as series.

There is a clear distinction of difference between the two technologies where WebGL had clearly outperformed Canvas in all browsers that were able to render WebGL. Comparing the performance of desktop and mobile browser performance that there is a clear advantage of performance on desktop browsers. However, both the Android Nexus 5 and iPhone 6 produced some decent results with Safari on the iPhone 6 and Firefox on the Nexus 5 coming on top.

6.3.4 Testing the Z-Buffer

The Z-buffer is now a key component in the graphics render pipeline when rendering 3D graphics. The Z-buffer is a buffer that will decide which 3D objects will render or not. As 3D objects that are in front of each other and whichever object is closest to the camera will be rendered and the object behind will not. This saves on computing power, as when the Z-buffer is enabled it will only render the objects that are visible within the view of the fulcrum.

This benchmark tests with both the Z-buffer enabled and disabled. The number of objects is increased and when enabled the hardware will start the process of culling the objects that do not need to be rendered when the objects are overlapped by newer objects. When the z-buffer is disabled all objects are rendered whether they are in view or not. Realistically, disabling the z-buffer is not recommended but for the purposes of benchmarking this may be useful to push the GPUs limit in rendering.

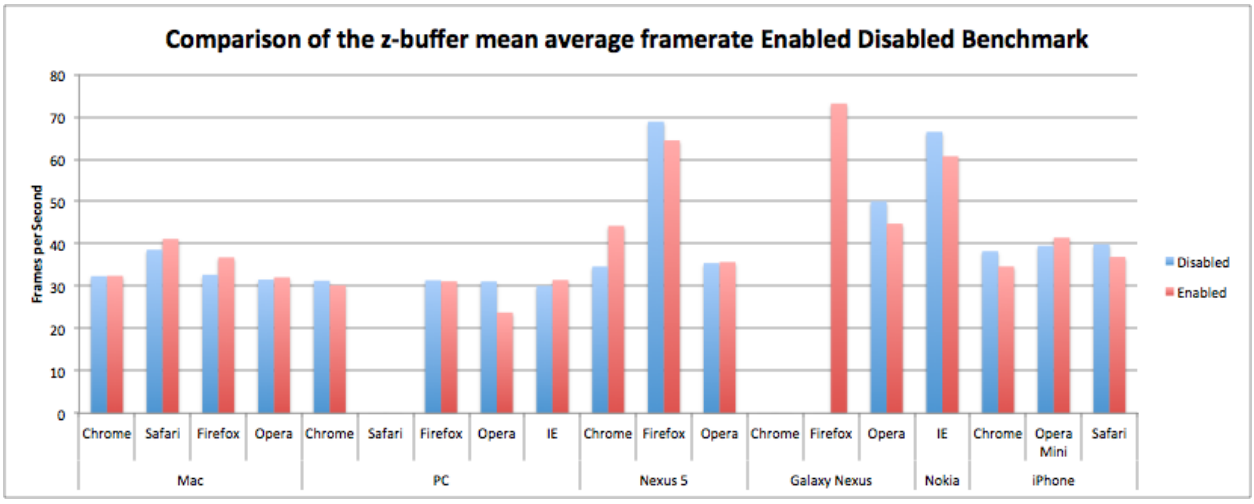


Figure 28: Frame rates for Z-buffer enabled and disabled results for all devices and browsers.

Figure 28 illustrates the differences between desktop and smartphones and compares each of the browser's performance. Both the disabled and enabled results are presented. Chrome on the Galaxy Nexus and Safari on the PC failed to produce a result. Firefox on the Nexus 5 outperformed both Mac and PC in terms of performance when the z-buffer was enabled but did not produce a result when disabled. Internet Explorer 11 on the Nokia Lumia 520 performed extremely well considering it is a budget smartphone. The iPhone also produced some good results when comparing to the desktop counterparts.

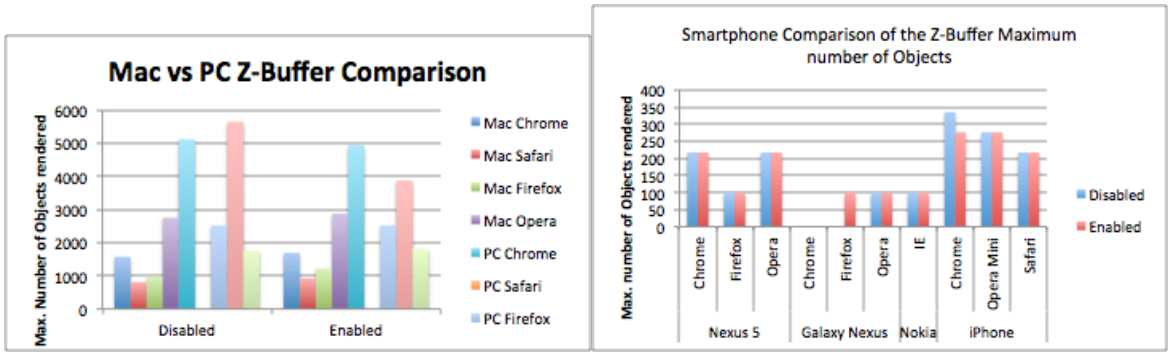


Figure 29: Frame rates and number of objects rendered for Z-buffer enabled and disabled results for all devices and browsers.

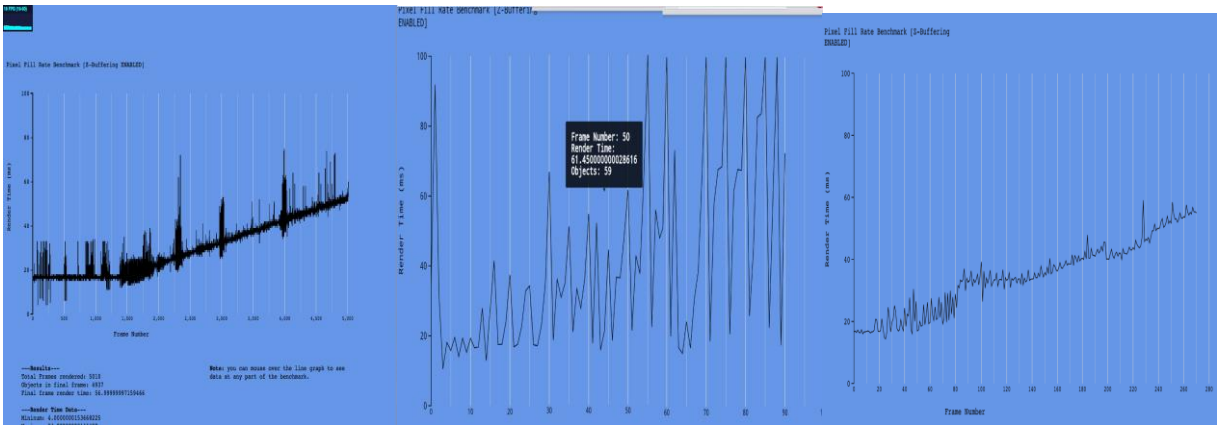


Figure 30: PC Chrome, Galaxy Nexus, iPhone 6 Z-buffer Enabled Charts

For each browser on each device a chart was compiled of how the browser performed during the test. As each object was added to the z-buffer the render time (in milliseconds) of how long it took to render that frame was recorded (the y-axis). The x-axis is the frame number where for each frame one object was added to the z-buffer. As can be seen in Figure 30 where I have selected three of the charts for comparison. Each of the charts is for Google Chrome with z-buffer enabled on PC, Galaxy Nexus and lastly the iPhone 6. Each of the devices produced different results. All the charts represent a similar pattern, whereas the frame number increases the render time increases for each frame. This is expected as by adding load to the GPU and increasing the amount of objects in the z-buffer will increase the render time. How they coped with the increase is what has differed. The PC has a gradual increase with occasional high spikes as I suspect this is due to the efficiency of the hardware graphics pipeline and the gradual increase is the compile time of the JavaScript engine. The iPhone 6 has a similar chart but with less spikes where at frames 80 to 140 it plateaued for a while. This shows efficient harmony between the software and the hardware in not increasing compile time as load is increased. On the other hand when viewing the chart (middle chart in Figure 30) for the Galaxy Nexus there seems to be a yo-yo effect as the render time goes up and down. This could be due to a couple of factors, first where device is overheating causing a spike in the render time and by the time the GPU has dissipated the heat it resumes back to efficient rendering speeds. The other factor may be due to a bottleneck as each object is added to the z-buffer, this causes an additional render time and by the time the garbage collection and culling has been performed it then resumes back to normal rendering speeds.

6.3.5 WebGL Boxes Benchmark

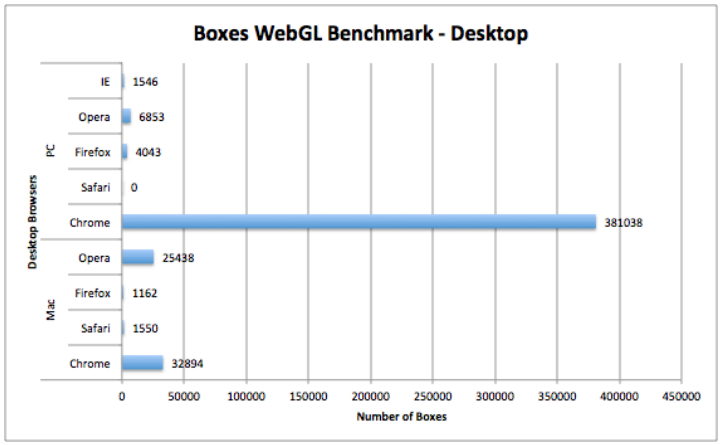


Figure 31: WebGL Performance Benchmark Boxes for desktop devices

In this benchmark suite as provided by Charles Lui and Bryon Kropf [77]. This has a series of spinning cubes that are added at regular intervals to the scene. The benchmark keeps on adding the cubes until the frame rate has stabilised between 45 and 50 frames per second. Chrome on PC handled this benchmark extremely well; to get the result I have had to leave the benchmark running overnight as it was coping with a six-digit figure of cubes where it had started to stabilise at about 381038. This may be due to the gaming spec graphics card and how that Chrome translates the WebGL directly to DirectX on windows platform through their Almost Native Graphics Layer Engine (ANGLE) Project [92] this is implemented in both the Chrome engine and Firefox engine on Windows platform only.

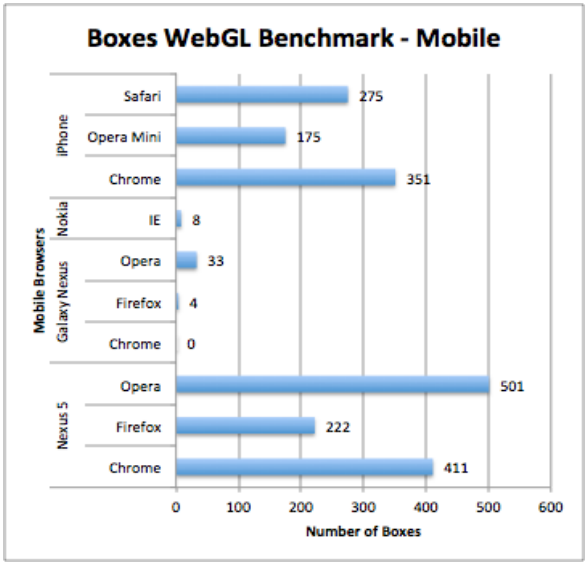


Figure 32: WebGL Performance Benchmark Boxes for mobile devices

To have a better understanding of the performance on smartphones through this benchmark, figure 32 represents the differences in performance between the browsers on all smartphones. The Nexus 5 had come out top with Opera producing the best result. The iPhone 6 came second and both the Galaxy Nexus and the Nokia Lumia 520 produced sub-par results. On both the iPhone and the Nexus 5, Chrome had outperformed the other browsers on the same platform.

The analysis from this is that this benchmark utilises the TDL [93] a low-level WebGL graphics library and some libraries are optimised and tested on smartphones whilst others are not. These benchmarks were not designed to be run on smartphones and hence why they produce different results than other frameworks. In comparison to the z-buffer test where the results that were obtained from the smartphones was comparable to the desktop.

6.3.6 Pixi.js and three.js Results

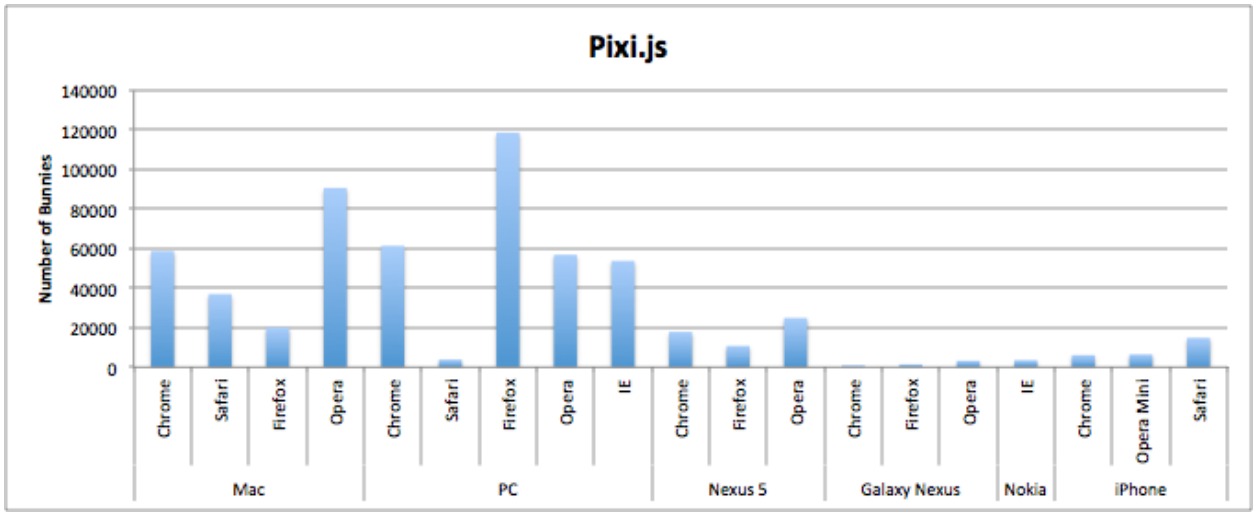


Figure 33: Performance comparison of the WebGL Pixi.js library

Figure 33 represents how many 2D sprites of a bunny have been drawn on the screen to reach a frame rate of 30 frames per second. The Pixi.js [26] framework will utilise WebGL if the browser supports it and if it does not support WebGL then it will fall back to using Canvas, similar to the Scirra framework discussed earlier. Hence why Safari on PC produced a result but not a very high score as the framework had reverted to Canvas to render the scene. Firefox on PC had outperformed all other browsers with Opera on Mac coming second. In most other tests Chrome seems to outperform other browsers but in this test it is difficult to determine who is the clear

winner. The results that were obtained from the Galaxy Nexus were too low to consider a success and the iPhone and Nexus 5 were similar in performance with Nexus 5 producing a slightly better outcome. Comparing the results from the smartphones to the desktop, it is clear that smartphone performance does not match desktop performance in this framework.

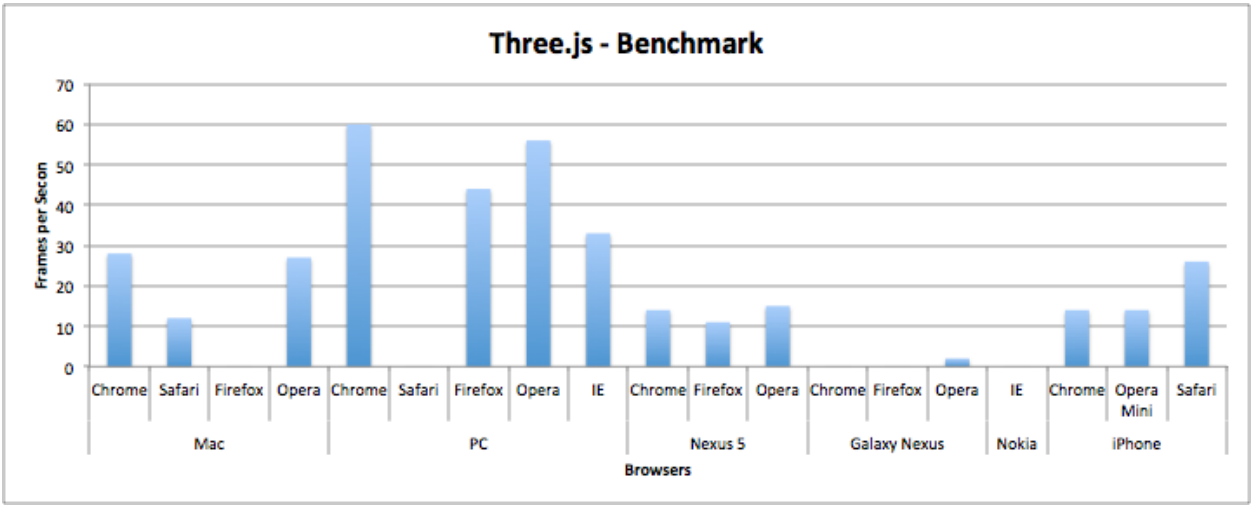
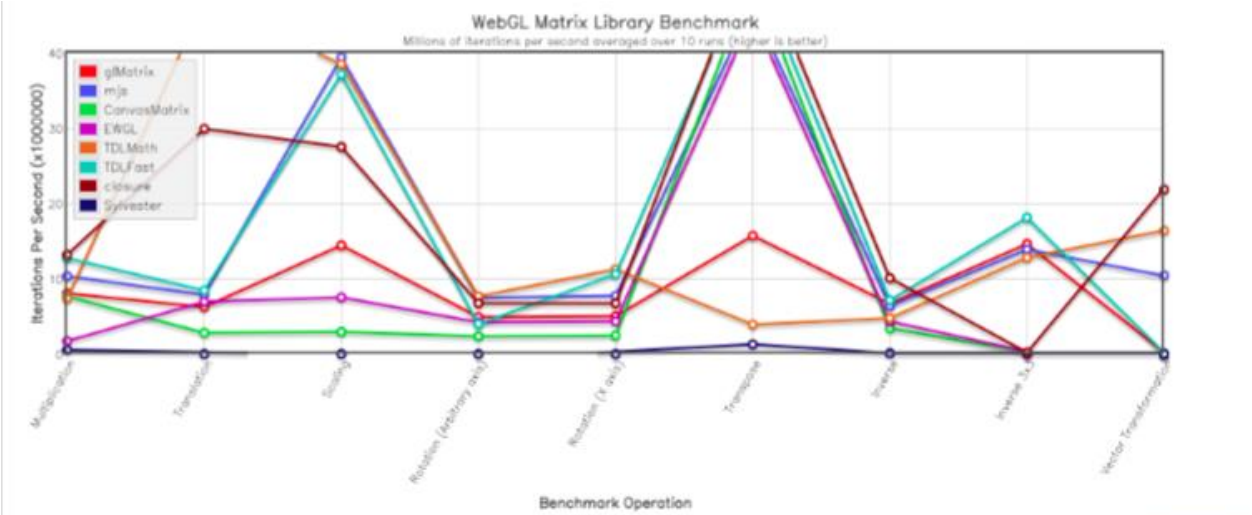


Figure 34: Performance comparison of the WebGL three.js library

The three.js library produced by Robert Cabello [82] is the most popular framework of WebGL on the web [94] and is widely used in commercial WebGL web development. The three.js benchmark is a complex detailed benchmark that utilises most of the features of the framework such as Phong shading, reflection and specular lighting. The three.js library does offer raytrace rendering but this is not suitable for real-time render display.

Surprisingly, Firefox on Mac did not render a result. The reason for this is unknown but this may be due to a software bug in that particular version release of Firefox 33.0.1 on Mac as this test was performed recently on a newer updated release of Firefox version 33.1.1 where it produced a result of 20 FPS. This benchmark did not produce a result on the Nokia Lumia 520 and two out of the three browsers on the Galaxy Nexus. On both the Nexus 5 and the iPhone it had rendered the scene successfully but with a low frame rate between 10 and 20 fps with one exception Safari on the iPhone where this had produced a result of 26 frames per second. During the time when the test was performed with the respective browser software releases, Safari on the iPhone 6 had outperformed Safari on the Mac platform.

6.3.7 WebGL Matrix Library Benchmark



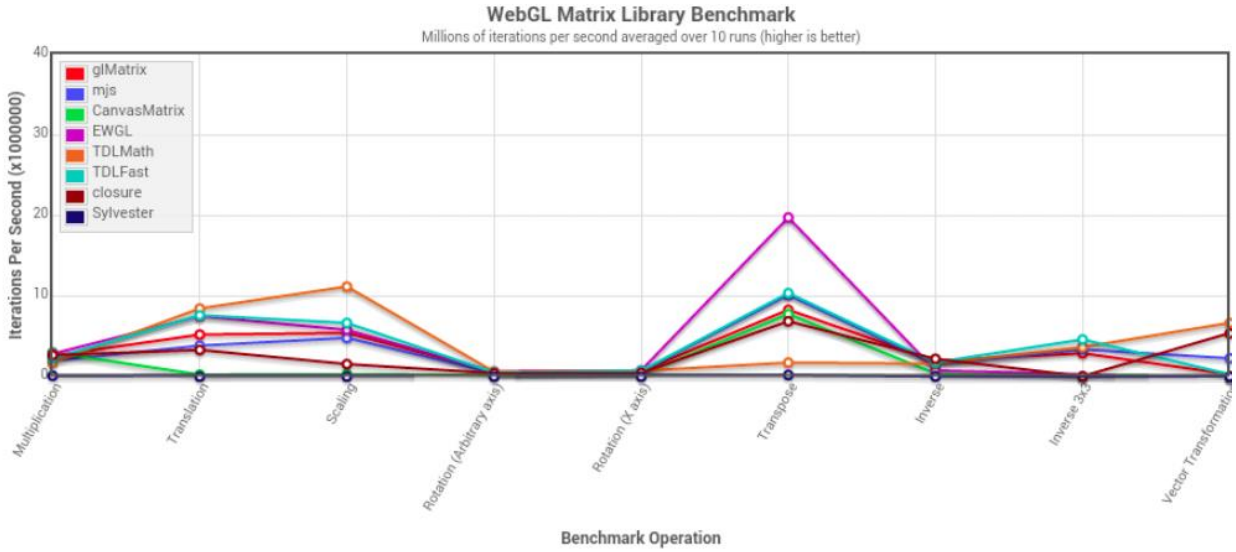
Library	Multiplication	Translation	Scaling	Rotation (Arbitrary axis)	Rotation (X axis)	Transpose	Inverse	Inverse 3x3	Vector Transformation	Average
closure	13.18	29.93	27.54	6.74	6.72	54.26	10.09	18.14	21.85	20.94
TDLFast	12.76	8.45	37.17	3.98	10.62	52.16	7.14			18.89
mjs	10.35	7.87	39.43	7.49	7.74	46.27	6.40 (bad)	13.89	10.43	16.65
TDLMath	7.39	46.90	38.53	7.67	11.27	3.91	4.76	12.81	16.40	16.63
EWGL	1.71	6.95	7.53	4.22	4.34	45.07	4.32			10.59
CanvasMatrix	7.67	2.81	2.95	2.36	2.45	50.23	3.41			10.27
glMatrix	8.05	6.24	14.45	4.91	5.02	15.73	6.67	14.63		9.46
Sylvester	0.53					1.28	0.10	0.18		0.53

Figure 35

Figure 35: WebGL Matrix Library Benchmark on iPhone Safari

This benchmark is a mathematical calculation test using different JavaScript libraries, some of the libraries relate to WebGL whilst others are common JavaScript libraries that can or are used for matrix calculations. This benchmark compares their performances by applying different functions. The same function is used on each library then compared. An average score is provided as an index measure for all of the calculations for one given library. The graph displays this as Iterations per Second (the higher the better) and as can be seen on the chart that the results differ greatly between one library and another for the same mathematical calculation. Figure 35 is a snapshot of the benchmark on the iPhone 6 using Safari. Sylvester, glMatrix and CanvasMatrix performed the worst out of the rest where closure provided the best performance. Closure is Google's generic library for common JavaScript functions that are widely used within their own Google Apps, such as Gmail and Google Calendar. This library is not a WebGL library

as such but is a JavaScript library that can perform matrix calculations. In comparison to the same calculations as glMatrix, glMatrix underperforms and glMatrix is a library designed primarily for the use of WebGL calculations.



Library	Multiplication	Translation	Scaling	Rotation (Arbitrary axis)	Rotation (X axis)	Transpose	Inverse	Inverse 3x3	Vector Transformation	Average
EWGL	2.83	7.51	5.79	0.61	0.77	19.73	0.76			5.43
TDLFast	2.25	7.61	6.66	0.52	0.73	10.37	1.76			4.27
TDLMath	1.73	8.44	11.17	0.60	0.71	1.74	1.60	3.65	6.64	4.03
glMatrix	2.57	5.22	5.42	0.62	0.69	8.29	1.96	2.89		3.46
mjs	1.96	3.84	4.81	0.47	0.66	10.14	1.50 (bad)	3.38	2.28	3.23
closure	2.67	3.31	1.56	0.41	0.46	6.86	2.21	4.61	5.36	3.05
CanvasMatrix	3.01	0.24	0.30	0.20	0.23	7.76	0.38			1.73
Sylvester	0.12					0.20	0.03	0.04		0.10

Figure 36: WebGL Matrix Library Benchmark on Chrome Nexus 5

When the same test was performed on the Nexus 5, the results were not as varied as the results from the iPhone. The calculations did not produce as high results as the iPhone. In comparison the worst 3 performing libraries are Sylvester, CanvasMatrix and closure whilst on the iPhone, closure had been the best performing library.

Chapter 7: Conclusion

This study has been an exploratory study in the performance of WebGL on smartphones and from the analysis of the results determining whether WebGL as a technology is suitable for use on modern day smartphones. The study focused on a collection of commonly used WebGL frameworks and libraries and produced a benchmark suite and reporting tools to test the performance of WebGL through these libraries and benchmarks. The study also compared smartphone WebGL performance to WebGL desktop performance and also compared WebGL to HTML5 canvas rendering.

The results from the benchmarks produced some expected and not so expected results but the conclusion from these results is that to effectively write WebGL for smartphones it is important to take extra consideration in choosing the right library for the appropriate browser and platform. Making sure the library used is one that has optimisations and tweaks specifically for smartphones. Due to WebGL only just surfacing on smartphones, this has not been the case on most WebGL frameworks as can be seen on the results. What can be said about the devices tested that in most results produced, both the LG Nexus 5 and the Apple iPhone 6 produced good solid results in performance with WebGL. Both those devices are flagship devices for the two dominating smartphone platforms Android and iOS.

The study also showed that smartphone devices are efficient at memory management and focusing the resources to where they are needed. This can be seen when the tests were repeated whilst having other apps running in the background and other tabs in the browser. Both, the Android and Apple iOS platform seemed unparalleled by having additional applications and tabs in the background. Smartphone operating systems have developed a system of having apps in different states depending on whether the app is used or not giving the app in focus priority in terms of processing.

To answer the question of whether WebGL is ready for smartphones and whether it is suitable for smartphones. From the results it can be seen that both the Nexus 5 and the iPhone 6 can handle WebGL at decent speeds. It is important that the WebGL that is developed is designed with mobile in mind as when developing modern websites. Most commercial sites that utilise WebGL

have only focused on developing and testing on desktop machines but this is only due to the fact that WebGL had not been around on smartphones till very recent.

Further research is needed in this field of study and further testing on a larger range of devices. The development of ‘real world’ benchmarks instead of synthetic benchmarks to test the effectiveness of WebGL on smartphones. Hopefully to see future benchmarks building and developing WebGL benchmarks that focus on testing the performance for smartphones.

WebGL is in its infancy. However, as the technology has arrived on iOS and Android on a mainstream scale, more and more developers will be developing on WebGL now that major browsers fully support WebGL on these devices. Initially, it had been an issue of having an acceptable level of performance on these devices as smartphones had limited capabilities in terms of rendering 3D graphics. There have been major advancements in hardware that can be demonstrated with such products as the iPhone 6. I predict future mobile phones arriving with similar graphics rendering capabilities or higher and expect to see increasingly more WebGL content on the web designed specifically for smartphones.

References

[1] J. D. Owens, D. Luebke, and N. Govindaraju, “A Survey of general-purpose computation on graphics hardware,” *Comput. Graph. ,* 2007.

[2] Khronos Group, “Khronos Releases Final WebGL 1.0 Specification - Khronos Group Press Release,” *Khronos Group*, 03-Mar-2011. [Online]. Available: https://www.khronos.org/news/press/khronos-releases-final-webgl-1.0-specification. [Accessed: 27-Oct-2014].

[3] Yahoo Inc., “Yahoo Web Page in 1995,” *CrazyEngineers*. [Online]. Available: http://www.crazyengineers.com/wp-content/uploads/2012/08/1995_469x458.gif. [Accessed: 23-Nov-2014].

[4] B. Segal, “A Short History of the Internet,” *CERN*, Apr-1995. [Online]. Available: http://ben.web.cern.ch/ben/TCPHIST.html. [Accessed: 23-Nov-2014].

[5] World Wide Web Consortium, “HTML,” *World Wide Web Consortium*. [Online]. Available: http://www.w3.org/History/19921103-hypertext/hypertext/WWW/MarkUp/MarkUp.html#4. [Accessed: 23-Nov-2014].

[6] World Wide Web Consortium, “HTML 4 Conformance,” *World Wide Web Consortium*. [Online]. Available: http://www.w3.org/TR/html401/conform.html#deprecated. [Accessed: 23-Nov-2014].

[7] “Brookfield Zoo Web Page 2009,” *Kentico*, 2009. [Online]. Available: http://www.kentico.com/i/SOY/2009/Nonprofit/brookfieldzoo.jpg. [Accessed: 23-Nov-2014].

[8] T. Berners-Lee and D. Connolly, “RFC 1866 - HyperText Markup Language 2.0,” *Internet Engineering Task Force*, Nov-1995. [Online]. Available: http://tools.ietf.org/html/rfc1866. [Accessed: 07-Nov-2014].

[9] TouchPhoto, “TouchPhoto 3D Cube in Flash,” *MotoCMS*. [Online]. Available: http://www.motocms.com/wp-content/uploads/2010/12/touchphoto.jpg. [Accessed: 23-Nov-2014].

[10] Millward Brown, “PC Penetration | Statistics | Adobe Flash Platform runtimes,” *Adobe Flash Platform Runtimes*, Jul-2011. [Online]. Available: http://www.adobe.com/mena_en/products/flashplatformruntimes/statistics.displayTab4.html. [Accessed: 07-Nov-2014].

[11] Adobe Systems Incorporated., “3D game development for Flash and video games,” *Adobe Developer Connection*, 01-Dec-2012. [Online]. Available: http://www.adobe.com/devnet/flashplayer/stage3d.html. [Accessed: 07-Nov-2014].

[12] B. Lawson and R. Sharp, *Introducing HTML5*. Pearson Education, 2011.

[13] Z. Johnson, “3D Canvas Galaxy,” *Zachstronaut*. [Online]. Available: http://www.zachstronaut.com/posts/images/3d-canvas-galaxy.png. [Accessed: 23-Nov-2014].

[14] Google Inc., “WebGL Eagle Image,” *Chrome Experiments*. [Online]. Available: http://www.chromeexperiments.com/img/webgl/webgl-header-eagle-img.jpg. [Accessed: 23-Nov-2014].

[15] The Khronos Group Inc., “WebGL 2 Specification,” *The Khronos Group Inc.* [Online]. Available: https://www.khronos.org/registry/webgl/specs/latest/2.0/. [Accessed: 26-Aug-2014].

[16] J. Gray, “Google Chrome: The Making of a Cross-platform Browser,” *Linux J.*, vol. 2009, no. 185, Sep. 2009.

[17] B. Danchilla, *Beginning WebGL for HTML5:* Apress, 2012.

[18] M. Segal, K. Akeley, C. Frazier, and J. Leech, “The OpenGL Graphics System: A Specification,” research.microsoft.com, 2004.

[19] D. Blythe, “OpenGL ES Common/Common-Lite Profile Specification,” 2004.

[20] C. Marrin, “WebGL Specification 1.0.2,” *Khronos*, 01-Mar-2013. [Online]. Available:

https://www.khronos.org/registry/webgl/specs/1.0.2/. [Accessed: 29-Oct-2014].
[21] Zygote Media Group Inc., “Zygote Body,” *Zygote Body*, 2012. [Online]. Available: bodybrowser.googlelabs.com. [Accessed: 27-Oct-2014].
[22] Adobe Inc., “Adobe - Flash Player,” *Adobe*, 2014. [Online]. Available: http://www.adobe.com/uk/software/flash/about/. [Accessed: 27-Oct-2014].
[23] Unity Technologies, “Unity - Game Engine,” *Unity*, 2014. [Online]. Available: http://unity3d.com/. [Accessed: 27-Oct-2014].
[24] Scirra Ltd., “Create Games with Construct 2 - Scirra.com,” *Scirra*, 2014. [Online]. Available: https://www.scirra.com/construct2. [Accessed: 27-Oct-2014].
[25] PlayCanvas, “PlayCanvas,” *PlayCanvas | 3D HTML5 & WebGL Game Engine*, 2014. [Online]. Available: https://playcanvas.com/. [Accessed: 27-Oct-2014].
[26] Goodboy Digital Ltd., “Pixi.js - 2D webGL renderer with canvas fallback,” *PixiJS*, 19-May-2013. [Online]. Available: http://www.pixijs.com/. [Accessed: 27-Oct-2014].
[27] M. Nobel-Jørgensen, “WebGL based 3D Game Engine,” master’s thesis, IT University of Copenhagen, 2012.
[28] D. Catuhe, D. Rousset, P. Lagarde, and M. Rousseau, “Babylon.js,” *Babylon.js*, 2014. [Online]. Available: http://www.babylonjs.com/. [Accessed: 27-Oct-2014].
[29] Google, “Step inside the map with Google MapsGL,” *Official Google Blog*, 12-Oct-2011. [Online]. Available: http://googleblog.blogspot.com/2011/10/step-inside-map-with-google-mapsgl.html. [Accessed: 28-Oct-2014].
[30] M. M. Mobeen, L. Feng, and S. Mieee, “Ubiquitous Medical Volume Rendering on Mobile Devices,” in *International Conference on Information Society (i-Society 2012)*, 2012, pp. 93–98.
[31] S. Adam, “Native WebGL: A giant leap for the web development community on the BlackBerry PlayBook,” *Inside BlackBerry - Developer Blog*, 22-Feb-2012. [Online]. Available: http://devblog.blackberry.com/2012/02/playbook-native-webgl-development/. [Accessed: 27-Oct-2014].
[32] B. Jones, “PSA: The latest Chrome Beta for Android has the chrome://flags/ page enabled,…,” *Google Chrome - Brandon Jones*, 25-Jan-2013. [Online]. Available: https://plus.google.com/+BrandonJonesToji/posts/LsHQCDk557P. [Accessed: 27-Oct-2014].
[33] Apple Inc., “Creating 3D Interactive Content with WebGL,” presented at the WWDC - Apple Developer, 04-Jun-2014.
[34] Apple Inc., “Apple Announces Record Pre-orders for iPhone 6 & iPhone 6 Plus Top Four Million in First 24 Hours,” *Apple (United Kingdom) - Apple Press Info*, 15-Sep-2014. [Online]. Available: https://www.apple.com/uk/pr/library/2014/09/15Apple-Announces-Record-Pre-orders-for-iPhone-6-iPhone-6-Plus-Top-Four-Million-in-First-24-Hours.html. [Accessed: 27-Oct-2014].
[35] J. Issa and S. Figueira, “Graphics Processor performance analysis for 3D applications,” in *Advances in Computational Tools for Engineering Applications (ACTEA), 2012 2nd International Conference on*, 2012, pp. 269–272.
[36] A. Benin, G. R. Leone, and P. Cosi, “A 3D Talking Head for Mobile Devices Based on Unofficial iOS WebGL Support,” in *Web3D 2012*, 2012, pp. 117–120.
[37] R. Taylor and X. Li, “A Micro-benchmark Suite for AMD GPUs,” in *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, 2010, pp. 387–396.
[38] S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, and W.-M. W. Hwu, “Optimization Principles and Application Performance Evaluation of a Multithreaded GPU Using CUDA,” in *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2008, pp. 73–82.
[39] J. Aycock, “A Brief History of Just-in-time,” *ACM Comput. Surv.*, vol. 35, no. 2, pp. 97–113, Jun. 2003.
[40] P. Ratanaworabhan, B. Livshits, and B. G. Zorn, “JSMeter: Comparing the behavior of JavaScript benchmarks with real web applications,” *conference on Web application …*, 2010.
[41] S. Niyogi, “The New JavaScript Engine in Internet Explorer 9,” *MSDN Blogs*, 18-Mar-2010.

[Online]. Available: http://blogs.msdn.com/b/ie/archive/2010/03/18/the-new-javascript-engine-in-internet-explorer-9.aspx. [Accessed: 28-Oct-2014].
[42] N. Leenheer, "Desktop Browser Scores," *HTML5test - How well does your browser support HTML5?*, 2014. [Online]. Available: http://html5test.com/results/desktop.html#scoreToggle. [Accessed: 28-Oct-2014].
[43] A. Kingsley-Hughes, "The BIG browser benchmark (January 2013 edition) | ZDNet," *ZDNet*, 14-Jan-2013. [Online]. Available: http://www.zdnet.com/the-big-browser-benchmark-january-2013-edition-7000009776/. [Accessed: 28-Oct-2014].
[44] J. L. Jacobi, "Browser comparison: How the five leaders stack up in speed, ease of use and more," *PCWorld*, 17-Sep-2014. [Online]. Available: http://www.pcworld.com/article/2605933/browser-comparison-how-the-five-leaders-stack-up-in-speed-ease-of-use-and-more.html. [Accessed: 29-Oct-2014].
[45] StatCounter, "Top 9 Desktop, Tablet, Mobile & Console Browsers from Sept 2013 to Sept 2014 | StatCounter Global Stats," *StatCounter Global Stats*, 2014. [Online]. Available: http://gs.statcounter.com/#all-browser-ww-monthly-201309-201409-bar. [Accessed: 26-Oct-2014].
[46] "The WebKit Open Source Project," *The WebKit Open Source Project*. [Online]. Available: http://www.webkit.org/. [Accessed: 28-Oct-2014].
[47] B. Lawson, "300 Million Users and Move to WebKit," *Dev.Opera*, 12-Feb-2013. [Online]. Available: https://dev.opera.com/blog/300-million-users-and-move-to-webkit/. [Accessed: 29-Oct-2014].
[48] "Chromium," *Chromium - The Chromium Projects*. [Online]. Available: http://www.chromium.org/Home. [Accessed: 28-Oct-2014].
[49] Mozilla Developer Network, "SpiderMonkey," *Mozilla Developer Network*, 23-Oct-2014. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey. [Accessed: 29-Oct-2014].
[50] A. Deveria and L. Schoors, "Can I use WebGL - 3D Canvas graphics," *Can I use*, 24-Sep-2014. [Online]. Available: http://caniuse.com/#feat=webgl. [Accessed: 29-Oct-2014].
[51] The Khronos Group Inc., "WebGL Conformance Test Runner Version 1.0.2," *Khronos*, 2012. [Online]. Available: https://www.khronos.org/registry/webgl/conformance-suites/1.0.2/webgl-conformance-tests.html. [Accessed: 29-Oct-2014].
[52] R. C. Hoetzlein, "Graphics performance in rich Internet applications," *IEEE Comput. Graph. Appl.*, vol. 32, no. 5, pp. 98–104, Sep. 2012.
[53] R. Yogya and R. Kosala, "Comparison of Physics Frameworks for WebGL-Based Game Engine," *EPJ Web of Conferences*, vol. 68, p. 00035, Mar. 2014.
[54] D. Catuhe, "Testing the Limits of WebGL: The Babylon.js Train Demo," *SitePoint*, 21-Nov-2013. [Online]. Available: http://www.sitepoint.com/testing-limits-webgl-babylon-js-train-demo/. [Accessed: 29-Oct-2014].
[55] A. Overa, "Performance Benchmarks: HTML5 Hardware Acceleration And WebGL - Web Browser Grand Prix 4: Firefox 4 Goes Final," *Tom's Hardware*, 04-Apr-2011. [Online]. Available: http://www.tomshardware.com/reviews/firefox-4-internet-explorer-9-chrome-10,2909-10.html. [Accessed: 29-Oct-2014].
[56] Unity Technologies, "Unity - Web Player Download," *Unity*, 2014. [Online]. Available: http://unity3d.com/webplayer. [Accessed: 30-Oct-2014].
[57] Unity Technologies, "Unity - Manual: Getting Started with Flash Development," *Unity Documentation*, 2014. [Online]. Available: http://docs.unity3d.com/Manual/flash-gettingstarted.html. [Accessed: 30-Oct-2014].
[58] J. Echterhoff, V. Balasevicius, K. Paskova, and B. Bibby, "Benchmarking Unity performance in WebGL," *Unity Technologies Blog*, 07-Oct-2014. [Online]. Available: http://blogs.unity3d.com/2014/10/07/benchmarking-unity-performance-in-webgl/. [Accessed: 30-Oct-2014].
[59] I. Elliot, "Support For Asm.js Growing?," *I Programmer - News*, 24-Sep-2014. [Online]. Available:

http://www.i-programmer.info/news/167-javascript/7786-support-for-asmjs-growing.html. [Accessed: 30-Oct-2014].
[60] A. Feldthaus, T. Millstein, A. Møller, M. Schäfer, and F. Tip, "Refactoring Towards the Good Parts of Javascript," in *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, 2011, pp. 189–190.
[61] D. Crockford, *JavaScript: The Good Parts: The Good Parts*. O'Reilly Media, 2008.
[62] Mozilla Developer Network, "WebGL best practices," *Mozilla Developer Network*, 23-Mar-2014. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/WebGL/WebGL_best_practices. [Accessed: 30-Oct-2014].
[63] Khronos Group, "WebGL now runs on Apple iOS 8 - khronos.org news," *Khronos Group - News*, 19-Sep-2014. [Online]. Available: https://www.khronos.org/news/permalink/webgl-now-runs-on-apple-ios-8. [Accessed: 31-Oct-2014].
[64] A. Gullen, "Boosting mobile HTML5 game performance with WebGL - Scirra.com," *Scirra Blog*, 26-Feb-2013. [Online]. Available: https://www.scirra.com/blog/107/boosting-mobile-html5-game-performance-with-webgl. [Accessed: 31-Oct-2014].
[65] Khronos Group, "Get WebGL," *Get WebGL*. [Online]. Available: http://get.webgl.org/. [Accessed: 01-Nov-2014].
[66] A. Salga, "Does My Browser Support WebGL?," *Does My Browser Support WebGL?* [Online]. Available: http://doesmybrowsersupportwebgl.com/. [Accessed: 01-Nov-2014].
[67] Analytical Graphics Inc., "WebGL Report," *WebGL Report*, 2014. [Online]. Available: http://webglreport.com/. [Accessed: 01-Nov-2014].
[68] Scirra, "ScirraMark browser score for HTML5 gaming," *Scirra*, 25-Feb-2013. [Online]. Available: http://www.scirra.com/labs/scirramark/. [Accessed: 01-Nov-2014].
[69] Autodesk, E. Haines, and G. Dekena, "Interactive 3D Graphics Course With Three.js & WebGL - Udacity." .
[70] T. Parisi, *WebGL: Up and Running*. O'Reilly Media, Incorporated, 2012.
[71] C. Bateman, "WebGL Workshop London," *Meetup*. [Online]. Available: http://www.meetup.com/WebGL-Workshop-London/. [Accessed: 07-Nov-2014].
[72] "Big Buck Bunny » Download." [Online]. Available: https://peach.blender.org/download/. [Accessed: 19-Oct-2014].
[73] "Lagomi Test Page." [Online]. Available: http://www.lagom.nl/lcd-test/all_tests.php#colorbands.png. [Accessed: 19-Oct-2014].
[74] D. Catuhe, P. Lagard, M. Rosseau, C. Joly, R. Rouhier, Enozone, and Progis, "Babylon.js - Train Demo," *Babylon.js*, 28-Oct-2013. [Online]. Available: http://www.babylonjs.com/index.html?TRAIN. [Accessed: 01-Nov-2014].
[75] Google Inc., "WebGL Aquarium Benchmark," *WebGL Samples*, 2009. [Online]. Available: http://webglsamples.org/aquarium/aquarium.html. [Accessed: 23-Nov-2014].
[76] tulr...@google.com, "webgl-bench -- WebGL performance info," *webgl-bench*, 29-Jun-2010. [Online]. Available: https://webgl-bench.appspot.com/. [Accessed: 23-Nov-2014].
[77] C. Lui and B. Kropf, "WebGL Performance Benchmark," *Github - Luic*, 25-Oct-2012. [Online]. Available: http://luic.github.io/WebGL-Performance-Benchmark/. [Accessed: 23-Nov-2014].
[78] Scirra, "Scirra Canvas 2D Benchmark," *Scirra*. [Online]. Available: http://www.scirra.com/demos/c2/renderperf2d/. [Accessed: 23-Nov-2014].
[79] Scirra, "Scirra WebGL Benchmark," *Scirra*. [Online]. Available: http://www.scirra.com/demos/c2/renderperfgl/. [Accessed: 23-Nov-2014].
[80] Good Boy Digital, "pixi.js bunnymark," *Pixi.js*. [Online]. Available: http://www.goodboydigital.com/pixijs/bunnymark/. [Accessed: 23-Nov-2014].
[81] S. Bannasch, "Matrix Benchmark," *Stepheneb - Github IO*, 14-Feb-2011. [Online]. Available: http://stepheneb.github.io/webgl-matrix-benchmarks/matrix_benchmark.html. [Accessed: 23-Nov-2014].
[82] R. Cabello, "three.js webgl - performance," *three.js*, 16-Feb-2012. [Online]. Available:

http://threejs.org/examples/webgl_performance_doublesided.html. [Accessed: 23-Nov-2014].
[83] S. Ludwig, “Mozilla CEO: ‘We refuse’ to bring Firefox to iOS until Apple lets us use our web engine,” 15-Apr-2013.
[84] “Desktop PC Image.” [Online]. Available: http://d48g9fi2crz4.cloudfront.net/wp-content/uploads/2014/04/cyberpower-gamer-9000-review.jpg.
[85] EveryMac, “MacBook Pro.” [Online]. Available: http://www.everymac.com/images/cpu_pictures/apple-macbook-pro-15_09.jpg.
[86] GSMArena, “Android Samsung Galaxy Nexus Image.” [Online]. Available: http://cdn2.gsmarena.com/vv/bigpic/samsung-galaxy-nexus-new.jpg.
[87] GSMArena, “Android LG Nexus 5 Image.” [Online]. Available: http://cdn2.gsmarena.com/vv/bigpic/lg-google-nexus-5-.jpg.
[88] GSMArena, “iPhone 6 Image,” *GSMArena*. [Online]. Available: http://cdn2.gsmarena.com/vv/bigpic/apple-iphone-6-3.jpg.
[89] GSMArena, “Nokia Lumia 520 Image,” *GSMArena*. [Online]. Available: http://cdn2.gsmarena.com/vv/bigpic/nokia-lumia-520.jpg.
[90] Google Inc., “Android updates: Nexus & Google Play edition devices - Nexus Help,” *Google Nexus Help*, 2014. [Online]. Available: https://support.google.com/nexus/answer/4457705?hl=en&ref_topic=3415518&rd=1. [Accessed: 02-Nov-2014].
[91] Apple Inc., “App Programming Guide for iOS: Strategies for Handling App State Transitions,” *iOS Developer Library*, 2014. [Online]. Available: https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/StrategiesforHandlingAppStateTransitions/StrategiesforHandlingAppStateTransitions.html#//apple_ref/doc/uid/TP40007072-CH8-SW1. [Accessed: 07-Nov-2014].
[92] Google Inc., “angleproject - ANGLE: Almost Native Graphics Layer Engine - Google Project Hosting,” *Google Code*, 13-Aug-2013. [Online]. Available: https://code.google.com/p/angleproject/. [Accessed: 27-Nov-2014].
[93] greggman, “greggman/tdl,” *GitHub*, 16-May-2011. [Online]. Available: https://github.com/greggman/tdl. [Accessed: 27-Nov-2014].
[94] S. Li, “3D development with WebGL, Part 2: Code less, do more with WebGL libraries,” *IBM*, 21-Jan-2014. [Online]. Available: http://www.ibm.com/developerworks/library/wa-webgl2/. [Accessed: 27-Nov-2014].