Measuring QoS in Web-Based Virtual Worlds - an Evaluation of Unity 3D Web Builds

Hussein Bakri, Colin Allison
School of Computer Science, University of St Andrews
St Andrews, United Kingdom
{hb,ca}@st-andrews.ac.uk

ABSTRACT

Web Based Virtual Worlds (WBVW) provide users with an immersive 3D experience through their regular browser. They can be seen as prototypes for the 3D Web. This paper uses key Quality of Service metrics to compare and present measurements of two major formats for WBVW–Unity Web Player and WebGL. Significantly, in terms of realizing the 3D Web, the former requires a plug-in whereas the latter is now directly supported by major browsers. Metrics include Frames per Second, Frame Time, CPU usage, GPU usage, memory usage and Initial Download Time. The WBVW used in these experiments is Timespan Longhouse, a virtual world hosted originally in OpenSim and then transformed into Unity 3D and WebGL. The ability to transform virtual worlds built in OpenSim/Second Life to Unity 3D and then to the web has great potential to bring 3D immersive interaction capabilities to all web users but our results show that there is a significant performance difference between Web Player (plug-in needed) and WebGL (no plug-in required), in terms of all the metrics listed above. This paper explores the performance characteristics of the respective formats and proposes possible optimizations to improve their performance.

CCS Concepts

•**Information systems** → *Internet communications tools;*
•**Computing methodologies** → *Virtual reality;*

Keywords

Web-Based Virtual Worlds, Unity 3D, Unity Web Player, WebGL, OpenSim, QoS

1. INTRODUCTION

The regular 2D web is gradually becoming more suitable for hosting immersive 3D content in the form of Web-Based Virtual Worlds. This is achieved through the use of Web3D technologies such as X3D[1], O3D[2], Oak3D[3], Unity 3D and WebGL[4]. Web-Based Virtual Worlds (WBVWs) are similar to traditional Multi-User Virtual Environments (MUVEs) such as Second Life[5] and OpenSim [6] but appear to be integrated into the standard web fabric from the perspective of the user. All the user has to do is to follow a standard web link to the WBVW and interact with the immersive 3D environment from within their web browser. Significantly in terms of growing the 3D Web, some formats require a plug-in to be installed, which can dissuade potential users, whereas others consist of WebGL which is now supported as standard in major browsers, and requires no plug-in.

OpenSim is a leading MUVE platform. It started as an open source copy of Second Life which was awkward to install and manage but has steadily improved and gained much credence in the MUVE community due to its many advantages over Second Life including programmability, scalability, extensibility, configurability and manageability[7]. In hyper-grid mode, it offers a prototype of how 3D Web immersive enviroments might be connected.

Unity[8] is a leading 3D software suite for creating immersive environments for games and WBVWs. It incorporates a powerful physics engine (NVidia PhysX) and gives developers the ability to publish any Unity game or 3D world in different formats for different platforms. It has the capability to host a 3D world in a browser through its Web Player plug-in or by generating WebGL.

This paper presents an initial investigation of WBVW Quality of Service (QoS) metrics: Frames per Second (FPS), Frame Time (FT), CPU, GPU, and physical memory usage, and Initial Download Time (IDT) in WBVW projects created in Unity 3D from OpenSim texture packets. FPS, FT and IDT are all important for the user experience. Understanding the demands on the CPU, GPU and memory are important as always with real-time 3D graphics.

IDT is important as existing 2D web pages aim to load within a few seconds at most, lest a visitor loses interest and goes elsewhere. Hence the proliferation of Page Load Time monitoring and analysis tools for the 2D web as evidenced by the built-in facilities in major browsers and web sites such as www.webpagetest.org. In strong contrast, OpenSim based virtual worlds IDTs can be relatively high, taking minutes for complex or large MUVEs such as the reconstruction of the St Andrews Cathedral[9, 10], an OpenSim *mega-region*. FPS and FT are important in real time immersive environments. In an OpenSim viewer each frame should complete in approximately 18.18ms (55 frames per second). "If total frame time is greater than this then simulator performance

MMVE'16, May 12 2016, Klagenfurt, Austria

DOI: http://dx.doi.org/10.1145/2910659.2910660

Figure 1: Timespan Longhouse

will be degraded"[11]. Similarly, in Unity3D web applications, 60 frames per second is recommended.

This paper presents the results and analysis of 3 experiments on Timespan Longhouse (TL, see Fig. 1)[12], a virtual world hosted originally in OpenSim and transformed into Unity 3D. Two web builds of TL are tested: a Unity Web Player (UWP) build and a Unity WebGL (U-WebGL) build. For both UWP and U-WebGL the first experiment measures the FPS and FT in the browser. The second experiment measures the Physical Memory Used (MB), Physical Memory load (%), Total CPU Usage (%), GPU core Load (%), GPU D3D usage (%) and Total GPU Memory Usage (%). The 3rd experiment measures the IDT of these worlds during normal network conditions.

The main contributions of this paper are:

- We describe a methodology which can be used in a client to obtain metrics for the quantitative assessment of WBVW inside web browsers.
- We present quantitative empirical measurements for different QoS metrics of 2 types of WBVW, those using the UWP plugin and those using U-WebGL, highlighting performance differences.
- This work discusses what could affect the QoS of WBVWs built in Unity and proposes possible optimizations to improve their performance.

This paper is organized as follows. Section 2 describes the experimental setup, the client specification, measurement tools and methodology. Section 3 presents the results in terms of Graphics, CPU, GPU, memory and network. These results are discussed in Section 4, along with possible optimizations. Importantly, section 5 highlights some limitations of this initial investigation. Finally, Section 6 concludes and outlines future work.

2. METHODOLOGY AND TESTBED

2.1 Client machine specification

Specification of the client machine used for the 3 experiments: Intel Core i5-440- 3.10 GHz with 16GB 1067 MHz DDR3 RAM. The graphics card of the machine is NVIDIA GeForce GTX 970 with 4GB of Video RAM. The client ran on a fresh installation of Windows 7 Enterprise 64 Bit Operating System with a minimal set of background processes running to avoid interference. The worlds were generated by Unity 3D engine version 5.2.0f3.

2.2 Measurement Tools

Experiment 1: Google Chrome version 44.0.2403.125 and Fraps were used to measures the FPS and FT. Fraps[13] is a popular real-time video and game benchmarking tool.

Experiment 2: This uses HWiNFO64 and TechPowerUp GPU-Z to measure: Physical Memory Used (MB), Physical Memory load (%), Total CPU Usage (%), GPU core Load (%), GPU Memory Usage (%) and GPU D3D Memory usage (%). HWiNFO64[14] is a system information utility tool that does in-depth hardware analysis, real time monitoring and reporting. TechPowerUp GPU-Z[15] is a lightweight system utility that provides real-time information about video cards and graphics processors. Both tools presented similar measurement values. As in Experiment 1 the client was Google Chrome v. 44.0.2403.125.

Experiment 3: This uses the built-in Network Monitor tool[16] in Mozilla Firefox version 39, Network Inspector in Opera 30[17] and the Network tool in Google Chrome[18] to measure the Initial Download Time. Two Firefox add-ons were also used to further check the results: app.telemetry Page Speed Monitor version 15.2.5[19]; and Extended Statusbar 2.0.3.1-signed extension[20].

2.3 Experimental Methodology

Experiment 1: A PowerShell script manages the timing of runs, opens the selected browser, and sets the specific link of each 3D World to navigate. The script runs Fraps and logs the data into CSV files. In this scenario pseudo-random navigation moves the avatar from non-dense areas toward areas dense in 3D objects. The PowerShell script then closes the browser. Fraps is configured to capture the FPS and FT every second. Frame time numbers are recorded in the CSVs as cumulative numbers. A simple repetitive subtraction equation is used to calculate the actual time of each frame numbered sequentially.

Experiment 2: A PowerShell script regulates timings, launches measurement applications (TechPowerUp GPU-Z and HWinfo64), the browser and 3D worlds, logs measurements into txt and CSV files and then closes after a predetermined time. Measurements were taken 10 times in each mode or scenario (listed below). We found that 10 times gave an acceptable variation for each mode. The modes are:

1: Baseline mode: Measurements of all CPU/GPU and Physical Memory metrics were made in the Operating System (OS) immediately after a fresh installation. No antivirus or other applications were running, no folder was opened and an absolute minimum of services and background processes were present. Each run's duration was 2 minutes.

2: Baseline mode + the Browser: Measurements of all CPU/GPU and Physical Memory metrics were done on the OS + only a web browser opened (Chrome). Each run's duration was 2 minutes.

3: Baseline mode + Browser + a 3D world: All measurements were taken for 2 minutes (Standing with Yawing) and 3 minutes Random Walking. Values were taken every 2 secs. Compared with mode #2 this gives the actual consumption in CPU/GPU and Physical memory of the 3D world in question.

Experiment 3: Browser caches were cleaned completely from inside the browsers themselves in addition to using CCleaner[21] before every run. Measurements were taken after everything was downloaded. The tools give the initial download time + all the timings of each resource in detail. The results from the range of tools used show that the measurements are accurate and reliable.

2.4 Avatar Mobility

Two mobility models were used in experiments 1 and 2:

1. **Standing:** Avatar remains standing still with continuous yawing for 2 minutes (Yaw is the change in avatar orientation or the change in the "look at" view).

2. **Random Walking:** Avatar randomly walks for 3 minutes in different directions (from non-dense towards dense areas) and with a constant speed.

2.5 Unity web-build sizes and complexity

This section describes the sizes and complexity of the worlds measured. Unity generates the HTML file for UWP. The default page is usually very simple. UWP is divided into 3 components: the mono, the player and the plugin. The plugin is either an ActiveX Control (OCX) in Internet Explorer in Windows, or a NPAPI-style DLL for other browsers like Chrome. On Mac OS it is a .plugin[22].

The WebGL capability of Unity is still being enhanced. At time of writing, version 5.x does not support some features including runtime generation of substance textures, movie textures and runtime global illumination. In Unity 5.2, WebGL 2.0 is supported in experimental mode. U-WebGL uses the emscripten compiler[23] to cross compile the runtime code into asm.js JavaScript code. A Unity WebGL project consists of several files: an index.html that embeds the contents; several JavaScript files which contain the code of the player and deals with its different functionalities; a .mem file which contains a binary that allocates the heap memory of the player and a .data file which contains all scenes and assets data and normally constitutes the majority of the size of the 3D world[22]. Sizes of the two builds of Timespan Longhouse are 40.1 MB for UWP and 353 MB for U-WebGL. Both builds of Timespan Longhouse are the defaults without any optimizations. The U-WebGL build is considerably larger than the UWP build of the same world. Table 1 shows the complexity of the virtual world in terms

Table 1: Rendering parameters of 2 minute random walk

Triangles	Vertices	Draw Calls
1,099,168	1,467,567	1,351

of the averages of the numbers of draw calls, and triangles and vertices rendered.

3. RESULTS

3.1 Frame Rates and Frame Times

FPS is the average number of frames rendered in a second while FT is the time taken to render a frame. These are key QoS indicators of the performance of a virtual world system. The boxplots in Figure 2 summarize the FPS distributions of Timespan Longhouse virtual world for both UWP and U-WebGL when the avatar is standing with random yaw and is walking randomly from a non-dense to a dense area. It can be seen that there are wide box-and-whisker diagrams with highs of around 60 FPS which is very good performance whether standing or walking and also very low FPS whiskers reaching 0. FPS of around 60 characterizes non-dense areas in TL whereas FPSs around 0 characterizes very complex regions with complex geometry where the number of triangles and vertices is high. It is interesting that the FPS median

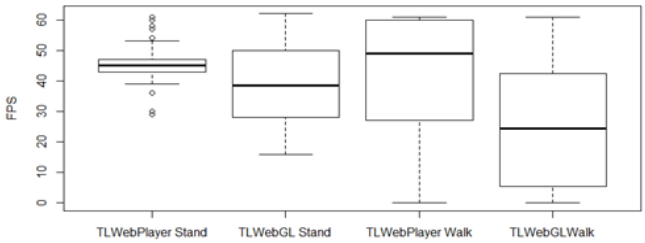

Figure 2: FPS

(50th percentile, the thick whisker line) while walking stays around 50 in the UWP version but less than 30 in the U-WebGL version. Figure 3 and Figure 4 show that the FT

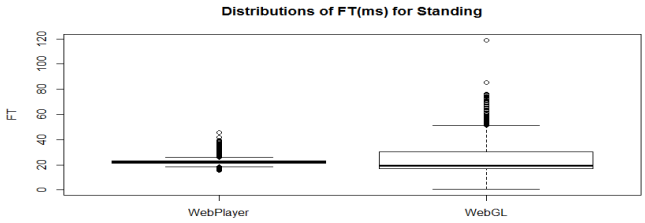

Figure 3: FT Distribution while avatar standing (ms)

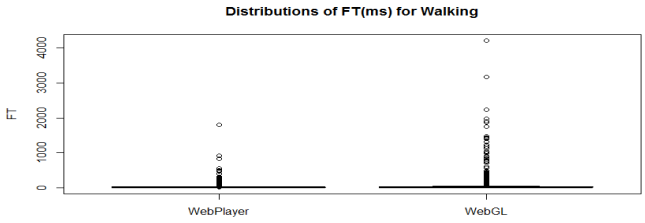

Figure 4: FT Distribution while avatar walking (ms)

has many outliers above 2000ms which correlates with the low values of FPS while walking and standing. Both versions reach FT values above 1000ms. The UWP version has a more compact box and whisker diagram and nearer to the high values of FPS than the U-WebGL version while both standing and walking. Thus, it can be seen that the UWP version outperforms U-WebGL in terms of FT and FPS.

3.2 CPU, GPU and Memory

Physical Memory Load (%) in the client machine and the Physical Memory Used in MB were measured. Both give the same information. Figures 5 and 6 show that no mat-

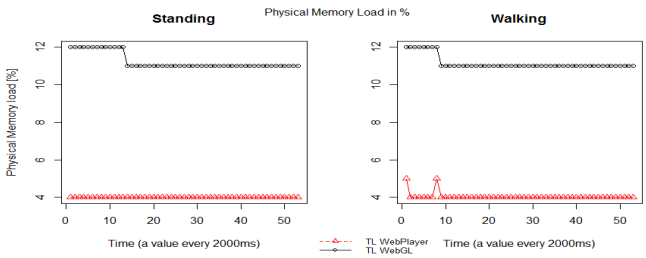

Figure 5: Physical Memory Load (%)

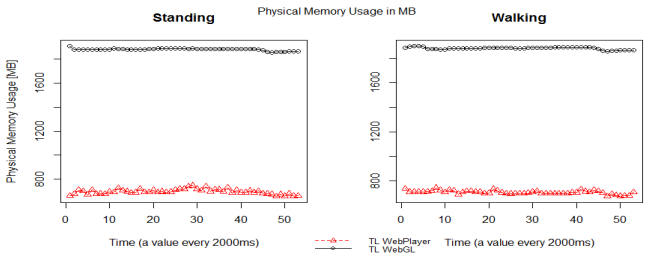

Figure 6: Physical Memory usage (%)

ter whether the avatar is walking or standing, the memory consumption does not change. However, the U-WebGL version has a higher usage (11%) compared to UWP one (4%). Figure 7 shows that U-WebGL consistently uses

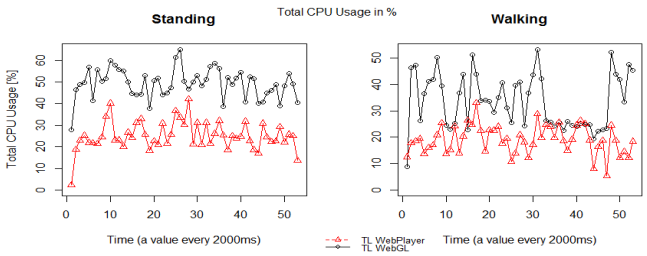

Figure 7: CPU usage (%)

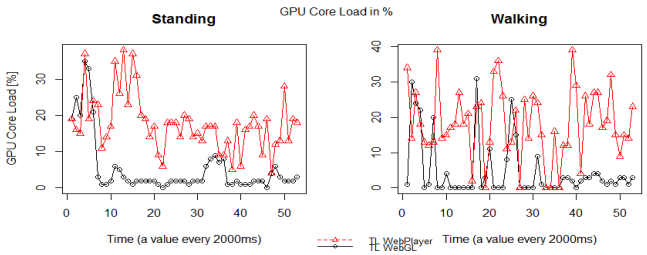

Figure 8: GPU Core Load (%)

more CPU whereas Figure 8 shows that GPU load usage of the UWP build is greater than U-WebGL. The spikes in U-WebGL occur when the avatar is walking. The U-WebGL API uses hardware accelerated rendering on the GPU: on Windows, DirectX is used for U-WebGL; on Linux and OS X, OpenGL is used[22]. Figure 9 shows the GPU utiliza-

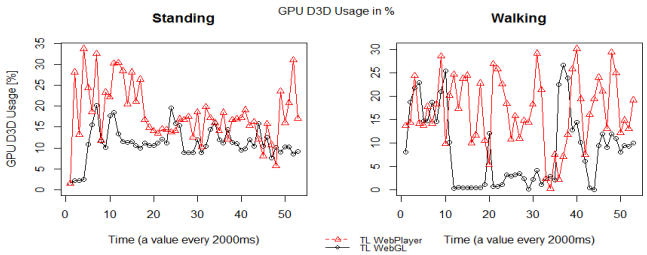

Figure 9: GPU D3D use (%)

tion via DirectX/Direct3D. This covers only the utilization of the DirectX/Direct3D interface subsystem and does not cover usage via other GPU interfaces. Figure 8 and 9 show similar patterns because the major usage is on the Direct3D interface of the GPU. Finally, Figure 10 shows the general GPU Memory usage as a percentage (memory allocated from

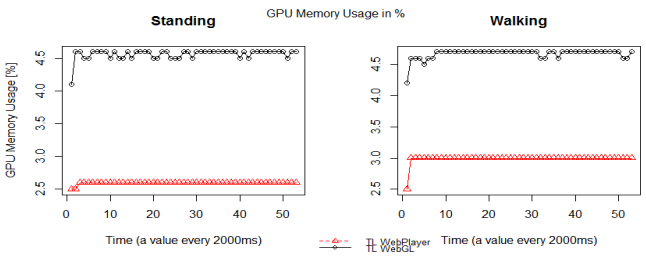

Figure 10: GPU memory usage (%)

Total GPU Memory available in client machine). The U-WebGL build consumes a little more, 1-1.5% GPU memory, than a UWP build of the Timespan Longhouse.

3.3 Initial Download Time

In the same way that the Page Load Time is an important measurement for the 2D Web, the IDT is important for WBVW. Too long a time will result the user losing interest and turning their attention elsewhere.

Table 2: Average RTT & Downlink bandwidths of actual files on the Server, without rendering or processing

Average RTT to Server	Sizes of Raw Files when downloaded	Average downlink bandwidth
0.271 ms	40 MB (UWP file)	11.2 MB/s
	129MB (U-WebGL .data file)	10.4 MB/s

The client and server were connected to different subnets of a campus network, which has minimum link speeds of 100Mb/s. Table 2 summarizes the characteristics of the network path when downloading files from server to client.

Table 3: Initial Download Times (ms) for TL Builds

	Average	Standard Deviation
UWP	3934.33	91.258
U-WebGL	20191.67	581.461

For the IDTs the caches were completely cleared prior to each run. Table 3 shows that the U-WebGL version takes over 20 seconds, whereas the UWP only takes around 4 seconds - a significant difference.

4. RESULTS DISCUSSION

This section discusses some performance issues and possible optimizations. These are based on our experience from conducting the experiments and also from the Unity documentation.

When comparing UWP and U-WebGL builds of the TL world, we notice that the size of the U-WebGL is considerably larger. FPS and FT vary in the same world quite considerably. It is possible to achieve 60 FPS (a very good performance in web browsers) in Unity worlds, as shown in Figure 2. On the other hand, FPS can reach 0 in extremely dense scenes with complex geometry in both U-WebGL and UWP builds. The Fraps measurements were been confirmed by values reported by Google Chrome and Firefox FPS meters. For less than 16 FPS, worlds become noticeably sluggish and scenes begin to take longer to render. A user feels that it takes a considerable amount of time to change the

orientation of her avatar in such low FPS scenes. At rates less than 16 FPS, especially less than 9 FPS, the avatar becomes unresponsive to commands of the user and freezes for several tens of seconds (shown by the very high frame times seen in Figures 3 and 4 - above 1000ms).

There is typically a better frame rate in the UWP version than the U-WebGL version. Walking can generate lower FPS and higher FT than standing when the avatar encounters lower FPS parts of a world. Dynamic batching and other optimizations explained in[24] can alleviate these frame rate bottlenecks. The frame rates and thus the frame times correlate with the complexity of the world, its composition and its size. More complex worlds or even complex scene geometries inside a world lead to lower FPS and higher FTs. Other parameters such as lighting, shadows, and reflection mechanisms are among many others that can influence those rates as well as influencing GPU and CPU consumption. Walking consumes almost the same amount of physical memory as standing because the Unity 3D world takes a fixed reserve share from the browser's memory regardless of the activity, regardless of whether the avatar navigates to a lower frame rate part of the world.

We discovered from our experiments that the U-WebGL version of the same world consumes more CPU, GPU memory and physical memory than the UWP version of the same world. U-WebGL builds have performance bottlenecks in physical memory consumption in their heap memory. U-WebGL memory problems may lead to crashes in browsers running out of memory when loading big U-WebGL projects. Factors to consider include: is the browser a 32 or 64 bit process; if the browser uses separate processes for each opened tab; how the memory is shared between opened tabs; how much memory is allocated for the JavaScript engine to parse and optimize the U-WebGL code. U-WebGL builds can easily produce millions of lines of JavaScript thereby exceeding the maximum amount of JavaScript normally budgeted for browsers. The JavaScript engine has to use large data structures to parse the code leading to considerable consumption of memory (sometime in Gigabytes). The emitted code of JavaScript always needs to be kept in check. Sometimes optimizations like changing the size of memory heap in U-WebGL builds can help in alleviating memory failures[22].

Unity has many techniques for optimizing WebGL projects. Features and techniques like "the optimization level" feature; the "striping level" feature (which strips any classes or components not used during a build) and the "Exception support" feature can all influence the performance and size of the builds and thus their correct configurations contribute to either fast or slow download times. Other modes tune how much exceptions are required in U-WebGL code and on which level and such modes increase/decrease the size of the builds. Unity can also generate a compressed version of the U-WebGL code using gzip for HTTP transfer[22].

Due to the fact that U-WebGL code has to be translated to asm.js, the behaviour of the JavaScript engine in the browser is crucial to the performance of any U-WebGL game. The browser has to have an optimized Ahead of Time Compilation for the asm.js code. Unity advises the use of Firefox as the best browser for this[22].

The Initial Download Time (IDT) of a U-WebGL world is significantly longer than the UWP version, as can be seen in Table 3. This time is being governed by 2 major file types. In a UWP build, IDT is governed by the time it takes to download the .unity3d file(s). On the other hand, in a U-WebGL build, IDT is governed by the time it takes to download the .data file(s) which are normally a lot bigger than their counterpart .unity3d files. Both types of files (.unity3d and .data) can be very big, especially from worlds developed in OpenSim. This can be very problematic with big Web-Based Virtual Environments especially when fetched on slow connections with high rate of packet loss and/or delay. In other words, by default in UWP builds, the entire 3D scene contained in the .unity3d file(s), is completely sent to the client. All files should be received by the client before the user can access or interact with the 3D environment. The progress bar seen by the user when loading a Unity Web-Based world, actually shows mainly the progress in transferring the .unity3d file(s) /.data file(s) and miscellaneous files and an additional time for browser processing. A particularly important file in UWP builds is Object2.js, which is responsible for detecting and communicating with the plugin, for customising the appearance of the loading screen and for embedding the Unity content.

The use of streaming mode[22] is advised in this case instead of downloading the complete files in the beginning of sessions. This allows the user to receive portions of the 3D scene progressively. It is based on the philosophy of making the user access the 3D world as soon as it is possible instead of waiting for a complete download of the world. It is important to think about users who access these 3D worlds on slow connections. The world can be accessed even after downloading 1 MB of data. The game or 3D world can be divided into different levels. Due to the large size of OpenSim projects, transforming these worlds to Unity 3D worlds can generate very large .unity3d or .data files. We advocate always using streaming mode for builds of considerable sizes that originated from Second Life/OpenSim and to divide the scenes into different levels.

Being able to transform worlds already built in OpenSim into Unity is potentially useful but to take advantage of the web, a redesign of the 3D world is needed. Textures take the majority of space. Textures in OpenSim/Second Life may be larger in size and higher in resolution than what is required for the web. Optimizing these textures for a web setting is always needed[22].

Caching would help in making the 3D Worlds run and execute faster becuase the .unity3d file can be stored in the browser cache. Sometimes the default settings whether in Unity builds themselves or on the server that hold Unity worlds do not provide caching capabilities and thus every time the worlds are fetched, all the files need to be sent the client. This could be solved by changing the Unity code itself (the caching class per example[25]) and by setting the headers adequately (Cache-Control header) in a web server directive especially for both .unity3d and .data files.

5. LIMITATIONS

Although this may be the first work to measure QoS metrics in Unity generated WBVW it is not a complete assessment. For example, additional performance measurements for more avatar mobility models and on different graphics cards would give a more comprehensive view. In addition, potential optimizations of worlds in both UWP and U-WebGL builds need to be explored. The worlds measured were transformed from OpenSim to Unity, but were not optimized and hence not necessarily representative of other

types of WBVW. We chose not to use the Unity profiler tool[26] which can generate metrics for CPU, GPU, physical memory, rendering, physics, and audio. Why? Firstly, the numbers that are displayed in the profiler (especially in the memory category) are not the same as the numbers given by operating system facilities such as Task Manager, Activity Monitor and similar tools. This is probably because some normal overhead is not taken into account by the profiler[27]. Also, the memory usage in the editor is different from that shown in the player. Secondly, the frame rates of 3D games in the editor might be a lot higher than the capped 60 frames per second rate in UWP on the majority of web browsers; see the online Unity guide[22]. Finally, we needed to measure QoS metrics from outside the Unity software ecosystem to obtain a degree of objectivity, and to see how these worlds performed in web browsers "in the wild".

6. CONCLUSION AND FUTURE WORK

In this work, we have made QoS measurements of Unity Web Player and Unity WebGL builds of the Timespan Longhouse virtual world. Timespan Longhouse was originally built in OpenSim and then transformed into Unity 3D. We have described a methodology to obtain metrics including FPS, FT, CPU, GPU, Physical & GPU memory usage, and IDT. We have compared the QoS of both types of build and have suggested optimizations and guidelines to improve performance. The main conclusion is that the UWP build has significant advantages in the majority of the QoS metrics used. From a 3D Web perspective however, the need to install the UWP plug-in is not nearly as attractive as simply visiting a virtual world as easily as any other web page.

Future work will consist of finer-grained analysis and measurements of different optimizations and configurations for the respective builds possible for use in standard browsers and also incorporate other types of virtual worlds.

7. REFERENCES

[1] What is X3D. http://www.web3d.org/x3d/what-x3d. Accessed: 2016-02-16.

[2] Peter Paulis. 3d webpages. *Študentská vedecká konferencia, FMFI UK, Bratislava*, pages 316–327, 2010.

[3] Oak3D. http://www.effecthub.com/t/oak3d. Accessed: 2016-02-16.

[4] WebGL - OpenGL ES 2.0 for the web. https://www.khronos.org/webgl/. Accessed: 2016-02-16.

[5] Linden lab - second life. http://secondlife.com/. Accessed: 2015-11-05.

[6] The Open Simulator project. http://opensimulator.org/wiki/Main_Page. Accessed: 2015-11-05.

[7] Colin Allison, Alan Miller, Thomas Sturgeon, Indika Perera, and John McCaffrey. The third dimension in open learning. In *Frontiers in Education Conference (FIE), 2011*, pages T2E–1. IEEE, 2011.

[8] Unity3d game engine. http://unity3d.com/. Accessed: 2016-02-05.

[9] Sheldon Kennedy, Richard Fawcett, Alice Miller, L Dow, R Sweetman, A Field, Arnett Campbell, I Oliver, J McCaffery, and C Allison. Exploring canons & cathedrals with open virtual worlds: The recreation of st andrews cathedral, st andrews day, 1318. In *Digital Heritage International Congress (DigitalHeritage), 2013*, volume 2, pages 273–280. IEEE, 2013.

[10] Colin Allison, Alan Miller, Iain Oliver, Rosa Michaelson, and Thanassis Tiropanis. The web in education. *Computer Networks*, 56(18):3811–3824, 2012.

[11] Opensim client side monitoring. http://opensimulator.org/wiki/Client_side_monitoring. Accessed: 2016-02-05.

[12] J McCaffery, Alice Miller, Sheldon Kennedy, Tom Dawson, C Allison, Anna Vermehren, C Lefley, and K Strickland. Exploring heritage through time and space supporting community reflection on the highland clearances. In *Digital Heritage International Congress (DigitalHeritage), 2013*, volume 1, pages 371–378. IEEE, 2013.

[13] Fraps web site. http://www.fraps.com/. Accessed: 2015-09-10.

[14] HWiNFO64. http://www.hwinfo.com/. Accessed: 2015-09-10.

[15] TechPowerUp GPU-Z. http://www.techpowerup.com/gpuz/. Accessed: 2015-09-01.

[16] Firefox developper tools - network monitor. https://developer.mozilla.org/en-US/docs/Tools/Network_Monitor. Accessed: 2015-09-20.

[17] Opera dragonfly documentation. http://www.opera.com/dragonfly/documentation/network/. Accessed: 2015-09-20.

[18] Chrome - evaluating network performance. https://developer.chrome.com/devtools/docs/network. Accessed: 2015-09-20.

[19] Firefox app.telemetry page speed monitor add-on. https://addons.mozilla.org/en-US/firefox/addon/apptelemetry/. Accessed: 2015-09-21.

[20] extended-statusbar firefox addon. https://github.com/kustodian/extended-statusbar. Accessed: 2015-09-21.

[21] CCleaner tool. https://www.piriform.com/ccleaner. Accessed: 2016-02-16.

[22] Unity documentation. http://docs.unity3d.com/Manual/index.html. Accessed: 2015-09-01.

[23] Emscripten compiler. http://kripken.github.io/emscripten-site/. Accessed: 2016-02-12.

[24] Unity documentation - optimizing graphics performance. http://docs.unity3d.com/Manual/OptimizingGraphicsPerformance.html. Accessed: 2015-10-20.

[25] Unity documentation - caching. http://docs.unity3d.com/ScriptReference/Caching.html. Accessed: 2015-10-10.

[26] Unity profiler. http://docs.unity3d.com/Manual/Profiler.html. Accessed: 2015-10-01.

[27] Memory measurement - unity profiler tool. http://docs.unity3d.com/Manual/ProfilerMemory.html. Accessed: 2015-10-01.