Browser Platform Assessment for X3Dom Graphics' Rendering Capabilities

Panagiotakis Spyros*, Vakintis Ioannis, Kapetanakis Kostas, Malamos Athanasios
Department of Informatics Engineering
Technological Educational Institute of Crete
Heraklion, Crete, Greece, GR 71004
*Email: spanag {at} ie.teicrete.gr

Abstract— **With the course of time, the process of creating and serving web pages is becoming more complicated, since some forms of modern web content are really demanding. 3D graphics are definitely included among them. In that context, the Extensible 3D (X3D) Graphics standard by the Web3D consortium provides a technology for easing the deployment of interactive 3D content over the web. In particular, a combination of technologies such as WebGL, X3D, X3Dom and JavaScript can provide 3D graphics on browsers without any requirement for plug-ins. In this paper, we at first introduce a methodology for efficient evaluation of browsers' performance as it concerns their capability to render X3Dom graphics. Our methodology measures how many frames per second (fps) a browser can render for a specific 3D content. Hence, the fps metric captures the actual 3D content's rendering frequency a browser can reach. Then, we present the results of our benchmark on the assessment of some popular web browsers as they render and display specified X3D graphics under controlled conditions. To this end, a series of experiments using different browsers on a variety of operating systems were conducted to measure browsers' achieved fps. The results underline the performance differences between the various browser platforms.**

Keywords- HTML5, X3DOM, 3D, X3D, Browser, Benchmark, JavaScript, web

I. INTRODUCTION

VRML (Virtual Reality Markup Language) was the first language for representing 3-Dimensional (3D) vector graphics over the web [1]. The acronym was a combination of HTML and 3D. Later, the term changed from Markup to Modeling considering that it implements recreation of a scene in the three dimensions. However, a common VRML format was never supported by the various browser manufacturers. Due to this fact, in 1997 [2] the Web3D Consortium released X3D. The X3D specification [3] established a technology for deploying interactive and animated 3D graphics and scenes over the web using XML. With X3D, the network transfers just a simple text file in XML format, since the whole process for the emergence of 3D graphics runs in the terminals. This is due to WebGL, a custom version of OpenGL, written in JavaScript, especially for web browsers. It contains a function that grants access to the computer hardware (GPU), providing thus to web applications the ability to use computational power at the client side to accelerate rendering.

Following this, the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG) cooperated to include into the emerged HTML5 specification an updated way for the presentation of 3D content in web browsers. HTML5 [4] [5] through its canvas element enabled the presentation of X3D graphics from web pages without requirement for any plugin. That is, all web browsers compatible with WebGL can render interactive 3D graphics. Definitely, the advent of HTML5 associated with other web technologies, such as X3Dom [6], can convert web browsers to 3D friendly multi-platforms with lots of capabilities. X3Dom is a proposed syntax model that translates X3D files to WebGL graphics. X3Dom is considered to be the state of the art technology for the visualization of X3D graphics on the canvas of a web page. With X3Dom's implementation as a JavaScript library, 3D graphics can be readily presented to browsers, without plugins, using only WebGL and JavaScript. More specifically, X3Dom acts as a broker between DOM (front end), which embodies X3D objects, and X3D runtime (backend). The X3Dom library is responsible for the conversion of X3Dom descriptions into the appropriate format for the X3D backend that renders the 3D objects. Hence, X3Dom library and X3D runtime can update the 3D scene whenever any change occurs. The X3Dom library is compatible with a variety of operating systems, devices and web browsers.

In this paper, we at first introduce a methodology for efficient evaluation of browsers' performance as it concerns their capability to render 3D graphics. In particular, we sequentially measure the achieved fps a browser can reach as a constantly incremental number of 3D triangles is drawn over an HTML5 canvas, until the browser reaches its limits. Almost all 3D visualizations involving surfaces are currently done via triangles. For the implementation of the test-bed's web application we have chosen the X3DOM framework because of its provided ability to deliver and render 3D graphics on the browser without any plugin running and consuming resources. In addition, we have developed a JavaScript function that increases the X3Dom workload with triangles and automatically captures the achieved fps. Then, we present the

results of our benchmark on the assessment of some popular web browsers as they run our test-bed web application under controlled conditions. The results underline the performance differences between the various browser platforms. The rest of the paper is organized as follows: sections II and III discuss some similar works and introduce to the architecture of web browsers, respectively. Section IV presents in detail our methodology. Sections V and VI depict the setup of our test-bed and the results from our benchmark, respectively. Finally, section VII concludes the paper.

II. RELATED WORK

A Web browser is an essential tool for every user to download, execute, and render web applications. The main criteria for selecting a browser include: how fast it executes client-side code, how fast it renders web pages and graphics and how optimum navigation experience it provides. Although web 3D content is becoming more and more popular, particularly in gaming, movies and advertising, there are not so many published works related to the performance of browsers when rendering 3D content.

A set of benchmarks for graphics performance based on various VRML browsers is presented in [7]. They use a simple animation construction that rotates a scene in an infinite loop to measure the performance of computers based on two criteria: memory consumption and speed of rendering. Scenes differ in many characteristics, either in quantity (number of polygons, lights, textures, etc.) or in specific features (materials, light sources, texture mapping).

In [8], the authors investigate to which extent web applications are capable of visualizing 3D data on mobile devices. To this end, they present a benchmark which compares the graphics performance between native applications built with OpenGL and web applications built with WebGL in different popular mobile browsers. Then, they also compare the performance of JavaScript with that of native languages for mobile devices. The first issue authors notice is that the performance of desktop and laptop computers is very high compared to that of smartphones. Another general conclusion is that Java and C applications perform significantly better than JavaScript ones, almost 9 to 15 times faster.

In [9] Microsoft announces online the results for the performance of Internet Explorer. Three workloads were used for stressing the browser and evaluating the browsing experience it provides: In HTML5-based FishBowl and Blizzard the workbench assesses how many fish or snowflakes, respectively, Internet Explorer can animate in 60fps. Obviously, this represents the rendering frequency the browser can reach. With Speed Reading it measures how long Internet Explorer takes in seconds to flip billboards of various lengths.

Brian Hook and Andy Bigos in [10] have created D3DBench, a benchmark of nine standardized tests, to measure the performance of some popular hardware 3D graphics accelerators, including 3Dlabs, ATI Technologies, Diamond and Cirrus Logic. Their scope is to provide game developers with a clearest view about expectations from the aforementioned accelerators. The tests check the rendering capabilities of accelerators in terms of triangle- and fill-rate throughput. The triangle throughput measures how many triangles per second a hardware accelerator can handle, while the fill rate throughput measures the pixels per second a hardware accelerator can render.

With respect to the combination of HTML5 with X3Dom technologies, the authors have presented in [11] an interactive 3D game for preschool education utilizing X3Dom and Websockets. An instructor uses a host application to display 3D animals to client-peers. Then the children have a variety of options to interact with the 3D scenery such as to send messages to the host when recognizing the animals placed by the instructor, to add more 3D animals or to rotate them.

With the benchmark we present in this paper, we attempt to assess the performance of some popular and X3Dom compatible desktop browsers in different operating systems when rendering X3Dom content. To the best of our knowledge it is the first benchmark in bibliography that attempts such an assessment since an official benchmark for X3Dom graphics is currently missing. The sections that follow elaborate on our work.

III. ANATOMY OF WEB BROWSERS

The Web browser is one of the most used software applications in computer history. Nowadays, web browsers run in almost every computing device from laptops and desktops, to tablets, phablets and smartphones. Millions of people use browsers, so competition between manufacturers is stupendous. In this section we present the generic architecture of a browser platform and compare from a structural point of view the browsers that take part in our benchmark to highlight their structural differences.

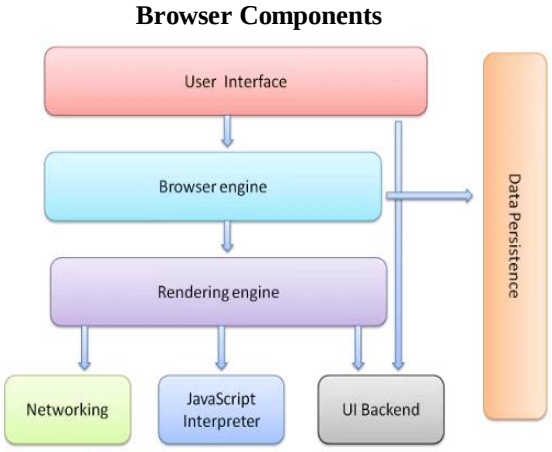

Figure 1. Generic Web browser's Architecture

The main function of a web browser is to display a web source (HTML document, image, video, PDF, etc.) into its main window. The source content is requested via HTTP signaling from a web server, which is running on a specific URI (Uniform Resource Identifier). All browsers implement the same standard specifications for interpreting, parsing and

displaying HTML content. These specifications are mostly developed by the W3C (World Wide Web Consortium) organization [12]. A Web browser consists of several structural parts including the User Interface, the rendering engine, the browser engine, the networking module, the User Interface (UI) backend, the JavaScript Engine and the data storage. Figure 1 depicts the generic browser's architecture.

The most critical components for a web browser are the User Interface, the rendering engine and the JavaScript interpreter. The former is due to the fact that the first criterion for an average user to choose a web browser is its layout. The other two concern technical issues with great impact on the perceived feeling from browsing since they strongly affect the performance of the browser. In the past the whole process was simpler but with the course of time browsing became more demanding due to the evolution of HTML and CSS. Today, every browser has its own rendering engine [13] which is considered to be a separate part from the browser itself. Chrome and Safari use Webkit, Mozilla uses Gecko, Internet Explorer uses Trident, Opera uses Presto. WebKit, for instance, is an open source rendering engine which started as an engine for the Linux platform but was modified later by Apple to support Mac and Windows [14].

Similarly, the execution of JavaScript code belongs to the technical aspects of browsing. Dedicated scripting engines [14] undertake to execute and interpret the nested JavaScript code in Web pages. Every browser use its own scripting engine: Firefox uses Spidermonkey, Chrome uses V8, Internet explorer uses Jscript, Safari uses JavaScriptCore and Opera uses Carakan. At a high-level, there is little difference between various JavaScript engines, but from a technical point of view critical differences can be found. For example, SpiderMonkey is the original JavaScript engine that was first introduced as part of the Netscape Navigator. Then, SpiderMonkey had a couple of updates to improve its performance: TraceMonkey and JägerMonkey with IonMonkey to be expected soon. On the other hand, V8 is relatively new, since it was first introduced as part of Google Chrome. The most important difference between them is in compilation. SpiderMonkey at first interprets JavaScript to an intermediate language, which is then compiled to machine language. V8 directly compiles JavaScript to machine instructions, eliminating thus the need for interpretation.

TABLE I. COMPARISON OF BROWSER PLATFORMS

Browser	Javascript Engine	Rendering Engine
Chrome	V8	Webkit
Mozilla	SpiderMonkey	Gecko
IE	JScript	Trident
Safari	JavascriptCore	Webkit
Opera	Carakan	Presto

Table I includes the browser platforms participated in our benchmark. The browsers are juxtaposed in terms of Javascript and rendering engines. Definitely, the most critical components when running such an X3Dom benchmark are the JavaScript and the Rendering engines.

IV. MEASUREMENTS METHODOLOGY

As we have already mentioned, with this benchmark we attempt to evaluate the performance of browsers as they render X3Dom graphics. In particular, we try to measure how many fps a browser can reach for a specific X3Dom workload. Our methodology at first includes the creation of an elementary 3D scene, over an HTML5 canvas, consisting of 3D triangles. Hence, for this scene we capture the achieved fps. Then we increase the 3D workload by one triangle and capture again the achieved fps. This process is constantly repeated until the browser reaches its limits. Definitely, the more the triangles in a scene, the less the achieved fps. Similarly, the more fps a browser achieves for a certain scene configuration, the more powerful it will be. Almost all sophisticated 3D visualizations involving surfaces are currently done via triangles. Really critical for such interactive visualizations with triangulated surfaces is rendering speed, which is limited by the rate at which triangulated data can be sent to the graphics subsystem for drawing. The rendering time is considerably reduced when polygonal models are partitioned into triangles [15]. Furthermore, our methodology, with its constantly increasing number of triangles in the scene, enables us to precisely monitor the behavior of a browser for any scene configuration: from scenes with just few triangles to scenes with thousands ones. This provided flexibility makes our methodology superior than any similar one using standard 3D models with different Level of Detail for benchmarking the browsers.

JavaScript is the basis for our benchmark. We have developed a JavaScript function that periodically increases the X3Dom workload with triangles and automatically captures the achieved fps for each configuration. In more specific, we are using JQuery to this end. The initial 3D scene for our benchmark displays just one triangle but with the course of time our JavaScript function periodically appends a new triangle in the canvas every 1000 ms. The value of 1000 ms as period is intentionally chosen large, so it assures independence from the underlying hardware. It denotes the required relaxation time between measurements so the operating system returns to idle situation after each measurement. The IndexedFaceSet object is a X3D tag which can be used for creating 3D shapes formed by polygons from vertices listed in the coord field. In particular, the coord field specifies coordinates, and the coordIndex defines the polygons. In our implementation we create triangles by setting the coordindex as "0" "1" "2" "-1". CoordIndex ‘’-1’’ at the end of a polygon signifies the end of the current face and the beginning of the next one. Every new triangle is moved at -0.5 points on each axis. Apart from coordinates we also use some other characteristics, such as size, angle, scale and shape, to fit triangles properly in the scene. The triangles are generated from the bottom left corner of the canvas to the right until the end of the line. Then they follow the same order at the upper line, and so on, until the canvas fulfills with triangles.

Figure 2 illustrates an abstract from the javascript code that runs behind our testing application. In particular; the abstract demonstrates the addition of a new triangle in the canvas. As the benchmark is running we count the achieved frames per second for each scene configuration. The fps method exists in the library X3DOM.js. Apart from fps we can also measure nodes, shapes, draws, points and triangles. Figure 3 illustrates a screenshot from our application.

Two alternative implementations of our testing application were developed: the first one stops only when the scene fills with triangles, the second includes a control mechanism for low fps, so when fps fall under value 1, the application to stop. Since there is not any practical reason for such low measurements, the results presented hereafter are from the implementation with the control mechanism for low fps values.

```
 <!-- ADD NODE -->
var root = document.getElementsByTagName('Scene')[0];
var TransfromElement =
  document.createElement('Transform');
var ShapeElement = document.createElement('Shape');
var AppearanceElement =
  document.createElement('Appearance');
var MaterialElement = document.createElement('Material');
var IndexedFaceSetElement =
  document.createElement('IndexedFaceSet');
var CoordinateElement =
  document.createElement('Coordinate');
   if (positionx>groups){
        positionx=initpositionx;
        positiony++;}
   if (positiony>groups){
        positiony=initpositiony;}
TransfromElement.setAttribute("translation", '
'+positionx/totalScale+' '+positiony/totalScale+ ' '+0 );
 positionx++;
TransfromElement.setAttribute("rotation", '1 0 0 1.571' );
TransfromElement.setAttribute("scale", '0.05 0.05 0.05' );
ShapeElement.setAttribute("DEF", 'trian'+looptime);
AppearanceElement.setAttribute("containerField", '
appearance');
MaterialElement.setAttribute('diffuseColor', 'green');
IndexedFaceSetElement.setAttribute('containerField',
"geometry");
IndexedFaceSetElement.setAttribute('creaseAngle', ".5236");
IndexedFaceSetElement.setAttribute('CoordIndex', "0 1 2 -1");
CoordinateElement.setAttribute("containerField", "coord");
CoordinateElement.setAttribute("point", " -.5 0 .5     .5 0 .5
.5 -0 -.5");
AppearanceElement.appendChild(MaterialElement);
IndexedFaceSetElement.appendChild(CoordinateElement);
ShapeElement.appendChild(AppearanceElement);
ShapeElement.appendChild(IndexedFaceSetElement);
TransfromElement.appendChild(ShapeElement);
root.appendChild(TransfromElement);
<!--END ADD NODE HERE-->
```

Figure 2. Abstract from the javascript code (addition of a new triangle)

In order to have a meter of the perceived multimedia content quality, although secure ranges are not provided, by various standardization fora, empirical studies from independent researchers propose 15 fps as the minimum threshold for smooth video playback [16]. For network computer games the respective threshold is at 30 fps [17].

V. SET UP OF THE TESTBED

The experiments were conducted in our lab, over a set of 10 identical desktop PCs to minimize the impact from hardware. Each such PC is an Intel Core i3 3220 (3.30 GHz), with 4 GB of RAM and an onboard graphics card with 1 GB RAM. Table II describes in detail the PC configuration. The experiments were also run on different operating systems including Windows 7 and Fedora. Table III presents the operating systems under evaluation. For each browser we used the latest stable version available at the benchmark time (November 2013) for each of the aforementioned operating systems. In particular we tested the Opera, Firefox, Chrome, Safari and Internet Explorer platforms juxtaposed in Table IV. We notice that at the benchmark time Safari and IE did not support HTML5 without a plugin.

TABLE II. PC HARDWARE CONFIGURATION

Motherboard	**CPU**	**RAM**	**Graphics card**
Dell, model: 042P49, version: A00	Intel Core i3 3220, 3.30GHz	DDR3, 4096 MB	Intel Graphics, 1024 MB

TABLE III. TESTED OPERATING SYSTEMS

Operation System	**Version**	**Service Pack**	**System type**
Windows 7	Profesional N	Service pack 1	64 bit
Linux	Fedora 18 , “Spherical Cow”	-	64 bit

TABLE IV. COMPARISON OF BROWSER PLATFORMS

Browser	**Version**	**X3Dom**	**Plugin**
Chrome	26.0.1410.43m	YES	-
Mozilla	20.0	YES	-
IE	8.0.6001.18702	YES	Flash 11
Safari	5.1.7	YES	Flash 11
Opera	12.14	YES	-

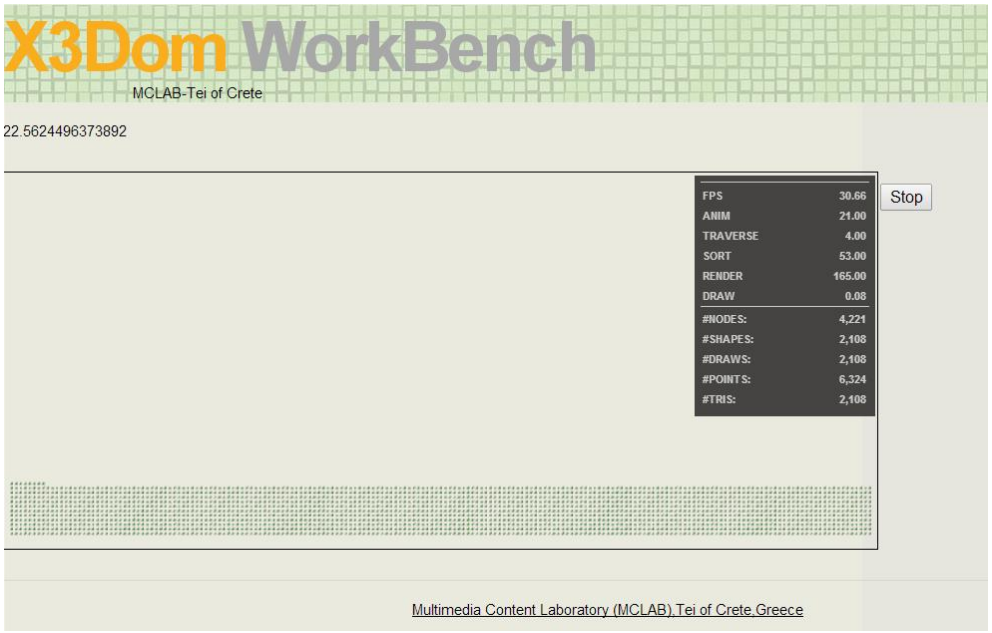

Figure 3. Screenshot from the testing application

Before every test we took care, so browsers, operating system and PCs were configured according to the following: i) The cache of each browser was cleared to avoid running content for our testing application from there. ii) The Screen Saver settings were disabled, so the screen luminosity remained unchanged after a long period of inactivity. iii) PCs remained inactive before measurements, so we could gain the best performance from hardware. Finally, all tests were executed for multiple times and on different PCs to assure consistency of our results.

The experiments were run into two rounds. The first round was a compatibility round to assure in practice whether a browser was able to complete the test satisfactorily. Indeed, only three of the five browsers managed to finish the process without crashing: Chrome, Firefox and Opera. Safari and IE that did not support HTML5 without a plugin crashed after some time. The second round was devoted to the winning browsers. The next section presents our outcomes.

VI. ASSESMENT OF THE BROWSERS' PERFORMANCE

A. *Objectives*

With our benchmark we attempt to answer the following questions:

- What is the best browser platform for an operating system?
- How a browser behaves when the triangles generation period changes from 1000 ms to 250 ms?
- Are there substantial differences between different versions of a browser?

The charts that are presented hereafter, illustrate the achieved fps as a function of the number of triangles in the scene. The maximum capacity of our canvas is 15089 triangles. However, our measurements stopped once a browser achieved rendering frequency of one frame per second. For consistency, our measurements concluded after ten sequential values of 1 fps.

B. *Best browser platform for an operating system*

Firefox is the definite winner in Windows 7. Firefox scores higher fps for both low and high number of triangles. Chrome is in the second place and Opera in the third. Figure 4 illustrates the outcomes for this set. In Fedora, Chrome is the winner, despite the fact that Firefox performance is almost equal to Chrome's one for the first scene configurations as Figure 5 illustrates. But as the number of triangles increases, Chrome achieves more fps. In this benchmark Opera did not participate as at the benchmark time Opera for Fedora did not support WebGL.

According to [18] "*the speed difference between Chrome and Firefox are probably due to the fact that Chrome sandboxes WebGL calls and translates all commands to DirectX, while Firefox runs WebGL in the same process*". However, we cannot confirm this with this benchmark. A very interesting outcome, however, is that almost all browsers start at the same fps value (near 60 fps). The only exception is Opera for Windows 7 that starts at 50 fps. However, the performance of all browsers degrades really fast, as the number of triangles increases, and practically, the achieved rates are very low for the very duration of the experiments. In particular, for scene configurations with more than 4000 triangles almost all browsers achieve rates less than 20 fps, and less than 10 fps for scenes with 7000 triangles or more.

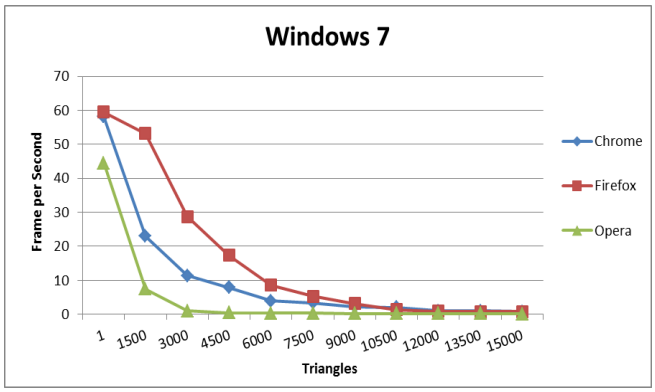

Figure 4. Browsers' assessment for Windows 7

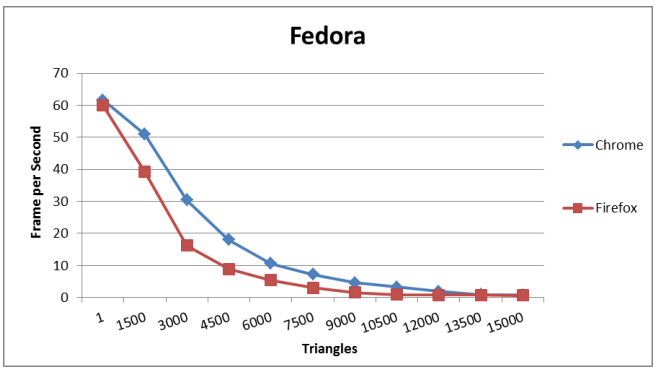

Figure 5. Browsers' assessment for Fedora

C. Browser behavior with the triangles generation period at 250 ms

This set of measurements attempts to bombard browsers with triangles decreasing the period at which they are appended in the scene. At the time that one triangle was appended so far, four triangles are generated now. Hence, the browsers are stressed further in order to let us examine with how robust, efficient and flexible design they have been engineered. Definitely, any browser that manages to handle this additional workload, without severely degrading its performance, it will be a proof of its responsiveness as well as of its robust software design. It is due to the fact that all tests run on the same hardware, hence its impact is standard for all the browser platforms under test.

Figures 6-10 depict the results of our tests for the versions of Chrome, Firefox and Opera we benchmarked. We observe that all browsers present serious degradation of their performance with almost identical curves for this new configuration. Among them, Chrome and Opera for Windows 7 are the worst with very low fps scores after 1500 triangles. Chrome for Fedora and Firefox for Windows 7 are those with highest fps for high number of triangles. In addition, Opera for Windows 7 presents almost identical behavior to its former one with minor difference between the two curves. However, it is questionable if it is due to its good design or due to its limited score at the previous tests (section 5.2).

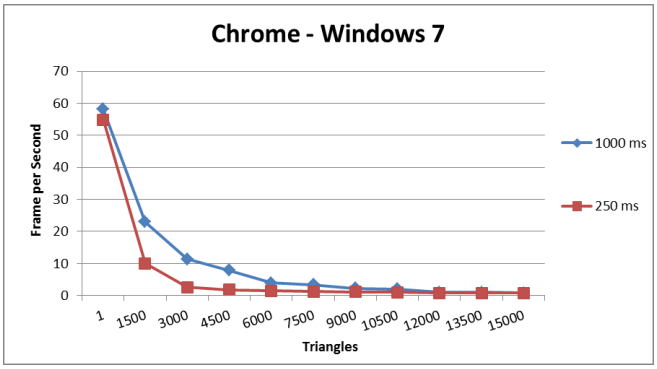

Figure 6. Figure 5: Stress test for Chrome – Windows 7

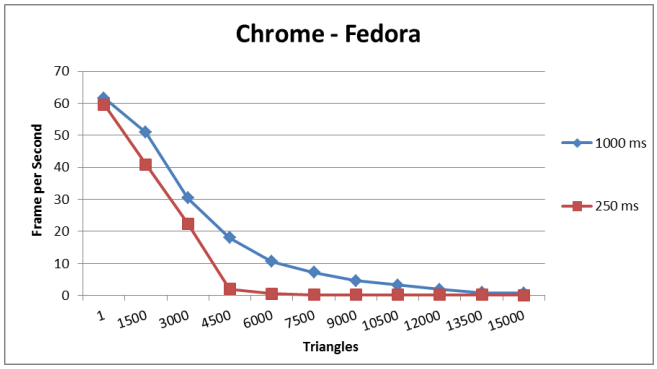

Figure 7. Stress test for Chrome – Fedora

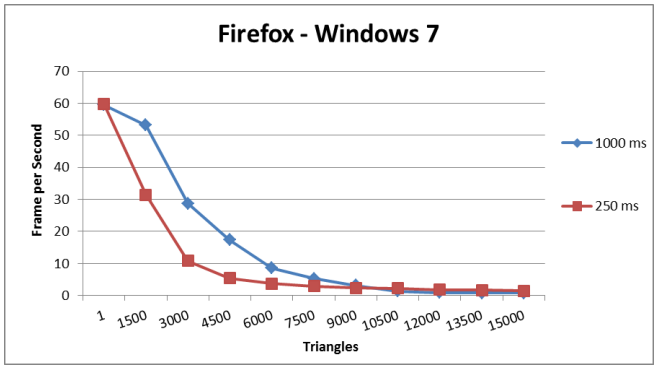

Figure 8. Stress test for Firefox – Windows 7

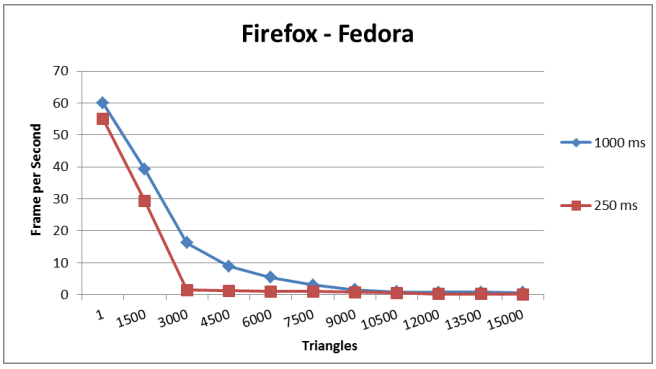

Figure 9. Stress test for Firefox – Fedora

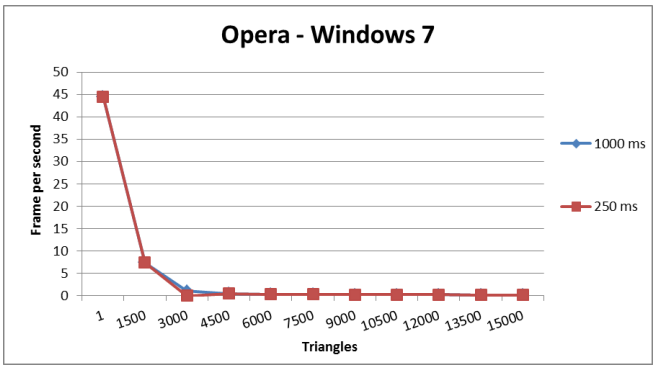

Figure 10. Stress test for Opera – Windows 7

D. Comparision between different versions of a browser

With this experiment we attempt to investigate if substantial difference between different versions of a browser should be expected as far as it concerns its capability to render X3Dom content. Hence, we have indicatively chosen Chrome and Firefox for Windows 7 to provide comparisons between their different versions.

Table V details in the juxtaposed versions for each browser. We understand that this experiment includes a great amount of stochasticity, since it is quite possible a new version of a browser to incorporate several innovations to this end and the next one any. However, such juxtapositions are an indication of the general philosophy behind the software design of the browser's rendering engines.

TABLE V. TESTED VERSIONS OF BROWSERS

Browser	**Version 1**	**Version 2**
Chrome	26.0.1410.43m	31.0.1650.57m
Firefox	20.0	25.0.1

Figures 11 and 12 depict our results. We notice that for Firefox almost nothing has changed but for Chrome a great improvement is audited. The impressive with Chrome is that the new curve has an almost linear change with the increase of the 3D scene, which underlines critical performance update to its rendering engine.

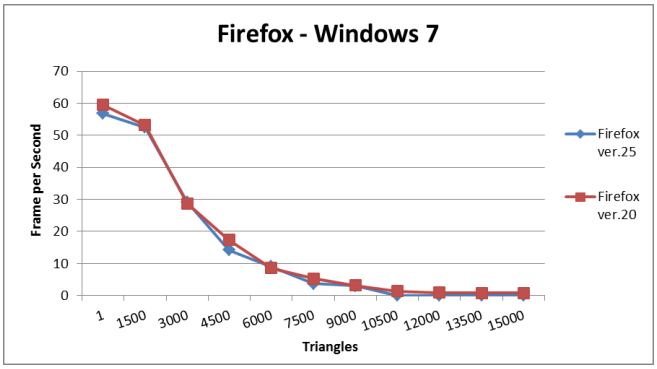

Figure 11. Juxtaposition between two different versions of Firefox for Windows 7

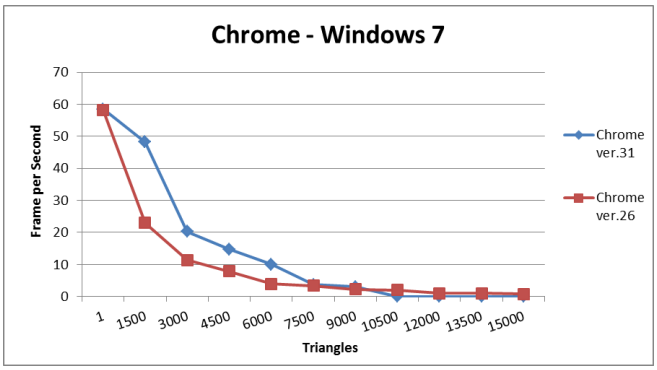

Figure 12. Juxtaposition between two different versions of Chrome for Windows 7

VII. CONCLUSION

This paper introduced a methodology for efficient assessment of browsers' performance as it concerns their capability to render 3D graphics. The methodology is based on a JavaScript function that increases the 3D workload with triangles and automatically captures the achieved fps. For the implementation of the test-bed's web application we have chosen the X3Dom framework because of its ability to deliver and render 3D graphics on a browser without any plugin. For demonstration purposes we also benchmarked some popular browser platforms to evaluate their performance on rendering X3D graphics. The experimental results showed significant performance differences not only between browsers but also between different versions of the same browser. In the future our benchmark could be extended to include some compatible mobile browsers. For example, Firefox and the native Android browser already support WebGL and X3Dom.

Our results, along with the web application we have created for running our tests, are publicly available via the web to anyone wishing to run its own online benchmarks. Information

about our benchmark can be found on http://www.medialab.teicrete.gr/minipages/x3domwb/.

ACKNOWLEDGMENTS

Work in this paper has been funded by the European Union and the Hellenic General Secretariat for Research and Technology (GSRT) under the ARCHEMEDES research framework, T.E.I. Crete project 32 - Acronym : DECO.

REFERENCES

[1] Broll, Wolfgang, and Tanja Koop. "VRML: today and tomorrow." Computers & graphics 20.3 (1996): 427-434.

[2] Brutzman, Don, and Leonard Daly. X3D: extensible 3D graphics for Web authors. Morgan Kaufmann, 2010.

[3] X3D specifications as visited on March 28, 2014, http://www.web3d.org/x3d/specifications

[4] HTML5 as visited on March 28, 2014, http://www.w3.org/html/wg/drafts/html/master/

[5] http://www.web3d.org/wiki/index.php/X3D_and_HTML5, as visited on March 28, 2014.

[6] BEHR, Johannes, et al. X3DOM: a DOM-based HTML5/X3D integration model. In: Proceedings of the 14th International Conference on 3D Web Technology. ACM, 2009. p. 127-135.

[7] Zara, Jiri, and Jaroslav Krivanek. "Graphics performance benchmarking based on VRML browsers." VRIC 2001 Proceedings (2001): 111-120.

[8] Moelker, Ruurd R., and Wilco E. Wijbrandi. "HTML5 data visualization capabilities of mobile devices." 9th SC@ RUG 2011 - 2012: 23.

[9] http://msdn.microsoft.com/en-usus/library/windows/hardware/jj130839.aspx, as visited on March 29, 2014.

[10] http://www.gamasutra.com/view/feature/131609/3d_ acceleration_demystified_part_.php?print=1, as visited on March 29, 2014.

[11] Kapetanakis, Kostas, Spyros Panagiotakis, and Athanasios G. Malamos. "HTML5 and WebSockets; challenges in network 3D collaboration." Proceedings of the 17th Panhellenic Conference on Informatics. ACM, 2013.

[12] W3C website, http://www.w3.org/, as visited on July 7, 2014.

[13] http://en.wikipedia.org/wiki/Web_browser_engine, as visited on July 7, 2014.

[14] https://www.webkit.org/, as visited on July 7, 2014.

[15] F. Evans, S. S. Skiena, and A. Varshney. Optimizing triangle strips for fast rendering. In R. Yagel and G. M. Nielson, editors, IEEE Visualization '96, pages 319-326, 1996.

[16] Yubing Wang , Mark Claypool, Zheng Zuo, "An empirical study of realvideo performance across the internet", in proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement (IMW '01), pp. 295-309, New York, NY, USA , 2001

[17] Frame Rate at wikipedia, http://en.wikipedia.org/wiki/Frame_rate, as visited on July 7, 2014.

[18] C. Hollemeersch, B. Pieters, A. Demeulemeester , P. Lambert , R. Van de Walle, "Real-time visualizations of gigapixel texture data sets using HTML5", Proceedings of the 18th international conference on Advances in Multimedia Modeling, January 04-06, 2012, Klagenfurt, Austria.