Augmented Reality on Mobile Browsers with WebGL

Vinamra Singh

School of Computing and Information, University of Pittsburgh, Pittsburgh, United States
Email: vinamrasingh19@gmail.com (V.S.)
Manuscript received February 17, 2024; revised March 21, 2024; accepted May 1, 2024; published November 13, 2024

***Abstract*—This research investigates the potential of Augmented Reality (AR) applications on mobile devices solely using web browsers, focusing on 3D augmentation via WebGL. It addresses the feasibility of AR in browsers, explores the integration of device sensors, and assesses the performance of WebGL on various mobile platforms. Despite technical challenges, such as camera integration limitations, the study demonstrates the successful implementation of other AR prerequisites on mobile devices. Results reveal significant performance variations between iOS and Android devices, with rendering capabilities generally meeting the task requirements, albeit with some limitations. Battery consumption during WebGL usage is relatively high compared to standard web browsing. Although AR standards in web browsers are not yet mature, and implementations remain unstable, this research suggests that WebGL, in its current state, can augment 3D objects within a user's mobile view. While widespread adoption may not be immediate, ongoing advancements in both mobile hardware and web standards may pave the way for ubiquitous AR applications in mobile web browsers in the near future.**

***Keywords*—WebGL, augmented reality, JavaScript, mobile devices, benchmarks**

I. Introduction

The evolution of technology continually reshapes the way individuals interact with their surroundings. Initially, the web primarily served static content, but it has since evolved into a platform where dynamic and interactive experiences abound. As smartphones became ubiquitous, native applications dominated the landscape. However, there is a notable shift towards browser-based applications, spurred by the increasing influence of mobile web browsers. This trend is expected to accelerate, with web applications projected to eventually supplant traditional binary software [1].

Web applications are increasingly blurring the lines between native and browser-based experiences. Services like Google Docs for word processing and 280 Slides for presentations offer functionalities previously exclusive to native applications, thus narrowing the gap between them [2]. The development of HTML5 and WebGL has further expanded browser capabilities, particularly in supporting three-dimensional graphics natively. This native integration of 3D functionality further enhances web applications, paving the way for richer user experiences.

Augmented Reality (AR) stands at the forefront of merging digital information with the physical world, enhancing users' views with additional context and data [3]. Its potential applications span diverse fields, including healthcare [4], education [5, 6], and tourism [7]. The popularity of AR apps has surged in recent years, thanks in part to the widespread adoption of smartphones. Leveraging AR, particularly through browserbased interfaces, holds promise in revolutionizing how information is accessed and presented, potentially transforming the user experience itself [8–10].

The rise of smartphones is growing at a fast and still accelerating pace and enabled displaying information in a new way in a truly mobile context to many people. Smartphones have been pivotal in reshaping how information is consumed, offering users a truly mobile web experience with the power of "real browsers." Their exponential growth, driven by advancements in technology, has led to unparalleled accessibility to information. However, disparities exist between desktop and mobile browsing experiences, influenced by screen size, processing power, and input methods.

This paper analyzes the status quo and potentials of WebGL on mobile devices and answers the following questions:

1) Are Augmented Reality applications possible on mobile devices using only the browser?
2) Is 3D-augmentation of these applications possible using WebGL?
3) How fast is WebGL on mobile devices?

The paper is structured as follows: Section II gives background information on the relevant topics of augmented reality and WebGL with mobile devices and their evaluation criteria; Section III gives implementation details of the WebGL environment used to capture the status quo; Section IV presents the evaluation as well as limitations to this study; and Section V gives a summary of the results and presents future research opportunities.

II. Background

In this chapter, we delve into key terms and technologies essential for addressing the research questions, namely WebGL and Augmented Reality (AR), and their respective evaluation criteria.

A. WebGL

WebGL represents a significant advancement in web technology by enabling the rendering of 3D graphics directly within web browsers. It is a cross-browser and cross-platform compatible software library and API, extending JavaScript's capabilities to generate websites with hardware-accelerated 3D content. Launched in 2011, WebGL leverages the standard HTML5 <canvas> element and exposes its functionality to developers through the Document Object Model (DOM) interfaces.

Design and Functionality: At its core, WebGL utilizes OpenGL ES 2.0 as its foundation and employs its shading language, GLSL, for graphics rendering. While it shares semantics with desktop OpenGL 2.0 to facilitate code portability, some distinctions exist, such as requirements for power-of-two textures and limited support for 3D textures.

doi: 10.18178/ijfcc.2024.13.4.620

The integration of WebGL into web content allows seamless combination with other web elements, facilitating layering of 3D content with existing web content using languages like JavaScript. This integration enables dynamic and interactive web experiences, empowering developers to create immersive environments where 3D graphics interact with traditional web elements.

Desktop and Mobile WebGL: WebGL enjoys widespread support on desktop browsers, with major players like Firefox, Safari, Chrome, and Opera incorporating compatibility. However, the landscape differs on mobile devices, where support remains limited. While browsers like Firefox and Opera Mobile 12 offer WebGL compatibility on Android platforms, Google Chrome for Android is still in beta and lacks WebGL support.

Apple introduced WebGL capabilities to iOS with iOS 4.2, primarily for use within Apple's iAd platform. However, enabling WebGL in the Safari browser requires non-public APIs and is not feasible for end-user applications due to Apple's App Store Review Guidelines. With a hack discovered by Nathan de Vries, WebGL can also be enabled in a UIWebView. Consequently, true WebGL support on iOS devices remains elusive, hindering the proliferation of WebGL-based applications in the mobile ecosystem.

As the desktop version of Internet Explorer does not support WebGL [2] and Microsoft considers WebGL "harmful," I probably won't see WebGL support on Windows Phone 7 in the near future.

Also, RIM's tablet, the BlackBerry PlayBook, supports WebGL in web applications.

Security Issues: Although the support of WebGL seems to be widespread, there are also critical voices, with Microsoft probably being their most prominent spokesperson. Microsoft believes "that WebGL will likely become an ongoing source of hard-to-fix vulnerabilities" because WebGL allows direct access to the computer's hardware from the web. WebGL security, therefore, relies on graphic card drivers and other third-party components to mitigate the risks.

Context, an information security consultancy recommending disabling WebGL in the browser, published and demonstrated two possible attack scenarios initiated by a malicious website. They were not only able to perform a successful Denial of Service attack, which led to the crashing of operating systems and freezes of desktops, but they were also able to gather confidential information by stealing the content of the graphic memory with which they were able to reconstruct screenshots of the desktop. Context states these issues are inherent to the WebGL specification and can't be resolved without significant changes in WebGL's architecture. Although there are countermeasures in development that could resolve these issues, at the moment, WebGL allows malicious programs access to the graphics hardware and software.

Status Quo and Evaluation criteria: Assessing the prevalence of WebGL support presents challenges, with unofficial trackers like WebGLStats offering insights into adoption rates. Despite approximately 51.1% of desktop users accessing WebGL-enabled browsers, mobile adoption remains low at only 2.3%.

Evaluating WebGL implementations on mobile devices involves various technical and performance assessments, ranging from compatibility with official WebGL desktop browser examples to performance tests and conformance tests [2]. Security concerns also factor into evaluations, focusing on potential vulnerabilities and mitigation strategies.

B. Augmented Reality

Augmented Reality (AR) seamlessly integrates digital information into the user's view of the physical world, bridging digital and physical environments. This technology holds vast potential across diverse application areas, ranging from healthcare and education to tourism and entertainment [3].

New technologies have always influenced human information behavior and will again with possible scenarios because of AR. For example, it may be possible to provide recommendation agents, which have been shown to reduce a consumer's information overload and search complexity [11], directly in-store. Acquiring product information in-store has often been linked to consumer decision-making and information processing [11–13]. Besides cognitive factors, the user's affection may be impacted as well; social aspects of mobile image recognition - attaching digital storytelling to physical products - have been shown to have an impact on a user's affection [14], e.g., trust, engage consumers to communicate and receive information about products [12].

In a survey about the expectations and usage of augmented reality applications [3], people expected them to enhance their lives by providing them with up-to-date information about and proactive functions to act upon context-relevant data that was not readily available before. Furthermore, users expected them to offer "stimulating and pleasant experiences, such as playfulness, inspiration, liveliness, captivation, and surprise." Applications dealing with practical problems were regarded higher than applications for pure entertainment.

The main negative aspects were information flood, user's loss of autonomy, and a possible switch from real to virtual experiences and information [3].

Augmented reality is occasionally used interchangeably with virtual reality because of their similarities, but they have an essential key difference. According to Yi Wu, a senior research scientist in interaction and experience research at Intel Labs, the difference is that augmented reality never substitutes your view of the actual physical world; it merely adds virtual information on top of the natural world. Virtual reality, however, has no natural physical objects in view.

History: Although augmented reality has existed for about half a century, the term was not coined before 1990 by a researcher at aircraft manufacturer Boeing, Thomas Caudell [15]. The definition has been slightly modified over time, but Caudell applied the term to "a head-mounted digital display that guided workers through assembling electrical wires in aircraft."

One of the first devices using augmented reality elements was a machine called Sensorama, built by Morton Helig. This machine was designed to give the users a cinematic experience that is more than just visual. The demo film was a cycle ride through Brooklyn where you were supposed to get the feeling you were the one on the ride (this was done by

vibrating one's seat, blowing wind in one's face, etc). These extra effects might be considered elements of augmented reality as, in the sense of the word, it augmented the reality of the experience.

Many current AR software, such as apps showing points-ofinterest (POI) or product-related information, are considered representatives of augmented reality applications by users. A lot of these AR applications, however, are technically *"pseudoAR"*, as "not all applications align information properly on top of the view of the real world, or the augmented information is not truly 3D" [15].

In April 2012, Google presented its vision of a future technology called "Project Glass." This futuristic vision includes an augmented reality device in the form of glasses, which should cover the functionality of current smartphones (video conferencing, chatting, navigation, messaging, scheduling meetings, etc.). However, constant wearing makes it possible to augment a user's vision continuously. Despite the positive first impressions, some critics have already emerged, targeting mainly the possibility of excessive advertising and generally overwhelming users with information, distraction, and concerns regarding the users' privacy.

Categories: AR applications can be categorized into two main classes: AR browsers and image recognition-based applications *AR browsers* and *image recognition-based AR applications*. AR browsers overlay digital information onto the user's view of the physical world, often sourced from web content. In contrast, image recognition-based applications trigger augmented information based on visual cues, such as QR codes or object recognition [15].

AR browsers are applications that augment the real world seen through the device's camera and other sensors - with digital information, usually from web sources, such as graphics, links, and other objects, similar to mashups [16]. Examples include touristic applications showing POI in the direction the device is oriented at, e.g., Layar (layar.com). This type of AR is usually called a "magic lens" in the literature [15].

Image recognition-based AR applications provide augmented information to everyday objects in the user's view based on visual recognition, which acts as a trigger for the digital information that is then presented, e.g., showing product information like prices and reviews or search results. Visual recognition can happen by identifying markers like QR codes or objects.

Given the two classes, the technical requirements for AR applications that can be deduced are access to the device's video camera, image registration, positioning, orientation, and 2D and/or 3D image overlay.

Native apps and web applications: So far, augmented reality applications are almost exclusively *native* apps, i.e., applications written in the system's native programming language (e.g., Objective-C for iOS or Java for Android). The main reason is that up to recently, the ordinary web browser couldn't access necessary system resources, such as the integrated video camera or the geolocation.

Lately, projects that target resolving this issue have begun to form; these are still in a very early development phase. For example, WebRTC (WebRTC) is a new standard that enables real-time communication (RTC), i.e., the utilization of the video camera and microphone using Javascript and HTML5 for a range of applications such as video chat or voice calls. WebRTC is still in the pre-alpha phase, and so far, only one smartphone web browser supports this technology (Opera 12).

Augmented reality web apps are beginning to emerge, e.g., Argon.

WebGL usage: Real augmented reality applications using WebGL are rare, but tech demos begin to appear, given that access to a device's video camera is now possible, e.g., HTML5WebRTC.

Still, in the beta phase, WebGL Earth webGLEarth is opensource software for visualizing maps similar to Google Earth. It relies purely on HTML5 canvas, WebGL, and JavaScript, which most modern browsers support. Nokia has also created a WebGL-based software for visualizing maps nokiaMap. As an extra feature, they have enabled stereoscopic 3D, if you have 3D glasses, allows you to view cities in 3D.

WebGL seems to spark the interest of game developers, too. There are now games with the aesthetic value seen on many games distributed as a native application on mobile devices. Seeing this trend, Opera Software wants to encourage new developers to this new paradigm.

Evaluation criteria: For AR evolution and adoption, user-oriented issues are critical [3]. AR research, however, "still lacks evaluation methods" [17], and metrics are still very abstract [15]. Obstacles to evaluating the usability of mixed reality systems are manifold and include "common testing platforms and benchmarks" [18]. Many researchers evaluate only early tech demos ([19]); some evaluations oriented towards user experience based on HCI research exist [18], and include subjective ratings, e.g., user surveys with open questions [10].

Many researchers evaluate only early tech demos ([19]); some evaluations oriented towards user experience based on HCI research exist [18], and include subjective ratings, e.g., user surveys with open questions.

C. UI and UX Research

User experience (UX) can be defined as 'a person's perceptions and responses that result from the use or anticipated use of a product, system or service" [20]. In the literature, instrumental and non-instrumental aspects are essential, i.e., pragmatic elements like utility and subjective elements like pleasure and aesthetics [21].

Research on human-computer interaction (HCI) has been traditionally rooted in cognitive psychology, engineering, and computer sciences. Besides these fields, research on emotional factors of design is growing [22], which marks a shift from *usability* to *user experience* analysis.

Academia evaluates user experience oftentimes by looking at the emotional state of the user [23], e.g., by letting users fill out pen-and-paper questionnaires during the usage or by capturing user-made video diaries [24, 25].

III. Implementation

Given the two classes of augmented reality applications, tests had to be made for the following requirements: access to

the device's video camera, image registration, positioning, orientation, and 2D and/or 3D image overlay. Furthermore, WebGL had to be tested as a means of 3D augmentation.

To test the different requirements, I implemented a room where you can look around and interact with several objects, like displaying additional information or starting a movie sequence. Geopositioning and orientation using the device's sensors and image overlays are integrated.

This section is structured as follows: I first describe the application implemented to use WebGL on iOS. Afterward, I go into detail, elaborating on which elements and their properties populate the room. At the end of the section, I explain how I fetch the user's location and how movement and viewing are done.

A. iOS WebGLViewer

As mentioned in section II-A2, implementing WebGL support on iOS posed significant challenges due to the absence of official browser support. Therefore, a custom iOS application named WebGLViewer was developed based on the methodology and instructions provided by Nathan de Vries. WebGLViewer consists of a UIWebView, a reload button, and an address bar. The UIWebView was modified to show WebGL content (compare section II-A2).

In addition to rendering WebGL content, WebGLViewer includes functionality to monitor battery consumption during WebGL usage. When the device is shaken while running WebGLViewer, the application displays information about battery usage, including the current battery level, the level upon page loading completion, the delta between these values, the duration of webpage loading, and the corresponding battery consumption rate per minute. Notably, due to iOS restrictions, battery level updates occur in 5% increments, limiting the precision of battery consumption measurements.

The WebGLViewer application *WebGLViewer* serves as a tool for viewing and evaluating WebGL content on iOS devices, providing insights into performance and resource utilization on mobile platforms.

B. Room

Our application, *Room.html* was developed to demonstrate the capabilities and limitations of WebGL-based augmented reality functionalities on mobile devices. The environment simulates a room where users can interact with various objects, including chairs, tables, a blackboard, and a rotating cube, to explore different AR interactions and experiences. The web application in question, the Room.html, is available on oslo.paulsteinhilber.de/room.html.

Fig. 1 shows a screenshot of room.html running in Chrome on a desktop computer while playing the video. Fig. 2 shows a screenshot of room.html running in the WebGLViewer on an iPhone.

The room environment incorporates primitive shapes and imported 3D models created using tools like Three.js and Blender. Basic shapes like planes and cubes form the room structure, while more complex objects like chairs and tables are imported from Blender. Each object is assigned specific properties and behaviors, such as rotation, video playback, and information display, to enable interactive experiences within the room.

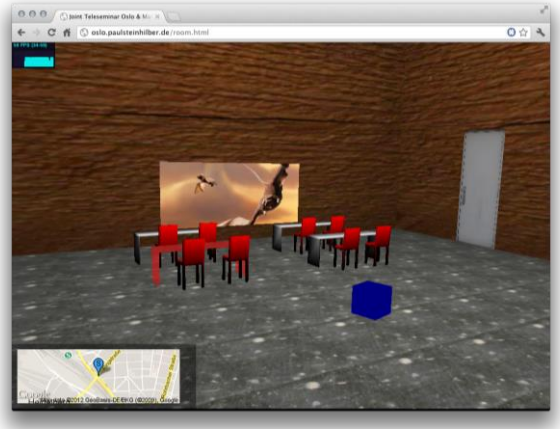

Fig. 1. Room.html with playing video in Chrome on a desktop computer.

Fig. 2. Room.html shown in the WebGLViewer on an iPhone.

The room is built using 6 planes put together to simulate a cube. Each plane has 4 vertices, creating a single polygon. The blackboard is also simple. It is just a single plane and a single polygon. To make the windows and the door, I added three additional planes with a texture, thus making the total number of polygons 9.

Objects: WebGL objects can be built from scratch using basic shapes or imported from 3D modeling software. For our room, I decided to use the tools provided by Three.js, a JavaScript 3D-library, to create the "primitives" (basic geometric shapes such as a plane or a cube) and use Blender, an open source 3D application, for objects of greater complexity (such as the chairs and tables).

The blackboard is a simple plane consisting of 4 vertices, which create a polygon together. I added a video element as a texture for the blackboard. The user can touch it to interact with the blackboard, which plays or pauses the video.

A single table contains 24 vertices, which creates 22 polygons. Considering that our scene contains 4 tables, this means that the desks are responsible for 88 polygons. They all have a grey Lambert material assigned, with the exception of one table, which has a red basic material. This is to clearly show that it differs from the others and is interactive. When this table is touched, a box appears explaining where such a table could be purchased.

A single chair consists of 56 vertices, which results in 57 polygons. I have eight chairs, so this amounts to 456 polygons. All chairs have a red Lambert material assigned, as no chair is interactive.

The cube is natively created in Three.js and consists of 8 vertices and 6 polygons. A basic red shader has been assigned, similar to the other interactive objects. When the cube is touched, it starts rotating and changes its color. Another touch stops the rotation and changes color one more time.

This gives us a total of 560 polygons in our application.

Interaction with Objects: With the *Ray* method/class the *Three.js* library provides a simple method for interacting with 3D objects. By specifying a starting point and the direction of the *Ray*, any objects caught in the ray's path will be added to an array. To identify these objects, I have created an additional array holding all our "assigned" interactable objects. The starting point will always be the camera position, and the direction will be related to the mouse click or touch. I am only interested in the first interacted object (or else, in a larger project, I might be clicking on something in a separate room). So, it is the only object I compare with our assigned interactable objects. When I have identified the object, I perform the corresponding function.

Textures: Textures in the implementation were imported using *THREE.ImageUtils.loadTexture()*, which creates a material based on the texture and combines it with the object. Although compressed textures are technically possible and are important for large applications, this technique was not used in this tech demo.

Accessing Geolocation: To demonstrate that JavaScript can access a mobile device's physical position, I show a map of its location at the bottom of the screen.

To get the device's location, the W3C Geolocation API was used. This API is independent of the underlying method to get the location. The location can be obtained from, for example, GPS or "inferred from network signals such as IP address, RFID, WiFi and Bluetooth MAC addresses, and GSM/CDMA cell IDs".

Moving and Looking around in the Room: Although data from a gyroscope is accessible with JavaScript and could be used in a WebGL application, I have only used data provided by a device's accelerometer since not all of our test devices included a gyroscope. The JavaScript *onDeviceMotion*-event is used to get the accelerometer data. On iOS, "the accelerometer measures the sum of two acceleration vectors: gravity and user acceleration". Using this data, I implemented the feature to look up and down by moving the device. It is implemented in such a way that if the device is flat on a surface, it shows the floor of the room. When holding the device vertically, you are looking at the blackboard.

The room application is reacting to touch events. You can swipe left or right to look to the left or right.

If the application is run from a browser on a stationary machine or a laptop, it is possible to move and look around in the room with the keyboard (W A S D and arrow keys).

IV. Evaluation

To present the findings in our work in a structured and readily available manner, I will devise a test matrix for our results. Since WebGL technology is still relatively young and its support varies from browser to browser and device, it would be tough to devise common criteria for all the devices. Therefore, I am approaching the problem by presenting the results separately for each device in Device - OS - Browser form. In that way, I can outline our observations in a much more structured way and stress the peculiarities of WebGL support on different devices.

Since we aim to investigate WebGL on mobile devices, I chose one desktop computer as a reference. Which OS (Mac OS, Windows or Linux) and particular machine (Mac or PC, laptop or stationary) is used is not that important since all modern computers should be powerful enough to render WebGL graphics. The important thing is to choose a browser that fully supports WebGL. I find that Google Chrome is a good choice for that. All the measurements and observations will be taken running the *Room.html*, as described in Section III-B, developed for the purpose of this paper.

The parameters I am observing are, for the most part, technical. As part of our application, I am measuring Frames Per Second (FPS), which gives us the number of times the browser refreshes the screen per second. In addition to the FPS, I am also considering parameters like CPU usage and battery consumption, but as noted before, the different parameters will be mentioned when applicable.

Furthermore, in a similar approach to Golubovic *et al.* [2], the common functionality of WebGL is tested by executing the lessons of the WebGL tutorial. These lessons are based on a popular OpenGL tutorial (*NeHe*, nehe.gamedev.net/) and cover most of the standard functions, and should be enough to put our target devices to the test. The results of these tests are presented in table III.

In addition to technical parameters, another exciting area of evaluation of WebGL is user experience. But since the most valid and exhaustive results in that field come from a large user survey [3], I will not do any work in that area.

A. Energy Consumption

The WebGLViewer application for iOS, as described in Section III-A, was used to measure battery usage. The device was fully charged. The accordant content was opened in the WebGLViewer application. The device was then unplugged, and the content reloaded. The time until the battery level decreased to 90% as well and the battery consumption per minute was measured and calculated by the WebGLViewer. Using Apple Instruments, a part of Apple's Developer Tools, I measured the *CPU Activity* (*Total Activity*, *Foreground App Activity* and *Graphics*) as well as the relative *Energy Usage* on a scale from 0 to 20. All tests have been performed on an Apple iPhone 4 with iOS 5.1 installed using the WebGLViewer. I have compared the battery usage and CPU activity of the implemented *Room.html*, the *Quake 3 WebGL Demo* by Brandon Jones (as shown in Fig. 3) and google.com as reference. The results are shown in Table 1.

Comparing the two WebGL applications to a regular website like google.com shows a huge difference in CPU activity. The overall CPU activity is below 10%, with a value below 1% for Foreground App Activity as well as *Graphics Activity*, while using google.com. In contrast, the overall CPU activity is over 70%, with about 60% for *Foreground*

App Activity and up to 25% for *Graphics Activity*, while using a WebGL application.

Table 1. Battery consumption and CPU activity of different WebGL applications

Test Case	Battery and Energy Consumption			CPU Activity		
	Time till 90 %	Battery Usage	Relative Energy Usage	Total	Foreground App	Graphics
room.html	36.03 min	0.278 *min/%*	17	100 %	64 %	20 %
Quake 3	34.67 min	0.289 *min/%*	16	70 %	60 %	8 %
google.com	78.93 min	0.127 *min/%*	11	6 %	0.2 %	0.5 %

If I look at the battery consumption, there is also a significant difference. The battery drains twice as fast while using a WebGL-enabled website compared to google.com. The difference regarding the *Relative Energy Usage* is not as big; however, I don't know how Apple calculates this value.

B. *Frames Per Second*

To test the performance of WebGL on different platforms, I have measured and compared frames per second (FPS) for different applications. FPS describes the frequency at which images are generated. The values below 10-12 FPS are recognizable by the human eye as separate images; for higher FPS, single images cannot be recognized, and they blend, creating motion [26]. Therefore, a higher number of FPS creates a more fluid animation and is considered better. Since the refresh rate of modern flat screens is 60 FPS, there is usually no need to render with more than 60 FPS.

In Table 2, I compared the FPS of the implemented *Room.html*, the *Quake 3 WebGL Demo* (as shown in Fig. 3), and an example of a spinning cube, the *SpiritBox* (as shown in Fig. 4), on different mobile devices as well as on one laptop computer (2010 MacBook) and stationary computer (2006 MacPro) as reference.

Table 2. Frames Per Second (FPS) of different WebGL applications on different devices

Device	Operating System	Browser	Launched	room.html	Quake 3	SpiritBox
Apple iPad 2	iOS 5.1	WebGLViewer	2011	61 FPS	61 FPS	60 FPS
HTC EVO 3D	Android 2.3.4	Firefox	2011	8 FPS	12 FPS	N/A [1]
Apple iPhone 4	iOS 5.1	WebGLViewer	2010	40 FPS	29 FPS	43 FPS
Apple iPod Touch (4th Gen.)	iOS 5.1	WebGLViewer	2010	21 FPS	25 FPS	43 FPS
HTC Desire	Android 2.2.2	Firefox	2010	2 FPS	N/A	8 FPS
Apple iPhone 3GS	iOS 5.1	WebGLViewer	2009	36 FPS	27 FPS	60 FPS
Reference Laptop [2]	Mac OS X 10.7.3	Google Chrome	2010	34 FPS	36 FPS	50 FPS
Reference Computer [3]	Mac OS X 10.7.3	Google Chrome	2006	58 FPS	59 FPS	85 FPS

[1] The FPS value constantly alternates between values in the range from 10 FPS up to over 200 FPS, making it impossible to determine a realistic value.

[2] Reference Laptop: MacBook 2010, Mac OS X 10.7.3, 2.26 GHz Intel Core 2 Duo, 4 GB DDR3 RAM

[3] Reference Computer: MacPro 2006, Mac OS X 10.7.3, 2x 2.0 GHz Dual-Core Intel Xeon, 6 GB DDR2 RAM

Fig. 3. Quake 3 WebGL Demo shown in the WebGLViewer on an iPhone.

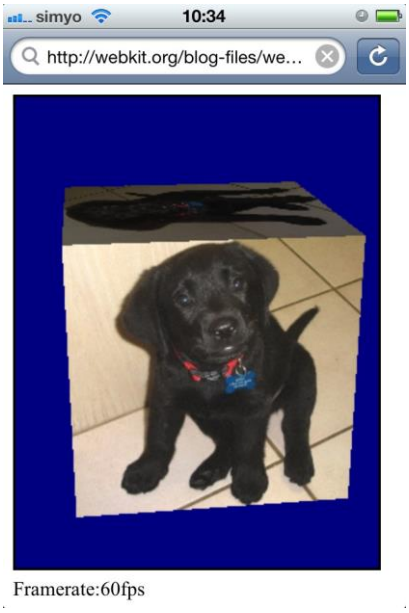

Fig. 4. WebGL demo SpiritBox.

Table 3. Frames per Second (FPS) using the lessons from the learning WebGL tutorial on different devices

8	Reference[1]	Desire	EVO 3D	iPhone 4	iPhone 3GS	iPad 2	iPod Touch
3 - Simple animations	59 FPS	13 FPS	16 FPS	57 FPS	57 FPS	58 FPS	40 FPS
4 - 3D animations	58 FPS	12 FPS	19 FPS	57 FPS	57 FPS	58 FPS	57 FPS
5 - Textures	59 FPS	13 FPS	16 FPS	40 FPS	57 FPS	58 FPS	40 FPS
6 - Texture filters and keyboard	59 FPS	12 FPS	18 FPS	40 FPS	57 FPS	58 FPS	40 FPS
7 - Basic Lighting	59 FPS	12 FPS	17 FPS	57 FPS	57 FPS	58 FPS	57 FPS
8 - Transparency and blending	59 FPS	11 FPS	16 FPS	58 FPS	57 FPS	58 FPS	58 FPS
9 - Particles	59 FPS	8 FPS	17 FPS	39 FPS	31 FPS	60 FPS	40 FPS
10 - Loading a map	58 FPS	11 FPS	15 FPS	57 FPS	57 FPS	58 FPS	40 FPS
11 - Sphere and rotation	59 FPS	12 FPS	15 FPS	40 FPS	57 FPS	58 FPS	57 FPS

12 - Point lighting	58 FPS	11 FPS	15 FPS	40 FPS	57 FPS	58 FPS	40 FPS
13 - Per-fragment lighting	59 FPS	10 FPS	17 FPS	36 FPS	40 FPS	58 FPS	39 FPS
14 - Specular highlights and JSON model	58 FPS	9 FPS	15 FPS	35 FPS	39 FPS	58 FPS	36 FPS
15 - Specular maps	58 FPS	8 FPS	16 FPS	22 FPS [2]	24 FPS [2]	58 FPS [2]	22 FPS [2]
16 - Render to texture	58 FPS	6 FPS	16 FPS	22 FPS	22 FPS	58 FPS	22 FPS

[1] Reference Computer: MacBook 2010, Mac OS X 10.7.3, 2.26 GHz Intel Core 2 Duo, 4 GB DDR3 RAM

[2] Rendering results aren't looking as expected

The lessons from learningWebGL.com have been used as a benchmark to test the performance on different mobile devices. For example, Fig. 5 shows a screenshot of lesson 9. The results are shown in Table 3.

Fig. 5. "Lesson 9" WebGL demo.

Opera has advertised the WebGL capabilities beginning with Opera Mobile 12, but the performance of WebGL content on this browser is much worse than in Firefox. Although all lessons from table III are rendered correctly, the frame rate was always below 10 FPS, compared with 16 FPS in average in Firefox on the HTC EVO 3D. On the other hand, the *SpiritBox* is not rendered correctly (only a blue square is visible) and the room.html doesn't load at all.

The most obvious result is the big performance difference between the iOS and Android devices. Even the HTC EVO 3D, launched in 2011, could not deliver results comparable to the iPhone 3GS launched in 2009. On the Android devices, the *Room.html* was unusable to interact with fluidly, whereas it was rendered smoothly on all iOS devices.

While Android devices have always had lower frame rates than iOS devices, they could render all lessons from Table 3 correctly. iOS-based devices, on the other hand, were not able to render lesson 15 correctly. Screenshots to compare the rendering of lesson 15 on Android and iOS are shown in Fig. 6 and Fig. 7, respectively.

Another interesting result is that the iPhone 3GS delivers significantly higher frame rates in some tests compared to the iPhone 4. An explanation could be that the iPhone 4's display resolution is significantly higher, thus requiring the calculation of four times more pixels.

Comparing the performance of our reference machine, the 2010 MacBook, with iOS-based mobile devices indicates that today's mobile devices are powerful enough to render 3D applications with an acceptable frame rate.

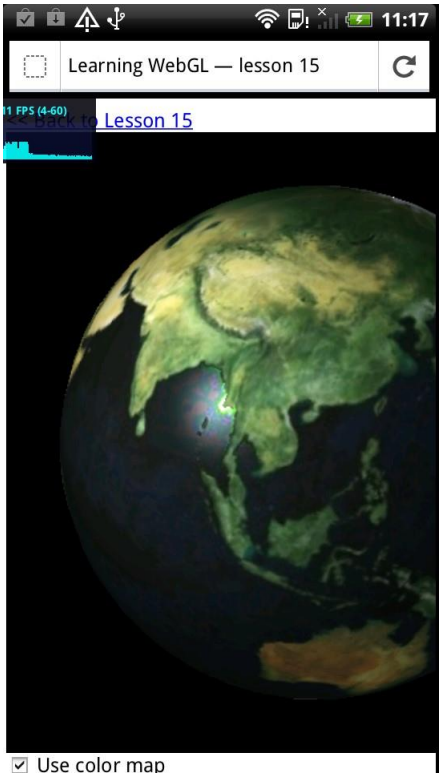

Fig. 6. "Lesson 15" WebGL demo on Android.

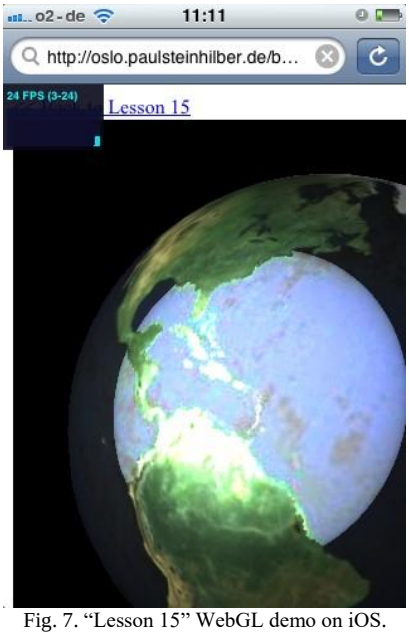

Fig. 7. "Lesson 15" WebGL demo on iOS.

C. Limitations

Although most of the tests I have performed were successful, I encountered a few limitations during our test phase. I am listing them further down the text.

Device Sensors Access: WebRTC, an upcoming standard for access to the device's video camera and microphone, is

currently only available on Opera Mobile 12, a browser currently not supported on iOS. I could not implement a WebRTC application with Android devices that lacked performance.

During our evaluation and testing on other devices, I realized that the accelerometer wasn't working as intended on Android using an HTC EVO 3D running the *Room.html* in Firefox. The room's floor is shown as expected when the device is flat on a surface. Holding it vertically in your hand, however, still shows the floor. The accelerometer is undoubtedly working, but it seems to be much less sensitive than the one on iOS-based devices. Therefore, the feature to look up and down with the accelerometer is deactivated when a non-iOS device is detected.

Key and Mouse Events: As expected, key and mouse events cannot be triggered in WebGL on mobile devices with touch screens, but replacing them with touch events is a working solution for this problem.

Video texture: Touching the blackboard should start playing a movie on it. It worked perfectly on desktop computers using Windows, Mac, and Linux as operating systems and Chrome, Safari, and Firefox as browsers. However, I could not get the video texture to work correctly on mobile devices.

Namely, neither audio nor video was reproduced on the mobile devices after touching the blackboard, except working audio on the iPad 2. Using an iPhone 4, a significant frame rate decrease could be observed after tapping the blackboard. After touching the blackboard, the frame rate dropped from 47 FPS to just 20 FPS. While measuring the CPU activity, after wiping the blackboard, the *Foreground App Activity* dropped from about 60% to 30% while the *Audio Processing* increased from 0% to about 10%, with an overall CPU activity of 100% on an iPhone 4.

The video played ideally using just the HTML5 video tag on iOS using Safari or WebGLViewer; hence, a wrong video encoding could be eliminated as a source of the problem. Since the video texture works on all browsers and all desktop operating systems, and the video's audio is even played on the iPad 2, I assume our implementation is correct.
Therefore, video texture not working for us is a limitation of WebGL on mobile devices.

V. Conclusion

In this paper, I investigated the feasibility of augmented reality (AR), specifically 3D augmentation, on mobile devices using web browsers. Through developing an application and rigorous testing, I aimed to assess the technical capabilities and limitations of implementing AR experiences in a browserbased environment.

The technical possibilities for accessing the device's sensors for interacting with the user's context, crucial to developing AR applications, are now given, even in an early phase. One of the most common properties associated with augmented reality is camera integration, and although it is technically possible to realize this using pre-alpha software, due to the limitations of the mobile devices I tested (as mentioned in section IV-C1), I were not able to push augmented reality in a browser using a camera. Other requirements for augmented reality applications on mobile devices were successfully shown.

As I can see from the results, there was a vast difference between iOS and Android about performance, but both seemed to handle the task, though with certain limitations, as discussed previously. The application renders on all our devices, making essential user interaction possible. Compared to a reference machine, I saw that the rendering cycle expressed through the FPS is a bit lower on mobile devices but still good (except Apple's iPad 2, whose frame rate is comparable to the one on our reference machine). Furthermore, I observed that battery consumption is relatively high while running WebGL applications compared to ordinary web surfing.

Our results, presented in this paper, indicate that WebGL in its current state, can be used on mobile devices to augment 3D objects to a user's view. The standards for AR applications in web browsers are not yet completed, and implementations are not stable yet, making widespread use nearly impossible. However, if Moore's law [27] applies to the progression of mobile devices as well as web standards, and mobile software continues to follow its current pace, I can be sure that augmented applications in web browsers could be ubiquitous on mobile devices in the near future.

Installing WebGLViewer

To install the WebGLViewer using the provided source code on an iOS device, a membership in Apple's iOS Developer Program is needed. However, compiling and running the WebGLViewer in the iOS Simulator is possible using XCode. As an alternative, a registered developer can provide you with a binary, which is built for your specific device. I used testflight.com to distribute such binaries. To request a binary, you must create an account on http://bit.ly/zWoZQJ and register your device as soon as you are accepted.

Conflict of Interest

The author declares no conflict of interest

References

[1] A. Taivalsaari, T. Mikkonen, M. Anttonen, and A. Salminen, "The death of binary software: End user software moves to the web," in *Proc. 2011 Ninth International Conference on Creating, Connecting and Collaborating through Computing*, 2011, pp. 17–23.

[2] D. Golubovic, G. Miljkovic, S. Miucin, Z. Kaprocki, and V. Velisavljev, "Webgl implemenation in webkit based web browser on android platform," in *Proc. Telecommunications Forum (TELFOR)*, 2011, pp. 1139–1142.

[3] T. Olsson and K. Va¨an¨ anen-Vainio-Mattila, "Expected user experience¨ with mobile augmented reality services. workshop of mobile augmented reality," *MobileHCI 2011*, 2011.

[4] W. L. D. Lui, D. Browne, L. Kleeman, T. Drummond, and W. H. Li, "Transformative reality: Augmented reality for visual prostheses," in *Proc. 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011.

[5] F. Mannuß, J. Rubel, C. Wagner, F. Bingel, and A. Hinkenjann, "Augmenting magnet field lines for school experiments," in *Proc. 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011.

[6] G. Liestol, "Learning through situated simulations: Exploring mobile augumented reality," *(ECAR Research Bulletin 1, 2011) Boulder*, CO: EDUCAUSE Center for Applied Research, 2011, pp. 1–14.

[7] A. Mulloni, H. Seichter, and D. Schmalstieg, "User experiences with augmented reality aided navigation on phones," in *Proc. 10th IEEE*

International Symposium on Mixed and Augmented Reality (ISMAR), 2011.

[8] T. Hollerer and S. Feiner, "Mobile augmented reality," in *Telegeoinformatics: Location-Based Computing and Services*, Taylor and Francis Books Ltd, 2004.

[9] P. Wellner, W. Mackay, and R. Gold, "Back to the real world," *Communications of the ACM*, vol. 36, no. 7, pp. 24–26, 1993.

[10] X. Qiao, R. Pei, S. Dustdar, L. Liu, H. Ma, and C. Junliang, "Web ar: A promising future for mobile augmented reality—state of the art, challenges, and insights," in *Proc. the IEEE*, vol. 107, pp. 1–16, 02 2019.

[11] T. Kowatch and W. Maass, "In-store consumer behavior: how mobile recommendation agents influence usage intentions, product purchases, and store preferences," *Computers in Human Behavior*, vol. 26, no. 4, pp. 697–704, July 2010.

[12] S. Karpischek and F. Michahelles, "my2cents—digitizing consumer opinions and comments about retail products," in *Proc. Internet of Things (IOT)*, 2010.

[13] B. Xiao and I. Benbasat, "E-commerce product recommendation engines: use, characteristics, and impact," *Management of Information Systems Quarterly*, vol. 31, no. 1, 2007.

[14] R. Barthel, A. Hudson-Smith, M. Jode, and B. Blundell, "Tales of things. the internet of 'old' things: collecting stories of objects, places and spaces," in *Proc. the Urban Internet of Things*, 2010.

[15] T. Olsson and M. Salo, "Online user survey on current mobile augmented reality applications," in *Proc. 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, October 2011, pp. 75–84.

[16] D. Schmalstieg, T. Langlotz, and M. Billinghurst, "Augmented reality 2.0," *Virtual Realities*, Springer, Vienna, 2011.

[17] M. G. et al., "Experience with an ar evaluation test bed: Presence, performance, and physiological measurement," *ISMAR 2010*, 2010, pp. 127–136.

[18] C. Bach and D. L. Scapin, "Obstacles and perspectives for evaluating mixed reality systems usability," in *Proc. the IUI-CADUI Workshop on Exploring the Design and Engineering of Mixed Reality Systems (MIXER)*, 2004.

[19] A. Dunster, R. Grasset, and M. Billinghurst, "A survey of evaluation¨ techniques used in augmented reality studies," in *Proc. ACM SIGGRAPH 2008*, 2008.

[20] I. organization for standardization, "Iso fdis 9241-210:2009. ergonomics of human system interaction—part 210: Human-centred design for interactive systems (formerly known as 13407)."

[21] M. Hassenzahl and N. Tractinsky, "User experience—a research agenda," *Behaviour and Information Technology*, vol. 25, no. 2, pp. 91–97, 2006.

[22] D. A. Norman, "Emotion and design: Attractive things work better," *Interactions*, vol. 9, no. 4, 2002.

[23] V. Roto, "Web browsing on mobile phones—characteristics of user experience," Ph.D. dissertation, Helsinki University of Technology, 2006.

[24] M. Csikszentmihalyi and R. Larson, "Validity and reliability of the experience-sampling method," *Journal of Nervous and Mental Diseases*, vol. 175, no. 9, pp. 526–536, September 1987.

[25] M. Isomursu *et al*., "Experience clip: method for¨ user participation and evaluation of mobile concepts," in *Proc. the Eighth Conference on Participatory design: Artful Integration: Interweaving Media, Materials and Practices*, 2004, pp. 83–92.

[26] P. Read and M.-P. Meyer, *Restoration of Motion Picture Film (Butterworth-Heinemann Series in Conservation and Museology)*. Butterworth-Heinemann, 2000.

[27] R. R. Schaller, "Moore's law: past, present and future," *IEEE Spectrum*, vol. 34, no. 6, pp. 52–59, June 1997.