An Adaptive Resolution Scheme for Performance Enhancement of a Web-based Multi-User VR Application

Rishabh Pathak[1], Anderson Augusto Simiscuka[1], *Member, IEEE*, and Gabriel-Miro Muntean[1], *Senior Member, IEEE*

***Abstract*—Over the last few years, several frameworks have been introduced to help developers build virtual reality (VR) experiences for the web. One such open-source web framework is the A-Frame framework, which is built on top of WebXR and Three.js. A-Frame can be used to create 3D scenes that can be rendered using compatible browsers such as Chrome and Firefox. The performance of web-based VR applications, however, can be affected due to the limitations of the CPU and GPU, especially in multi-user applications. In this paper, an A-Frame-based multi-user VR application is developed and the performance is analyzed under different scenarios, demonstrating how an increase in the number of users affects metrics related to VR quality of experience (QoE). Then, an Adaptive Resolution Scheme for VR (ARS-VR) is proposed, which improves the VR performance in terms of frame rate and frame latency on remote devices with limited processing and display features.**

***Index Terms*—VR, WebXR, A-Frame, Three.js, frame rate, resolution, adaptation**

I. Introduction

WEBXR, formerly known as WebVR, is an application programming interface (API) that has made possible the development of VR applications on the web. As the name suggests, it is a combination of two technologies - Web and VR. The web takes care of the HTML, CSS styling, and JavaScript while VR is the usage of virtual reality devices such as Oculus and Google Cardboard. It is used in conjunction with WebGL, which is a JavaScript API that can be used for rendering 2D and 3D graphics in various compatible web browsers. These technologies together enable the user to experience a VR scene on their browsers. While VR applications built using development engines such as Unity and Unreal Engine need to be downloaded and installed on the user devices, with WebXR it is possible to access a VR application in a web browser.

The work presented in [1] showcases a Unity-based VR application wherein they evaluate the throughput and round trip time (RTT) against an increasing number of simulated users and server refresh rate. Authors in [2] attempt to evaluate A-Frame [3] as a VR technology by creating a 360-degree virtual tour consisting of panoramic equirectangular photos of the Conímbriga Monographic Museum in Portugal. A-Frame can be tested in terms of frame rate to demonstrate the quality and responsiveness of the experience [4], [5]. This indicates the need for approaches that increase the frame rate metric, especially in devices with limited processing resources.

[1]R. Pathak, A. Simiscuka and G.-M. Muntean are with the Performance Engineering Laboratory, School of Electronic Engineering Dublin City University, Glasnevin, Dublin 9, Ireland. (e-mails: rishabh.pathak2@mail.dcu.ie, anderson.simiscuka2@mail.dcu.ie and gabriel.muntean@dcu.ie)

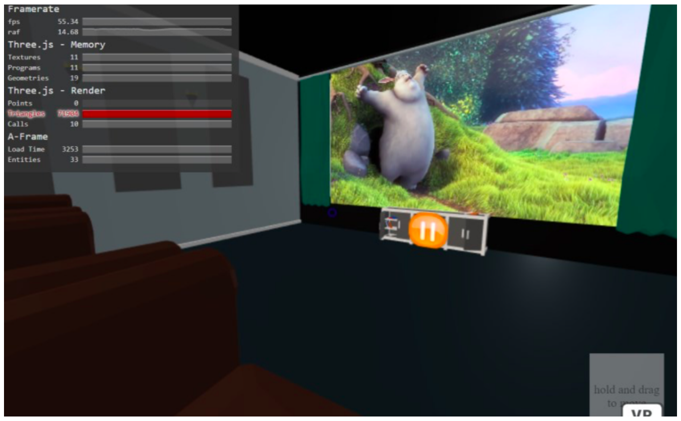

Fig. 1. Multi-user VR application theater room

As demonstrated by [1] and [6], rendering VR graphics in web browsers can be a graphics processing unit (GPU) intensive job and requires high-end graphics and processing hardware to provide a stable frame rate, which is the minimum requirement for a smooth user experience.

This paper describes the design and implementation of a multi-user VR application developed using the A-Frame library, rendered as a 360-degree experience within a browser. The application employs a novel Adaptive Resolution Scheme for VR (ARS-VR), which improves the VR performance in terms of frame rate and frame latency on remote devices with limited processing and display features.

A-Frame offers cross-platform support, being compatible with iOS, Android and Windows devices with an active internet connection. Fig. 1. shows the proposed application, a VR movie theater-like setting. Multiple remote users can join the VR scene with or without a VR headset and can watch a movie together. This allows for analysis of the performance of the application with multiple users connected via different devices consuming rich media content in VR.

The remaining sections of this paper are organized as follows. Section II presents related works and Section III discusses the application design and implementation. Section IV describes the performance analysis. Conclusions and directions for future work end this paper in Section V.

II. Related Work

A number of approaches have been reviewed in relation to the scheme proposed in this paper. For instance, the authors in [7] built a synchronization test-bed consisting of a web application that uses A-Frame based HTML pages. These pages create a binding between the user interface and the core of another application that delivers instructions to IoT devices in a VR-IoT synchronization system [8]. The research focuses on achieving timely synchronization between IoT devices in the real and virtual world.

Authors in [6] proposed a multi-user VR application to facilitate inter-communication between different VR hardware vendors such as HTC Vive and Oculus Rift. Whereas authors in [1] analyzed the performance of a Unity-based VR application by spawning multiple moving users that can share streamable files in a virtual space through a mobility algorithm. However, the impact on the frame rate with an increasing number of users on the same machine has not been considered as the author has attempted to evaluate the performance on the network level.

The users in the VR environment as demonstrated in [6] and [1] are simulated within the VR environment in the form of capsule-like avatars, but the users might feel socially detached. The scheme proposed in [9] tries to tackle this problem by using a green screen to blend people into the VR environment developed using A-Frame. They observed certain resolution problems with the videos and background images. Also, the browser runs at a frame rate of 50-60 frames per second (fps) which is less than they anticipated nevertheless, it is difficult to say if the frame rate was supposed to be more than that unless they had mentioned the specifications (primarily the screen refresh rate) of the laptops used.

Most standard laptops have a 60Hz refresh rate and a frame rate above the laptop's refresh rate can create screen tears and visual lag unless vertical sync (Vsync) [10] is enabled. Some gaming laptops provide up-to 165Hz refresh rate even though only a few of those are fully capable of supporting VR because of differences in the GPU. Less than 1% of sold computers and 6.8% of sold smartphones are VR ready, representing only 13 million computers and just under 200 million smartphones [11], [12]. The number was estimated to reach 100 million by 2020 which is still a small number as from 2015 to 2019, roughly 1.5 billion PCs were in use worldwide [13]. Regarding VR support on multiple devices, [14] also talks about problems with VR performance on mobile devices in multi-user distributed VR spaces. This paper proposes a novel adaptive resolution scheme within a multi-user VR application that aims at providing the best possible frame rate against different number of users, which if integrated with the WebXR API in future could help support VR for low-end devices.

III. Application Design and Implementation

The 360-degree scene in the application was principally designed using the A-Frame framework. It primarily uses HTML and has several Javascript libraries supporting it to take care of the 3D boilerplate and bi-directional communication between the clients and the application server. These libraries and their respective versions used have been listed in Table I. Some of the minified versions of the Javascript files were imported with their respective content delivery network (CDN) links.

TABLE I
JavaScript Libraries Employed

Library	**Version**
A-Frame	1.0.4
three.js [16]	r119
WebGL [17]	2.0
WebVR PolyFill [18]	0.10.4
Networked A-Frame [19]	0.7.1
socket.io [20]	2.2.0
node.js [21]	12.18.3
npmjs [22]	6.14.7

The application allows users to watch a film in a synchronized manner, in a virtual cinema room, as seen in Fig. 1. The film used for the tests is the Big Buck Bunny animation [15].

As network security is a crucial factor in multi-user VR systems [1], [6], [9], the application keeping in mind security over the public internet uses a reverse proxy service called ngrok to expose the local application server to a firewall on the public internet using secure tunnels [23]. Therefore, users that are not on the same network as the server can access the application by querying the link to the secure tunnel. The reason this tunnel is called secure is because it uses Transport Layer Security (TLS version 1.2 - RFC 5246) to send the application data.

A. Client-Server Architecture

Two laptops were used as the application server and the client with their respective specifications shown in Tables II and III.

Additionally, a smartphone was used to simulate a client from a different network than the server. The configuration is given in Table IV. Fig. 2 demonstrates the architecture or the testbed containing the application server and the clients. The external clients access the application server through the secure tunnel as shown in the figure.

B. Client-Server Operation

Before the users can join the VR scene, the server is started by using the NPM JavaScript package manager so that it can pre-load all the dependencies inside a JavaScript Object Notation (JSON) file for Networked A-Frame. Networked A-Frame is another library that is built on top of A-Frame and helps with the creation of templates for the position synchronization of the avatars and video playback. Once the server starts listening on port 8080, the client can create a ngrok tunnel instance to open the application server port to receive public requests. Ngrok allows the use of mobile networks for the testing of the application, and a 4G connection was used.

Once the user opens a browser window, the request is sent to the server based on the source. If the request is made locally, a direct connection with the server is initiated, but if the request is made via a public network, it is re-directed through the

TABLE II
LAPTOP 1 - SPECIFICATIONS

Parameter	Value
Model	Dell Inspiron 3551 15
Processor	Intel Pentium Quad Core N3540
Memory (RAM)	4GB
GPU	Intel Integrated HD Graphics
Screen Refresh Rate	60Hz
Screen Resolution	1366x768
Aspect Ratio	16:9
Operating System	Windows 10

TABLE III
LAPTOP 2 - SPECIFICATIONS

Parameter	Value
Model	Dell Inspiron 13-5378
Processor	Intel Core i7-7500U
Memory (RAM)	16GB
GPU	Intel HD Graphics 620
Screen Refresh Rate	60Hz
Screen Resolution	1280x720
Aspect Ratio	16:9
Operating System	Windows 10

TABLE IV
SMARTPHONE - SPECIFICATIONS

Parameter	Value
Model	Motorola One Power
Processor	Qualcomm Snapdragon 636
Memory (RAM)	4GB
GPU	Qualcomm Adreno 509
Screen Refresh Rate	60Hz
Screen Resolution	360x749
Aspect Ratio	19:9
Operating System	Android 10

tunnel to the server. Once a connection is established, the server starts sending application data to the client and Adaptive Resolution Scheme for VR (ARS-VR) starts. The ARS-VR algorithm is deployed via JavaScript functions and it adjusts the resolution of the application within the browser window by looking at the frame rate (see Algorithm 1). The aim is to adapt the resolution of the rendered content in order to increase the amount of frames rendered per second, which is an important metric in 3D and VR applications. The higher the frame rate, the higher the user perceived quality, in relation to the fluidity of the graphics.

Before the algorithm can start adjusting the resolution, all textures in the scene, images, video elements need to be resized to the power of 2 in case they are not. This is done by the WebGLRenderer instance of three.js. In order to maintain compatibility with multiple devices, inconsistencies are handled by the PolyFill instance so that the content works on multiple platforms regardless of the WebVR support. Once the dependencies load and the scene starts to render, the client is added to the room and a corresponding avatar is spawned based on the avatar template. As long as the current client browser window remains open, the frame rate data is continuously fetched from a JavaScript function.

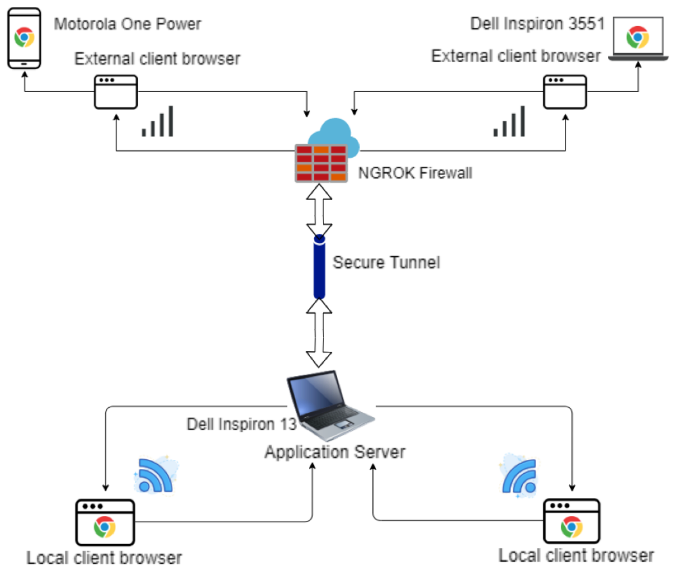

Fig. 2. Application client-server architecture

At this point, two variables are initialized with the product between the device pixel ratio and the screen width and height respectively. Another function calculates the greatest common divisor between the two newly initialized variables. The variables are now divided by the divisor to get the aspect ratio. The numerator and denominator of the aspect ratio are multiplied with the frame rate to provide a dynamic resolution. This algorithm was found to be suitable for laptop 2 with a widescreen aspect ratio. In the case of smartphones, the screen height is usually more than the width therefore, the algorithm first takes a ratio of the width to the height and if it is less than 1, it adjusts the resolution using the screen height and width of the smartphone. The resolution, in this case, is not dependent on the frame rate.

In both cases, the user sees a drop in the resolution but in a multi-user environment, it intends to keep a stable average frame rate between 40-60fps. The reason why the resolution is dropped for smartphones is that by default, the application is rendered at a resolution of 1920x1920px [24], which is way more than the physical resolution of mid-range smartphones. One might argue that because of high number of pixels per inch (ppi) animations will be sharper and crisp on a small size display, but unless the device has a high-end processing unit to support it, it will only make the application lag and cause motion sickness if the user is wearing a VR headset.

C. User Mobility and Audio-Video

Playback Simulated users can move around in the scene using the standard movement keys "w", "a", "s" and "d". Apart from the libraries mentioned in Table I, two more open source projects were used for allowing users to exchange voice messages and control playback of the video inside the scene [25], [26]. The WebRTC adapter enables microphone access in the browser for voice chat. The other project is a Javascript file based on top of the Networked-A-Frame library which defines a custom A-Frame component for broadcasting the video playback events to all the clients on the same network.

Algorithm 1: Adaptive Resolution Scheme for VR

```
while browser_window==open do
    Input: framerate; screen_width;
            device_pixel_ratio; window_height
    Output: maxCanvasSize
    Function greatest_common_divisor(w,
     h):
        var w = screen_width*device_pixel_ratio
        var h = window_height*device_pixel_ratio
        var ratio = screen_width/screen_height
        var gcd = greatest_common_divisor(w, h)
        var num = w/gcd
        var den = h/gcd
        return num, den, ratio, w, h
    Function Main:

    if ratio <1 then
        maxCanvasSize = {height: h/2, width: w/2}
    else
        maxCanvasSize ={height: framerate*den,
          width: framerate*num}

        return maxCanvasSize
    end
end
```

It roughly synchronizes the time each client is into the video since they entered the scene so that everyone is on the same point in the video once they join the room. The networked A-Frame library also uses the Open-EasyRTC server API for video synchronization [27]. Another custom component was added to the HTML front-end to allow users to broadcast play-pause events in the video.

IV. Performance Analysis

The testbed was set up as illustrated in Fig. 2 and analysis was performed after measuring QoS parameters for multiple users served by the same application server. Although A-Frame supports most browsers such as Chrome, Firefox, and Internet Explorer, Chrome was used as the client browser for all the experiments to keep the results consistent as all browsers have different rendering capabilities.

A. Frame Rate and Resolution Analysis

The application server was started and multiple browser windows were opened once a client avatar appeared for each window. The application was allowed to run at its default resolution for approximately 120 seconds with the algorithm code commented out and the video inside the scene playing. The frame rate data was sent to the console output every second.

The browser windows were closed and console data was collected. The application window was opened once again. However, this time, the JavaScript code for the algorithm was allowed to run. Fig. 3 shows the average frame rate for 4

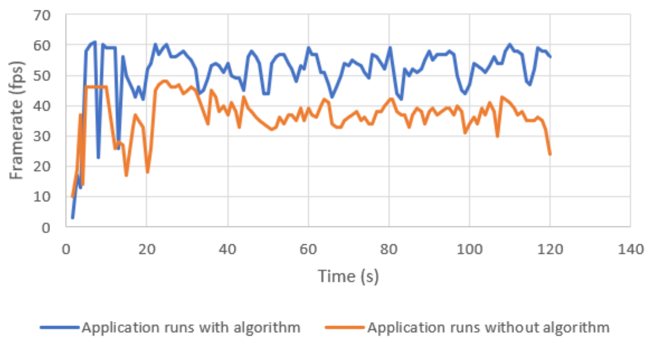

Fig. 3. Variation of frame rate for 4 users

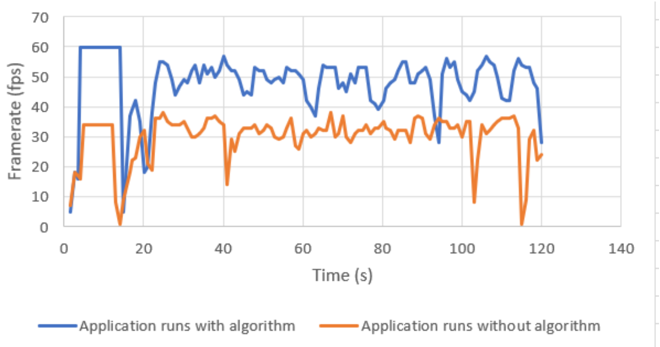

Fig. 4. Variation of frame rate for 6 users

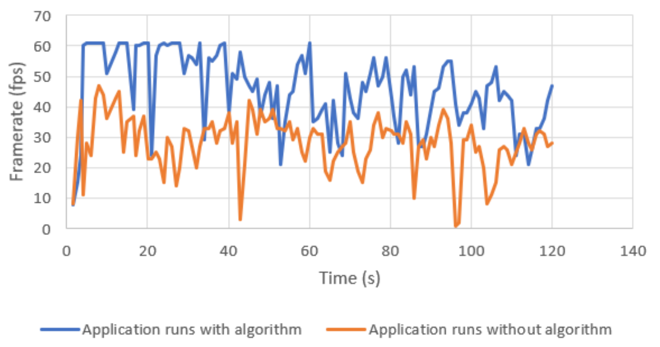

Fig. 5. Variation of frame rate for 8 users

simulated users with and without the algorithm running behind the scenes. The graphs indicate that the frame rate in the former case started well nevertheless, it dropped a few times and could peak up-to 50fps. In the latter case, the frame rate starts great and remains steady between up to 60fps for the whole duration. The same experiment for the same duration was repeated for 6 and 8 users. Figs. 4 and 5 show the variation of the frame rate against time in each case, respectively.

The downward spikes in Fig. 5 just after 40s is due to the frame rate getting dropped every time a new client was added. More downward spikes correspond to more users being added. The other downward spike just before 100s is because of the navigation between browser windows. The average frame rate in default resolution and adjusted resolution for each case is shown in Table V.

These results indicate that if the ideal frame rate for the 60Hz client monitor is considered to be 60fps, the algo-

TABLE V
AVERAGE FRAMERATE (HIGHER IS BETTER)

No. users	Avg. fps	Avg. fps-ARS	Avg. resolution	FPS loss reduction
4	36.65	50.94	806x453	23%
6	29.69	47.33	801x450	30%
8	27.94	45.23	667x375	28%

TABLE VI
STANDARD DEVIATION

No. of users	STDEV	STDEV-ARS
4	8.7078	8.7078
6	7.6326	8.7078
8	8.7078	8.7078

rithm reduces the fps loss in the scenarios with 4, 6, and 8 simultaneous users by 23%, 30%, and 28%, respectively. Beyond 8 users, the CPU of the used computer reaches its full functioning capacity and after a point, the system starts to lag and either crashes or stops responding. In spite of having a reasonable GPU, the client system is bound by its CPU.

In order to verify that the values of the frame rate were not too spread out from the mean value, the standard deviation for frame rate of all scenarios was calculated, as seen on Table VI.

The impact of the algorithm on resolution over the 120-second tests is shown in Figs. 6, 7 and 8, in the scenarios with 4, 6 and 8 users, respectively. The resolution variation has a transitory initial period (visible in all graphs), followed by a relatively stable period, once the framerate is also adjusted. Without ARS-VR, the default resolution of 1920x1920px is rendered by A-Frame, with the FPS being affected by performance issues. With ARS-VR, the resolution is adjusted so the FPS is improved. As seen in Table V, the resolutions, are on average 806x453px, 801x450px and 667x375px for the scenarios with 4, 6 and 8 users, respectively.

B. *Frame Latency Analysis*

For the analysis of frame latency, a WebVR API method [28] was used to provide a Document Object Model (DOM) timestamp. This method calls another method [29] to tell the browser that animation will be performed. The first method is then called once again to calculate the timestamp difference between the two values (once before the animation and once after) which gives the requested animation frame latency (RAF) also updated in real-time in the application window. This value is printed to the console output against a generated frame serial number starting from 1 for every user.

The same procedure for server and tunnel initiation was repeated and browser windows were opened on the client laptop. The console output was averaged with the number of users or windows and resulting values were plotted against the frame number. The results are shown in Figs. 9, 10 and 11.

The plots indicate that the frame latency peaks up-to 800 and 700 milliseconds respectively in the case of 4 and 6 users. The average frame latency when the application runs without

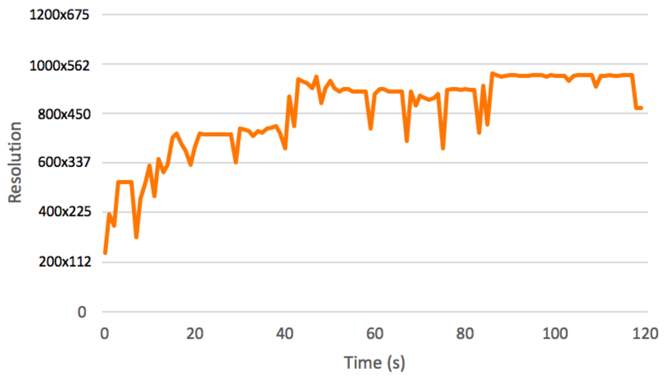

Fig. 6. Resolution Variation for 4 users

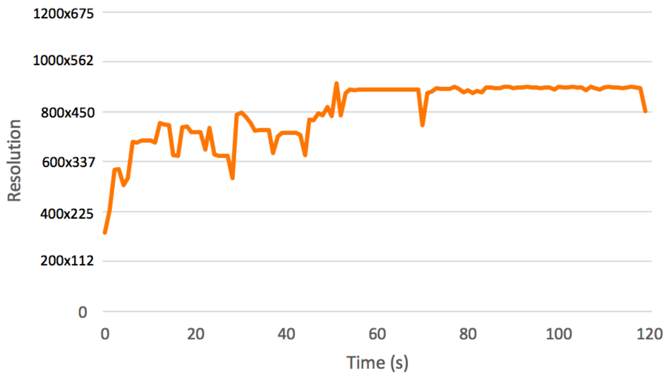

Fig. 7. Resolution Variation for 6 users

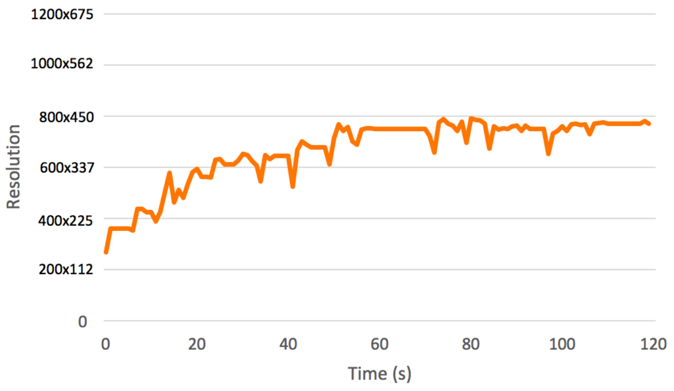

Fig. 8. Resolution Variation for 8 users

the algorithm was observed to be between 90-100 milliseconds whereas when it runs with the algorithm, it was observed to be between 70-80 milliseconds. However, when the users were increased to 8, there is another peak of the latency reaching as high as 1500 milliseconds which is due to the larger processing power needed at this point.

Thus, by performing these experiments, it was possible to analyze the performance of the application under various scenarios by observing the variation of frame rate and frame latency on a laptop and a smartphone having a wide screen and a short width aspect ratio respectively. ARS-VR shows good improvement in the frame latency on both the devices. On the smartphone, the frame rate peaks up-to 60fps, a good improvement given the graphics configuration of the device.

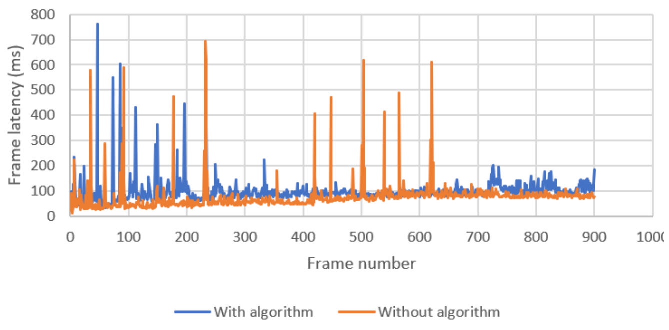

Fig. 9. Variation of frame latency for 4 users

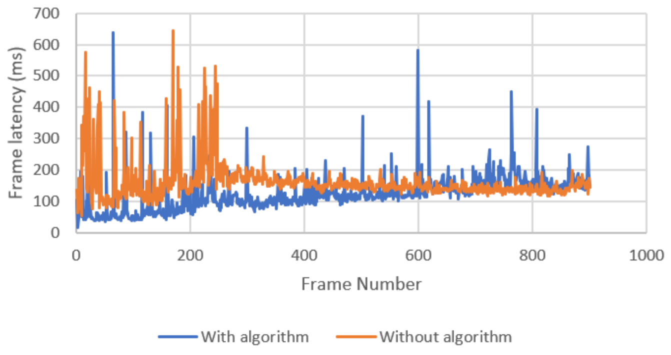

Fig. 10. Variation of frame latency for 6 users

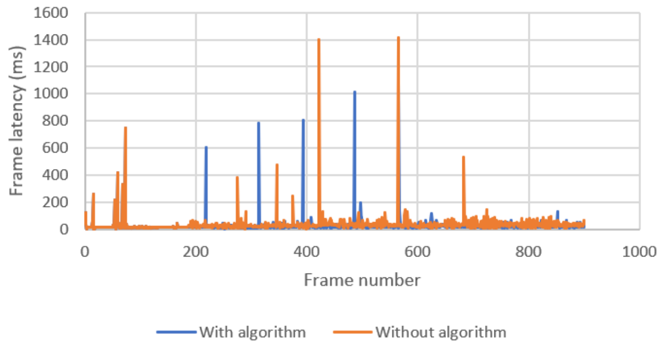

Fig. 11. Variation of frame latency for 8 users

V. Conclusions and Future Research Directions

This work presented an A-Frame browser-based multi-user VR application rendered on remote devices with limited processing and display features. The ARS-VR algorithm was also introduced, aimed at improving VR performance in terms of frame rate and frame latency on such devices, with the dynamic adaptation of the visuals resolution. ARS-VR significantly reduces the fps loss in the scenarios of 4, 6, and 8 simultaneous users by 23%, 30%, and 28%, respectively.

Future work will improve the algorithm to support devices with different screen height and width ratios. Other user applications, such as video conferencing in A-Frame and other device types will also be studied.

Acknowledgements

This work was supported by the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement no. 870610 for the TRACTION project. The support of the Science Foundation Ireland (SFI) Research Centres Programme via grants SFI/12/RC/2289_P2 (Insight) and SFI/16/SP/3804 (ENABLE), co-funded by the European Regional Development Fund, is also gratefully acknowledged.

References

[1] V. Parthasarathy, A. A. Simiscuka, N. O'Connor, and G.-M. Muntean, "Performance Evaluation of a Multi-User Virtual Reality Platform," in *International Wireless Communications and Mobile Computing (IWCMC)*, 2020, pp. 934–939.

[2] S. G. Santos and J. C. Cardoso, "Web-based virtual reality with A-frame," in *Iberian Conference on Information Systems and Technologies, CISTI*, 2019.

[3] "A-frame." [Online]. Available: http://www.aframe.io/

[4] R. Marx, S. Vanhove, W. Vanmontfort, P. Quax, and W. Lamotte, "DOM2AFRAME: Putting the web back in WebVR," in *International Conference on 3D Immersion (IC3D)*, 2017, pp. 1–8.

[5] M. Letić, K. Nenadić, and L. Nikolić, "Real-time map projection in virtual reality using webvr," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2018, pp. 1439–1443.

[6] S. Abbas, A. A. Simiscuka, and G. M. Muntean, "A Platform Agnostic Solution for Inter-Communication between Virtual Reality Devices," in *IEEE 5th World Forum on Internet of Things, WF-IoT 2019 - Conference Proceedings*. IEEE, 2019, pp. 189–194.

[7] A. A. Simiscuka and G. M. Muntean, "Synchronisation between Real and Virtual-World Devices in a VR-IoT Environment," in *IEEE International Symposium on Broadband Multimedia Systems and Broadcasting, BMSB*, 2018.

[8] A. A. Simiscuka and G.-M. Muntean, "REMOS-IoT-A Relay and Mobility Scheme for Improved IoT Communication Performance," *IEEE Access*, vol. 9, pp. 73 000–73 011, 2021.

[9] S. Gunkel, M. Prins, H. Stokking, and O. Niamut, "WebVR meets WebRTC: Towards 360-degree social VR experiences," *Proceedings - IEEE Virtual Reality*, pp. 457–458, 2017.

[10] T. Lacoma, "What Is VSync, and When Should You Use It?" *Digital Trends*, Aug 2020. [Online]. Available: https://www.digitaltrends.com/computing/what-is-vsync/

[11] "Less than 1% of pcs can run virtual reality, bbc news," vol. 04-Jan-2016. [Online]. Available: https://www.bbc.com/news/technology-35220974

[12] J. B. E. 19th April 2017, "6.8% of smartphones are ready for vr." [Online]. Available: https://www.gamesindustry.biz/articles/2017-04-19-6-8-percent-of-smartphones-are-ready-for-vr

[13] "Pcs installed base worldwide 2013-2019." [Online]. Available: https://www.statista.com/statistics/610271/worldwide-personal-computers-installed-base

[14] B. Maclntyre and T. F. Smith, "Thoughts on the Future of WebXR and the Immersive Web," in *IEEE International Symposium on Mixed and Augmented Reality, ISMAR 2018*, 2018, pp. 338–342.

[15] "Blender Cloud - Big Buck Bunny." [Online]. Available: https://cloud.blender.org/films/big-buck-bunny

[16] "Three.js." [Online]. Available: https://threejs.org

[17] "Webgl." [Online]. Available: https://www.khronos.org/webgl/wiki

[18] Immersive-Web, "immersive-web/webvr-polyfill," *GitHub*. [Online]. Available: https://github.com/immersive-web/webvr-polyfill

[19] H. L. Networked-Aframe, "networked-aframe/networked-aframe." [Online]. Available: https://github.com/networked-aframe/networked-aframe

[20] *Socket.IO*, Aug 2020. [Online]. Available: https://socket.io/

[21] Node.js. [Online]. Available: https://nodejs.org/en/

[22] [Online]. Available: https://www.npmjs.com/

[23] "What is ngrok?" [Online]. Available: https://ngrok.com/product

[24] Aframevr, "Ability to cap the canvas size by benjaminleonard, Pull Request 3641, aframevr/aframe." [Online]. Available: https://github.com/aframevr/aframe/pull/3641/files

[25] machenmusik. [Online]. Available: https://glitch.com/edit/!/networked-aframe-synced-video-example?path=public/video-transport.js:5:28

[26] [Online]. Available: https://webrtc.org/

[27] Open-Easyrtc, "open-easyrtc/open-easyrtc." [Online]. Available: https://github.com/open-easyrtc/open-easyrtc

[28] "performance.now()." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Performance/now

[29] "Window.requestanimationframe()." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame