HYBRID MOBILE AUGMENTED REALITY: WEB-LIKE CONCEPTS APPLIED TO HIGH RESOLUTION 3D OVERLAYS

A.-M. Boutsi[1], C. Ioannidis[1], S. Soile[1]

[1]Laboratory of Photogrammetry, School of Rural and Surveying Engineering, National Technical University of Athens, Greece; iboutsi@mail.ntua.gr, cioannid@survey.ntua.gr, ssoile@survey.ntua.gr

Commission II

KEY WORDS: Mobile Augmented Reality, 3D overlays, cross-platform, marker-based, pose estimation

ABSTRACT:

Mobile Augmented Reality (MAR) aligns toward current technological advances with more intuitive interfaces, realistic graphic content and flexible development processes. The case of overlaying precise 3D representations exploits their high penetration to induct users to a world where data are perceived as real counterparts. The work presented in this paper integrates web-like concepts with hybrid mobile tools to visualize high-quality and complex 3D geometry on the real environment. The implementation involves two different operational mechanisms: anchors and location-sensitive tracking. Three scenarios, for indoors and outdoors are developed using open-source and with no limit on distribution SDKs, APIs and rendering engines. The JavaScript-driven prototype consolidates some of the overarching principles of AR, such as pose estimation, registration and 3D tracking to an interactive User Interface under the scene graph concept. The 3D overlays are shown to the end user i) on top of an image target ii) on real-world planar surfaces and iii) at predefined points of interest (POI). The evaluation in terms of performance, rendering efficacy and responsiveness is made through various testing strategies: system and trace logs, profiling and 'end-to-end" tests. The final benchmarking elucidates the slow and computationally intensive procedures induced by the big data rendering and optimization patterns are proposed to mitigate the performance impact to the non-native technologies.

1. INTRODUCTION

Immersive computing exploits machine learning, sensors technology and computer vision techniques to affect and alter the intuitive perception and cognition. Virtual Reality (VR), Augmented Reality (AR) and Mixed Reality (MR) represent scalable immersion adaptations of the real and virtual world. Each technology enhances user's immediate context through digital data in a divergent way. From the computer-generated 3D simulations and artificial senses of VR, more responsive to physical space experiences are attained by the latter technologies. Digital content is superimposed in the dynamic and ever-changing live view of the camera facilitating knowledge dissemination and emotional engagement. Especially when this content is interactive-driven and spatially aware, user transits to the state of MR and the served purpose is even more meaningful.

The delivery of these reality-based services and functions entails a powerful processor, a display system and sensors. The robustness and interoperability between these hardware components consolidate the involved AR software and determine the completeness, applicability and potential of the ultimate implementation. Through the integrated headsets and MR platforms like Microsoft Hololens, Magic Leap One, Glass Enterprise Edition 2, Epson Moverio and Vuzix Blade, the full potential over AR experiences is reached. However, their high price confines their use to industrial, commercial and enterprise sector. The lack of the idealized hardware form factor to consumer devices and the absence of standards, complicate the process of unifying, credible and accessible solutions for the public. Mobile Augmented Reality (MAR) eliminates these deficiencies with the handheld display and the built-in gear of smartphones or tablets. The characteristics of mobility, low-cost and ubiquity in everyday life along with their technical specifications constitute MAR platforms a promising alternative (Brondi et al., 2012).

In the context of mobile development environment, AR applications are driven by a dedicated Software Development Kit (SDK) on top of the device's operating system. The advent of ARCore and ARKit SDKs on Android and iOS platforms respectively introduces new and consistent functionalities. They handle the interaction between the internal hardware with the computer vision algorithms to extend the device's environmental awareness and responsiveness. The reality-based implementations distinguish in either location-sensitive or marker-based. The first operating principle leverages the transformation of data received through GPS indicators and other built-in sensors while the second one rests upon anchoring and real objects recognition in real-time view. The toolset that exposes and abstracts these underlying mechanisms is determined by the targeted platform for either native or cross-platform approaches.

An intriguing concept comprises the standard web technologies of front-end development with native performance. Such an AR prototype application is developed in this paper. Leveraging open source frameworks and rendering engines a holistic approach to MAR is built with modular and reusable code. It aims to fuse large-scale and high-resolution 3D models with the real world within an interactive User Interface (UI). Their seamless integration to the real environment supplies in-situ visualizations with practical use cases for big data and photogrammetric outputs. Both marker-based and mark-less augmentation is implemented within three different mechanisms: image recognition, surface detection and geospatial localization. The high-level programming and the exploitation of Inertial Measurement Unit (IMU) are delegated to ViroReact platform, a React Native component while the world-scale location services are provided by Mapbox SDK and geolocation modules. The evaluation lies in the compliance with specific criteria: visual quality, response times and resources usage. Among the addressed issues, two are the main directions of the current work:

- Low-cost and versatile AR schema based on JavaScript and React
- Accurate registration and qualitive visualization of photorealistic 3D overlays.

The rest of the paper is structured as follows: in Section 2 a literature review is presented regarding the aforementioned directions and relative endeavors in the field of MAR. Section 3 analyzes the workflow, the system's structure and the data management of the application. The setup, the design and the utilities that it serves are described in Section 4 and finally, Section 5 evaluates the performance of the proposed convergent strategy of AR and 3D detailed content.

2. RELATED WORK

Recent years have witnessed significant progress and adaptability of Augmented Reality technologies on smartphones/tablets and a great range of applications from academic research to healthcare (Vávra et al., 2017), industry (Fraga-Lamas et al., 2018) and education (Challenor and Ma, 2019). Their operation mode is principally specified by the platform and the system's architecture. Initially, an augmented experience can be accessed by the device's browser. Mobile web AR deploys the sufficient computational resources of a server and the GPU-accelerated rendering of WebGL API to accommodate its services to user's devices (Qiao et al., 2019). A vital constraint is the network dependency and its essential connectivity for data streaming. The self-contained methods run locally and distinguish in native and cross-platform development. Native applications are based on the programming language required by the platform they are built for: Objective-C or Swift for iOS and Java or Kotlin for Android. Their cross-platform counterparts further involve bridging systems like React Native, Flutter, Apache Cordova and its distribution Adobe PhoneGap, Ionic, NativeScript and Xamarin that absolve development from platform-specific code (Mesfin et al., 2014). Among the aforementioned, emphasis is given to open-source and JavaScript-based solutions that relies on API bindings to access device's IMU, data and network status. React Native and the WebView-based Apache Cordova represent the most popular hybrid and interpreted approaches respectively (Biørn-Hansen and Ghinea, 2018). Unlike the latter, the multi-threading technology and the way its components are linked to native UI views endorse React Native with optimized refresh rates, close to 60 fps (Kämäräinen et al., 2016). Cardoso et al. (2018) test the applicability of six MAR frameworks in vision-based approach for an outdoor archaeological site and the three prevailing formulate a multi-platform application in the context of Cordova. ViroReact, a React Native's integration, is exploited for AR utilities based on physical markers and geographic positioning of points of interest (Feierherd et al., 2018). However, research on ViroReact's adoption in the AR landscape is limited. In accordance with the criteria of free use, full-fledged and Android compatibility, ARCore, ARToolKit and EasyAR are the Software Development Kits (SDKs) that can underpin the desired functionality within the AR application. Several comparative analyses and usability studies are encountered in the literature: review and comparison of ARCore and ARKit (Nowacki et al., 2020) and assessment of eleven development frameworks in terms of recognition and tracking for educational purposes (Herpich et al., 2017).

Typically, the overlaid information into the real scene may comprise text, images, animation and video. While every data type is subject to a certain management strategy, the potential to blend the third dimension imposes further constrains. Most endeavors orientate to 3D graphics, CAD (Palma et al., 2019), wireframe or other forms of simplified and lightweight overlays. Physical objects are enriched with heterogeneous or out of people's insight aspects like virtual reconstructions (Boboc et al., 2019), historical recreations (Panou et al., 2018), supporting simulations, etc. However, a noteworthy number of implementations align towards current advances in photogrammetry and computer vision with realistic, geometrically accurate and detailed 3D models. A UNESCO monument's model, generated by the Structure from Motion (SfM) technique, is superimposed using ARCore and Tango platform and the evaluation held by the end users showcases the profound impact of reviving Cultural Heritage assets (Voinea et al., 2019). A more extensive topographic and photogrammetric survey yields photorealistic 3D models for on-site and off-site AR (Canciani et al., 2016) while an analogous digitized statue composes a realistic AR scene with proper illumination and occlusion (Carrión-Ruiz et al., 2019). On a more general note, the practical challenges that arise from the synergy of big data and AR are discussed extensively by recent studies. From perceptual ambiguities (Bermejo et al., 2017) to tracking, registering and rendering limitations (Olshannikova et al., 2015) large sets of data dictate specific methodological and technical considerations.

3. METHODOLOGY

The operation mode of the proposed application integrates a computing and a 3D rendering procedure. Each one adheres to a different workflow; the setup of the software frameworks and the management of the visualized data, respectively.

3.1 System's architecture

The chosen of the system's architecture depends entirely upon the low-cost factor and the need to handle large 3D models and to conform to their precise requirements. In the context of web development, JavaScript and its library React.js endorse 3D visualizations with improved loading, real-time interactivity and highly dynamic UI. The optimal way to integrate conceptually similar capabilities to mobile devices and bundle them with augmentation is provided by the open-source frameworks React Native and ViroReact, respectively. Besides handling the scene-graph and the rendering, the AR system needs on-the-fly information retrieval and access to low-level procedures. ARCore SDK interfaces with these native mobile controllers and the code is rendered directly, fast and efficiently. The bidirectional communication of the application's state with the hardware is established by a bridge, a package that wraps Java native threads in a JavaScript module via JSON and AMQP protocol. For the global pose estimation, in which the device accurately figures out world position, a location data platform is employed. Mapbox is a mapping service that provides building blocks for position-based features and open source libraries for interactivity and control (Figure 1). The proposed system is compatible with every Android device that supports ARCore SDK and Android 7.0 Nougat release or later.

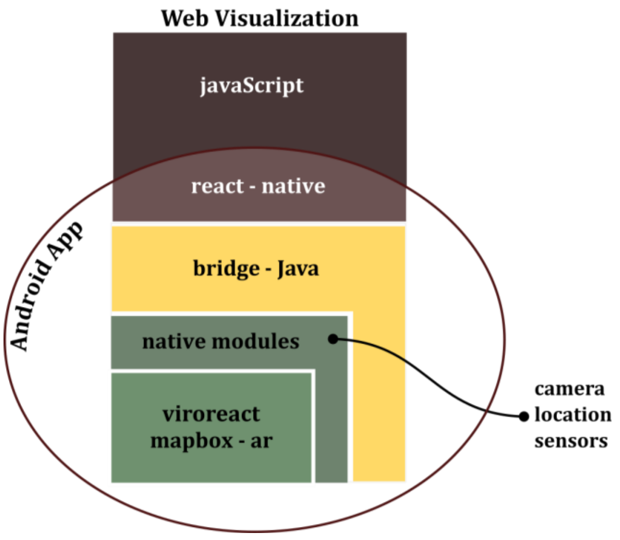


Figure 1. Architecture of the Augmented Reality application

3.2 High-resolution 3D overlays

The projection of a 3D photorealistic object on top of the real world entails misconceptions, information overload or perplexity. Simplicity, aesthetic appeal and clarity in visualization elicit correct responses to the created scene. The developed MAR prototype is pivoted on a relevant case study.
The 3D overlays are part of the geometric documentation of the archaeological site of Meteora, Greece, a UNESCO Cultural Heritage site. They have been generated using image-based photogrammetric techniques and computer vision algorithms, constituting high-quality and photorealistic 3D modeling products. Two meshes from the dataset are used to evaluate the functionality and performance of the proposed application (Figure 2). A post-processing phase is carried out in the software Geomagic Wrap 2017 for simplification and refinement. The number of poly-faces of the meshes is decreased by merging poly-vertices and equally, preserving the geometric features shapes (Table 1). This method results in the reduction of the meshes' size by selecting a specific ratio and controlling the targeted face count. The original shape in the following-up modeling processes is maintained and eventually only the noisy faces are removed. The workflow aims to maintain an adequate proximity level of display resolution and equally to remove areas with large polygons' concentration. Normal mapping is applied to textures for effectiveness in render time and surface detail enhancement.

	3D Model (a)	3D Model (b)
Geometry	700K faces, 360K vertices	420K faces, 190K vertices
RGB	Texture mapping	Texture mapping
Format	OBJ /MTL	OBJ /MTL
Size	81 MB	45 MB

Table 1. Characteristics of the 3D overlays

3.3 AR schema and data structure

The integral parts of MAR technology involve pose and position tracking and the system that estimates them precisely

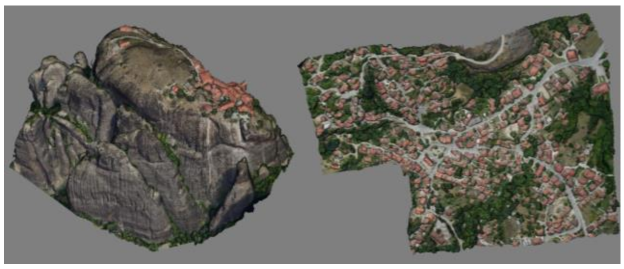

(a) (b)

Figure 2. Input 3D models of the AR application

in the presented prototype is ARCore SDK. Initially, it identifies image features using motion tracking by Visual-Inertial Odometry (VIO). The combination of the recognizable features with the information of device's orientation and acceleration keeps tracking updated. The algorithm is further exploited to determine the device's 6 degrees of freedom and estimate its pose. ViroReact is the rendering engine and development platform that exposes these functionalities to support surface and feature detection, full 3D rendering (lighting, surface texturing, etc.), anchoring and real world effects. It inherits React Native's declarative paradigm and the scene graph concept. Top-level components such as ViroARSceneNavigator work alongside with React Native to display native UI elements, respond to user interaction and make scenes dynamic. All the content that ViroReact renders in AR is defined by the ViroScene logical container. Specifically, the manipulation and loading of the 3D models is assigned to the Viro3DObejct component that resides in the scene graph. The OBJ files are loaded directly by setting the corresponding materials and the attributes of transformation in the AR scene.

The loading of the 3D models is performed asynchronously to avert rendering lag and consequent frame drops. Even if ViroReact does not limit the number of object' faces, the framerate may be dictated by the scene's complexity. The potential of degraded performance is enhanced concerning the heavy loads of data required to render to the end user. Another issue arises from the storage, the memory consumption and device's RAM usage. In order to control data flow, shared components and frequent updates the Redux state container is exploited. Events are declared as "actions", their storage is centralized and re-rendering is triggered only on demand. A caching strategy to speed up loading is provided by the library Redux Persist. It saves the Redux state object to persisted local storage and on application launch it dispatches it back to Redux.

4. IMPLEMENTATION

4.1 Initial configuration

The development is conducted on Samsung Galaxy Tab S4, an ARCore supported model with Android 9 Pie. The implementation runs under a packager server using ngrok tool and the testbed feature of Viro Media application. JavaScript's code is executed in Node.js run-time environment. The subsequent phases of geo-location and application's evaluation impose source's compilation and packaging on Android Studio IDE. All the tasks are performed on release/production builds in order to mitigate bugs and errors that arise from packages version incompatibility. The setup of the AR scene along with its properties and callbacks is defined in the entry point of

ViroARSceneNavigator. In the specific case three independent states are set to define the three AR scenarios and the flow functions that update these states. Redux sets also an initial state as a homepage. The User Interface of homepage determines the type of experience to launch in (Figure 3). On each experience launch its persisted state is retrieved by Redux Persist and saved back to Redux management layer.

4.2 AR implementation mechanisms

Each AR scene that corresponds to a different state entails a specific interaction pattern and a customized selection of modules and libraries. The first two are subsumed under the concept of anchors detection while the third one handles geo-location. Moreover, virtual buttons are created to convey an accessible role to users enabling transition from one scene to another. The buttons' icons have a click handler that invokes "jumping" (jump method) to the next or previous scene on the navigation stack.

4.2.1 Marker-based: The first approach of overlapping the 3D models to the real-world is to listen for visual markers detected by the AR system. The marker-based scene lies on image recognition in order to trigger where to position the 3D assets. The target used is a distinguishable logo that can be easily perceived by the camera. Its properties are specified in the ViroARTrackingTargets component. Then, detection, rendering and tracking are delegated to ViroARImageMarker component. It comprises the scene graph where the overlay's structure along with the parameters of orientation, scale and position are defined. Every time the reference image is encountered a callback is received in which the 3D model is attached (Figure 4).

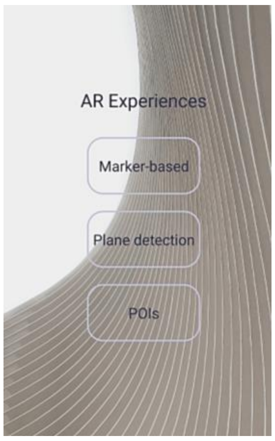


Figure 3. Homepage

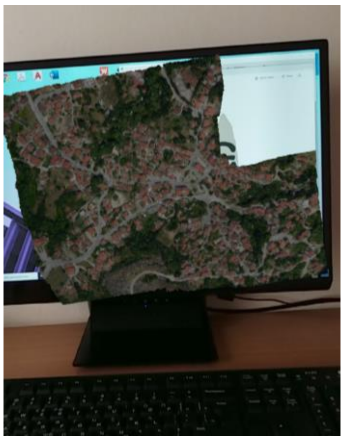
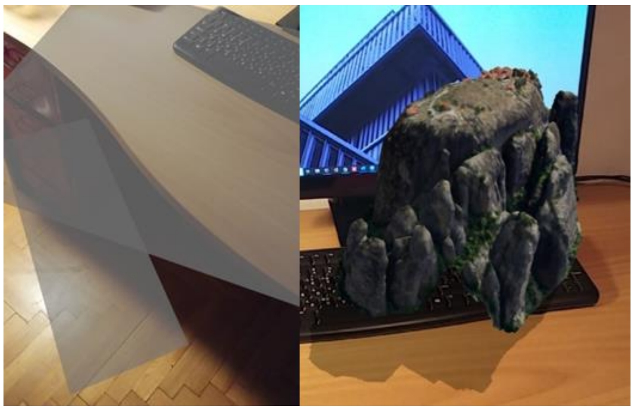


Figure 4. 3D model anchored to image marker

4.2.2 Plane Selection: The second scene displays the 3D model on top of a plane of the real-world, selected by the user. Planes are flat shaded meshes, reconstructed by sets of features points. Picking lies in the fact that a point on the 2D screen corresponds to a 3D ray of the real space. The point of touch in two dimensions is projected into the scene by casting a ray. The technique checks the virtual ray by collision detection. In Viro, the ViroARPlane component divulges the knowledge of the real-world surfaces. These functionalities can be wrapped to ViroARPlaneSelector to allow real-time interaction. Upon the mechanism activation, the AR system detects at runtime physical surfaces and highlights them in a semi-transparent grey color (Figure 5a). The experience starts when user selects an AR surface that represents an attachment point. At the given pose in the world coordinate space the 3D model is displayed with fixed position (Figure 5b). Then, the user can interact with the object by dragging it across the plane. Besides lighting, shadows are generated to add realism and depict model's volume.



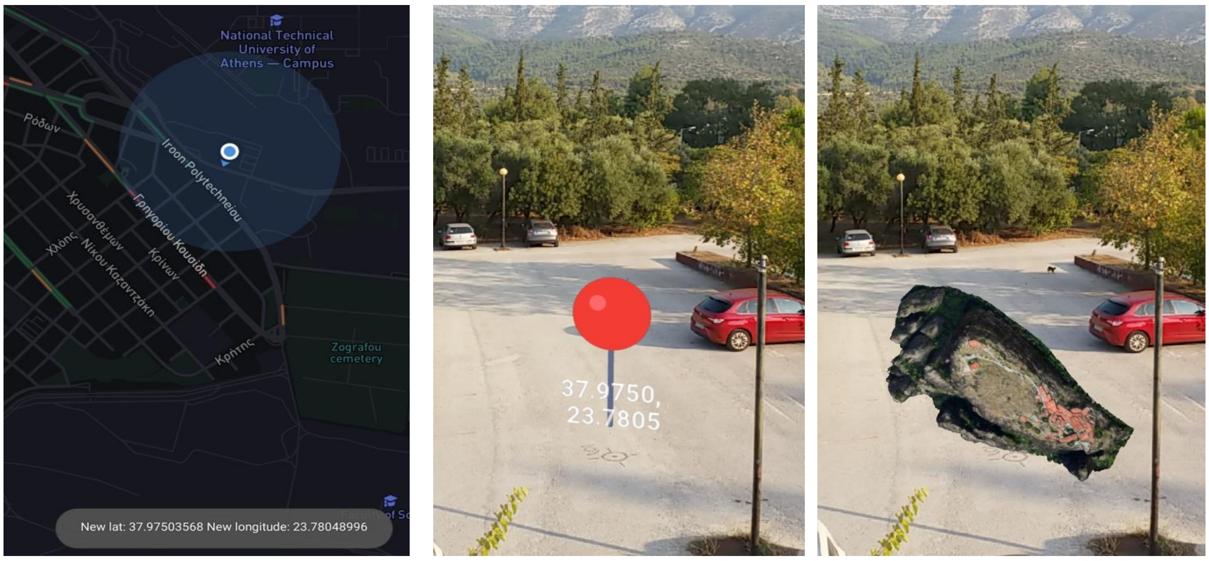
(a) (b)

Figure 5. The plane selection scene: (a) semi-transparent surfaces are displayed to indicate the real ones (b) 3D model anchored to the selected plane

4.2.3 Geospatial localization: The third implementation places the 3D model in a specific POI and bridges geo-location with AR. In order to position the overlay, a conversion from the Geographic coordinate system to the AR space is needed. The AR space is the reference system of the device that is centered at the user's initial position when the application launches. For the conversion, mathematical packages and modules like mercator projection and geolocation are used. The mapping resources, including the latitude and longitude of the current position, are delivered by the integrated Mapbox module. Initially, the Geographic coordinates are projected to the Web Mercator system. The projected coordinates are translated by the device's initial position that is also transformed to the same 2D projection by the Geolocation module. The output is a vector that corresponds to the distance from the 3D overlay to the device. Similarly, the Viro coordinates are converted from device's compass direction to device's orientation in AR space. This conversion involves a rotation by the true north's angle that is obtained by IMU data and inertial tracking. The activation of the third AR experience initializes a Mapbox's map that displays and tracks user's location (Figure 6a). The targeted POI is indicated by a pin. The camera is enabled when the pin is selected and the AR experience when user approaches the location. An image of a "pin" icon and a string with the spherical coordinates are drawn to the real scene (Figure 6b). When loading is completed the 3D model is displayed at the specified global position (Figure 6c).

5. EVALUATION

The evaluation of the developed application concerns the performance of React Native's and ViroReact's processing phases and delves into their most costly operations. Further tests are executed to report on its overall efficiency regarding CPU activity, loading times and energy usage of the end user's device. The Android device that hosts the prototype is Octa-core (4×2.35GHz for performance and 4×1.9 GHz for



(a) (b) (c)

Figure 6. Outdoors location-based approach: (a) The Mapbox's map detects and tracks user location by GPS (b) On the reference POI ViroImage and ViroText components are superimposed until model's loading (c) 3D overlay at world's fixed position

efficiency) with 4 GB of RAM and 7,300 mAh battery. React Native's profiler, Jest test runner and Perfetto's trace processing are used for the aforementioned analysis. A critical performance indicator lies on the rendering ability and responsiveness the application demonstrates in each AR implementation mechanism. React's built-in monitor tracks and displays issues like frame drops and stuttering and provides the UI and JS frame rates of the main and JavaScript thread respectively (Table 2). UI thread is used for native rendering involving layout parameters and draw processes while JS thread for JavaScript's code execution. The Android device displays 60 frames per second (fps) and the application is working close to this boundary. Good performance is maintained when the JS thread sends batched updates to UI thread in less than 16.67 ms, which corresponds to the next frame rendering deadline. The unresponsiveness for a frame is considered as a dropped frame. Validation of results is achieved through five sets of measurements and real device interaction. The tool is enabled by accessing every scene from the Homepage to ensure that no resources are shared. Each table cells shows two average values: the first one concerning the scenario's initialization and the second one related to the end of the 3D rendering (display of 3D model).

	Marker-based		Plane detection		POIs	
Frame drops	2	18	6	21	20	**57**
Stutters	2	3	2	2	2	2
JS (fps)	54.1	58	46.1	51.6	54,2	**48.5**
UI (fps)	60	58	60	55	59	**50**

Table 2. Low-level performance metrics recorded by the built-in performance monitor: the first column corresponds to the metrics on scenario initialization and the second one to the same metrics after 3D models' drawn time

It can be observed that the application successfully manages to deliver an average UI rate of 60 fps but drops frames during rendering. The initialization of "Plane detection" results in 6 frames being dropped with 100 ms time elapsed. Accordingly, the 3D model visualization causes 21 frame drops. These drops are normal considering that the re-rendering imposed by state calls or navigation is computationally expensive. The end-user will not feel the stutter. However, the performance of POIs scenario suffers greatly when the 3D object is displayed, as it is denoted by the bold values in Table 2. The 57 frame drops and the definitive delay of 3.5 s are perceived and any animations appear to freeze during that time. The duration of loading and rendering phases while the end-user waits before seeing the virtual scene is indistinct. Also, non-specific are the reasons of the latency until the 3D visualization happens. These observations stimulate further benchmarking to determine the CPU usage in the JS thread and the GPU load. Bottlenecks and slow procedures are defined accurately by profiled code blocks. The profiling is piloted to the POIs implementation and handled by two open-source project for performance instrumentation, Perfetto and Systrace. A trace involving the specific rendering phase is collected and analyzed. While the JS thread is executing close to the frame boundaries, the UI and Render threads run almost all the time. Part of render thread's operations is shown in Figure 7. Long periods of time are spent in DrawFrame process that exceeds the frame boundaries. During this time GPU drains its command buffer from the previous frame. This latency indicates an increased per-frame load on the GPU and problem lying in the native views being rendered. Figure 8 illustrates precisely the implication of time-consuming or expensive processes to CPU. Most performance issues are emerged after 3, 5 and 8 seconds from launch. At this time, Mapbox's map, GPS tracking and 3D model's display are enabled respectively. It is clear that the main activity, namely the large-sized and complex meshes, imposes high CPU usage, extra load consumption on the GPU and stutters on loading.

For accurate time values, React's performance tools are used to record these long running processes between rendering, states update and modules' loading. Initially, this method records each AR scenario as an independent session with exactly the same 3D assets. After benchmarking common components, an end-to-end test is carried out to examine navigation's transitions. Rendering the 3D model's view takes up to 2.27 s and 2.56 s in

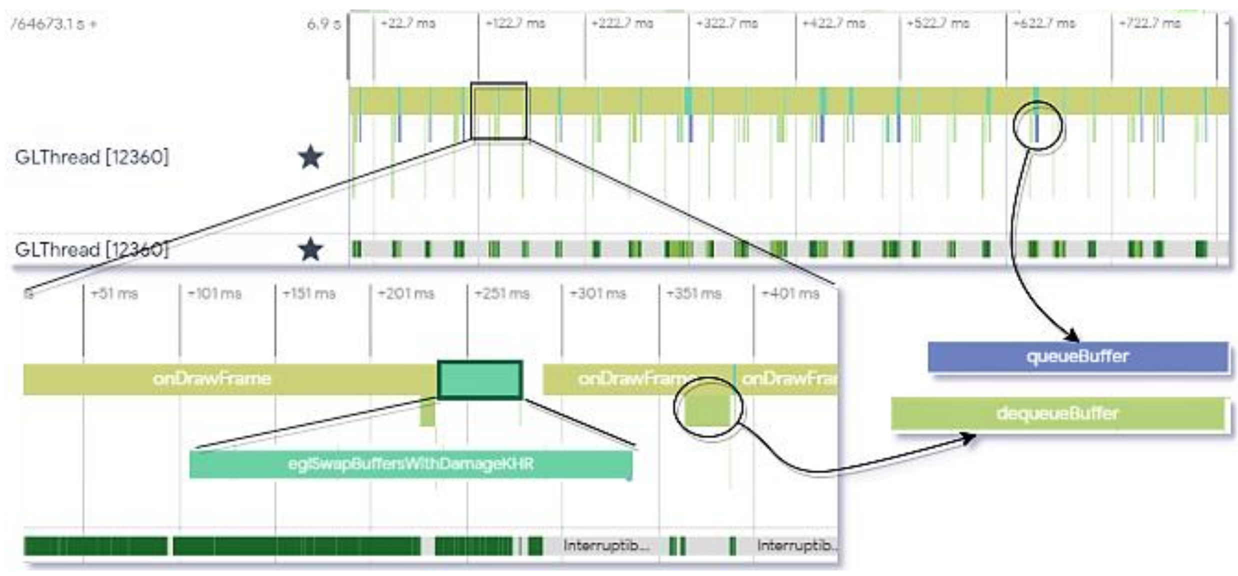


Figure 7. GLThread's process that represents the OpenGL commands used to draw the UI thread. Crossing frame boundaries and long onDrawFrame procedures can be noticed. The SwapBuffersWithDamageKHR represents the content that must be recomposed.

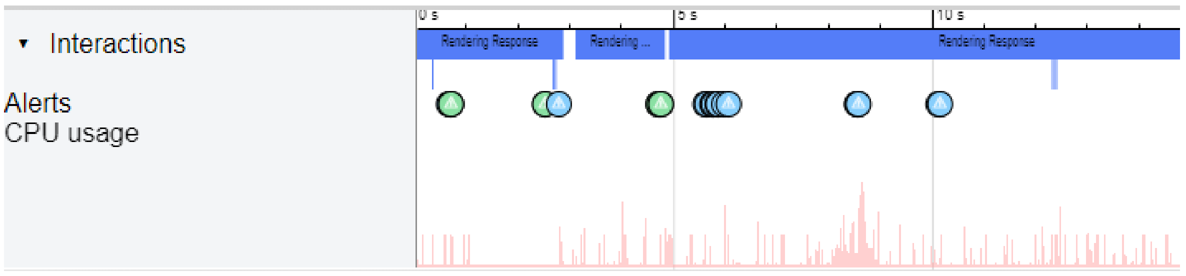


Figure 8. Alert states that indicate performance issues while rendering the relevant frames, provided by Android's Systrace analysis

the first two scenarios and 4.28 s to the third. Regarding the stack navigation measurements, Viro React is fast enough to be considered as instantaneous with an average of 400 ms transition animation. Moreover, cache by Redux Persist reduces interruptions during loading.

A second test measures the intuitive wasted rendering cycles and their re-rendering implications to performance. Overall, 23 ms, 14 ms and 66 ms are spent during each scenario rendering respectively on components with no actual engagement on rendering and DOM changing. Inner loops and nested views' instantiation should be minimized to optimize timing and CPU performance. One more test is handled by the Android Studio regarding the overall prototype's performance. It inspects energy use and battery resources of the device while application is running (Figure 9). Battery life drain is rather marginal as the discharge rate is sufficient considering the AR tasks. By tracing method executions and hardware utilization it seems that rendering on JS thread and GPS's activation account for increased battery consumption. The total size of the application is 118, 92 MB with 99,2 MB memory consumption.

```
CPU usage from 0ms to 10013ms ago:
 13% 101/system_viroserver: 6.8% user + 5.5% kernel / faults:
563 minor
 Battery Level: between 73 and 65
Discharge rate: 4.793 % / hour (217,49 mA)
```

Figure 9. Power consumption while application is running for 10 minutes

Finally, each implementation mechanism is subject to specific difficulties in computing and user experience. Specifically, the marker-based tracking is sensitive to occlusion while the planes' tracking entails slow device movements by the user in order ARCore detects the real surfaces with accuracy. The monitoring of global position and the identification of spatial coordinates are dynamic and ever-changing tasks that dictate accuracy and speed. The testing device supports up to five satellite networks and its dedicated GPS receiver has an average accuracy of 4 meters. Therefore, the geo-location accuracy that corresponds to the confidence level of GPS receiver is 4 meters. The accuracy attained in the current implementation is 7 meters. Factors that cause uncertainty like clock drift and residual geo-location error have not been examined at the current phase.

6. CONCLUSIONS

The presented AR system's outreach is twofold: qualitative visualization of photogrammetric 3D overlays and optimal monitoring of native services and modules by cross-platform JavaScript tools. The realized prototype proves that a low-cost AR workflow with open-source components can serve the most of use cases including vision-based and hybrid tracking methods. The amount of customization is high. The adaptive to various situations display can be enriched with interactive and responsive to user's needs features and future research could delve into Viro's features detection.

Performance analysis shows that ViroReact succeeds in delivering a good frame latency distribution and maintaining the frame rate close to 60 fps. However, overlaying big and complex data is a memory-intensive task that imposes pressure to processor core and GPU. Currently, GPU wastes resources to render faces in areas that are not displayed at the current view. Instead of loading a single chunk on the order of hundreds of megabytes, various Levels Of Detail (LOD) for each model could be built. Performance and visual quality could be optimized by loading the appropriate version for the appropriate detail level on subsequent requests. Moreover, reducing unnecessary processes and data that the application does not need at the current time optimizes initial loading time and rendering lifecycle. The integration of Redux Persist for centralized storage prevents unnecessary re-renders and facilitates testing as the UI thread and data management are separated. Finally, for the optimal development strategy is suggested that resource intensive computations can be performed in native code, and then only the native part of the application has to be written individually for each platform.

ACKNOWLEDGEMENTS

This research has been co-financed by the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH – CREATE – INNOVATE (project code: T1EΔK-02859).

REFERENCES

Bermejo, C., Huang, Z., Braud, T., Hui, P., 2017. When Augmented Reality meets Big Data, 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW). IEEE, Atlanta, GA, USA, pp. 169–174. doi.org/10.1109/ICDCSW.2017.62

Biørn-Hansen, A., Ghinea, G., 2018. Bridging the Gap: Investigating Device-Feature Exposure in Cross-Platform Development. Hawaii International Conference on System Sciences. doi.org/10.24251/HICSS.2018.716

Boboc, R., Duguleană, M., Voinea, G.-D., Postelnicu, C.-C., Popovici, D.-M., Carrozzino, M., 2019. Mobile Augmented Reality for Cultural Heritage: Following the Footsteps of Ovid among Different Locations in Europe. *Sustainability* 11, 1167. doi.org/10.3390/su11041167

Brondi, R., Carrozzino, M., Tecchia, F., 2012. Mobile Augmented Reality for cultural dissemination. ECLAP 2012 Conference on Information Technologies for Performing Arts, Media Access and Entertainment, Firenze, pp. 113-117.

Canciani, M., Conigliaro, E., Del Grasso, M., Papalini, P., Saccone, M., 2016. 3D survey and augmented reality for cultural heritage - The case study of Aurelian wall at Castra Praetoria in Rome. *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci*. XLI-B5, pp. 931–937. doi.org/10.5194/isprsarchives-XLI-B5-931-2016

Cardoso, J.C.S., Belo, A., 2018. Evaluation of Multi-Platform Mobile AR Frameworks for Roman Mosaic Augmentation. Eurographics Workshop on Graphics and Cultural Heritage, 10 p. doi.org/10.2312/GCH.20181348

Carrión-Ruiz, B., Blanco-Pons, S., Weigert, A., Fai, S., Lerma, J.L., 2019. Merging Photogrammetry and Augmented Reality: The Canadian Library of Parliament. *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.*, XLII-2/W11, pp. 367-371. doi.org/10.5194/isprs-archives-XLII-2-W11-367-2019

Challenor, J., Ma, M., 2019. A Review of Augmented Reality Applications for History Education and Heritage Visualisation. *MTI* 3(39). doi.org/10.3390/mti3020039

Feierherd, G., Viera, L., González, F., Huertas, F., Delía, L., Depetris, B., 2018. Value enhancement of the Artistic Heritage of Tierra del Fuego using Augmented Reality. XXIV Congreso Argentino de Ciencias de la Computación, La Plata, pp. 1160-1169.
Fraga-Lamas, P., Fernandez-Carames, T.M., Blanco-Novoa, O., Vilar-Montesinos, M.A., 2018. A Review on Industrial Augmented Reality Systems for the Industry 4.0 Shipyard. IEEE Access 6, pp. 13358–13375. doi.org/10.1109/ACCESS.2018.2808326

Herpich, F., Guarese, R.L.M., Tarouco, L.M.R., 2017. A Comparative Analysis of Augmented Reality Frameworks Aimed at the Development of Educational Applications. CE 08, pp. 1433–1451. doi.org/10.4236/ce.2017.89101

Kämäräinen, T., Siekkinen, M., Ylä-Jääski, A., Zhang, W., Hui, P., 2016. Dissecting the End-to-end Latency of Interactive Mobile Video Applications. Proceedings of the 18th International Workshop on Mobile Computing Systems and Applications. ACM, NY, USA, pp. 61–66. doi.org/10.1145/3032970.3032985

Mesfin, G., Ghinea, G., Midekso, D., Grønli, T.-M., 2014. Evaluating Usability of Cross-Platform Smartphone Applications, in: Awan, I., Younas, M., Franch, X., Quer, C. (Eds.), *Mobile Web Information Systems*. Springer International Publishing, Cham, pp. 248–260. doi.org/10.1007/978-3-319-10359-4_20

Nowacki, P., Woda, M., 2020. Capabilities of ARCore and ARKit Platforms for AR/VR Applications, in: Zamojski, W., Mazurkiewicz, J., Sugier, J., Walkowiak, T., Kacprzyk, J. (Eds.), *Engineering in Dependability of Computer Systems and Networks*. Springer International Publishing, Cham, pp. 358–370. doi.org/10.1007/978-3-030-19501-4_36

Olshannikova, E., Ometov, A., Koucheryavy, Y., Olsson, T., 2015. Visualizing Big Data with augmented and virtual reality: challenges and research agenda. *Journal of Big Data*, 2(22). doi.org/10.1186/s40537-015-0031-2

Palma, V., Spallone, R., Vitali, M., 2019. Augmented Turin baroque atria: AR experiences for enhancing cultural heritage. *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci*. XLII-2/W9, pp. 557–564. doi.org/10.5194/isprs-archives-XLII-2-W9-557-2019

Panou, C., Ragia, L., Dimelli, D., Mania, K., 2018. An Architecture for Mobile Outdoors Augmented Reality for Cultural Heritage. *IJGI* 7, 463. doi.org/10.3390/ijgi7120463

Qiao, X., Ren, P., Dustdar, S., Liu, L., Ma, H., Chen, J., 2019. Web AR: A Promising Future for Mobile Augmented Reality—

State of the Art, Challenges, and Insights. Proc. IEEE 107, pp. 651–666. doi.org/10.1109/JPROC.2019.2895105

Vávra, P., Roman, J., Zonča, P., Ihnát, P., Němec, M., Kumar, J., Habib, N., El-Gendi, A., 2017. Recent Development of Augmented Reality in Surgery: A Review. *Journal of Healthcare Engineering*, pp. 1–9. doi.org/10.1155/2017/4574172

Voinea, G.-D., Girbacia, F., Postelnicu, C.C., Marto, A., 2019. Exploring Cultural Heritage Using Augmented Reality Through Google's Project Tango and ARCore, in: Duguleană, M., Carrozzino, M., Gams, M., Tanea, I. (Eds.), *VR Technologies in Cultural Heritage*. Springer International Publishing, Cham, pp. 93–106. doi.org/10.1007/978-3-030-05819-7_8