

Universal Web-Based Tracking for Augmented Reality Applications

Yannic Bonenberger, Jason Rambach, Alain Pagani, and Didier Stricker

German Research Center for Artificial Intelligence (DFKI)
Kaiserslautern, Germany
`Jason.Rambach@dfki.de`

Abstract. Augmented Reality (AR) is a growing technology which begins to reach its maturity and address a broad spectrum of areas. However, current augmented reality applications still tend to be confined to a single use case or a single set of devices. In this paper, we explore web-based augmented reality systems using a single cross-platform binary to address a wide range of devices, which can dramatically decrease the developmental effort to create applications and therefore help to satisfy the growing demand for them. To this extent, we discuss the implementation of a feature tracking system using WebAssembly and evaluate its real-time capabilities on a wide range of devices and operating systems. Additionally, we also demonstrate a simple AR application making use of our tracker.

Keywords: Augmented Reality · Web-based · 6DoF Pose Tracking

1 Introduction

Augmented Reality (AR) is an emerging technology combining virtual scenes with the real world [5]. With the progress in mobile computing devices like smartphones or tablets in recent years, there is a growing demand in AR applications in various consumer-oriented fields like entertainment or education, and also in other areas like industrial construction and maintenance or medicine and rehabilitation [11, 8, 26, 10].

The main enabling technology for an AR system is a 6 Degree of Freedom (6DoF) pose estimation and tracking system. Knowledge of the camera pose allows for correct placement of virtual augmentations in the real world [23]. Model-based tracking approaches use a predefined target model for localization [25] while Simultaneous Localization and Mapping (SLAM)-based approaches operate without prior knowledge of the environment meaning that it also needs to be uncovered in parallel to the localization [20]. An advantage of model-based systems is the reduced complexity and the ability to create AR content that is specific to the tracked model, while SLAM systems can provide localization out-of-the-box without any user involvement.

There is a large number of devices like smartphones or tablets with at least one camera and sufficient computing power which are capable of running AR



Fig. 1. Our application running on a smartphone.

software. These devices have one major issue: They run many different operating systems and therefore have their own ecosystem, most of them not compatible with other ones. An example thereof is ARKit [9] which is developed by Apple and only supports the latest generation of iOS devices. Users with older devices will not be able to run applications using these features, even though lots of these devices have sufficient computation power to run them. Smartphones and tablets from other brands are also excluded from running these applications. For users with Android devices, there is ARCore [13] which only supports the latest generations of Android smartphones from vendors like Samsung or Google. It is even more complicated if developers want to bring their applications to the desktop. There is, however, one ecosystem that all of the mentioned devices support: And this is the WorldWideWeb. The WorldWideWeb was developed by Tim Berners-Lee and Robert Cailliau at CERN in 1990 [6, 7] and grew from a platform to access static HTML files to a platform with support for large applications which can do most of what native applications can [14].

A significant milestone was the recent addition of WebAssembly, “a binary instruction format for a stack-based virtual machine” [3, 17] which aims to execute at native speed on a wide range of platforms. It is available in all major browsers including Google Chrome, Microsoft Edge, Mozilla Firefox and Apple Safari [4]. With these technologies, it is possible to write fully functional AR applications which can replace native applications on the desktop and mobile devices. This has the potential of reducing the amount of work and therefore the costs to develop an augmented reality application, which is eventually necessary to satisfy the growing demand.

In this paper, we investigated whether it is possible to develop augmented reality applications using only technologies provided by the modern web platform without additional plugins or usage of particular browsers with built-in support for AR [22, 15, 16]. To demonstrate this, we developed a simple model based AR application using traditional web languages like HTML5 and ECMAScript as well as compiled languages like C, which could traditionally only be used to create native applications, and compile them to WebAssembly using Emscripten (Figure 1). To evaluate whether the performance of such applications is sufficient for use in AR, we compared the performance of our implementation if we compile to a native binary to the performance of our implementation if we compile to a WebAssembly binary.

2 Related Work

Building AR systems using web technologies has traditionally been very challenging because browsers did not provide sufficient speed to run these computationally heavy applications. To our knowledge, this is the first paper presenting a purely web-based approach running in standard browsers without additional plugins. However, some approaches for applications using a browser to display their interface have been presented. In [31], a prototype of an AR system with a web-based client was presented. To use this application, users use the front-end to capture images of the scene which are then uploaded to a server, processed, and the augmented result is sent back to the client. While this approach can provide high-quality augmentations, it has the disadvantage that only images can be processed and, due to possibly slow network connections, real-time processing is impossible.

Previous research also investigated whether it is possible to use browser plugins like Flash which have more computational power than ECMAScript for AR systems [29, 21]. While this approach is similar to ours, it has the disadvantage that it is not available in all browsers across all platforms and that users must install plugins before they can use the application.

Another highly interesting concept are browsers with built-in support for AR [22, 15, 16]. However, these approaches require that users install dedicated applications on their devices to be able to use such systems. To our knowledge, none of these applications are available for download and must be compiled from source which makes them unusable for arbitrary users. It is also very likely that these dedicated browsers will be abandoned once browsers add native support for AR [19].

Other research in the field of AR and the web investigated how web-based systems can be utilized to create content for native augmented reality applications [12, 30] or whether it is possible to embed web technologies into native AR systems to create interactive augmentations [18]. Researchers have also investigated how web-based AR systems can protect the privacy of their users [24].

3 Implementation

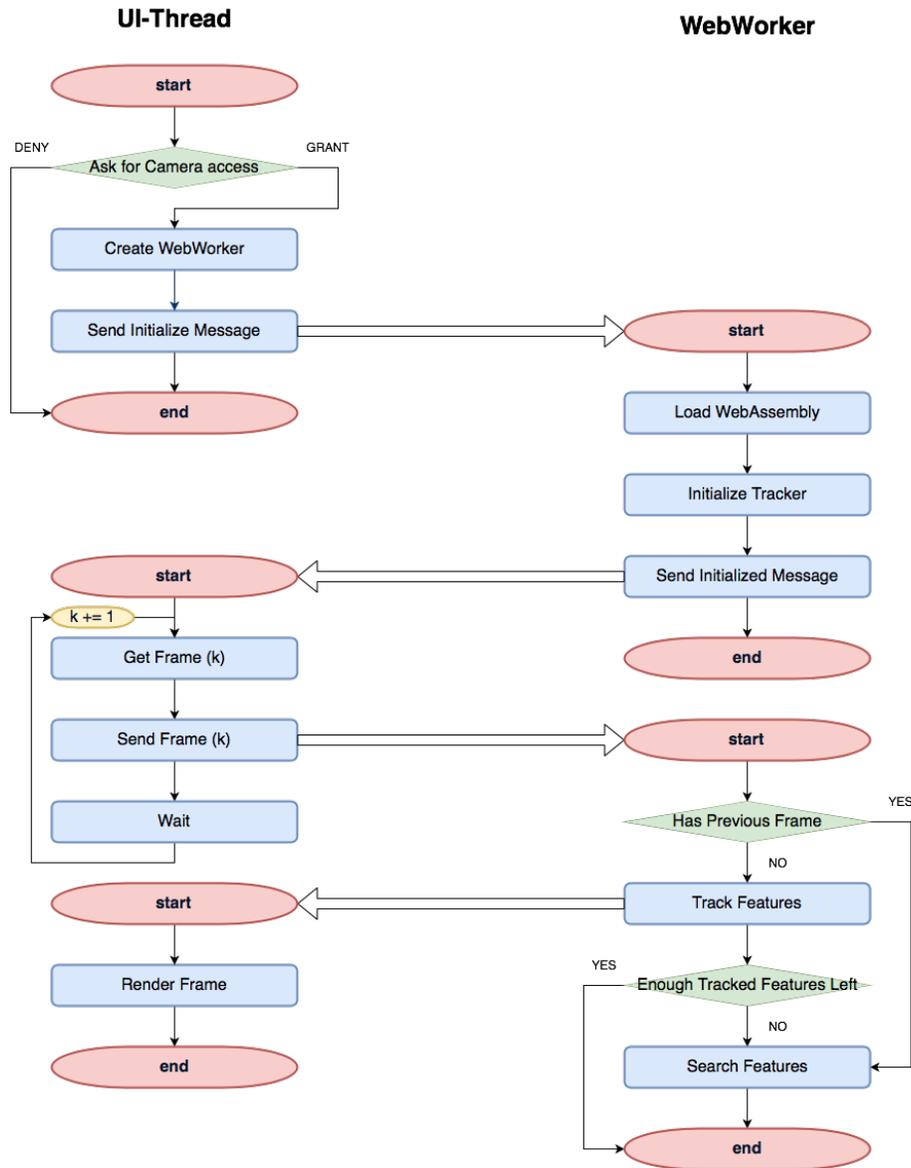


Fig. 2. Architecture of our System, split into UI-Thread (left) and Web-Worker (right).

In this section, we will discuss the architecture and implementation of our application and the difficulties we encountered. For our implementation we used

a combination of well-established web technologies like HTML5 or ECMAScript and technologies more recently added to the web platform like WebRTC, which provides access to the camera, or WebAssembly, which is a platform independent binary instruction format that aims to achieve native-like execution speed.

The content of this section is divided into these three parts: First, we will present the overall architecture of our system and discuss the challenges of building AR systems in the browser. Then, we will present the point tracking system we used in our application, and in the end, we will briefly discuss the details of our simple web-based AR system.

3.1 System Architecture

The main challenges to building web-based AR are tight constraints of computational power available in the browser and the unique concurrency model of ECMAScript. ECMAScripts concurrency is based on a queue-based model often called event-loop. This means that the runtime “contains a message queue which stores a list of messages to be processed and their associated callback functions. These messages are queued in response to external events (...) given a callback function has been provided” [27]. With the more recent addition of Promises, which are so-called microtasks and have entirely different semantics than regular tasks in the event-loop, ECMAScripts execution model got even more convoluted. To complicate things further, the main thread is also used by the browser to parse pages from HTML into a DOM tree or to calculate the layout of a website. In order to be able to implement AR systems on a platform using this concurrency model, we split our application into multiple independent subsystems which can run completely independently and executed them on two different threads: A UI-thread, which is also the default thread browsers use to execute code and render the page on, and a background thread we created using the WebWorker API [2]. As it can be seen in Figure 2, we reduced the number of computations on the UI-thread to a minimum and only executed what is directly related to the user-interface of our application. All computations related to tracking objects between frames are executed in a dedicated WebWorker which communicates with the other thread asynchronously.

After the browser finished downloading the initial website, we ask the user for their permission to use their camera. Since this is mandatory for our application, we do not proceed further if this permission is denied. When our application has the permission to use the camera, a dedicated WebWorker is created and the browser downloads and executes our separate worker script. Additionally, we immediately send an initialization message to the newly created worker which then downloads the external WebAssembly module. When the worker is fully initialized, it sends a message back to the UI-thread to indicate that it is ready to receive frames. When this message is received by the UI-thread, we start capturing frames from the camera and send them to the worker thread for processing. For convenience, we use a fixed frame rate, augment the message to the worker with the current timestamp and drop the frame in the worker if it is older than a predefined amount of time. All other frames are used to track

features and then send back to the UI-thread. If too many features are lost, or if there was no previous frame, the current frame is also used to search for new features. Every time the UI-thread receives a processed frame, the user interface is updated and the image is shown to the user. With this architecture, we were able to achieve sufficient computational power and have an interface which is responsive to interactions with the user.

3.2 Tracking

We decided to use the well known Kanade-Lucas-Tomasi (KLT) algorithm [28] for our tracker. We use KLT tracking to follow image patches between consecutive frames by performing a local search to minimize the photometric error between the patch in the previous image and its match in the current one (see Figure 3). Initial evaluations revealed that processing a single image is sufficiently fast and we do not need to split the algorithm into independent parts which can be executed asynchronously to use KLT in our applications. However, existing implementations such as the one from OpenCV could not be ported easily, thus we decided to make our own implementation. To be able to perform a fair evaluation of the performance of our system, we ensured that the implementation of our tracker does not make any assumptions about the architecture it is executed on.



Fig. 3. KLT feature tracker, from the first frame on the left to the last frame on the right.

3.3 Application

The architecture of our application is based on the tracker architecture presented in 2. As a marker for pose estimation, a 2D image rich in texture is used. Thus, the estimated pose is given with respect to a coordinate system defined by this marker. The use of an image with natural features allows to have a registration step where a user can select his own image to be used for tracking. The tracking application follows the principles of other systems such as [25]. A set of ORB features are stored along with their 3D positions in order to register the tracking target. To start the application, we initialize the camera and ask the user for

their permission. If the user grants us permission to use the camera, we periodically capture a frame, extract features from this frame and match them to registered ORB features of our marker. Once the number of matches is above a certain threshold, we compute a homography between the marker and the picture, project the boundaries of the marker into the frame, and filter outliers which are outside this projection, as it can be seen in Figure 4. Having estimated an initial pose once enough inliers are found, we use KLT to track them from frame to frame which ensures a fast update of the pose. The tracked pose can be used to render virtual augmentations on the marker. As an example, we draw a virtual cube over the marker as can be seen in Figure 4.

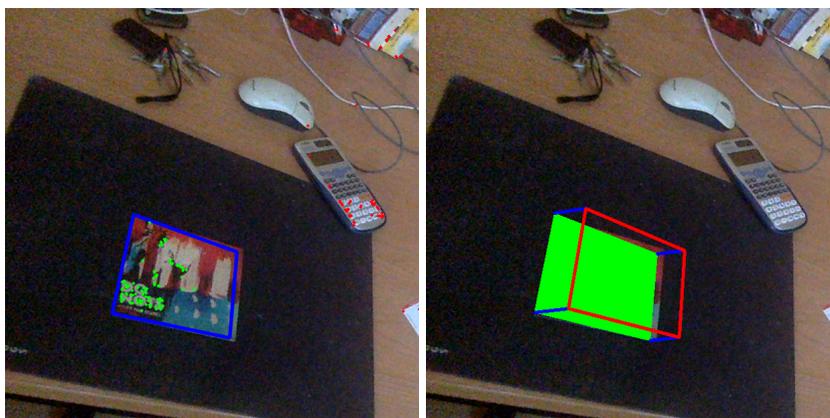


Fig. 4. On the left, projection of the boundaries of the marker into the scene and an example of a virtual cube rendered on top of our marker on the right.

4 Evaluation

In this section, we present and discuss the results of our runtime evaluation of the proposed KLT tracker. To get a comprehensive overview of the performance of our application, we executed our tests on various common devices and operating systems and across all popular browsers available on these platforms. Primarily, we used a MacBook Pro (Mid 2015) with an Intel Core i7 processor at 2.8 GHz and 16 GB RAM, running macOS 10.13, a desktop PC with an Intel Xeon E5 processor at 2.6 GHz running Windows 10 respectively Ubuntu 16.04, an iPhone X running iOS 11 and a Samsung Galaxy S8 running Android 7 to run the tests. We also tested our application on various other devices, including low-end mobile devices, which showed results comparable to the results we observed during our runtime evaluation.

The runtime performance was evaluated using a single test binary, once compiled to a native binary and once to a WebAssembly module. Both binaries were produced using the highest optimization level available. To account for differences in the architectures of the underlying platforms, we used a fixed set of test images which were inlined into the binary.

Analysis of the raw data we collected during our tests revealed that the time required to track features in the first frame is roughly double the time required to track features in all subsequent frames. Investigations revealed that this is caused by the fact that we compute the gradient image for both pictures for the first image, and only for one picture in all other images. Due to this, we decided to exclude the first image from our analysis.

Table 1. Runtime performance across different browsers and operating systems.

		 Chrome 66	 Edge 17	 Firefox 60	 Safari 11.1	 Node.JS 10
macOS	7.2 ms	13.6 ms	N/A	9.6 ms	10.1 ms	10.9 ms
Windows	9.5 ms	17.6 ms	14.1 ms	13.9 ms	N/A	15.8 ms
Ubuntu	8.0 ms	13.2 ms	N/A	13.1 ms	N/A	11.4 ms
iOS	N/A	12.4 ms	12.4 ms	12.3 ms	12.5 ms	N/A
Android	N/A	15.2 ms	N/A	14.9 ms	N/A	N/A

In Table 1, we present the average time in ms to perform the computations to track points from one image to another, which is the central part of our application. We observe that the runtime of the native binary is 36.9% lower than the runtime in browsers. However, the average time of 13.4 ms required for the tracking in the browser indicates that a frame rate of 74 fps can be achieved. This high frame rate leaves sufficient time for the rendering of quality augmentations and other application content.

Further investigations revealed that roughly 80% of the time we need to process a frame is spent computing the gradient image for the KLT. Given that we currently run these computations on the CPU, we expect that the frame rate will further increase when we can use OffscreenCanvas [1] to accelerate this on a GPU. It is also worth mentioning that getting precise timestamps in browsers requires several switches in the execution context and the operation is therefore slightly more expensive than its native equivalent. To mitigate the effects of the recently published processor exploits spectre and meltdown, browser vendors also reduced the maximum precision of timestamps in the browser. However, investigations of the impact of these changes revealed that we were still able to compute time differences with sub-millisecond precision. Given that our measurements showed a runtime that was a lot higher than that, we think that these changes did not affect our evaluation.

5 Conclusion

In this paper, we investigated the possibility to develop a universal cross-device, cross-browser based AR application using only web development tools. Our proposed system is built using WebAssembly, and implements a marker based KLT tracker for AR without using common computer vision libraries that are still unavailable or partially functional for Web Assembly. A runtime evaluation performed a selection of commonly used devices for AR, based on different operating systems proved the general feasibility of the approach. Future work includes dealing with the rendering of more complex virtual models for more demanding AR applications, and the improvement of tracking and user experience by further optimization, addition of camera intrinsics self-calibration and usage of 3D objects as tracking targets instead of 2D marker images.

Acknowledgments

This work has been partially funded by the Federal Ministry of Education and Research of the Federal Republic of Germany as part of the research projects PROWILAN and BeGreifen (Grant numbers 16KIS0243K and 16SV7525K).

References

1. Offscreencanvas (2018), <https://developer.mozilla.org/en-US/docs/Web/API/OffscreenCanvas/>
2. Web worker api (2018), https://developer.mozilla.org/de/docs/Web/API/Web_Workers_API/
3. Webassembly (2018), <https://webassembly.org>
4. Webassembly (2018), <https://developer.mozilla.org/en-US/docs/WebAssembly/>
5. Barfield, W.: Fundamentals of wearable computers and augmented reality. CRC Press (2015)
6. Berners-Lee, T., Cailliau, R.: Worldwideweb: Proposal for a hypertext project (11 1990), <https://www.w3.org/Proposal.html>
7. Berners-Lee, T., Fischetti, M.: Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor. DIANE Publishing Company (2001)
8. Billinghamurst, M., Clark, A., Lee, G., et al.: A survey of augmented reality. *Foundations and Trends in Human-Computer Interaction* **8**(2-3), 73–272 (2015)
9. Buerli, M., Misslinger, S.: Introducing ARKit-Augmented Reality for iOS. In: *Apple Worldwide Developers Conference (WWDC17)*. pp. 1–187 (2017)
10. Chen, L., Day, T., Tang, W., John, N.: Recent Developments and Future Challenges in Medical Mixed Reality. In: *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. pp. 123–135. IEEE (2017)
11. Dunleavy, M., Dede, C.: Augmented reality teaching and learning. In: *Handbook of research on educational communications and technology*, pp. 735–745. Springer (2014)
12. Feuerstack, S., de Oliveira, Á., dos Santos Anjo, M., Araujo, R.B., Pizzolato, E.B.: Model-based design of multimodal interaction for augmented reality web applications. In: *Proceedings of the 20th International Conference on 3D Web Technology*. pp. 259–267. ACM (2015)

13. Google: Arcore overview (2017), <https://developers.google.com/ar/discover/>
14. Google: Web fundamentals (2018), <https://developers.google.com/web/fundamentals/>
15. Google: Webaronarcore (2018), <https://github.com/google-ar/WebARonARCore/>
16. Google: Webaronarkit (2018), <https://github.com/google-ar/WebARonARKit/>
17. Haas, A., Rossberg, A., Schuff, D., Titzer, B., Holman, M., Gohman, D., Wagner, L., Zakai, A., Bastien, J.: Bringing the web up to speed with WebAssembly. In: Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 185–200. ACM (2017)
18. Hill, A., MacIntyre, B., Gandy, M., Davidson, B., Rouzati, H.: Kharma: An open kml/html architecture for mobile augmented reality applications. In: 9th IEEE International Symposium on Mixed and Augmented Reality (ISMAR). pp. 233–234. IEEE (2010)
19. Jones, B., Waliczek, N.: Webxr device api (2018), <https://immersive-web.github.io/webxr/>
20. Klein, G., Murray, D.: Parallel tracking and mapping for small AR workspaces. In: 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, ISMAR. pp. 225–234. IEEE (2007)
21. Leppänen, T., Heikkinen, A., Karhu, A., Harjula, E., Rieki, J., Koskela, T.: Augmented reality web applications with mobile agents in the internet of things. In: Eighth International Conference on Next Generation Mobile Apps, Services and Technologies (NGMAST). pp. 54–59. IEEE (2014)
22. MacIntyre, B., Hill, A., Rouzati, H., Gandy, M., Davidson, B.: The Argon AR Web Browser and standards-based AR application environment. In: Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on. pp. 65–74. IEEE (2011)
23. Marchand, E., Uchiyama, H., Spindler, F.: Pose estimation for augmented reality: a hands-on survey. IEEE transactions on visualization and computer graphics (2015)
24. Molnar, D., Vilck, J., Ofek, E., Moshchuk, A., Wang, J., Gal, R., Shapira, L., Burger, D.C., MacIntyre, B., Livshits, B., et al.: Protecting privacy in web-based immersive augmented reality (Jun 13 2017), uS Patent 9,679,144
25. Rambach, J., Pagani, A., Stricker, D.: [POSTER] Augmented Things: Enhancing AR Applications leveraging the Internet of Things and Universal 3D Object Tracking. In: 2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct). pp. 103–108. IEEE (2017)
26. Schneider, M., Rambach, J., Stricker, D.: Augmented Reality based on Edge Computing using the example of Remote Live Support. In: IEEE International Conference on Industrial Technology (ICIT) (2017)
27. Swenson-Healey, E.: The javascript event loop: Explained (2013), <https://blog.carbonfive.com/2013/10/27/the-javascript-event-loop-explained/>
28. Tomasi, C., Kanade, T.: Detection and tracking of point features (1991)
29. Vert, S., Dragulescu, B., Vasii, R.: LOD4AR: Exploring Linked Open Data with a Mobile Augmented Reality Web Application. In: International Semantic Web Conference (Posters & Demos). pp. 185–188. Citeseer (2014)
30. Walczak, K., Wiza, W., Wojciechowski, R., Wójtowicz, A., Rumiński, D., Cellary, W.: Building Augmented Reality Presentations with Web 2.0 Tools. In: International Joint Conference. pp. 595–605. Springer (2015)
31. Wang, C., Feng, Y., Guo, Q., Li, Z., Liu, K., Tang, Z., Tung, A., Wu, L., Zheng, Y.: ARShop: a cloud-based augmented reality system for shopping. Proceedings of the VLDB Endowment **10**(12), 1845–1848 (2017)