Project Esky: an Open Source Software Framework for High Fidelity Extended Reality

Damien Constantine Rompapas
hyperlethalvector92@gmail.com

Charlton Rodda
charlton.rodda@gmail.com

Bryan Christopher Brown
bryan@highrockstudios.com

Noah Benjamin Zerkin
noazark@gmail.com

Alvaro Cassinelli
cassinelli.alvaro@gmail.com

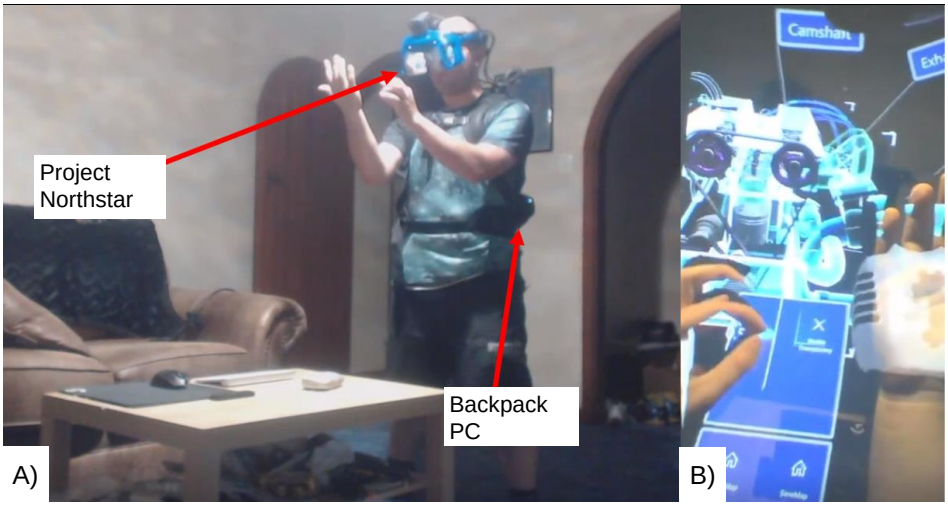

Figure 1: We present Project Esky, an eXtended Reality (XR) framework that enables high fidelity XR content on any hardware configuration. A) A user wearing an OST-HMD with a 6DOF head tracker, hand tracker, and PC is able to develop high fidelity experiences with hand-gesture interactions. B) An image through the lens of the OST-HMD, showing a user interacting with the engine e-learning example displayed in the accompanying video.

ABSTRACT

Recent years have seen an explosion in eXtended Reality (XR) experiences. Until now, researchers and developers were limited to relatively expensive, closed-source, and consumer-level hardware when creating high-fidelity XR experiences, as these were the only devices that can create a high-fidelity XR experience out of the box with minimal setup times. In this paper, we present Project Esky, a complete open-source modular software framework that enables the rapid creation of high-fidelity XR experiences on any combination of display and tracker hardware. Our framework includes several components that handle 6DOF head tracking with head pose prediction to minimize latency, hand interactions with virtual content, and network components to facilitate between-user co-located experiences. Our framework also includes an asynchronous rendering pipeline that simplifies the representation of optical distortions, as well as a simplified planar-based temporal re-projection technique to minimize rendering latency. It is our hope that the techniques described in this paper, and the open source software implementations that support it, will be a stepping stone towards bringing high-fidelity XR content to researchers and hobbyist users alike.

CCS CONCEPTS

• **Software and its engineering** → **Development frameworks and environments**; • **Human-centered computing** → **Interactive systems and tools**; **Mixed / Augmented reality**.

KEYWORDS

Augmented Reality, Mixed Reality, Extended Reality, Development tools, Software Frameworks

1 INTRODUCTION

Recent years saw an explosion in eXtended Reality (XR) experiences for consumers, typically developed on commercial grade optical see-through devices such as the Microsoft HoloLens 2 [11] and Magic Leap [5]. These devices enable end-users to engage with high-fidelity XR content via hand interactions or a hand tracked 6DOF controller, and are capable of high-fidelity immersive experiences with minimal set-up times. Alternative XR display hardware is available in open source domains [15, 19]. These systems provide an open end-to-end hardware solution for XR content, but at reduced fidelity, as many of the technical software advancements

available in industry-standard headsets and frameworks are missing. Legacy frameworks, such as DWARF [2] and MORGAN [14], lack compatibility with modern hardware. Alternatively, modern OpenXR [7] frameworks such as Monado [12] rely on mesh based representations of optical systems. For new optical distortion models, Monado requires a new distortion rendering driver implementation, increasing the time taken to test and deploy new optical configurations. Furthermore, Monado does not include techniques for head pose prediction, and temporal re-projection. ILLIXR [8] is the most similar to Project Esky as a framework, with many of its modules handling tracking, pose prediction, temporal re-projection and undistortion, but lacks any means of calculating the extrinsic parameters between attached sensors. This is especially required for XR setups that show a view of the real world, otherwise virtual content may not be correctly registered to the desired real world locations. HoloKit[1] is a mobile OpenXR alternative, but is also missing temporal re-projection and head pose prediction, limiting the fidelity of the experiences it provides. In regards to temporal re-projection, while several platforms include it as part of their pipeline, they are closed-source with implementation details largely unknown. OpenWarp [18], is an open source alternative that utilizes raymarching or mesh based techniques with currently no known integration into an OpenXR solution, furthermore, the raymarching approach can be simplified further with the assumption that our rendered content lies on the same plane as our display's focal depth with the caveat that there is a possible loss of visual quality.

In this paper, we describe a complete open source software framework for developing XR applications and experiments on any display and tracker combination that addresses the shortcomings of related work. We do this by including a means of between sensor calibration, a simplified undistortion representation, and a simplified planar based temporal re-projection technique. Akin to ILLIXR, we pipe the native texture pointers from a rendering engine into a separate native, asynchronous renderer, as well as running tracker communications as a low-level API set, we are able to achieve the minimum latency for any given hardware configurations, and obtain the benefits the tracker provides (spatial mapping, relocalization, and video passthrough previews). Our framework also supports co-located experiences by saving the map provided by the tracker, sharing, and re-localization against the loaded map, in combination with built in networking libraries. We also support hand interactions with virtual content by means of the Mixed Reality Toolkit and a content authoring pipeline that allows users to create persistent XR content. Esky aims to be the software solution that enables any combination of display, head tracking, and hand tracking hardware with an easy to use game engine, minimizing setup and deploy times. The result allows for any hobbyist, developer, or researcher to achieve any XR related goals. This paper expands upon the previous demonstration [6] implementing temporal re-projection and head pose prediction to deal with tracker and renderer latency as described in the paper's future work section. Overall, the core contributions of this work are:

- The description and open source (See Appendix A) implementation of an modular framework enabling game engine and rendering pipelines high fidelity XR capabilities.
- A simplified approach to represent optical distortions, universalizing the render pipeline to fit any optical distortion model without modifying the source code.
- An automated between-sensor calibration pipeline, allowing us to accurately place the eye rendering cameras where we expect the user's real eye to be, and calculate the tracker offsets relative to the user's eye.
- A simplified temporal re-projection technique based on plane-ray intersection to account for frame render latencies, crucial for minimizing visual latency of XR content.

While this work is in its early stages, and lacks formal verification of the XR content quality we achieve, the results of our framework allow anyone with any display and tracking hardware configuration to create experiences with visual XR fidelity comparable to high end XR hardware such as the Microsoft Hololens and Magic Leap, with relative ease. Finally we present the source code of the implementation, allowing anyone to take the knowledge obtained from this paper and cross reference it to a practical example, customizing it for their needs. It is our hope that the work described in this paper, along with the open source solution shared, will be a stepping stone towards bringing high-fidelity XR content to researchers and users alike.

2 SOFTWARE FRAMEWORK DESIGN

The following section describes the design and implementation details for Project Esky. Any XR framework requires a means of connecting a 6 Degree of Freedom (6DOF) head tracking unit, input tracking (in our case, hand tracking) module, and a rendering pipeline for projecting undistorted images into our real world space. These components require us to first model the distortion properties of the optical system, then render an undistorted image. Once that is achieved we need to align all tracking sensors, so that they orient around the head worn display's pivot point, typically located at the bridge of the nose or the center of the eye. Finally, optimal XR experiences requires a motion to photon latency of less than 20ms [10] to prevent motion sickness, and therefore require a means of predicting head poses and some means of compensating for renderer frame latencies during runtime. Each component is built separate, with minimal data transmission between components to allow for full modularity. Figure 2 shows the complete pipeline for Esky, including all components and their communications.

2.1 Modelling the optical system for distortion

Before the HMD can be used, we need to model the optical system so that the image presented to the user appears undistorted, resulting in all straight rendered lines appearing straight in the user's view. Project Esky is compatible with the following methods of calibration and undistortion.

V1: 3D representation of the display undistortion. Previously developed by leapmotion [13]. This pipeline uses two stereo cameras that are calibrated together for intrinsic and extrinsic parameters, then placed behind the lens of the OST-HMD, so that it can observe a monitor placed in front of the HMD. Then, by rendering an inverted virtual monitor on the headset in the same position as the real monitor in the world, one can optimize the placement of the real vs virtual screens. If the two versions of the monitor matched

[1]HoloKit: https://holokit.io

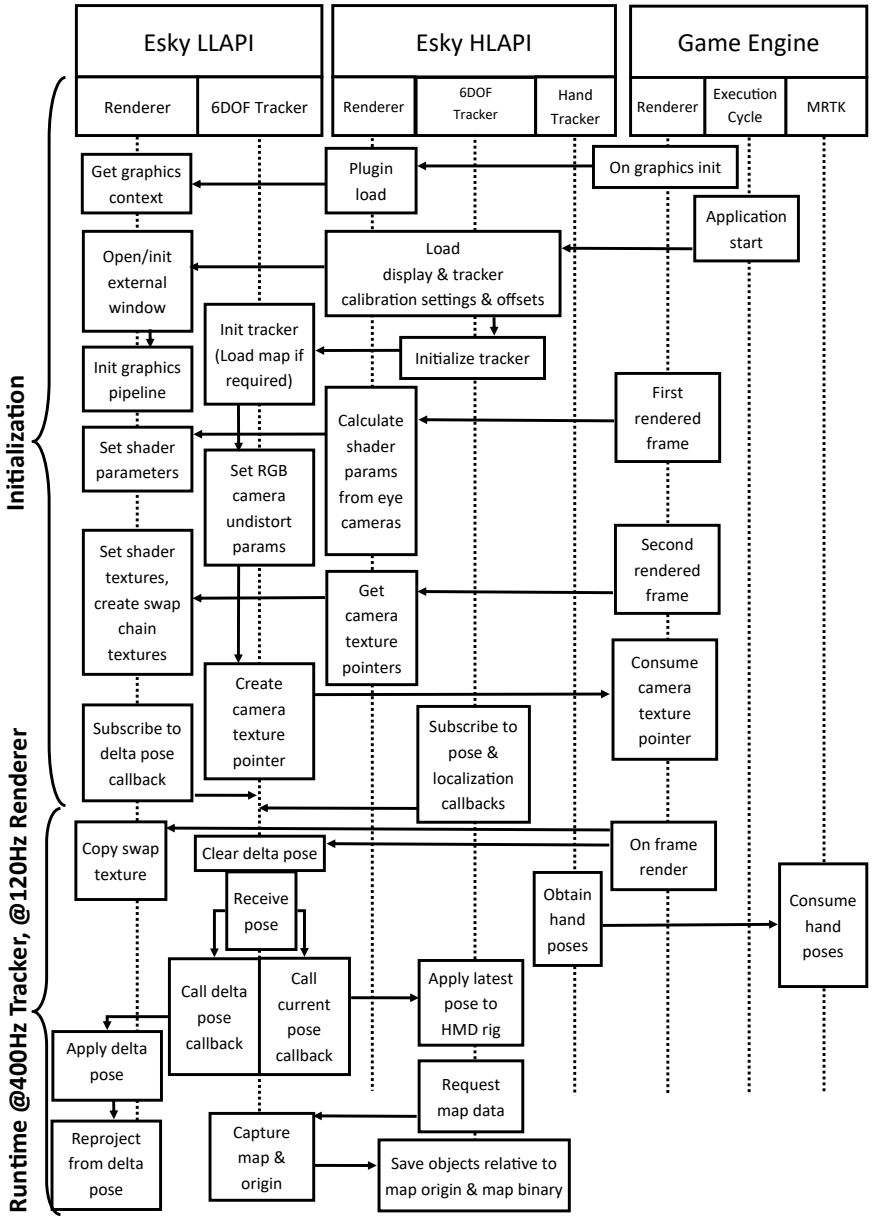

Figure 2: The framework pipeline. Project Esky's Low-Level Renderer consumes the native texture pointers of the hooked rendering/game engine. The rendering/game engine also subscribes to the LLAPI's tracker component to obtain 6DoF head poses. The hand tracking component transforms the poses of the attached hand tracking hardware (in our case, UltraLeap's Leap Motion Controller), feeding it into Microsoft's Mixed Reality Toolkit.

up perfectly, they would additively cancel out to uniform white. The virtual monitors will jitter within each step, performing a check to see if the optimized solution produces a canceled out image, eventually converging on the correct solution, outputting a series of look up values for creating an undistortion mesh. The rendering pipeline then projects the results of the left and right eye render textures onto the undistortion mesh to produce an undistorted image (then displayed on screen). This render pipeline produces the most accurate undistortion result since it calculates the screen and eye positions in real world space, and allows adjustment for several inter pupilary distances. However, it requires a complex set up that might not be available to all end-users.

V2: 2D representation of the display undistortion. The V1 calibration has an expensive requirement of two stereo cameras, and an additional monitor for displaying a calibration pattern, often out of reach for most hobbyists. This required us to simplify the process

down to a single stereo camera pair, placed behind the view of the headset. We utilize binary-coded structured light homography [9] to compute the depth of each pixel displayed on the HMD, in order to collect a series of undistortion points. These points allow us to create a look up table, resolving a 3rd-order 2D polynomial from the 2D screen space to the corresponding point on the now undistorted virtual viewport. While this calibration method is faster to perform compared to previous undistortion mapping algorithms, it is not perfect, and exhibits distortion at the extreme edges of the display. It also limits the Inter Pupillary distance to the stereo camera's between sensor distance (typically 64mm).

V3: Look Up Texture undistortion. If we were to represent the optical distortion of the headset with a different approach to the ones described above, we would need to significantly modify the underlying renderer to support the newer technique. While not necessarily a calibration pipeline in itself, we can instead render the results of the new optical calibration pipeline as a 2D (or 3D for multiple IPD representations) look up table texture, allowing us to simplify any undistortion model into a much simpler look up technique. With this technique, the incoming screen-space UV coordinate is transferred to an output screen-space UV coordinate via a texture that acts as a look up table. This frees the dependency of creating a specific render pipeline for any newer calibration techniques, simplifying and universalizing the rendering process.

2.2 Computing the relative offsets for the head and hand tracking sensors

Before the 6DOF and hand tracking can be utilized with our now-calibrated HMD, their offsets from the user's viewport need to be calculated. Esky has two approaches to compute the relative offsets.

Online Hand-based Alignment: A user holds their hand in front of the leap motion tracker, which shows the virtual hand with some offset, then the user presses a button, freezing the virtual hand in place, allowing the user to match the virtual hand's pose and orientation with their real hand. Doing this several times captures the corresponding points between the initial fingertip pose and orientation. Then by using a simplified Kabsch Solve method to align the two point sets, we can resolve the transform between the hand tracker and the user's eye, as well as the 6DOF tracker with respect to the hand tracker. While this method does work, and requires less hardware to perform, it can lead to errors generated by user input.

Offline Visual Marker Alignment: This process involves a stereo camera placed behind the lens of the OST-HMD, with an ARuCO marker placed in the view of all sensors. The relative pose between each sensor is computed automatically by inferring the transform between each sensor and the detected ARuCO marker. This technique involves the use of extra hardware, but allows for an automated calculation of the required pose offsets. This also has the benefit of calculating where the user's eyes are expected to be placed behind the headset (therefore allowing us to place the stereo rendering cameras where the user's eyes would be in virtual space).

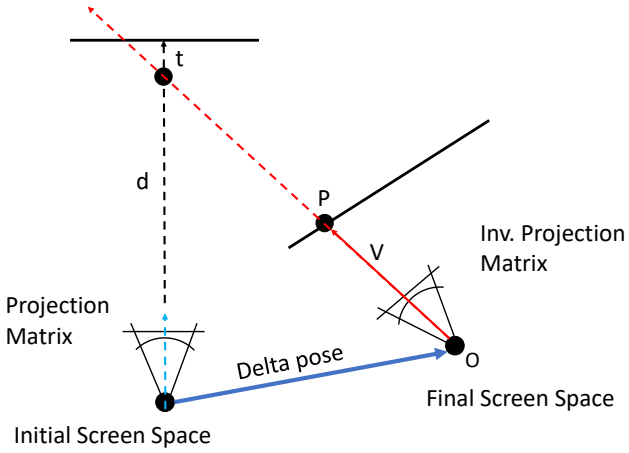

Figure 3: Our ray-plane intersection based temporal re-projection technique. We calculate a pixels point in the new pose frame as the intersect between the new view ray and a projected plane from the origin of the initial pose frame.

2.3 Rendering the image, compensating for tracking jitter, and tracking/rendering latencies

One of the core challenges behind any XR framework is dealing with the latency from the rendering engine used, and incoming poses from the tracker that typically exhibits jitter. Project Esky includes a set of low level API libraries that handle rendering, and tracking components. The feature set of the renderer and tracking include techniques to correct for between-frame latencies, and jitter in the poses received from tracking sensors.

To handle the jitter of incoming poses, we filter the output 6DOF pose using a one-euro-filter [4], filtering the translation and orientation to smooth out values. Then, to compensate for tracking latency, we predict the user's head pose by calculating between-frame positional and angular velocities, projecting our head poses to future frames until we receive newer poses.

Once we have a frame-perfect set of poses, we need to compensate for rendering delays, in particular when the rendering engine used experiences low frame rates (and therefore large delays in rendered frames). Like ILLIXR [8], Project Esky's LLAPI renderer is an asynchronous rendering pipeline that consumes two native texture pointers passed from a rendering/game engine (one for each eye) and uses a fragment shader to undistort and render the images to screen. Additionally, we use a plane-ray intersection technique for temporal re-projection, taking a previously rendered frame, and reprojecting it to future poses based on a delta pose between the previously rendered frame, and current timestep. By assuming that our rendered content will lie at the same depth as our display's focal point, we simplify the raymarching approach of OpenWarp [18] at the potential cost of visual quality. Figure 3 shows a diagram for how our simplified temporal re-projection algorithm works. Overall, we perform the following steps:

(1) Calculate the delta between the pose of the initially rendered frame, to the current timestamp.

(2) Resolve the look up for undistortion
(3) Project a depth probe along the Z-forward axis of the initial frame from screen space into world space using the virtual projection matrix, and an assumed plane depth (in our case, 25cm from the user, the focal length of our OST-HMD).
(4) Calculate the ray from the initial pose origin, and the projected depth probe as normalized vector **d**
(5) Calculate the ray from the final pose origin **O**, and it's projected depth probe as **V**
(6) Calculate the intersection point between ray **d** and **V** as time t
(7) Obtain the newly reprojected point as $\mathbf{V} * t$
(8) Back project the newly reprojected point into screen space using the inverse virtual projection matrix

The result allows us to smoothly reproject previously rendered frames into new coordinate spaces, effectively interpolating between low-update rendered frames. Please see the accompanying video for an example result.

2.4 Working with the Esky XR framework

Currently, Esky's Low Level API (LLAPI) C++ libraries have been integrated into the Unity game engine, with High Level API (HLAPI) C# wrappers, although any game engine or rendering pipeline can consume the LLAPIs. Figure 2 shows the initialization process of first obtaining the graphics context from the game engine for use by our asynchronous renderer, then opening the external render window on the headset, setting any tracker offsets, and setting any shader parameters for undistortion and temporal re-projection. Also during this initialization process, the Esky HLAPI wrappers subscribe to pose, sensor image, and localization events passed by the LLAPI tracker components. During runtime, the LLAPI passes pose received events with the latest poses received by the 6DOF tracker, applying it to the relative unity Transform. Additionally, the tracker module passes the delta between the last rendered pose, and future poses directly to the renderer module, with every frame rendered by unity resetting the initial pose saved by the tracker module. It is this separation of code, with pointer functions being passed between components that allows for Esky to be modular. Finally, Esky includes code to parse hand tracked input from a leap motion controller into Microsoft's Mixed Reality Toolkit (MRTK), facilitating natural hand-gesture based interactions with virtual content, however replacing this component is straightforward by feeding in the calculated hand joint data (for example feeding hand tracked joints from MediaPipe [20] as a replacement for the LeapMotion joints).

The re-localization events along with the MRTK allow users to pre-author their environment, and save the map data from the tracker, along with the pose of each virtual object with respect to the origin of the tracked map, into a single, sharable binary file. Additionally, Esky includes a networking library for co-located experiences. By sharing the map used for tracking between two devices, we are able to co-localize the devices in the same virtual space, and share the poses relative to the origin of the tracked map. This includes the techniques described by Rompapas et al. [17], compensating for synchronized pose latency via pose prediction, and interpolation. The result is a smooth between-device co-located AR experience.

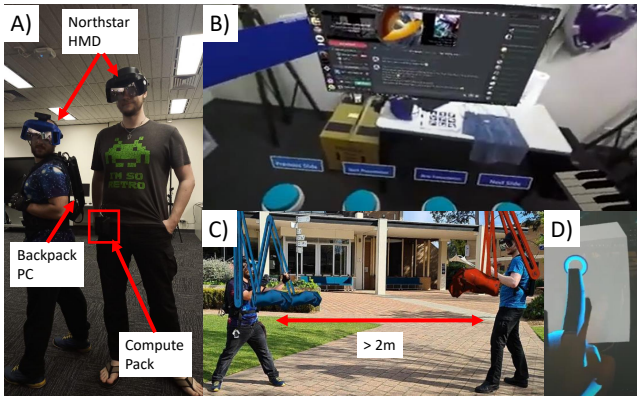

Figure 4: A) The hardware we deploy Project Esky on. B) A virtual desktop and powerpoint presentation application. C) RockemBot Boxing, a co-located social-distancing safe boxing game. D) VisuoTouch, a visuo haptic system to visually induce haptic feedback effects.

3 HARDWARE DEPLOYMENT AND USE CASES

Currently, although not limited to the devices below, we have deployed Project Esky on the following Hardware Configurations:

- Computer Hardware
  - Backpack PC 1: HP VR G2,
  - Backpack PC 2: MSI VR One, CPU: i7-6820HK, GPU: GTX 1070, Ram: 16GB
  - Customized Compute pack: AMD Ryzen Embedded V-Series V1605B, Radeon Vega 8 Graphics, 2 GHz, 4GB RAM
- Trackers
  - Intel Realsense T265/T261
  - Stereolabs ZED/ZED Mini/ZED 2
  - Optitrack Tracking Systems
- Displays
  - Project Northstar [19]
  - Project Triton [1]

Using these hardware configurations, several demonstrations were presented at the ACM Interactive Surfaces and Spaces (ISS) 2020 and the IEEE Interactive Symposium on Mixed Reality (ISMAR) 2020 conferences, each focusing on the different components and features that Esky provides.

Desktop Mirroring as part of Powerpoint Presentation control in AR: A part of the Project Esky demonstration in ISS 2020[6]. Part of the demonstration included the usage of a plane placed in the view of the user, which mirrored the desktop of the backpack PC. This effectively rendered a virtual monitor in front of the user, along side controls for power point presentation. The demonstration showcased the basic functionality of Esky, including hand interactions with virtual content, tracking map saving and loading, and spatial mapping.

Co-located shared XR experiences: RockemBot Boxing [3] is a co-located collaborative experience where two users who stand at distances larger than 1.5 meteres (as part of the current COVID-19 social distancing requirements), and throw virtual punches beyond hand tracking volumes using repeated hand motions towards the opponent. To facilitate the between-user interaction, the authors used Esky's built in networking solution for hand and head pose synchronization with respect to the co-located map's origin space.

Experimental visual pseudo haptic experiences: One of the critical issues in XR is the inability to determine whether we have interacted with a mid-air virtual object if the user does not have any haptic feedback equipment (such as a vibration motor, or haptic controllers). VisuoTouch [16] aimed to alleviate this limitation by a visual means, rendering a virtually rigged IK finger that rests against a physical object during mid air interactions, replicating the visual feedback of a real finger's action against a real world object. This rendering allows users to accurately gague how far they've pressed into a virtual object.

4 CONCLUSIONS AND FUTURE WORK

In this paper, we describe a complete open source software framework for developing high-fidelity XR experiences comparable to off-the-shelf high-fidelity XR solutions. Our software framework includes a simplified rendering pipeline capable of representing any distortion model required by an optical see-through setup, and a simplified planar-based temporal re-projection technique to minimize rendering and tracking latency. Our framework also includes calibration methods to compute the between-sensor offsets, integration with Microsoft's Mixed Reality Toolkit for hand-gesture based interactions, and a means of creating co-located experiences through map and pose sharing techniques.

This work is still in its early stages and requires formal verification via visual quality metrics, frame timing, tracking latency, and an evaluation of the modularity and effectiveness of the toolkit. ILLIXR already implements these validation metrics. Therefore, re-implementing in Esky is a viable approach to validation. Nonetheless, we hope that the open source implementation of our framework described in this paper will enable researchers and hobbyists to develop high-fidelity XR experiences on any combination of display and tracking hardware.

REFERENCES

[1] Graham Atlee. Last Accessed 2021, Feb. Project Triton. https://project-north-star.gitbook.io/project-north-star/

[2] Martin Bauer, Bernd Bruegge, Gudrun Klinker, Asa MacWilliams, Thomas Reicher, Stefan Riss, Christian Sandor, and Martin Wagner. 2001. Design of a component-based augmented reality framework. In *Proceedings IEEE and ACM International Symposium on Augmented Reality*. IEEE, 45–54.

[3] James Cambpell, Eleanor Barnes, Jack Douglas Fraser, Bradley Twynham, Xuan Tien Pham, Nguyen Thu Hien, Geert Lugtenberg, Nishiki Yoshinari, Sarah Al Akkad, Andrew Gavin Taylor, et al. 2021. RockemBot Boxing: Facilitating Long-Distance Real-Time Collaborative Interactions with Limited Hand Tracking Volumes. *IEEE International Symposium on Mixed and Augmented Reality (ISMAR) 2020* (2021).

[4] Géry Casiez, Nicolas Roussel, and Daniel Vogel. 2012. 1€ filter: a simple speed-based low-pass filter for noisy input in interactive systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2527–2530.

[5] Magic Leap Company. Last Accessed 2021, Feb. Magic leap. https://www.magicleap.com/en-us

[6] Rompapas Damien Constantine, Daniel Flores Quiros, Charlton Rodda, Bryan Christopher Brown, Noah Benjamin Zerkin, and Alvaro Cassinelli. 2020. Project Esky: Enabling High Fidelity Augmented Reality on an Open Source Platform. In *Companion Proceedings of the 2020 Conference on Interactive Surfaces and Spaces* (Virtual Event, Portugal) *(ISS '20)*. Association for Computing Machinery, New York, NY, USA, 61–63. https://doi.org/10.1145/3380867.3426220

[7] The Khronos Group. Last Accessed 2021, Feb. OpenXR standard. http://www.open-xr.com/

[8] Muhammad Huzaifa, Rishi Desai, Xutao Jiang, Joseph Ravichandran, Finn Sinclair, and Sarita V. Adve. 2020. Exploring Extended Reality with ILLIXR: A new Playground for Architecture Research. arXiv:2004.04643 [cs.DC]

[9] Douglas Lanman and Gabriel Taubin. 2009. Build your own 3D scanner: 3D photography for beginners. In *ACM SIGGRAPH 2009 Courses*. 1–94.

[10] Steven M LaValle, Anna Yershova, Max Katsev, and Michael Antonov. 2014. Head tracking for the Oculus Rift. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 187–194.

[11] Microsoft. Last Accessed 2021, Feb. Microsoft HoloLens 2. https://en.wikipedia.org/wiki/HoloLens_2

[12] Monado. Last Accessed 2021, Feb. OpenXR Runtime. https://monado.dev/

[13] Leap Motion. 2019. Bending Reality: North Star's Calibration System. https://blog.leapmotion.com/bending-reality-north-stars-calibration-system/

[14] Jan Ohlenburg, Iris Herbst, Irma Lindt, Torsten Fröhlich, and Wolfgang Broll. 2004. The MORGAN framework: enabling dynamic multi-user AR and VR projects. 166–169. https://doi.org/10.1145/1077534.1077568

[15] Wayne Piekarski and Bruce Thomas. 2002. ARQuake: the outdoor augmented reality gaming system. *Commun. ACM* 45, 1 (2002), 36–38.

[16] GS Rajshekar Reddy and Damien Constantine Rompapas. 2020. VisuoTouch: Enabling Haptic Feedback in Augmented Reality through Visual Cues. *IEEE International Symposium on Mixed and Augmented Reality (ISMAR) 2020* (2020).

[17] Damien Constantine Rompapas, Christian Sandor, Alexander Plopski, Daniel Saakes, Joongi Shin, Takafumi Taketomi, and Hirokazu Kato. 2019. Towards large scale high fidelity collaborative augmented reality. *Computers & Graphics* 84 (2019), 24–41.

[18] Finn Sinclair. Last Accessed 2021, Feb. Open Warp. https://github.com/Zee2/openwarp

[19] Ultraleap. Last Accessed 2021, Feb. Project North Star. https://project-north-star.gitbook.io/project-north-star/

[20] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. 2020. MediaPipe Hands: On-device Real-time Hand Tracking. *arXiv preprint arXiv:2006.10214* (2020).

A ONLINE RESOURCES

The source code implementations of the work described in this paper, and any other support pages/information can be found at the following web pages:
LLAPI: https://github.com/HyperLethalVector/ProjectEskyLLAPI
HLAPI (Unity Integration):
https://github.com/HyperLethalVector/ProjectEsky-UnityIntegration
Project Esky setup (Unity Integration):
https://project-north-star.gitbook.io/project-north-star/software/esky
Discord server for support: https://discord.gg/UcUmgJAa
Supplimental Video (Online): https://youtu.be/EyobMiSPp_I